

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA



TESIS

**HYBRID INTELLIGENT SYSTEMS
FOR MEDICAL IMAGE CLASSIFICATION**

PRESENTADA POR

JESÚS ROGELIO GARCÍA AGUIRRE

**COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE
DOCTORADO EN INGENIERÍA ELÉCTRICA**

SEPTIEMBRE 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



HYBRID INTELLIGENT SYSTEMS FOR MEDICAL IMAGE
CLASSIFICATION

BY

JESÚS ROGELIO GARCÍA AGUIRRE

FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

IN

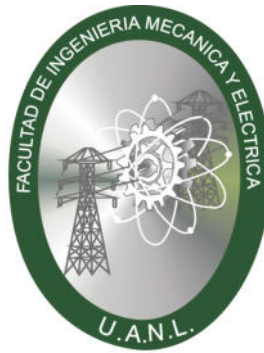
Electrical Engineering

SAN NICOLÁS DE LOS GARZA, NUEVO LEÓN, MÉXICO
SEPTEMBER 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



HYBRID INTELLIGENT SYSTEMS FOR MEDICAL IMAGE
CLASSIFICATION

BY

JESÚS ROGELIO GARCÍA AGUIRRE

FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

IN

Electrical Engineering

SAN NICOLÁS DE LOS GARZA, NUEVO LEÓN, MÉXICO
SEPTEMBER 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
Facultad de Ingeniería Mecánica y Eléctrica
Posgrado

Los miembros del Comité de Evaluación de Tesis recomendamos que la Tesis "HYBRID INTELLIGENT SYSTEMS FOR MEDICAL IMAGE CLASSIFICATION", realizada por el estudiante Jesús Rogelio García Aguirre, con número de matrícula 1983174, sea aceptada para su defensa como requisito parcial para obtener el grado de Doctorado en Ingeniería Eléctrica.

El Comité de Evaluación de Tesis

Dr. Luis Martín Torres Treviño
Director

Dra. Eva María Navarro López
Co-director

Dra. Griselda Quiroz Compeán
Revisor

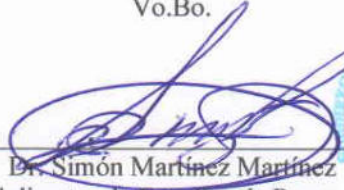
Dr. Alberto Cavazos González
Revisor


Dr. Juan Ángel Rodríguez Liñán
Revisor

Dr. Romeo Sanchez Nigenda
Revisor

Dr. Erick de Jesús Ordaz Rivas
Revisor

Vo.Bo.


Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado



Institución 190001

Programa 514601

Acta Núm. 362

Ciudad Universitaria, a 28 de Octubre de 2024.

Acknowledgement

I would like to express my gratitude to the Consejo Nacional de Humanidades, Ciencias y Tecnologías in Mexico for their financial support for my studies. I also thank the Rochester Institute of Technology for granting me access to their facilities and providing additional financial assistance that significantly contributed to my research.

The PhD journey is a wild one. Each of us who embarks on this path faces different challenges and deals with a fair share of failures and successes. Yet, pursuing a PhD is an adventure that should not be undertaken alone. Those of us who are lucky (and stubborn) enough to reach the end of this journey have learned from many people and have had the support of even more. I want to express my gratitude to everyone involved in my journey, from friends and family to colleagues, evaluators, and everyone in between. In the following paragraphs, I will acknowledge those who played a crucial role in my odyssey.

I want to begin by thanking my thesis directors, Dr Eva Navarro-López and Dr Luis Torres-Treviño. Dr Luis, thank you for allowing me to pursue my own research agenda and supporting my decisions along the way. Dr Eva, I do not have the words to thank you properly. You have supported me beyond what anyone can expect, taught me so much, made me do better, and accompanied me through. You showed me the type of academic that I hope to be. Without your engagement and dedication to your students, this thesis would be a lesser version of itself.

Thank you to all my friends for keeping me sane during these years. Ana and Manuel, thank you for accompanying me through the ordeals of the PhD and for being like my family in Monterrey. Pablo, Paco, and Diana thank you for your friendship and support all these years. You always cheer me up and are like a lifesaver whenever I feel lost in my head. Miguel, thank you for always listening

to me going on and on about the same things and pushing some sense into me when I need it. Karla, thank you for cheering for me since we were kids. One can not ask for a better cousin. Lastly, I want to thank Mike and Eva for their friendship and for treating me like family while I was abroad. You always made me feel at home.

Finally, I want to thank my family. Mom and Dad, thank you for raising me the way you did, allowing me to be myself while keeping me on the right path. Mom, thank you for all your love, time, and advice. Dad, thank you for being my role model and always giving me all of your support and trust. Karen and Claudia, my sisters, thank you for showing me the way and for helping me to shape my own. Sergio and Trini, for always supporting me in every way you can and being like my brothers.

*To Leo, Emma, and Teo, my beloved nephews,
thank you for bringing joy to my life and
reminding me what is important.*

Contents

Abstract	10
Resumen	12
Glossary	14
1 Introduction	17
1.1 Core concepts and current context of the field	18
1.1.1 Image classification	18
1.1.2 Deep learning	20
1.1.3 Automatic medical image classification	21
1.1.4 Intelligent systems for medical image classification: Cur- rent state and open challenges	23
1.2 Thesis overview	28
1.3 Contributions of the thesis	29
1.4 Publications list	31
2 Background	32
2.1 Machine Learning	35
2.2 Convolutional Neural Networks	38
2.2.1 Fully connected layers	39
2.2.2 Convolutional layers	40
2.2.3 Backpropagation	43
2.2.4 Activation functions	54
2.2.5 Downsampling operations	58

2.2.6	Other elements	60
2.3	Learning procedure	63
2.3.1	Loss function	64
2.3.2	Optimizer	65
2.3.3	Transfer learning	68
2.4	Hyperparameters	69
2.4.1	Design hyperparameters	69
2.4.2	Learning hyperparameters	73
2.5	Evaluation metrics for image classification	78
3	Genetic algorithm-based hyperparameter optimization	80
3.1	Introduction	81
3.2	Hyperparameter optimization	83
3.2.1	Genetic algorithms for hyperparameter optimization	86
3.3	Methodology	88
3.3.1	Search space	88
3.3.2	Genetic algorithm	91
3.3.3	Training resulting model	97
3.4	Results	97
3.4.1	Datasets	97
3.4.2	Experimental setting	102
3.4.3	Testing the convergence of Gen-CNN	102
3.4.4	Assessing Gen-CNN against state-of-the-art hand-design models for medical image classification	103
3.5	Discussion and conclusions	105
4	Artificial astrocytes to complement artificial neural networks	113
4.1	Introduction	114
4.2	Artificial Neuron-Glia Networks	115
4.3	Methodology	118
4.3.1	Artificial astrocyte	118
4.3.2	Analysis of the CNNs	121
4.4	Results	122

4.4.1	Datasets	122
4.4.2	Experiments	122
4.5	Discussion and conclusions	132
5	Hybrid learning with model fusion and hyperparameter optimization	138
5.1	Introduction	139
5.2	Methodology	140
5.2.1	Genetic algorithm	140
5.2.2	Hybrid learning with weight averaging	142
5.3	Results	144
5.3.1	Datasets	144
5.3.2	Experiments	145
5.3.3	Data augmentation	147
5.4	Discussion and conclusions	149
6	Discussion and conclusions	153
6.1	Hyperparameter optimization	153
6.2	Neuro-inspired models	155
6.3	Model fusion	157
6.4	Ethical implications	158
6.5	Conclusions	158
	Bibliography	161
	List of Figures	187
	List of Tables	190

Abstract

This century's technical advances in hardware and the advent of big data set the stage for the thriving of deep learning, an archetype of machine learning (a sub-field of artificial intelligence). Therefore, there are high expectations for deep learning applications in areas where data is rich and fundamental, such as medical image analysis. Nevertheless, deep learning alone might not be enough to fulfill the requirements of real-world applications, and factors like the system's complexity and high resource demand start to inhibit the adoption of this technology.

Hybrid intelligent systems combine different artificial intelligence paradigms to overcome the limitations of singular methods. This thesis covers the research of hybrid approaches that incorporate deep learning models for the automatic classification of medical images with other artificial intelligence approaches to improve the efficiency and practicalness of the final system.

The issue of the increased complexity of these systems is tackled by researching the pivotal characteristics of these types of models and developing an optimization algorithm that automatically generates effective and low computational complexity models for a target task, permitting the practical use of this technology to non-expert machine learning users. Then, a neuro-inspired novel unit is proposed to transform the current paradigm of convolutional neural networks, which are mainstream for image analysis systems. The proposed unit aims to align artificial neural networks more closely with the learning mechanisms of the human brain.

Finally, a hybrid learning method is proposed to automatically generate optimized models of image classification by merging the previously proposed approaches and model fusion, a strategy that has been successful in recent years in

generating deep learning models with better classification performance than singular models and lower computational demand than other prevailing methods in the field.

Keywords:

Image classification, Medical image analysis, Convolutional neural networks, Hyperparameter optimization, Genetic algorithm, Artificial astrocytes, Model fusion, Hybrid intelligent systems

Resumen

Los avances técnicos de este siglo en equipo computacional y la llegada de la era de big data sentaron las bases para el progreso del aprendizaje profundo, un arquetipo del aprendizaje automático (un subcampo de la inteligencia artificial). Por esta razón, existen grandes expectativas para las aplicaciones de aprendizaje profundo en áreas donde los datos son abundantes y fundamentales, como el análisis de imágenes médicas. Sin embargo, el aprendizaje profundo por sí solo podría no ser suficiente para satisfacer los requisitos de las aplicaciones del mundo real, y factores como la complejidad de estos sistemas y su alta demanda de recursos comienzan a inhibir la adopción de esta tecnología.

Los sistemas inteligentes híbridos combinan diferentes paradigmas de inteligencia artificial para superar las limitaciones de los métodos singulares. Esta tesis aborda la investigación de enfoques híbridos que incorporan modelos de aprendizaje profundo para la clasificación automática de imágenes médicas con otros enfoques de inteligencia artificial para mejorar la eficiencia y la practicidad del sistema en su conjunto.

La problemática causada por la creciente complejidad de estos sistemas se aborda investigando las características fundamentales de este tipo de modelos y desarrollando un algoritmo de optimización que genere automáticamente modelos efectivos y de baja complejidad computacional para una tarea específica, lo que permite el uso práctico de esta tecnología a usuarios inexpertos en aprendizaje automático. Luego, se propone una nueva unidad de inspiración neurológica para transformar el paradigma actual de las redes neuronales convolucionales, que son componentes fundamentales de sistemas de análisis de imágenes actualmente. La unidad propuesta tiene como objetivo alinear las redes neuronales artificiales más estrechamente con los mecanismos de aprendizaje del

cerebro humano.

Finalmente, se propone un método de aprendizaje híbrido para generar automáticamente modelos optimizados de clasificación de imágenes mediante la combinación de los enfoques propuestos anteriormente y la fusión de modelos, una estrategia que ha tenido éxito en los últimos años en la generación de modelos de aprendizaje profundo con mejor desempeño de clasificación que los modelos singulares y menor demanda computacional que otros métodos prevalecientes en el campo del aprendizaje profundo.

Palabras clave:

Clasificación de imágenes, Análisis de imágenes médicas, Redes neuronales convolucionales, Optimización de hiperparámetros, Algoritmo genético, Astroцитos artificiales, Fusión de modelos, Sistemas inteligentes híbridos

Glossary

AI artificial intelligence.

ANN artificial neural network.

AutoML automated machine learning.

BO bayesian optimization.

CAT-ACC categorical accuracy.

CBAM convolutional block attention module.

CCA canonical correlation analysis.

CNGN convolutional neuron-glia network.

CNN convolutional neural network.

conv convolutional.

CT computed tomography.

CV computer vision.

DL deep learning.

F1 macro-average F_1 score.

FC fully connected.

FLOP floating point operations.

FPS frames per second.

GA genetic algorithm.

GELU gaussian error linear unit.

GS grid search.

h-swish hard-swish.

HPO hyperparameter optimization.

LLM large language model.

Macro-REC macro-recall.

Macro-PREC macro-precision.

Macro-SPEC macro-specificity.

MCC Matthew's correlation coefficient.

ML machine learning.

MLP multi layered perceptron.

MRI magnetic resonance imaging.

NAS neural architecture search.

NFLT no free lunch theorem.

NGN neuron-glia network.

NLP natural language processing.

PAPA population parameter averaging.

PWCCA projection weighted canonical correlation analysis.

ReLU rectified linear unit.

RS random search.

SeLU sigmoid linear unit.

SGD stochastic gradient-descent.

SNN spiking neural networks.

SVM support vector machine.

TanH hyperbolic tangent.

ViT vision transformer.

Chapter 1

Introduction

Artificial intelligence (AI) is perhaps the field that holds more expectations currently. Namely, machine learning (ML), a subfield of AI encompassed by data-driven approaches, enjoys a prosperous present thanks to the technological advances of the century, the well-established research that started several decades ago, and the advent of the big data era. However, no progress goes without challenges, and those of modern AI are many. AI has previously experienced periods of apparent prosperity that generated anticipation about the possible advances it would bring to several fields, which failed to deliver. Now, from the hand of deep learning (DL), a subtype of ML, AI, has a new opportunity to transform almost every aspect of the contemporary world.

DL has been particularly successful in applications with spatially arranged data, such as images. Therefore, this technology holds great promise as a means to develop the next generation of systems for medical image analysis, with significant expected outcomes, including increased diagnosis efficiency, empowerment of physicians, and enabling low-cost universal access to medical diagnosis. However, DL, like any other approach, has its weaknesses, and it alone might not fulfill the requirements of medical image analysis applications. We shall learn from past failures and not put all our hopes in a single strategy. Hence, hybrid approaches take the spotlight as an alternative to reach the promised land.

This thesis covers the research for hybrid intelligent systems that fulfill the current required and desired features of medical image classification systems. As

the name suggests, intelligent hybrid systems integrate different approaches or paradigms of AI to improve the characteristics and performance of the singular methods.

Initially in this thesis, two distinct branches are investigated with different objectives. The first research direction regard is automated machine learning (AutoML) based on the optimization of simple convolutional neural network (CNN) models, offering a solution to the problem of the increased complexity of state-of-the-art DL systems that inhibits its scalability and practical use in limited-resources conditions, as well as its spreading to the less-experienced community of users, which slows down its application in other research areas. The second research path involves developing and analyzing a new brain-inspired unit compatible with current DL systems, bringing them closer to the brain’s neural networks, which are their original inspiration, intending to improve the current learning paradigm of DL. Finally, this thesis culminates by presenting a third approach, connecting the two former researched methods on a single framework that automatically produces CNN models for image classification. The proposed approach consists of the hybrid learning of the optimal model architecture and training conditions (hyperparameters) along with its parameters for the target task’s domain.

We shall start at the beginning. So, the following section of this introductory chapter serve to acquaint with the fundamental concepts for the subjects treated in this thesis and the current state, challenges, and expectations of intelligent systems for medical image classification.

1.1 Core concepts and current context of the field

1.1.1 Image classification

Image classification can encompass many different tasks in the automatic interpretation of an image. However, its essence is assigning labels to images from a predefined set of categories [1]. To analyze an image, we use the characteristics of the objects in it. Perceiving such characteristics, like colors, textures, spatial locations, etcetera, seems trivial for humans since evolution increased our

dependency on visual information, and our brains adapted to process, analyze, and interpret visual cues [2]. However, for machines, this has proven to be an extraordinary endeavor. Computer vision (CV) focuses on the inverse problem of retrieving information from images and reconstructing the properties of the objects or situations that they depict [3].

The very first step to processing images using machines is representing visual information in a manner computers can comprehend. Here, we introduce the term digital image. A finite number of spatially-arranged elements called pixels compose a digital image [4]. Pixels are discrete finite quantities that describe the intensity of a given color in the spatial location of the image they represent. Color images use three color intensities (red, blue, and green) to recreate the colors in the visible spectrum. Digital images that use this system consist then of three different stacked color maps (called channels). There are other types of image representations. In this thesis, we use the above-described one.

By this, digital images are just arrangements of pixels: how can we give meaning to these quantities to describe the properties we see in the images they describe? The difference between how humans perceive an image and its computational representation is known as the semantic gap [1]. To close this gap, we encode the digital representations of an image to quantify its attributes. This process is called feature extraction and is central for any CV task, including image classification.

CV relies on physics-based and probabilistic models [3] to extract the features from a digital image to analyze its contents by quantifying its characteristics. For image classification, the features are the input for a classification algorithm (often simply referred to as a classifier), which typically is a ML model. The classifier outputs the label to categorize the image from a predefined set of categories.

There are two types of features in the context of image classification: (1) hand-engineered or hand-crafted features, which are extracted using image processing algorithms and mathematical techniques, and (2) machine-learned features that, as the name suggests, are automatically learned and extracted using ML. The last decade's technological advances allowed the application of DL for automatic feature extraction with remarkable success [5–8]. Thus, CV is a multidisciplinary field closely related to AI.

1.1.2 Deep learning

Deep learning is an ML subfield that uses stacked layers of artificial neural networks (ANNs) to learn patterns from vast amounts of data [7, 8]. Even though ANNs draw inspiration from the brain, they do not try to mimic its functionality realistically [1]. Instead, ANNs are basic models with parallelisms with the behavior of the brain [1]. The field of ANNs has been around since the 1940s, with its origins in studies of other related fields, such as cybernetics and connectionism [1, 9].

In 1943, McCulloch and Pitts [10] came up with the first neural network model capable of statistical learning [1, 11]. Rosenblatt [12] made the next breakthrough for ANNs during the 1950s with the Perceptron model, which architecture is still the ground for many ANNs models. Also, it set the basis of the gradient-based training methods for learning the synaptic weights on ANNs [1].

In the 1960s, the field of AI experienced its first boom, a period of big hype impulsed by expert systems (another AI subfield). CV started as a formal study field at the beginning of the 1970s as a consequence of the first AI boom, and it was initially thought of as the perceptual component of an over-enthusiastic agenda to empower machines with human intelligent behavior [3].

Yet, soon after, Minsky and Papert proved that the perceptron (a simple neural network model with a single layer) was a linear classifier incapable of solving non-linear problems [13]. This situation, along with the disillusion provoked by the unreal expectations in expert systems [11], caused the beginning of the first AI winter, a period of nearly a decade where the funding and interest for the field almost disappeared [1, 11].

Then, in the 1980s, expert systems were once again responsible for a shift in the field of AI. Cases of success of expert systems on industrial applications renovated the interest of investors in AI, and a new wave of funding for research impulsed the field [11]. This period is known as the second AI boom [11].

During these years, the development of the back-propagation algorithm, the application of non-linear activation functions on the neurons, and the technological progress allowed ANNs researchers to train multi-layer feedforward neural networks capable of learning non-linear functions [1, 7]. Thanks to these advance-

ments, ANNs became universal approximators capable of approximating any continuous function [1]. Nonetheless, there is no assurance that a specific ANN can learn the parameters needed to represent a given function [1].

At the end of the 1980s, the difficulty of translating the new research to the industry and the maintenance costs of the expert systems that originated the second AI boom set the start of the second AI winter [11]. That marked the end of the expert systems' predominance in the field. ML approaches have received more attention [11] ever since.

Nowadays, we are living in the third AI boom, driven by DL [1, 11]. The availability of fast computers, specialized hardware, and big training datasets made it possible to train networks with many more hidden layers capable of hierarchical learning [11]. A particular type of ANN led the way for applied DL: CNNs, which advanced the field of CV due to their powerful properties for image analysis [1, 14, 15].

CNNs became the standard for CV applications during the last decade [14, 16, 17], leaving many other CV techniques superfluous. Nevertheless, the advent of transformers [18], another type of ANN, ushered in a revolution for DL applications. Transformers were originally conceived to process sequential data, especially for natural language processing (NLP). Yet, due to their outstanding performance, this type of ANN quickly spread to other tasks, including image classification [14, 19].

1.1.3 Automatic medical image classification

Nowadays, AI is extending throughout almost every aspect of our lives. And, people have come to believe, expect, and accept that it will have a key supporting role in medicine [20–22].

In medical care, the critical tasks of diagnosis and prognosis continually rely on the analysis of different types of medical images, such as X-rays, magnetic resonance imaging (MRI), and computed tomography (CT), among others, which routinely, physicians assess based on their knowledge and experience [23, 24]. ML systems are powerful support tools for human experts in medical image analysis [20, 22–26], and, at research level, DL applications have been exceptionally

successful in image-intensive medical specialties [8, 20–23, 27] for image classification, segmentation, detection, and localization [22–24, 28].

Different studies show that DL has matched or even exceeded human performance in image classification [7, 8, 29], including those of the medical domain [20, 22, 23, 25, 26, 30]. For instance, a study compared the performance of a CNN against 21 board-certified dermatologists in the classification of biopsy-proven clinical images, finding that the DL algorithm has comparable performance to that of the specialist physicians [30]. Therefore, there are high hopes for future DL applications in the medical field [8, 11, 20–23].

Many authors have pointed out the appeal of assessing medical images using AI-based algorithms. Some of the most expected outcomes are:

1. Increased efficiency and objectivity: AI-based systems are much more efficient than actual physicians at analyzing massive amounts of data quickly [8, 20, 22, 23, 26, 29], without affecting their analysis by tiredness or personal biases. Nevertheless, DL approaches are data-driven and can be affected by biased sampling [8, 11, 21], and the resulting models are only as good as the data used for their development [7, 20, 31]. Biased sampling is notably dangerous in the development of ML systems with medical applications since it can have repercussions on the diagnosis, prognosis, or treatment quality of patients.
2. Algorithms for medical image classification capable of operating on mobile devices can have a prominent social impact by enabling low-cost universal access to medical diagnosis [30]. Still, these systems should have a supporting role in the medical practice, and they can not replace physicians [23].
3. AI-based systems for medical image analysis can cooperate with physicians during practice as a second observer, boosting their performance [8, 22, 25, 26, 28] or even for educational purposes [32].

A desired property of future AI-based systems for medical image classification is the capability of performing in real-time in a clinical setting [25, 28]. As stated before, in the near future, AI will have a supporting role in medicine. There are areas such as gastroenterology where physicians have to make decisions in

real-time during exploratory procedures according to the findings in the images [25, 28].

A worry that keeps coming up about DL in medicine is the black-box condition [8, 11, 21, 22, 25–27], which refers to the lack of interpretability in DL models. Even so, interpretability is hardly ever defined, and it is not a monolithic concept since it has many different interpretations [33]. In that sense, for DL applications in medicine, interpretability corresponds to the possibility for a human to understand the relationship between the system’s predictions and the information that led to those outcomes [34].

1.1.4 Intelligent systems for medical image classification: Current state and open challenges

Convolutional Neural Networks vs Transformers

Let us begin by addressing the fundamental issue. At present, there is no prevailing DL paradigm for CV applications. A lot has happened in this regard during the realization of the research of this thesis. Let us briefly cover the last few years of the transformers vs CNNs dilemma.

The self-attention mechanism is the foundation of transformers [18]. Soon after this type of ANN overcame the other methods in NLP, many studies adapted the technique of self-attention to CNNs [19]. However, the vision transformer (ViT) [19] released in 2020 marked a new era by challenging the supremacy of CNNs in CV applications [17]. The ViT architecture makes the fewest changes possible to the original Transformer to use it for CV tasks, namely, image classification. Hence, ViT does not include any image-specific inductive bias beside the first layer that *patchifys* the input [17].

ViTs achieve outstanding results when pre-trained with large datasets and transferred to tasks with fewer data [19]. The idea that ViTs scaling properties enabled them to outperform standard CNNs models on image classification with larger models when big datasets are available became popular. Nevertheless, ViTs did not spread as the standard backbone for CV applications, and CNNs prevailed in the field for their desirable properties, such as better computational

efficiency [17]. A major limitation of the standard ViT in CV is its global attention design that delivers quadratic complexity with respect to the input size, which is a problem as the resolution of the inputs increases [14, 17].

To overcome the vanilla ViT limitations for CV, hierarchical transformers adopt a hybrid method, which introduces sliding window behavior (similar to that of CNNs) to transformers [17]. A successful example of this approach is the swin transformer [14], which achieves state-of-the-art performance in many CV tasks besides image classification [17]. Nevertheless, this success indicates that: *“the essence of convolution is not becoming irrelevant, rather, it remains much desired and has never faded”*, Zuang et al. [17].

Transformers were starting to relegate CNNs for their better performance in some CV tasks despite the other advantages that these have. Then, a study [17] demonstrated that the dominance of transformers approaches in CV tasks was not specifically for the scaling properties of transformers but also due to differences in the training procedures and design choices of their elements. This study updated a standard CNN with state-of-the-art computational modules and training techniques, achieving competitive or even better performance than transformer-based models while preserving the efficiency of CNNs.

The debate about whether it is better to use CNNs or transformers for image classification is still open [35, 36]. Numerous recent studies are comparing the performance of both paradigms, delivering different conclusions [35–42]. Although the majority claims that Transformer-based approaches show better properties than CNNs in CV tasks, many studies deliver misleading results, affected by the choice of architectures, training conditions, and evaluation metrics [35]. A recent comparative analysis between the state-of-the-art architectures of both paradigms found no meaningful differences in their performance in different CV tasks when trained in optimal conditions for each of them [35].

Ensamble models and model fusion

Ensemble models are an approach that fuses several models to perform a single task [43, 44]. Ensemble models usually have better classification performance than individual models at the cost of an increase in computational demand and

model size [28, 43–45]. Yet, it is a prevalent practice for the development of AI solutions [7, 43, 44, 46, 47], including those for medical image classification [28, 46–48].

In the last few years, new types of model fusion have arisen, particularly weight averaging [43–45, 49, 50], a technique that fuses the parameters of similar models, resulting in an individual model with better performance, preventing the characteristic overload of computational resources generated by traditional ensemble techniques.

Brain-inspired approaches

As stated before, the current ANN paradigm uses connectionist models that are a simplified abstraction of their biological counterparts. DL uses arrangements of stacked ANNs. The specific configuration of an arrangement of ANNs is referred to as architecture. The everlasting pursuit to improve the performance of these models in terms of accuracy has led to an increase in the complexity of their architectures, provoking issues of interpretability, generalization, and efficiency.

Since AI has historically found inspiration in the human brain function, many authors have stated that new brain-inspired solutions can be found to these current issues [51–53]. Therefore, research in this direction has acquired importance during the last few years.

Neuromorphic computing and spiking neural networks (SNN), a type of neural network based on the dynamic modeling of the biological behavior of neurons, have gained attention as a solution for the development of models for real-world applications due to their higher energetic and computational efficiency when compared to traditional computing and ANNs [54–56]. However, SNNs have not reached the classification performance of ANNs and are complicated to train because of their inadequacy to function with gradient-based optimization [54, 55], which is the core of modern ML training algorithms.

On a related topic, there is research attempting to close the gap between the traditional connectionist models of neurons that prevail in today’s ANNs and contemporary neuroscience findings that have shifted the previously believed paradigms about how the brain learns and forms memories. Some exciting new

methods try to introduce new elements into the current ANN paradigm that account for elements that are now known to be fundamental for the brain’s information processing. Two predominant examples are studies developing and implementing units that mimic the dendrites’ function into ANNs [57–59] and research on artificial glial units to complement ANNs [60–64].

AI hybrid systems

In the context of AI, hybrid systems, hybrid approaches, hybrid methods, or hybrid frameworks denote the combination of different knowledge representation schemes, AI models, and learning procedures to solve a task [65]. Several authors have identified the potential of hybrid approaches to enhance the performance of intelligent systems for image classification, particularly in the medical domain [8, 22, 66–69]. Hybrid methods are viewed as a possible solution to overcome the scarcity of data and interpretability issues [8].

Either of the discussed current research directions seems to point in a similar direction: hybrid systems. Whether the combination of the inductive biases of CNNs and the self-attention mechanism of transformers, the combination of multiple ML algorithms with ensemble approaches or fusion techniques, or the combination of paradigms jointing the connectionist models with brand-new neuroscience-inspired units.

On the other hand, the complexity of hybrid models can be a limitation to their adoption in medical image analysis in real-world settings, particularly if they pose a liability for their computational demand and because their complexity may lead to suboptimal performance and generalizability [8].

Open challenges and main takeaways

Image classification confronts many challenges, including the selection of training data and methods for image processing and pattern recognition [8, 66, 70], and, in the medical domain, validating the algorithm with medical expert knowledge is particularly noteworthy [11, 21, 22, 25, 28, 29, 66]. For DL developments in the medical field, a central challenge is the availability and size of datasets, reasons why DL models for medical image classification must rely on techniques

such as transfer learning and data augmentation [8, 28, 29].

Besides achieving high classification performance, medical image classification algorithms must identify the significant regions of the images for classification [66]. Class activation maps or saliency maps have been recognized as a solution to the interpretability issue of DL for medical image analysis [25, 34, 71]. Quoting Reyes et al. (2020) [34]: “*saliency maps can be integrated easily into the radiology workflow because they work at the pixel level; hence, these visualization maps can be fused or merged with patient images and computer-generated results.*”

Another concern of implementing DL in medicine is the computational resources requirements, which may hamper the scalability and practical use in limited-resources environments [7, 8, 29, 31]. Previous studies have demonstrated that a proper configuration of hyperparameters is a determinant factor for the final classification performance of a DL model in the medical field [8].

Zhuang et al. [17] reached an intriguing conclusion in their research that culminated in a new family of CNNs that surpassed the state-of-the-art models for image classification (Transformer- and CNN-based). It points out that the techniques and methods they used for their development were not at all new since they have been studied individually before but not collectively. This argument is influential for this thesis since it suggests that we might already have the tools we need to achieve the desired performance for a given application of AI. We only need to find the correct configuration of existing techniques. A clinically competent DL method should consider classification performance in terms of accuracy and inference speed. An optimized combination of CNN models can be the solution to this [28].

Judging the capacity of a DL model only by its number of parameters or size is misleading since any ANN is a universal approximator (having enough number of parameters), yet the interaction of inductive biases and training procedures is fundamental in finding models that effectively generalize outside the scope of the training data [35]. Previous studies have demonstrated that a proper configuration of hyperparameters is key for the final classification performance of a DL model in the medical field [8].

These arguments motivated this research for a hybrid approach to optimizing smaller CNNs models to meet the classification performance and inference speed

requirements for medical image classification.

1.2 Thesis overview

A summary of the content of the thesis is presented to serve as a guide for the reader. Chapter 1 handles an introductory passage that offers an overview of AI history with a focus on CV and DL to conclude with the current state, open challenges, and expectations of AI for medical image analysis, providing the context in which this thesis has been done. The first chapter also helps to get acquainted with fundamental concepts, such as image classification, CV, ML, DL, etcetera.

Chapter 2 describes in detail all the needed concepts for an in-depth understanding of the research displayed in this thesis from a technical perspective. Readers with experience in ML and DL can skip this chapter, and readers who are novices to the subject of AI, ML, or DL can skim through the chapter and go back to specific sections when needed. Not the whole disclosed information is needed for every single other chapter, but all the information that the reader may need is there to consult at any time.

The research component of the thesis is presented in the subsequent three chapters (3, 4, and 5), each of which stands alone. However, it is recommended that the reader follows them in the order they appear for a better understanding of the significance and scope of the contributions of this thesis as a whole.

Chapter 3 tackles the relevant issue of hyperparameter optimization (HPO). Any ML algorithm comes with the ordeal of selecting the appropriate hyperparameters to extract the capacity of the models in the singular conditions of the task at hand. The hyperparameter configuration of DL systems is particularly challenging because of the complexity of the systems. During this chapter, an analysis is made to determine the influential hyperparameters that shall be optimized to increase the classification performance of CNN models. Then, a genetic algorithm (GA) is proposed as the backbone of a brand-new framework that automatically generates optimized models for image classification. Hence, this chapter covers the development of a hybrid intelligent system that combines a GA for HPO and CNNs for image classification.

A novel type of ANN is proposed in Chapter 4. Recent neuroscience findings

have shifted the way we understand brain operation. DL has its origins in connectionist models of neurons from decades ago. Therefore, it is logical to update these models with contemporary knowledge. This chapter explores the development and analysis of a new artificial unit that resembles astrocytes, a type of glial cell that is now known to be fundamental in learning and memory. The proposed artificial astrocyte is introduced into the existing DL models, creating a new paradigm: convolutional neuron-glia networks (CNGNs).

Chapter 5 takes on the research made in chapters 3 and 4, soothing the weaknesses of the proposed systems by merging them using a model fusion technique, combining their respective attributes in a new hybrid framework that learns both the parameters and hyperparameters of models at the same time and produces a single optimized model for image classification.

Finally, Chapter 6 encompasses the conclusions of the performed research as a whole and provides a discussion of the implications of the results, their significance, and the author's interpretation of them. Additionally, some possible future research directions are analyzed, considering the current context of AI as a technological tool and its transformation potential in our society.

1.3 Contributions of the thesis

This thesis deals with the research and development of hybrid intelligent systems, which combine different methods or paradigms of AI. Therefore, throughout the phases of the thesis, contributions to a variety of AI subfields are made. The specific research contributions are elaborated and discussed in each research chapter (chapters 3, 4, and 5). However, the most notable contributions of the thesis are listed in order of appearance in the following:

- From the vast universe of hyperparameters, a few of them are identified and shown to be enough to shape the classification performance of CNN models when using the techniques of transfer learning and fine-tuning for the image classification task.
- It is proven that models of a single CNN architecture can match or even outperform more computationally complex frameworks and ensemble systems

by properly selecting the found set of influential hyperparameters.

- This thesis includes the development of a GA capable of optimizing categorical, discrete, and continuous hyperparameters, including the selection of the CNN architecture, a unique attribute of the proposed algorithm compared to other HPO schemes in the current literature.
- The GA discussed in the previous point is utilized as the backbone for the development of a framework for the automatic generation of CNN models for image classification, considering the computational requirements of the target task, enabling the application of DL to non-ML-expert users and fastening the spreading of this technology to other fields. The library of the developed framework, Gen-CNN is openly available in [GitHub/RogeGar/Gen-CNN](#).
- By revisiting prior research studies on artificial neuron-glia networks, this thesis proposes an artificial astrocyte unit that mimics the brain's multi-time-scale neuroplasticity and is compatible with the current DL architectures and standard learning algorithms.
- A new archetype of ANNs, CNGNs, is created by introducing the proposed artificial astrocyte model into conventional CNN architectures. The code of the artificial astrocyte, the scheme to transform CNNs into CNGNs, and the training methods are available in [GitHub/RogeGar/Artificial-astrocytes-CNGNs-](#).
- An analysis of the CNGNs is carried out by comparing their learned representations to those of equivalent CNN models, finding that the proposed artificial astrocyte can guide the models to better-performing capabilities without extending their capacity, suggesting that the full potential of the current CNN models is yet to be exploited.
- A new hybrid framework has been designed to simultaneously learn parameters and hyperparameters of CNN models, using the HPO algorithm proposed in this thesis and a model fusion technique. Additionally, this new approach is compatible with the proposed artificial astrocyte unit, working

synergetically with it by optimizing the artificial astrocyte hyperparameters, whereas the astrocytes provide an essential component for the model fusion technique.

1.4 Publications list

Preprints and freely published research articles:

- *Automatic generation of optimized convolutional neural networks for medical image classification using a genetic algorithm.* Rogelio García-Aguirre, Luis Torres-Treviño, Eva María Navarro-López, José Alberto Gonzalez-González. SSRN, <https://ssrn.com/abstract=4167905>, 2022.

Conference articles published in special issues:

- *Hyperparameter optimization for transfer learning in gastrointestinal image classification using an evolutionary algorithm: proof of concept.* Rogelio García-Aguirre, Luis Torres-Treviño, Alberto González-González. 20th Mexican International Conference on Artificial Intelligence. Research in Computing Science, ISSN 1870-4069, 2021.
- *Towards an Interpretable Model for Automatic Classification of Endoscopy Images.* Rogelio García-Aguirre, Luis Torres-Treviño, Eva María Navarro-López, José Alberto Gonzalez-González. Lecture Notes in Artificial Intelligence, doi:10.1007/978-3-031-19493-1_24, 2022. **Best paper award (3rd place) at the 21st Mexican International Conference on Artificial Intelligence.**

Research articles in indexed journals:

- *Development of a muscle electrical stimulation parameter selection method with an intelligent system.* Rogelio García-Aguirre, Luis Torres-Treviño, Griselda Quiroz-Compeán, Angel Rodríguez-Liñan. Engineering Applications of Artificial Intelligence, doi:10.1016/j.engappai.2023.106167, 2023.
- *Gen-CNN: A framework for the automatic generation of CNNs for image classification.* Rogelio García-Aguirre, Eva María Navarro-López, Luis Torres-Treviño. Neural Computing and Applications. Accepted on August 20th, 2024, awaiting publication.

Chapter 2

Background

This chapter elaborates on pivotal concepts and definitions of this thesis research subject from a technical perspective. Since it covers the development of hybrid methods to tackle some of the main current limitations of intelligent systems for medical image classification, DL is a central topic of this thesis. DL is a subfield of ML. Hence, a solid understanding of the ML and ANNs principles is required to navigate the presented research.

For the reader's convenience, the information is presented sequentially, beginning with general ML concepts and moving toward more particular topics necessary to engage with the research subjects of the following three chapters. This chapter offers all the information needed to comprehend every aspect of the research element of this thesis. DL-experienced readers can skip this chapter and come back to check on specific sections when necessary. Newcomers to the ML and DL study are advised to skim through the chapter to get acquainted with the nomenclature and then return to review particular sections in detail if required.

List of Symbols for Chapter 2

This chapter makes use of an extensive number of symbols. For the sake of clarity and the reader's convenience, a list of symbols and their definition is provided in order of appearance.

E	Experience
T	Task
P	Performance metric
D	Target task's data domain
S	Experience in ML (training dataset), $S \subset D$, $S = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$
N	Number of samples or data points in S
x	The data in S that corresponds to the inputs (also known as features, covariates, and predictors)
y	The data in S that corresponds to the outputs (also referred to as labels, targets, or responses) of an ML model
\mathcal{X}	Real-valued input space, $\mathcal{X} \subset S$
\mathcal{Y}	Real-valued output space, $\mathcal{Y} \subset S$
f	Function that maps perfectly \mathcal{X} onto \mathcal{Y} , $f : \mathcal{X} \rightarrow \mathcal{Y}$ for the target task's data domain
h	Hypothesis function that maps \mathcal{X} onto \mathcal{Y} , $f : \mathcal{X} \rightarrow \mathcal{Y}$ for the training dataset and approximates f in the target task's data domain
K	Number of unordered and mutually exclusive labels in the training dataset
C	Number of channels in an image or number of elements in the first dimension a n -dimensional matrix
H	Hight (in pixels) of an image or number of elements in the second dimension of a n -dimensional matrix
W	Widht (in pixels) of an image or number of elements in the third dimension of a n -dimensional matrix
L	Number of layers in an ANN
l_ϕ	ϕ -th layer of an ANN, where l_0 is the layer that receives the inputs and l_L the layer that procudes the network's output \widehat{y}
\widehat{y}	Output of an ANN
g^{l_ϕ}	Output of the ϕ -th layer of an ANN
X^{l_ϕ}	Intput to the ϕ -th layer of an ANN

$\bullet^{conv/FC, l_\phi}$	The superscript <i>conv</i> or <i>FC</i> indicates the type of layer l_ϕ to which an arbitrary variable (\bullet) belongs
θ	Set of all the trainable parameters of an ANN
b	Independent bias term of a neuron
$a(\cdot)$	Arbitrary (non-linear) activation function
$y^{l_\phi, FC}$	Output of the l_ϕ (FC) layer previous to the application of its activation function
$\mathbf{W}^{l_\phi, FC}$	Real-valued set of synaptic weights of an <i>FC</i> layer l_ϕ
$\mathbf{B}^{l_\phi, FC}$	Real-valued set of bias terms of an <i>FC</i> layer l_ϕ
$*$	Convolution operator
$y^{l_\phi, conv}$	Output of the l_ϕ (conv) layer previous to the application of its activation function
$\mathbf{K}^{l_\phi, conv}$	Real-valued set of weights in the linear filters of the kernels in a conv layer l_ϕ
$\mathbf{B}^{l_\phi, conv}$	Real-valued set of bias terms of a conv layer l_ϕ
l_W^ϕ	Width of the layer l_ϕ : number of neurons in the layer l_W^ϕ
\mathcal{E}	Estimated error produced by the network when comparing the produced output \widehat{y} and the training targets y using an arbitrary cost or loss function
I	Input size ($X_C^{l_\phi}$) of layer l_ϕ
J	Output size ($g_C^{l_\phi}$) of layer l_ϕ
$\nabla_{\mathfrak{D}}(\kappa)$	Matrix of partial derivatives of an arbitrary variable κ along the respective variable axes of an arbitrary matrix \mathfrak{D}
\star^{full}	Full-convolution operator
y^{BN}	Output the batch normalization unit inside an arbitrary layer with batch normalization
X^A	Input to the activation function inside an arbitrary layer
X_{ReLU}	Input to a ReLU activation function
$\widehat{y}^{Softmax}, \widehat{y}^S$	Output of the softmax function
μ_β	During trining, mean of the batch of data (X^{l_ϕ}) input to a batch normalization unit. During inference (not training), mean of all the batches of data (X^{l_ϕ}) used during training

σ_β	During training, standard deviation of the batch of data (X^{l_ϕ}) input to a batch normalization unit. During inference (not training), standard deviation of all the batches of data (X^{l_ϕ}) used during training
ϵ	A small positive scalar used to prevent dividing by zero inside a batch normalization unit
\mathcal{E}^{C-ent}	Softmax cross-entropy loss
β	Batch of data
θ'	Set of new parameters computed with an optimization algorithm during training of an ANN, θ' that becomes θ for the next optimization iteration
y^β	Training targets of a batch of data β
\widehat{y}^β	Network's output for a batch of input data x^β
a_{GELU}	GELU activation function
X_{GELU}	Input to the GELU activation function
$\Phi(X_{GELU})$	Standard Gaussian cumulative distribution function of X_{GELU}
X_{SeLU}	Input to the SeLU activation function
$\sigma(X_{SeLU})$	Logistic sigmoid function of X_{SeLU}
$X_{h-swish}$	Input to the h-swish activation function
\mathcal{E}_{WD}	Cost or loss function with weight decay factor
λ	Scalar (hyperparameter) that controls the contribution of the L2 norm of the parameters θ to the loss with weight decay factor
\mathcal{B}	Set of all the bias factors of an ANN, $\mathcal{B} \subset \theta$

2.1 Machine Learning

ML is an AI subfield that deals with algorithms that exploit data to extract information and acquire knowledge to perform a task on a given domain [9, 72]. Essentially, ML uses algorithms with a probabilistic programming core to create models that fit the known data and uses those models to infer things from new, unseen data [73].

An ML algorithm learns from data. However, learning can denote many things. For the sake of clarity and formal analysis, it is imperative to stick with a definition. In the context of ML, a common formal one is that of Mitchell (1997) [74], which says: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .” To define a learning problem, the three components: $\langle T, P, E \rangle$ must be specified [74].

In this scenario, learning is the medium to gain the capability to perform a task [9]. There are different categorizations of ML depending on the task, experience, and performance measure configuration [75]. However, ML algorithms are commonly categorized as supervised, unsupervised, and semi-supervised learning [1].

Supervised learning is the most frequent type of ML, and the image classification systems that this thesis studies are built upon this ML archetype. Therefore, this chapter elaborates on this class of ML. Supervised ML utilizes the target task’s data domain D . Hence, the experience E is a subset S of all the data of the domain D , $S \subset D$ [72], and S is referred to as training dataset [75]. N samples or data points consisting of inputs x (also referred to as features, covariates, or predictors) and their corresponding outputs y (also referred to as labels, targets, or responses) comforms S , $S = \{(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)\}$.

Let us consider then that the target task’s data domain D consists of the input space \mathcal{X} and the output space \mathcal{Y} , whose elements are real-valued. Supervised ML tries to find the mapping function $f : \mathcal{X} \rightarrow \mathcal{Y}$ using the subset S . By the no free lunch theorem (NFLT), no system can unmistakably generalize to all the data points in one domain without experiencing them all [74]. Hence, ML can not find the mapping function f . Instead, it approximates f with another function $h : \mathcal{X} \rightarrow \mathcal{Y}$ that is called a model or hypothesis.

Fundamentally, ML is founded on optimization algorithms that minimize the gap between f and h [73]. The learning algorithms find parameter values that characterize a model that is defined by their hypothesis space (the type of functions that the ML model corresponds to). Therefore, ML has a strong component of parametric programming [74]. Additionally, ML can be analyzed from a probabilistic perspective since it relies on the probabilistic inference of the learned

hypothesis from the training dataset and prior probabilities [74,75].

There are three inevitable error sources in any ML system [74]:

1. The bias during learning towards a given hypothesis in the domain space. It is a source of error when the elected hypothesis fails to represent all the elements in the domain due to underrepresented domain subsets in the training data or misleading evaluation metrics in the training set [74].
2. The variance in the training data can lead to suboptimal models [74].
3. Unavoidable error implicit when trying to learn a non-deterministic function [74].

This thesis deals with the single-label multi-class classification task. Thus, the output space \mathcal{Y} is a set of K unordered and mutually exclusive labels [75]. Specifically, in image classification, the input space \mathcal{X} is the set of images. Consider that an image has three channel colors $C = 3$ (RGB) and a resolution of $H \times W$ pixels. Hence, $\mathcal{X} \in \mathbb{R}^{C \times H \times W}$. Therefore, the input to the first layer (hence to the network) has three channels $X_C^{l_0} = 3$. The number of channels in the input of subsequent layers is equal to the number of channels in the previous layer output $g_C^{l_{\phi-1}} = X_C^{l_\phi}$, with $\phi \in 1, 0, \dots, L$. Note that pixels have integer values in the interval $[0, 255]$, but the assumption of real values is made for the sake of notational simplicity [75].

Again, by the NFLT, no model prevails over all others for all tasks. Yet, for model selection, one has to consider model types whose inductive biases are suitable for the target domain. As exposed in Section 1.1.3, DL models have thrived in CV applications, including medical image classification, and nowadays are amid the CNNs vs Transformers dilemma (Section 1.1.4). Nonetheless, this thesis aims to devise methods for medical image classification, and prominent considerations in this field are the computing resources required for inference and the explainability of results (Section 1.1.3). CNNs have proven to be more computationally efficient [17], are a more mature field with a deeper understanding of their operation, and have been recognized to be able to provide interpretable results for medical applications through saliency maps [25, 34, 71]. Additionally, the CNNs inductive biases are particularly advantageous for image analysis [15, 17, 76, 77], and pose an advantage to Transformers when there is scarce training data [40, 78].

For all these reasons, CNNs are the predominant backbone of the image classification systems studied in this thesis.

2.2 Convolutional Neural Networks

CNNs are a type of ANN inspired by the way that the visual cortex of animals processes images [9, 24, 73]. These networks specialize in processing data with a spatially local structure [9, 76]. CNN are like other ANNs with the quirk of using in at least one of their layers the convolution operation (hence the name), which is a type of linear operation [9].

DL models display a layered architecture in which the initial layers in CNNs specialize in learning low-level features such as edges and textures [1, 8]. As the information flows deeper into the network, the features become more and more abstract [1, 8]. Finally, the high-level representations at the final layers contain conceptual information about the training data, which allows the network to automatically perform different tasks on the images like detection, segmentation, or classification [8]. This hierarchical feature learning of CNNs works remarkably well to capture and exploit the intrinsic spatial dependencies in medical images [8].

From a reductive perspective and the image classification point of view, CNNs have two principal components: a feature extractor and a classifier. The feature extractor receives the pixels as inputs that go by a sequence of operations, including convolution, to extract the images' features. Then, the extracted features are the inputs for the classifier, which regularly is a multi layered perceptron (MLP) but can be any other type of classifier, such as a support vector machine (SVM) or a decision tree. The feature extractor is an arrangement of stacked convolutional filters, downsampling (pooling) operations, and non-linear activation functions, amongst other elements.

Let us examine the elements and operation of CNNs. CNNs have at least convolutional (conv) layers and linear layers (generally fully connected (FC)), which for simplicity we will refer to as FC layers. Let $L \in \mathbb{N}$ be the total number of layers in the network. Individual layers are identified as l_0, l_1, \dots, l_L , where l_0 is the first layer (the one that receives the image as input), and l_L is the last layer where we

obtain the network's output \widehat{y} . Each layer l_ϕ , $\phi \in \{0, 1, \dots, L\}$ in the network performs a non-linear transformation $g^{l_\phi}(\cdot)$ on its inputs X^{l_ϕ} . This transformation is typically a composite function of different operations. Let us analyze the specific cases for FC and conv layers.

A note on notation: a superscript indicates the layer and the type of layer to which an arbitrary variable belongs, and subscripts are used for indexing when used with a lowercase symbol and to refer to the dimensions of a matrix when used with an uppercase symbol. For instance, $g_C^{l_\phi, conv}$ is the size of the first dimension (number of channels) of the output of the ϕ -th layer, which is a conv layer. The bold font identifies subsets of the network's trainable parameters set θ on a particular type of layer (the parameters characterize the model, and the training process updates the parameters values to approximate the hypothesis function h to the mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$).

2.2.1 Fully connected layers

The output of a standard neuron is the linear weighted sum of its inputs X^{l_ϕ} and an independent bias term. These weights are referred to as synaptic weights. Considering that an FC layer l_ϕ^{FC} is an arrangement of standard neurons where the network topology is all-to-all, and all the neurons use the same activation function $a(\cdot)$. The output of such an FC layer is

$$g^{l_\phi, FC} = a(y^{l_\phi, FC}) \quad (2.1)$$

where $y^{l_\phi, FC}$ is

$$y^{l_\phi, FC} = \mathbf{W}^{l_\phi, FC} \cdot X^{l_\phi} + \mathbf{B}^{l_\phi, FC} \quad (2.2)$$

where X^{l_ϕ} is the real-valued input vector to layer l_ϕ , $\mathbf{W}^{l_\phi, FC} \in \mathbb{R}^{\|X^{l_\phi}\| \times \|y^{l_\phi, FC}\|}$ is the matrix of synaptic weights of all neurons in l_ϕ and $\mathbf{B}^{l_\phi, FC} \in \mathbb{R}^{\|y^{l_\phi, FC}\|}$ is a vector containing the bias terms for every neuron in l_ϕ . Then, $\mathbf{W}^{l_\phi, FC}$ and $\mathbf{B}^{l_\phi, FC}$ are the trainable parameters of an arbitrary l_ϕ^{FC} . Note that $\|X^{l_\phi}\|$ is the number of elements in the layer's input (number of inputs), and $\|y^{l_\phi, FC}\|$ is the number of elements in the layer's output (number of outputs). For FC layers, inputs and

outputs have one channel and a width of one (are vectors).

2.2.2 Convolutional layers

In CNNs, a conv layer takes a three-dimensional real-valued matrix as input $X^{l_\phi} \in \mathbb{R}^{C \times H \times W}$. These dimensions are referred to as depth C (for the number of channels that define the depth), height H , and width W . Figure 2.1 shows an example of a three-dimensional matrix. If the three-dimensional matrix corresponds to an image, the input data has $C_{in} = 3$ for the three channels in RGB images. The H and W are the height and width of the image in pixels.

$x_{1,1,1}$	$x_{1,1,2}$...	$x_{1,1,W}$		$x_{2,1,1}$	$x_{2,1,2}$...	$x_{2,1,W}$		$x_{C,1,1}$	$x_{C,1,2}$...	$x_{C,1,W}$
$x_{1,2,1}$	\ddots	$x_{1,2,W-1}$	$x_{1,2,W}$		$x_{2,2,1}$	\ddots	$x_{2,2,W-1}$	$x_{2,2,W}$		$x_{C,2,1}$	\ddots	$x_{C,2,W-1}$	$x_{C,2,W}$
\vdots	\ddots	\ddots	\vdots		\vdots	\ddots	\ddots	\vdots	...	\vdots	\ddots	\ddots	\vdots
$x_{1,H,1}$...	$x_{1,H,W-1}$	$x_{1,H,W}$,	$x_{2,H,1}$...	$x_{2,H,W-1}$	$x_{2,H,W}$,	$x_{C,H,1}$...	$x_{C,H,W-1}$	$x_{C,H,W}$

Figure 2.1: Representation of a three-dimensional matrix of depth C , height H , and width W .

Convolutional layers are similar to FC layers. But, instead of applying a simple weight to their inputs X^{l_ϕ} , they apply the convolution operation on them, using sets of two-dimensional real-valued linear filters, also known as kernels.

Let us define the (two-dimensional) convolution operation $(*)$ as the sum of the element-wise products of each pair of elements in two matrices. The matrices involved in the convolution do not require to have the same shape. One matrix slides over the other, performing the operation until all the elements of both matrices are covered. Figure 2.2 shows an example of the convolution of two matrices.

The output of the convolution operation in a conv layer with a three-dimensional input X^{l_ϕ} with $X_C^{l_\phi}$ channels and an arbitrary kernel \mathbf{K} is denoted by

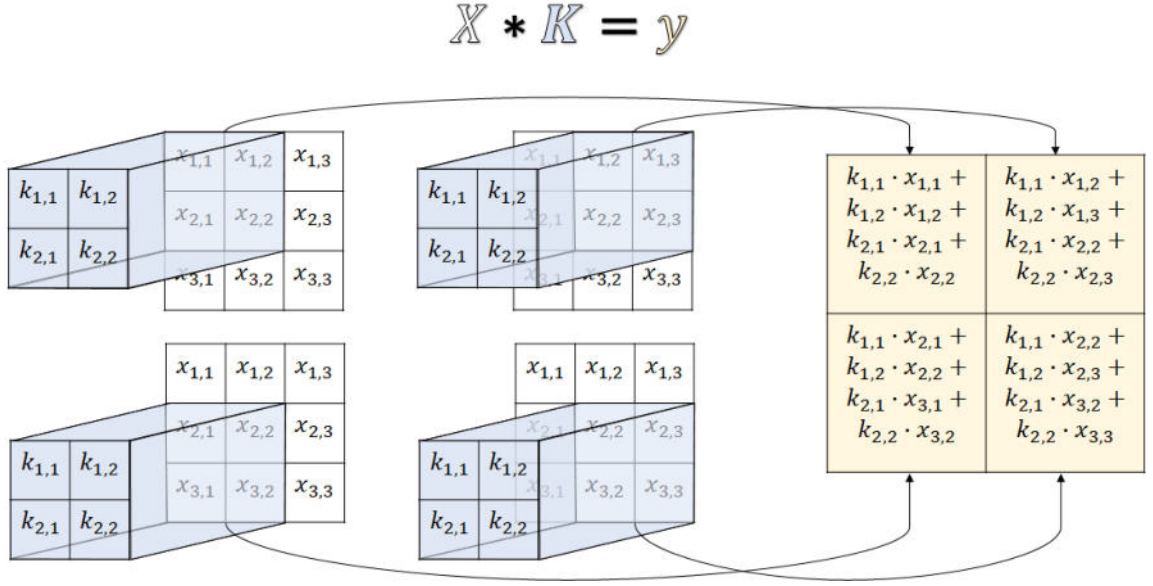


Figure 2.2: Example of the convolution operation ($*$) using two matrices. $X * K = Y$ (X in white convolves with K in blue and produces Y in yellow).

$$y^{l_\phi, \mathbf{K}} = (X^{l_\phi} * \mathbf{K})(i, j) \triangleq \sum_{c=1}^{X_C^{l_\phi}} \sum_{m=1}^{\mathbf{K}_H} \sum_{n=1}^{\mathbf{K}_W} X_{c, m+i-1, n+j-1} \mathbf{K}_{c, m, n} \quad (2.3)$$

where \mathbf{K}_H and \mathbf{K}_W are the height and width of the kernel, respectively. Note that since the input to layer is three-dimensional, the kernel also is three-dimensional, being a set of as many two-dimensional linear filters as the number channels in the input, and the output of the convolution between the input and a kernel is the summation of the convolution of each channel in the input with its respective two-dimensional linear filter in the kernel. Therefore, the output of a convolutional operation between the two three-dimensional data matrices (input and kernel) results in a two-dimensional matrix. Finally, i and j are the indexes of the two-dimensional output values. This operation that many ANNs libraries perform is actually the cross-correlation function [9].

In a conv layer, a neuron is a set of a kernel and a bias term. Therefore, the output produced by a single neuron is the sum of the bias and the output of the convolution operation of its input and its kernel. Hence, the bias is a two-

dimensional matrix of the same height and width as the output of the convolution. A conv layer l_ϕ^{conv} is an arrangement of this type of neurons, and the output of a conv layer is the three-dimensional matrix resulting from stacking the outputs of its neurons and the application of a non-linear activation function $a(\cdot)$. Thus, the output of such a conv layer is

$$g^{l_\phi, conv} = a \left(\begin{bmatrix} b_1 + y^{l_\phi, \mathbf{K}_1} \\ b_2 + y^{l_\phi, \mathbf{K}_2} \\ \vdots \\ b_{l_W^\phi} + y^{l_\phi, \mathbf{K}_{l_W^\phi}} \end{bmatrix} \right) \quad (2.4)$$

where l_W^ϕ represents the width of the layer, indicating the number of neurons in the layer (thus, number of kernels in conv layers).

Conv layers have two kinds of trainable parameters: the kernels' weights $\mathbf{K}^{l_\phi, conv}$ and biases $\mathbf{B}^{l_\phi, conv}$. The individual kernels are three-dimensional real-valued matrices with the same depth as the input ($\mathbf{K}_C^{l_\phi, conv} = X_C^{l_\phi, conv}$). On the other hand, they have an arbitrary height ($\mathbf{K}_H^{l_\phi, conv}$) and width ($\mathbf{K}_W^{l_\phi, conv}$) typically of the same value, $\mathbf{K}_H^{l_\phi, conv} = \mathbf{K}_W^{l_\phi, conv}$, which is the same for all the kernels in a conv layer.

Each kernel has a matrix of bias terms of the same shape as the feature maps produced with the convolution (output of the convolution). The shape of a conv layer output depends on many things. Overall, the output's shape depends on some design criteria, like the kernel's $\mathbf{K}_H^{l_\phi, conv}$ and $\mathbf{K}_W^{l_\phi, conv}$. The depth of the output ($g_C^{l_\phi, conv}$) is equal to the width of the conv layer (l_W^ϕ). Thus, the set of weight of all the kernels in a conv layer is a four-dimensional matrix $\mathbf{K}^{l_\phi, conv} \in \mathbb{R}^{X_C^{l_\phi, conv} \times \mathbf{K}_H^{l_\phi, conv} \times \mathbf{K}_W^{l_\phi, conv} \times l_W^\phi}$, and $\mathbf{B}^{l_\phi, conv} \in \mathbb{R}^{l_W^\phi \times \mathbf{K}_H^{l_\phi, conv} \times \mathbf{K}_W^{l_\phi, conv}}$, as Figure 2.3 depicts. However, commonly in practice, all the elements in the bias matrix are equal for a given kernel, so it is the same as reducing two dimensions and using scalar biases for every kernel. Thus, for a conv layer, the biases become a vector $\mathbf{B}^{l_\phi, conv} \in \mathbb{R}^{l_W^\phi}$.

Convolutional layers bring principally three advantages for DL models. (1) Sparse interactions [7,9,73], meaning that the kernels' size usually is much smaller than the input size, with the convolution the kernel slides over the input and not every element of the input and output has to have a specific synaptic weight for

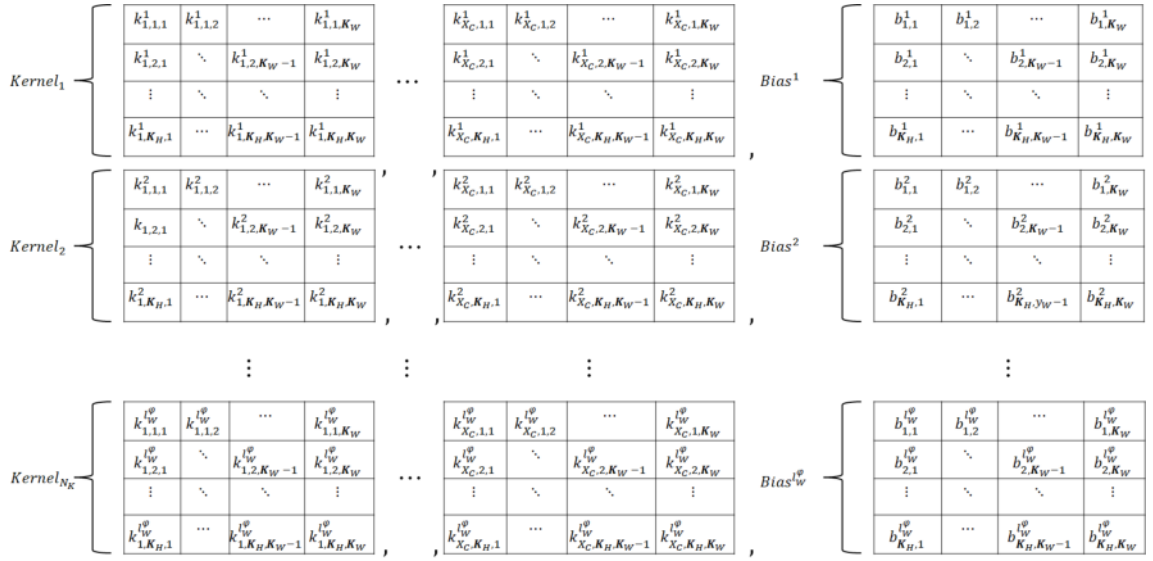


Figure 2.3: Trainable parameters of a convolutional layer with l_W^ϕ kernels of height K_H and width K_W and takes an input with X_C channels. In the figure, the superscript denotes the kernel number, and the subscripts index over the kernel's depth, height, and width, respectively. In practice, every neuron has only one associated bias. Hence, all the elements in a bias's two-dimensional matrix are the same.

their interaction, significantly reducing memory necessities for the model and increases its efficiency. (2) Parameter sharing, the trainable parameters of the kernel are used for different interactions with the inputs during the convolution, so a few parameters correspond to hundreds of interactions instead of having one parameter for the connection of every element of the input and output (such as in FC layers) [7, 9, 72, 73]. (3) Translation equivariance, the parameter sharing attribute causes that if an input changes, the output changes in the same way [7, 9]. That is helpful in the context of image analysis since it does not matter if the zone of interest for the given task is in different zones of the images.

2.2.3 Backpropagation

CNNs, like any other machine learning scheme, need to go through a training process. The training of ANNs is a cycle of three steps: (1) forwarding the training data into the network to obtain the network's output $\widehat{y}(x, \theta)$ (note that the

network's output depends on the inputs and the learned parameters θ), (2) calculating the "error" \mathcal{E} of the network against the desired targets y using an arbitrary cost function $\mathcal{E}(\widehat{y}(x, \theta), y)$, and (3) updating the network's parameters using a gradient-based optimization algorithm to minimize the cost function.

The rest of this section delivers a nuanced explanation of the backpropagation of error in ANNs, a fundamental step for using gradient-based algorithms to train neural networks, which are currently the core of the standard learning (training) algorithms. The intelligent systems investigated in this thesis utilize such algorithms. Moreover, Chapter 4 covers the development of a new element that engages in the learning procedure. However, the proposed element is compatible with the backpropagation algorithm and does not modify its functionality.

A profound understanding of backpropagation is superfluous to grasp the core of the performed research. Therefore, the rest of this section (2.2.3) is intended only for readers who are inexpert in gradient-based optimization applied to the training of neural networks and desire to learn about its operation principle in detail. In some of the following sections of this chapter, the backpropagation algorithm for specific units or elements of CNNs is also elaborated on. The reader can also skip these parts if they have no interest in knowing in depth about it.

For notational convenience, θ denotes all the network's trainable parameters. All the parameters are real-valued, and θ is a set of tensors with different dimensions. The superscript l_ϕ indicates the layer in the network, and $I = X_C^{l_\phi}, J = g_C^{l_\phi}$ are the input and output size of layer l_ϕ , respectively.

The gradient-based algorithms use the gradient of the produced error with respect to the parameters $\frac{\partial \mathcal{E}}{\partial \theta}$ to update the parameters' values during the optimization. Hence, computing $\frac{\partial \mathcal{E}}{\partial \theta}$ is essential for the training of ANNs, regardless of the type or architecture. The chain rule comes in handy for backpropagating the error.

First, let us define the nabla operator (∇) as the matrix of partial derivatives along the respective variable axes,

$$\nabla_{\vartheta}(\kappa) \triangleq \begin{bmatrix} \frac{\partial \kappa}{\partial \vartheta_{1,1}} & \frac{\partial \kappa}{\partial \vartheta_{1,2}} & \cdots & \frac{\partial \kappa}{\partial \vartheta_{1,\vartheta_W}} \\ \frac{\partial \kappa}{\partial \vartheta_{2,1}} & \ddots & \frac{\partial \kappa}{\partial \vartheta_{2,\vartheta_W-1}} & \frac{\partial \kappa}{\partial \vartheta_{2,\vartheta_W}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial \kappa}{\partial \vartheta_{\vartheta_H,1}} & \ddots & \frac{\partial \kappa}{\partial \vartheta_{\vartheta_H,\vartheta_W-1}} & \frac{\partial \kappa}{\partial \vartheta_{\vartheta_H,\vartheta_W}} \end{bmatrix} \quad (2.5)$$

where κ can be any arbitrary matrix of variables and ϑ any other arbitrary matrix of variables. $\nabla_{\vartheta}(\kappa)$ has the same shape as ϑ . So, in this example, ϑ is a matrix of depth $\vartheta_C = 1$, height ϑ_H , and width ϑ_W .

Then, the gradients of interest are estimated using the chain rule

$$\begin{aligned} \nabla_{\theta^{l_\phi}}(\mathcal{E}) &= \nabla_{g^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\theta^{l_\phi}}(g^{l_\phi}) = \\ &\nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\theta^{l_\phi}}(g_1^{l_\phi}) + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\theta^{l_\phi}}(g_2^{l_\phi}) + \cdots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\theta^{l_\phi}}(g_J^{l_\phi}) \end{aligned} \quad (2.6)$$

where g^{l_ϕ} is the output of the ϕ -th layer and θ^{l_ϕ} are the trainable parameters of such a layer, and J is the number of outputs of the layer ($J = g_C^{l_\phi}$). During training, the optimizer updates the parameters' values in each layer l_ϕ in accordance with the gradients in Eq. 2.6. The "backpropagation" of the gradients in one layer enables the estimation of the gradients in the previous layer. Analyzing a feed-forward neural network as shown in Figure 2.4, the output of a layer becomes the input of the next one during inference and vice versa during the backpropagation process. Thus

$$\nabla_{g^{l_{\phi-1}}}(\mathcal{E}) = \nabla_{X^{l_\phi}}(\mathcal{E}) \quad (2.7)$$

And, by the chain rule

$$\nabla_{X^{l_{\phi-1}}}(\mathcal{E}) = \nabla_{g_1^{l_{\phi-1}}}(\mathcal{E}) \cdot \nabla_{X^{l_{\phi-1}}}(g_1^{l_{\phi-1}}) + \nabla_{g_2^{l_{\phi-1}}}(\mathcal{E}) \cdot \nabla_{X^{l_{\phi-1}}}(g_2^{l_{\phi-1}}) + \cdots + \nabla_{g_J^{l_{\phi-1}}}(\mathcal{E}) \cdot \nabla_{X^{l_{\phi-1}}}(g_J^{l_{\phi-1}}) \quad (2.8)$$

According to the above, the gradients $\nabla_{g^{l_\phi}}(\mathcal{E})$, $\nabla_{\theta^{l_\phi}}(g^{l_\phi})$, and $\nabla_{X^{l_{\phi-1}}}(g^{l_\phi})$ are needed to train an ANN. The $\nabla_{g^{l_\phi}}(\mathcal{E})$ are known because of the backpropagation of

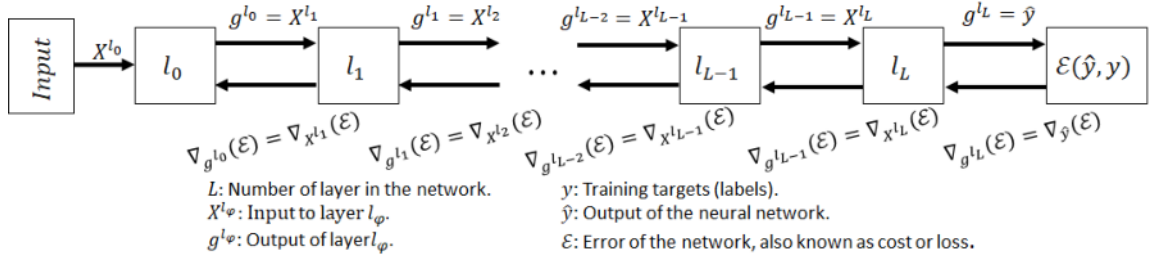


Figure 2.4: Schematic of the information flow in a standard ANN.

$\nabla_{\hat{y}}(\mathcal{E})$ (see Figure 2.4), which is computed at the beginning of the backpropagation and is dependent on the elected cost function \mathcal{E} . As for now, let us assume that $\nabla_{g^{l_\phi}}(\mathcal{E})$ is available and that the activation function $a(\cdot)$ in all layers is the identity function. Let us focus on the gradients $\nabla_{\theta^{l_\phi}}(g^{l_\phi})$ and $\nabla_{X^{l_{\phi-1}}}(g^{l_\phi})$.

Analyzing the specific case of an FC layer, extending Eq. 2.2 from the matrix notation to aid the visualization of its terms

$$\begin{bmatrix} g_1^{l_\phi} \\ g_2^{l_\phi} \\ \vdots \\ g_J^{l_\phi} \end{bmatrix} = \begin{bmatrix} W_{1,1}^{l_\phi} & W_{1,2}^{l_\phi} & \dots & W_{1,I}^{l_\phi} \\ W_{2,1}^{l_\phi} & \ddots & W_{2,I-1}^{l_\phi} & W_{2,I}^{l_\phi} \\ \vdots & \ddots & \ddots & \vdots \\ W_{J,1}^{l_\phi} & \dots & W_{J,I-1}^{l_\phi} & W_{J,I}^{l_\phi} \end{bmatrix} \begin{bmatrix} X_1^{l_\phi} \\ X_2^{l_\phi} \\ \vdots \\ X_I^{l_\phi} \end{bmatrix} + \begin{bmatrix} b_1^{l_\phi} \\ b_2^{l_\phi} \\ \vdots \\ b_J^{l_\phi} \end{bmatrix} \quad (2.9)$$

where $I = X_C^{l_\phi}$ is the number of inputs to the layer l_ϕ and $J = g_C^{l_\phi, FC}$ the number of produced outputs, $W_{j,i} \in \mathbf{W}^{l_\phi, FC}$ and $b_j \in \mathbf{B}^{l_\phi, FC} \forall j \in \{1, 2, \dots, J\} \wedge i \in \{1, 2, \dots, I\}$, and $X_i \in X^{l_\phi, FC} \forall i \in \{1, 2, \dots, I\}$. In $\nabla_{\theta^{l_\phi}}(g^{l_\phi})$, $\theta^{l_\phi, FC}$ stands for all the trainable parameters in the layer l_ϕ . For an FC layer, the trainable parameters are the synaptic weights $\mathbf{W}^{l_\phi, FC}$ and the biases $\mathbf{B}^{l_\phi, FC}$. Hence, for the synaptic weights $\mathbf{W}^{l_\phi, FC}$, the needed gradients are defined by

$$\nabla_{\mathbf{W}^{l_\phi, FC}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{W}^{l_\phi, FC}}(g_1^{l_\phi}) + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{W}^{l_\phi, FC}}(g_2^{l_\phi}) + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{W}^{l_\phi, FC}}(g_J^{l_\phi}) \quad (2.10)$$

Assuming the gradients $\nabla_{g^{l_\phi}}(\mathcal{E})$ are available, and extending Eq. 2.10,

$$\begin{aligned}
\nabla_{\mathbf{W}^{l\phi,FC}}(\mathcal{E}) = & \nabla_{g_1^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_1^{l\phi}}{\partial W_{1,1}^{l\phi}} & \frac{\partial g_1^{l\phi}}{\partial W_{1,2}^{l\phi}} & \dots & \frac{\partial g_1^{l\phi}}{\partial W_{1,I}^{l\phi}} \\ \frac{\partial g_1^{l\phi}}{\partial W_{2,1}^{l\phi}} & \ddots & \frac{\partial g_1^{l\phi}}{\partial W_{2,I-1}^{l\phi}} & \frac{\partial g_1^{l\phi}}{\partial W_{2,I}^{l\phi}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial g_1^{l\phi}}{\partial W_{J,1}^{l\phi}} & \ddots & \frac{\partial g_1^{l\phi}}{\partial W_{J,I-1}^{l\phi}} & \frac{\partial g_1^{FC}}{\partial W_{J,I}^{l\phi}} \end{bmatrix} + \nabla_{g_2^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_2^{l\phi}}{\partial W_{1,1}^{l\phi}} & \frac{\partial g_2^{l\phi}}{\partial W_{1,2}^{l\phi}} & \dots & \frac{\partial g_2^{l\phi}}{\partial W_{1,I}^{l\phi}} \\ \frac{\partial g_2^{l\phi}}{\partial W_{2,1}^{l\phi}} & \ddots & \frac{\partial g_2^{l\phi}}{\partial W_{2,I-1}^{l\phi}} & \frac{\partial g_2^{l\phi}}{\partial W_{2,I}^{l\phi}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial g_2^{l\phi}}{\partial W_{J,1}^{l\phi}} & \ddots & \frac{\partial g_2^{l\phi}}{\partial W_{J,I-1}^{l\phi}} & \frac{\partial g_2^{FC}}{\partial W_{J,I}^{l\phi}} \end{bmatrix} + \\
& \dots + \nabla_{g_J^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_J^{l\phi}}{\partial W_{1,1}^{l\phi}} & \frac{\partial g_J^{l\phi}}{\partial W_{1,2}^{l\phi}} & \dots & \frac{\partial g_J^{l\phi}}{\partial W_{1,I}^{l\phi}} \\ \frac{\partial g_J^{l\phi}}{\partial W_{2,1}^{l\phi}} & \ddots & \frac{\partial g_J^{l\phi}}{\partial W_{2,I-1}^{l\phi}} & \frac{\partial g_J^{l\phi}}{\partial W_{2,I}^{l\phi}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial g_J^{l\phi}}{\partial W_{J,1}^{l\phi}} & \ddots & \frac{\partial g_J^{l\phi}}{\partial W_{J,I-1}^{l\phi}} & \frac{\partial g_J^{l\phi}}{\partial W_{J,I}^{l\phi}} \end{bmatrix} \quad (2.11)
\end{aligned}$$

$$\begin{aligned}
\nabla_{\mathbf{W}^{l\phi,FC}}(\mathcal{E}) = & \nabla_{g_1^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} X_1^{l\phi} & X_2^{l\phi} & \dots & X_I^{l\phi} \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 \end{bmatrix} + \nabla_{g_2^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} 0 & 0 & \dots & 0 \\ X_1^{l\phi} & \ddots & X_{I-1}^{l\phi} & X_I^{l\phi} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 \end{bmatrix} + \dots + \\
& \nabla_{g_J^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ X_1^{l\phi} & \ddots & X_{I-1}^{l\phi} & X_I^{l\phi} \end{bmatrix} \quad (2.12)
\end{aligned}$$

$$\nabla_{\mathbf{W}^{l\phi,FC}}(\mathcal{E}) = \nabla_{g^{l\phi}}(\mathcal{E}) \cdot \begin{bmatrix} [X^{l\phi}]_1^T & [X^{l\phi}]_2^T & \dots & [X^{l\phi}]_J^T \end{bmatrix}^T \quad (2.13)$$

where $[X^{l\phi}]^T$ is the tranpose of the vector of inputs $X^{l\phi}$. Similarly, the gradients for the biases are

$$\nabla_{\mathbf{B}^{l_\phi, FC}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi, FC}}(g_1^{l_\phi}) + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi, FC}}(g_2^{l_\phi}) + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi, FC}}(g_J^{l_\phi}) \quad (2.14)$$

$$\nabla_{\mathbf{B}^{l_\phi, FC}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_1^{l_\phi}}{\partial b_1^{l_\phi}} \\ \frac{\partial g_1^{l_\phi}}{\partial b_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_1^{l_\phi}}{\partial b_J^{l_\phi}} \end{bmatrix} + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_2^{l_\phi}}{\partial b_1^{l_\phi}} \\ \frac{\partial g_2^{l_\phi}}{\partial b_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_2^{l_\phi}}{\partial b_J^{l_\phi}} \end{bmatrix} + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_J^{l_\phi}}{\partial b_1^{l_\phi}} \\ \frac{\partial g_J^{l_\phi}}{\partial b_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_J^{l_\phi}}{\partial b_J^{l_\phi}} \end{bmatrix} \quad (2.15)$$

$$\nabla_{\mathbf{B}^{l_\phi, FC}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad (2.16)$$

$$\nabla_{\mathbf{B}^{l_\phi, FC}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \quad (2.17)$$

Lastly, the other set of needed gradients for this type of layer

$$\nabla_{X^{l_\phi}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \nabla_{X^{l_\phi}}(g_1^{l_\phi}) + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \nabla_{X^{l_\phi}}(g_2^{l_\phi}) + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \nabla_{X^{l_\phi}}(g_J^{l_\phi}) \quad (2.18)$$

$$\nabla_{X^{l_\phi}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_1^{l_\phi}}{\partial X_1^{l_\phi}} \\ \frac{\partial g_1^{l_\phi}}{\partial X_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_1^{l_\phi}}{\partial X_J^{l_\phi}} \end{bmatrix} + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_2^{l_\phi}}{\partial X_1^{l_\phi}} \\ \frac{\partial g_2^{l_\phi}}{\partial X_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_2^{l_\phi}}{\partial X_J^{l_\phi}} \end{bmatrix} + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_J^{l_\phi}}{\partial X_1^{l_\phi}} \\ \frac{\partial g_J^{l_\phi}}{\partial X_2^{l_\phi}} \\ \vdots \\ \frac{\partial g_J^{l_\phi}}{\partial X_J^{l_\phi}} \end{bmatrix} \quad (2.19)$$

$$\nabla_{X^{l_\phi}}(\mathcal{E}) = \nabla_{g_1^{l_\phi}}(\mathcal{E}) \begin{bmatrix} W_{1,1}^{l_\phi} \\ W_{1,2}^{l_\phi} \\ \vdots \\ W_{1,I}^{l_\phi} \end{bmatrix} + \nabla_{g_2^{l_\phi}}(\mathcal{E}) \begin{bmatrix} W_{2,1}^{l_\phi} \\ W_{2,2}^{l_\phi} \\ \vdots \\ W_{2,I}^{l_\phi} \end{bmatrix} + \dots + \nabla_{g_J^{l_\phi}}(\mathcal{E}) \begin{bmatrix} W_{J,1}^{l_\phi} \\ W_{J,2}^{l_\phi} \\ \vdots \\ W_{J,I}^{l_\phi} \end{bmatrix} \quad (2.20)$$

$$\nabla_{X^{l_\phi}}(\mathcal{E}) = [\mathbf{W}^{l_\phi, FC}]^T \nabla_{g^{l_\phi}}(\mathcal{E}) \quad (2.21)$$

where $[\mathbf{W}^{l_\phi, FC}]^T$ is the tranpose of the matrix of synaptic weights $\mathbf{W}^{l_\phi, FC}$.

Now, analyzing the case of a conv layer, the trainable parameters are the kernels' weights $\mathbf{K}^{l_\phi, conv}$ and the biases $\mathbf{B}^{l_\phi, conv}$. Assuming that the layer $l_{\phi+1}$ provides $\nabla_{g^{l_\phi}}(\mathcal{E})$. For the sake of clarity, and just for the particular case of the analysis of the backpropagation on the conv layers, let the superscripts index the channels of kernels and input in the same layer. Still, $X_C^{l_\phi}$ is the number of channels in the layer's input, and $I_C^{l_\phi}$ is the number of channels in the layer output. Let us begin estimating the gradients

$$\nabla_{\mathbf{K}^{l_\phi, conv}}(\mathcal{E}) = \begin{bmatrix} \nabla_{\mathbf{K}^{l_\phi, 1, 1}}(\mathcal{E}) & \nabla_{\mathbf{K}^{l_\phi, 1, 2}}(\mathcal{E}) & \dots & \nabla_{\mathbf{K}^{l_\phi, 1, X_C^{l_\phi}}}(\mathcal{E}) \\ \nabla_{\mathbf{K}^{l_\phi, 2, 1}}(\mathcal{E}) & \dots & \nabla_{\mathbf{K}^{l_\phi, 2, X_C^{l_\phi}-1}}(\mathcal{E}) & \nabla_{\mathbf{K}^{l_\phi, 2, X_C^{l_\phi}}}(\mathcal{E}) \\ \vdots & \ddots & \ddots & \vdots \\ \nabla_{\mathbf{K}^{l_\phi, I_C^{l_\phi}, 1}}(\mathcal{E}) & \dots & \nabla_{\mathbf{K}^{l_\phi, I_C^{l_\phi}, X_C^{l_\phi}-1}}(\mathcal{E}) & \nabla_{\mathbf{K}^{l_\phi, I_C^{l_\phi}, X_C^{l_\phi}}}(\mathcal{E}) \end{bmatrix} \quad (2.22)$$

All the elements in the matrix in Eq. 2.22 are defined by

$$\nabla_{\mathbf{K}^{l_\phi, j, i}}(\mathcal{E}) = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{1,1}^{l_\phi, j, i}} & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{1,2}^{l_\phi, j, i}} & \dots & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{1, K_W^{l_\phi}}^{l_\phi, j, i}} \\ \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{2,1}^{l_\phi, j, i}} & \dots & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{2, K_W^{l_\phi}-1}^{l_\phi, j, i}} & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{2, K_W^{l_\phi}}^{l_\phi, j, i}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{K_H^{l_\phi}, 1}^{l_\phi, j, i}} & \dots & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{K_H^{l_\phi}, K_W^{l_\phi}-1}^{l_\phi, j, i}} & \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{K_H^{l_\phi}, K_W^{l_\phi}}^{l_\phi, j, i}} \end{bmatrix} \quad (2.23)$$

$\forall j \in \{1, 2, \dots, l_W^\phi\} \wedge i \in \{1, 2, \dots, X_C^{l_\phi}\}$. At the same time, every element in the matrix in Eq. 2.23 is defined by

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} &= \frac{\partial \mathcal{E}}{\partial g_{1,1}^{l_\phi,1}} \frac{\partial g_{1,1}^{l_\phi,1}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} + \frac{\partial \mathcal{E}}{\partial g_{1,2}^{l_\phi,1}} \frac{\partial g_{1,2}^{l_\phi,1}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} + \dots + \frac{\partial \mathcal{E}}{\partial g_{1,g_W}^{l_\phi,1}} \frac{\partial g_{1,g_W}^{l_\phi,1}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} + \\ &\dots + \frac{\partial \mathcal{E}}{\partial g_{2,1}^{l_\phi,1}} \frac{\partial g_{2,1}^{l_\phi,1}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} + \dots + \frac{\partial \mathcal{E}}{\partial g_{g_H,1}^{l_\phi,1}} \frac{\partial g_{g_H,1}^{l_\phi,1}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} + \dots + \frac{\partial \mathcal{E}}{\partial g_{g_H,g_W}^{l_\phi,l_\phi}} \frac{\partial g_{g_H,g_W}^{l_\phi,l_\phi}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} \end{aligned} \quad (2.24)$$

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}} &= \frac{\partial \mathcal{E}}{\partial g_{i,q}^{l_\phi,j}} (X_{d,q}^{l_\phi,i} + X_{d,q+1}^{l_\phi,i} + \dots + X_{d,\max\{\mathbf{K}_W^{l_\phi}, X_W^{l_\phi}\}}^{l_\phi,i} + X_{d+2,q}^{l_\phi,i} + \dots \\ &\quad + X_{\max\{\mathbf{K}_H^{l_\phi}, X_H^{l_\phi}\}, \max\{\mathbf{K}_W^{l_\phi}, X_W^{l_\phi}\}}^{l_\phi,i}) \end{aligned} \quad (2.25)$$

$\forall d \in \{1, 2, \dots, \mathbf{K}_H^{l_\phi}\} \wedge q \in \{1, 2, \dots, \mathbf{K}_W^{l_\phi}\}$. Replacing all the elements in the matrix of Eq. 2.23 with their extended form represented in Eq. 2.25 results in the convolution of the matrix of size $1 \times 1 \frac{\partial \mathcal{E}}{\partial \mathbf{K}_{d,q}^{l_\phi,j,i}}$ with the channel $X^{l_\phi,i}$ of the input X^{l_ϕ} .

Generalizing to the matrix of size $d \times q \nabla_{\mathbf{K}^{l_\phi,j,i}}(\mathcal{E})$ and get a solution to all the elements of the matrix of gradients in Eq. 2.22,

$$\nabla_{\mathbf{K}^{l_\phi,j,i}}(\mathcal{E}) = X^{l_\phi,i} * \nabla_{g^{l_\phi,j}}(\mathcal{E}) = X^{l_\phi,i} \quad (2.26)$$

where $i \in [1, 2, \dots, X_C^{l_\phi}]$ and $j \in [1, 2, \dots, l_W^\phi]$.

Similar to the biases of an FC layer, the bias gradients in a conv layer are equal to the output gradients

$$\nabla_{\mathbf{B}^{l_\phi,conv}}(\mathcal{E}) = \nabla_{g_{1,1}^{l_\phi,1}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi,conv}}(g_{1,1}^{l_\phi,1}) + \nabla_{g_{1,2}^{l_\phi,1}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi,conv}}(g_{1,2}^{l_\phi,1}) + \dots + \nabla_{g_{1,g_W}^{l_\phi,1}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_\phi,conv}}(g_{1,g_W}^{l_\phi,1})$$

$$\begin{aligned}
& + \dots + \nabla_{g_{2,1}^{l_{\phi},1}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_{\phi},conv}}(g_{2,1}^{l_{\phi},1}) + \dots + \nabla_{g_{g_H, g_W}^{l_{\phi},1}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_{\phi},conv}}(g_{g_H, g_W}^{l_{\phi},1}) + \nabla_{g_{1,1}^{l_{\phi},w}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_{\phi},conv}}(g_{1,1}^{l_{\phi},2}) \\
& + \dots + \nabla_{g_{g_H, g_W}^{l_{\phi}, l_W^\phi}}(\mathcal{E}) \cdot \nabla_{\mathbf{B}^{l_{\phi},conv}}(g_{g_H, g_W}^{l_{\phi}, l_W^\phi}) \quad (2.27)
\end{aligned}$$

$$\begin{aligned}
\nabla_{\mathbf{B}^{l_{\phi},conv}}(\mathcal{E}) = \nabla_{g_{1,1}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_{1,1}^{l_{\phi},1}}{\partial b_1^{l_{\phi},1}} \\ \frac{\partial g_{1,1}^{l_{\phi},1}}{\partial b_2^{l_{\phi},1}} \\ \vdots \\ \frac{\partial g_{1,1}^{l_{\phi},1}}{\partial b_{l_W^\phi}^{l_{\phi},1}} \end{bmatrix} + \nabla_{g_{1,2}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_{1,2}^{l_{\phi},1}}{\partial b_1^{l_{\phi},1}} \\ \frac{\partial g_{1,2}^{l_{\phi},1}}{\partial b_2^{l_{\phi},1}} \\ \vdots \\ \frac{\partial g_{1,2}^{l_{\phi},1}}{\partial b_{l_W^\phi}^{l_{\phi},1}} \end{bmatrix} + \dots + \nabla_{g_{1,1}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} \frac{\partial g_{1,1}^{l_{\phi}, l_W^\phi}}{\partial b_1^{l_{\phi}, l_W^\phi}} \\ \frac{\partial g_{1,1}^{l_{\phi}, l_W^\phi}}{\partial b_2^{l_{\phi}, l_W^\phi}} \\ \vdots \\ \frac{\partial g_{1,1}^{l_{\phi}, l_W^\phi}}{\partial b_{l_W^\phi}^{l_{\phi}, l_W^\phi}} \end{bmatrix} \quad (2.28)
\end{aligned}$$

$$\nabla_{\mathbf{B}^{l_{\phi},conv}}(\mathcal{E}) = \nabla_{g_{1,1}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \nabla_{g_{1,2}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \dots + \nabla_{g_{1,1}^{l_{\phi},1}}(\mathcal{E}) \cdot \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad (2.29)$$

$$\nabla_{\mathbf{B}^{l_{\phi},conv}}(\mathcal{E}) = \nabla_{g^{l_{\phi}}}(\mathcal{E}) \quad (2.30)$$

Note that in Eq. 2.28 and Eq. 2.29, the biases are vectorized for notational simplicity, and since, in practice, they have the same values for a given kernel.

Now, only the estimation of $\nabla_{X^{l_{\phi}}}(\mathcal{E})$ is left. Let us start defining

$$\nabla_{X^{l_\phi}}(\mathcal{E}) = \begin{bmatrix} \nabla_{X^{l_\phi,1}}(\mathcal{E}) \\ \nabla_{X^{l_\phi,2}}(\mathcal{E}) \\ \vdots \\ \nabla_{X^{l_\phi,l_W^\phi}}(\mathcal{E}) \end{bmatrix} \quad (2.31)$$

As with the analysis of the kernels' gradients, the superscripts serve to index the channels of the input and output in the same layer. Still, $X_C^{l_\phi}$ is the number of channels in the layer input, and l_W^ϕ is the number of channels in the layer output. All the elements in the vector in Eq. 2.31 are given by

$$\nabla_{X^{l_\phi,i}}(\mathcal{E}) = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial X_{1,1}^{l_\phi,i}} & \frac{\partial \mathcal{E}}{\partial X_{1,2}^{l_\phi,i}} & \cdots & \frac{\partial \mathcal{E}}{\partial X_{1,X_W^{l_\phi}}^{l_\phi,i}} \\ \frac{\partial \mathcal{E}}{\partial X_{2,1}^{l_\phi,i}} & \cdots & \frac{\partial \mathcal{E}}{\partial X_{2,X_W^{l_\phi}-1}^{l_\phi,i}} & \frac{\partial \mathcal{E}}{\partial X_{2,X_W^{l_\phi}}^{l_\phi,i}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial \mathcal{E}}{\partial X_{X_H^{l_\phi},1}^{l_\phi,i}} & \cdots & \frac{\partial \mathcal{E}}{\partial X_{X_H^{l_\phi},X_W^{l_\phi}-1}^{l_\phi,i}} & \frac{\partial \mathcal{E}}{\partial X_{X_H^{l_\phi},X_W^{l_\phi}}^{l_\phi,i}} \end{bmatrix} \quad (2.32)$$

$\forall i \in \{1, 2, \dots, l_W^\phi\}$. And at the same time, every element in the matrix in Eq. 2.32 is defined by

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial X_{d,q}^{l_\phi,i}} &= \frac{\partial \mathcal{E}}{\partial g_{1,1}^{l_\phi,1}} \frac{\partial g_{1,1}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} + \frac{\partial \mathcal{E}}{\partial g_{1,2}^{l_\phi,1}} \frac{\partial g_{1,2}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} + \cdots + \frac{\partial \mathcal{E}}{\partial g_{1,g_W}^{l_\phi,1}} \frac{\partial g_{1,g_W}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} + \\ &\cdots + \frac{\partial \mathcal{E}}{\partial g_{2,1}^{l_\phi,1}} \frac{\partial g_{2,1}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} + \cdots + \frac{\partial \mathcal{E}}{\partial g_{X_H^{l_\phi},1}^{l_\phi,1}} \frac{\partial g_{X_H^{l_\phi},1}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} + \cdots + \frac{\partial \mathcal{E}}{\partial g_{X_H^{l_\phi},X_W^{l_\phi}}^{l_\phi,1}} \frac{\partial g_{X_H^{l_\phi},X_W^{l_\phi}}^{l_\phi,1}}{\partial X_{d,q}^{l_\phi,i}} \end{aligned} \quad (2.33)$$

where $d \in [1, 2, \dots, X_H^{l_\phi}]$ and $q \in [1, 2, \dots, X_W^{l_\phi}]$. Let us apply the Eq. 2.33 to the convolution shown in Figure 2.2 to help us visualize the elements involved in the operation. Then, estimating the gradients element-wise for the y in Figure 2.2

$$\begin{bmatrix} \frac{\partial \mathcal{E}}{\partial x_{1,1}} & \frac{\partial \mathcal{E}}{\partial x_{1,2}} & \frac{\partial \mathcal{E}}{\partial x_{1,3}} \\ \frac{\partial \mathcal{E}}{\partial x_{2,1}} & \frac{\partial \mathcal{E}}{\partial x_{2,2}} & \frac{\partial \mathcal{E}}{\partial x_{2,3}} \\ \frac{\partial \mathcal{E}}{\partial x_{3,1}} & \frac{\partial \mathcal{E}}{\partial x_{3,2}} & \frac{\partial \mathcal{E}}{\partial x_{3,3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial y_{1,1}} k_{1,1} & \frac{\partial \mathcal{E}}{\partial y_{1,1}} k_{1,1} + \frac{\partial \mathcal{E}}{\partial y_{1,2}} k_{1,1} & \frac{\partial \mathcal{E}}{\partial y_{1,2}} k_{1,2} \\ \frac{\partial \mathcal{E}}{\partial y_{1,1}} k_{2,1} + \frac{\partial \mathcal{E}}{\partial y_{2,1}} k_{1,1} & \frac{\partial \mathcal{E}}{\partial y_{1,1}} k_{2,2} + \frac{\partial \mathcal{E}}{\partial y_{1,2}} k_{2,1} + \frac{\partial \mathcal{E}}{\partial y_{2,1}} k_{1,2} + \frac{\partial \mathcal{E}}{\partial y_{2,2}} k_{1,1} & \frac{\partial \mathcal{E}}{\partial y_{1,2}} k_{2,2} + \frac{\partial \mathcal{E}}{\partial y_{2,2}} k_{1,2} \\ \frac{\partial \mathcal{E}}{\partial y_{2,1}} k_{2,2} & \frac{\partial \mathcal{E}}{\partial y_{2,1}} k_{2,2} + \frac{\partial \mathcal{E}}{\partial y_{2,2}} k_{2,1} & \frac{\partial \mathcal{E}}{\partial y_{2,2}} k_{2,2} \end{bmatrix} \quad (2.34)$$

Analyzing carefully Eq. 2.34, we can see that the computation of the gradients $\frac{\partial \mathcal{E}}{\partial X}$, using the example shown in Fig. 2.2, resembles a convolution. Therefore, let us define a new operation called "full-convolution" (\ast^{full}); this operation acts like the convolution with the difference that the sliding matrix slides with each of its components over the second matrix, as illustrated in Figure 2.5.

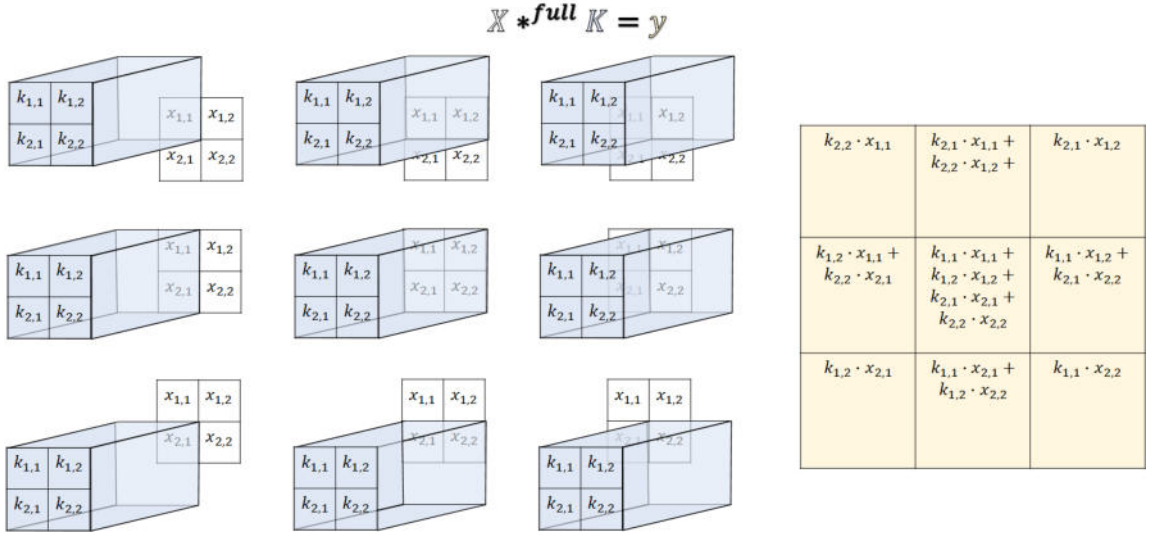


Figure 2.5: Example of the full-convolution operation (\ast^{full}) using two matrices. $X \ast^{full} K = Y$ (X in white fully-convolves with K in blue and produces Y in yellow).

However, the elements displayed in Eq. 2.34, are not exactly the result of a full-convolution of the gradients with K , but with K rotated 180deg. Let us represent the rotation by 180deg of K with $rot_{180}(K)$. The analysis that leads to Eq.

2.34 using the convolution displayed in Figure 2.2, considers a single kernel with only one channel, but it is the same case for every $K^{l_\phi,i,j} \in \mathbf{K} \forall i \in \{1, 2, \dots, X_C^{l_\phi}\} \wedge j \in \{1, 2, \dots, l_W^\phi\}$. For estimating the gradients for a given $X^{l_\phi,i}$ then, all the elements of the output contribute to the gradients. Therefore,

$$\nabla_{X^{l_\phi,i}}(\mathcal{E}) = \sum_{j=1}^{l_W^\phi} \frac{\partial \mathcal{E}}{\partial g_l^{l_\phi,j}} *^{full} rot_{180}(K^{l_\phi,i,j}) \quad (2.35)$$

2.2.4 Activation functions

Recapitulating, each layer output g^{l_ϕ} is a composite function of different operations. Up to this point, we have analyzed the transformations performed using FC and conv layers. However, regardless of the type of layer, an arbitrary activation function performs a non-linear transformation on y^{FC} and y^{conv} . The activation function can be analyzed independently from the previous operations. Figure 2.6 shows how a layer can be decomposed into the different operations that it performs, showing an example of an FC layer. Yet, it is the same scenario with conv layers. The batch norm step at the beginning of the layer is customary in modern CNNs. Section 2.2.6 further elaborates on this. Let us overlook it in the meantime. The activation function does not have any trainable parameters. Therefore, only the gradients $\nabla_{X^{l_\phi}}(\mathcal{E})$ is needed for the backpropagation process

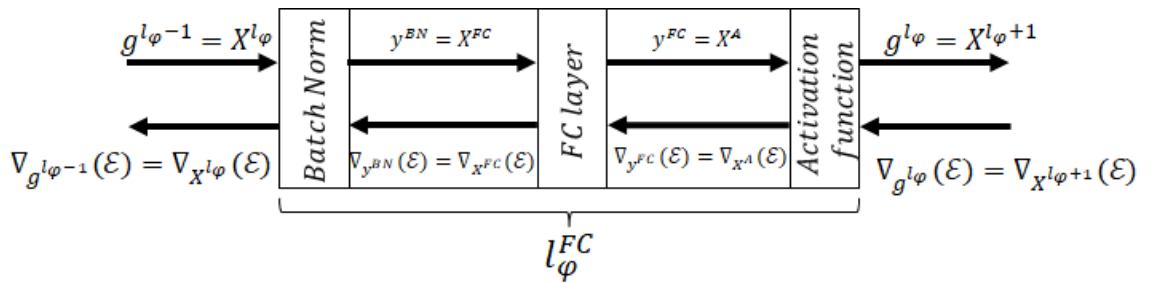


Figure 2.6: Example of the composition of an arbitrary FC layer l_ϕ^{FC} . A layer can have different operations and in different order.

Activation functions have two purposes in a neural network: (1) to induce a non-linear behavior and (2) to bind the neurons' outputs [73]. There are many

different activation functions. Let us analyze the standard activation function for current DL models, the rectified linear unit (ReLU). The ReLU function is defined as,

$$a_{ReLU}(X_{ReLU}) = \max(0, X_{ReLU}) \quad (2.36)$$

For backpropagation in a layer that uses this activation function, the estimation of the gradients is given by

$$\nabla_{X_{ReLU}}(\mathcal{E}) = \nabla_{a_{ReLU}}(\mathcal{E}) \nabla_{X_{ReLU}}(a_{ReLU}) \quad (2.37)$$

Since the following layer provides $\nabla_{a_{ReLU}}(\mathcal{E})$, $\nabla_{X_{ReLU}}(a_{ReLU})$ is the only gradient that requires analysis. In case $X_{ReLU} \leq 0$

$$\nabla_{X_{ReLU}}(a_{ReLU})0 = 0 \quad (2.38)$$

and the case when $X_{ReLU} > 0$

$$\nabla_{X_{ReLU}}(a_{ReLU})X_{ReLU} = 1 \quad (2.39)$$

So,

$$\nabla_{X_{ReLU}}(a_{ReLU}) = \begin{cases} \nabla_{a_{ReLU}}(\mathcal{E}) & \forall X_{ReLU} > 0 \\ 0 & \forall X_{ReLU} \leq 0 \end{cases} \quad (2.40)$$

There are many other activation functions. However, the ReLU activation function relegated previously used functions such as hyperbolic tangent (TanH) and Hyperbolic. The reason is that these last two are a type of logistic function whose output is bounded at both sides of the domain (see Figure 2.7), causing the gradients $\nabla_{X^{l\phi}}(g^{l\phi})$ to be near zero [75]. Therefore, the gradient diminishes over the layer and starts to disappear, which is an issue (known as the vanishing gradient problem) for the gradient-based optimization process that is training [75].

Nowadays, some variations of ReLU with mechanisms to enhance different aspects of training are commonly employed in the context of DL. Section 2.4.1 displays some of the most used ones.

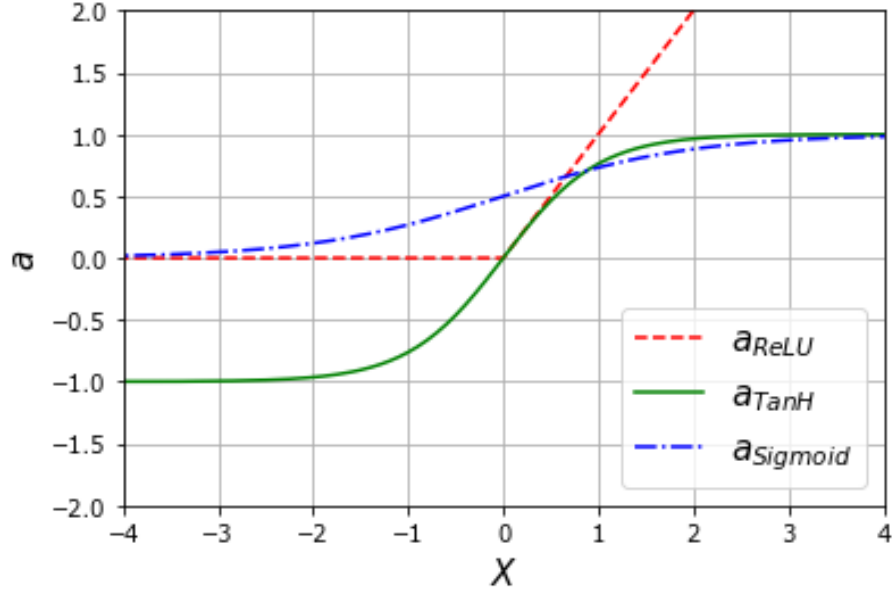


Figure 2.7: ReLU, TanH, and Sigmoid activation functions.

Softmax function

Neural networks have different types of special activation functions in the last layer, depending on the target task. The softmax function is the standard activation function for multi-class classification since the output resembles the probability of the inputs belonging to the different possible categories or classes. The softmax function is defined as

$$\widehat{y}_i^{Softmax} = \frac{e^{X_i^{l_\phi}}}{\sum_{j=1}^J e^{X_j^{l_\phi}}} \quad (2.41)$$

where $J = X_C^{l_\phi}$ is the size of the input vector to the softmax function that is the same as the last layer output size $g_C^{l_L}$. In ANNs, the softmax function typically is the activation function of the last layer, which is an FC layer.

The softmax function is just a type of activation function. Hence, it does not have any trainable parameters, and only $\nabla_{X^{l_\phi}}(\mathcal{E})$ is needed for the backpropagation of gradients. For notational convenience, let us abbreviate $\widehat{y}^{Softmax}$ as \widehat{y}^S .

$$\nabla_{X^{l\phi}}(\mathcal{E}) = \nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_1^s) + \nabla_{\widehat{y}_2^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_2^s) + \cdots + \nabla_{\widehat{y}_J^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_J^s) \quad (2.42)$$

Analyzing the first element $\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_1^s)$ of Eq. 2.42

$$\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_1^s) = \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \begin{bmatrix} \frac{\partial}{\partial X_1^{l\phi}} \frac{e^{X_1^{l\phi}}}{e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}}} \\ \frac{\partial}{\partial X_2^{l\phi}} \frac{e^{X_1^{l\phi}}}{e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}}} \\ \vdots \\ \frac{\partial}{\partial X_J^{l\phi}} \frac{e^{X_1^{l\phi}}}{e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}}} \end{bmatrix} \quad (2.43)$$

$$\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_1^s) = \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \begin{bmatrix} \frac{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})(e^{X_1^{l\phi}}) - (e^{X_1^{l\phi}})(e^{X_1^{l\phi}})}{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})^2} \\ \frac{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})(0) - (e^{X_1^{l\phi}})(e^{X_2^{l\phi}})}{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})^2} \\ \vdots \\ \frac{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})(0) - (e^{X_1^{l\phi}})(e^{X_J^{l\phi}})}{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})^2} \end{bmatrix} \quad (2.44)$$

$$\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l\phi}}(\widehat{y}_1^s) = \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \begin{bmatrix} \frac{e^{X_1^{l\phi}}}{e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}}} - \left(\frac{e^{X_1^{l\phi}}}{e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}}} \right)^2 \\ \frac{-(e^{X_1^{l\phi}})(e^{X_2^{l\phi}})}{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})^2} \\ \vdots \\ \frac{-(e^{X_1^{l\phi}})(e^{X_J^{l\phi}})}{(e^{X_1^{l\phi}} + e^{X_2^{l\phi}} + \cdots + e^{X_J^{l\phi}})^2} \end{bmatrix} \quad (2.45)$$

$$\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l_\phi}}(\widehat{y}_1^s) = \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \begin{bmatrix} (1 - \widehat{y}_1^s)\widehat{y}_1^s \\ -\widehat{y}_1^s\widehat{y}_2^s \\ \vdots \\ -\widehat{y}_1^s\widehat{y}_J^s \end{bmatrix} \quad (2.46)$$

$$\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l_\phi}}(\widehat{y}_1^s) = \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \widehat{y}_1^s \begin{bmatrix} (1 - \widehat{y}_1^s) \\ -\widehat{y}_2^s \\ \vdots \\ -\widehat{y}_J^s \end{bmatrix} \quad (2.47)$$

That is the case of $\nabla_{\widehat{y}_1^s}(\mathcal{E})\nabla_{X^{l_\phi}}(\widehat{y}_1^s)$. However, it holds for any $\nabla_{\widehat{y}_j^s}(\mathcal{E})\nabla_{X^{l_\phi}}(\widehat{y}_j^s) \forall j \in \{1, 2, \dots, J\}$. So, applying the form of Eq. 2.47 to every element in Eq. 2.42, the gradients are estimated with

$$\nabla_{\widehat{y}^s}(\mathcal{E})\nabla_{X^{l_\phi}}(\widehat{y}^s) = \begin{bmatrix} \widehat{y}_1^s(1 - \widehat{y}_1^s) & \widehat{y}_1^s(-\widehat{y}_2^s) & \dots & \widehat{y}_1^s(-\widehat{y}_J^s) \\ \widehat{y}_2^s(-\widehat{y}_1^s) & \widehat{y}_2^s(1 - \widehat{y}_2^s) & \dots & \widehat{y}_2^s(-\widehat{y}_J^s) \\ \vdots & \vdots & \ddots & \vdots \\ \widehat{y}_J^s(1 - \widehat{y}_1^s) & \dots & \widehat{y}_J^s(-\widehat{y}_{J-1}^s) & \widehat{y}_J^s(1 - \widehat{y}_J^s) \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial \widehat{y}_1^s} \\ \frac{\partial \mathcal{E}}{\partial \widehat{y}_2^s} \\ \vdots \\ \frac{\partial \mathcal{E}}{\partial \widehat{y}_J^s} \end{bmatrix} \quad (2.48)$$

2.2.5 Downsampling operations

A convolutional layer generally has at least three components. As Figure 2.8 shows, these are the convolution filters, the activation function, and a downsampling operator [9]. The downsampling operator reduces the size of the outputs by applying a pooling function and is usually performed at the end of the layer [1, 7, 73].

Pooling operations reduce the spatial size of the inputs (width and height) by replacing a subset of their elements with a single value that is statistically representative of the neighborhood [9]. There are different types of pooling functions. Some of the most common ones are average pooling and max pooling, which Figure 2.9 shows.

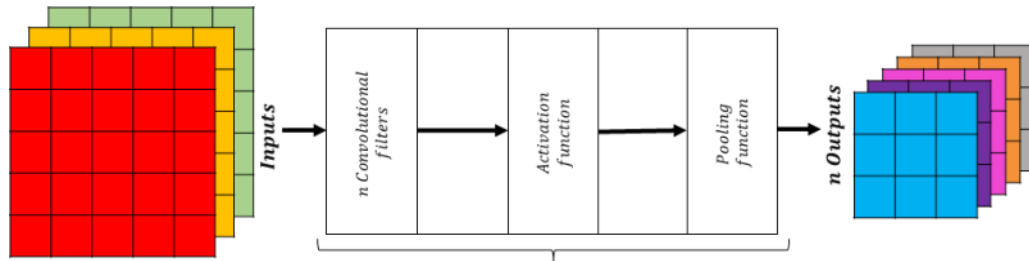


Figure 2.8: Example of the composition of a basic conv layer.

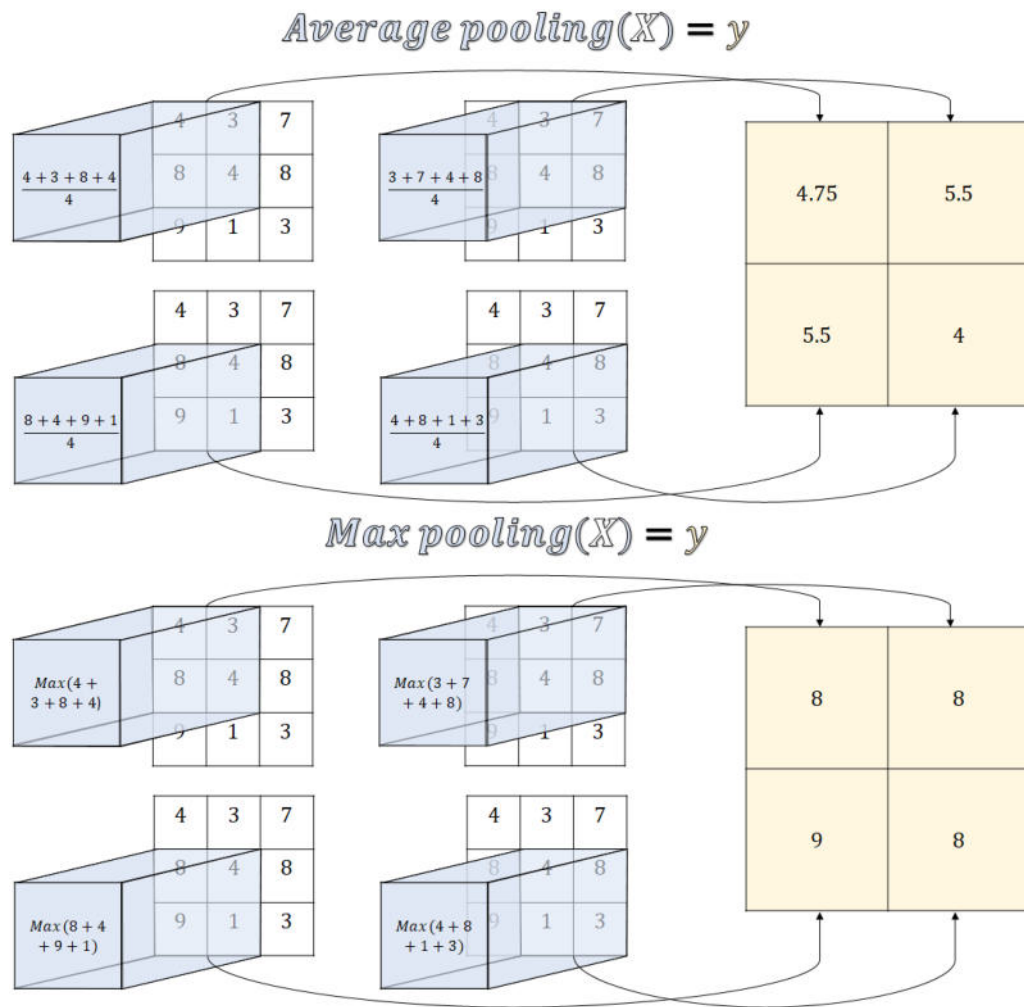


Figure 2.9: Example of commonly used pooling functions. The pooling operation operates channel-wise and slides with an arbitrary stride over the input channels. The size of the pooling receptive field (width and height of the striding window) is also arbitrary.

By drastically reducing the size of the feature maps with respect to the input size of the network, CNNs achieve approximate local invariance, meaning that they can detect relevant information in the inputs, regardless of their spatial location [1, 9, 72, 75]. Additionally, progressively reducing the spatial size of the data also reduces the number of required parameters and computation of the network [1, 9].

Another benefit of using pooling operations in CNNs is that they enable the network to process inputs of different sizes [9]. By defining the receptive field and stride of the pooling functions, one can control the size of the layer output.

2.2.6 Other elements

We have covered the essential elements of CNNs. However, they have many other types of components. Overall, the other elements in CNNs can be grouped into flattening segments, normalization layers, and regularization elements. Let us describe each of these.

Flattening

The flattening operation is necessary for any CNN for image classification. As exposed in Section 2.2, CNNs have two main parts: a feature extractor composed of conv layers and a classifier, typically an MLP (an arrangement of a few FC layers.). Nevertheless, the outputs of conv layers are three-dimensional, and the inputs to FC layers are one-dimensional. Thus, a flattening segment connects the two parts of a CNN. This part deals with converting the three-dimensional feature map of the last conv layer into a one-dimensional vector to feed the classifier.

The usual technique to flatten the features map in the last conv layer is concatenating all their elements into a single vector. However, other approaches could be used, such as applying a global pooling operation over each channel in the feature map and concatenating the scalar output values for every channel. Figure 2.10 shows an example of the flattening operation.

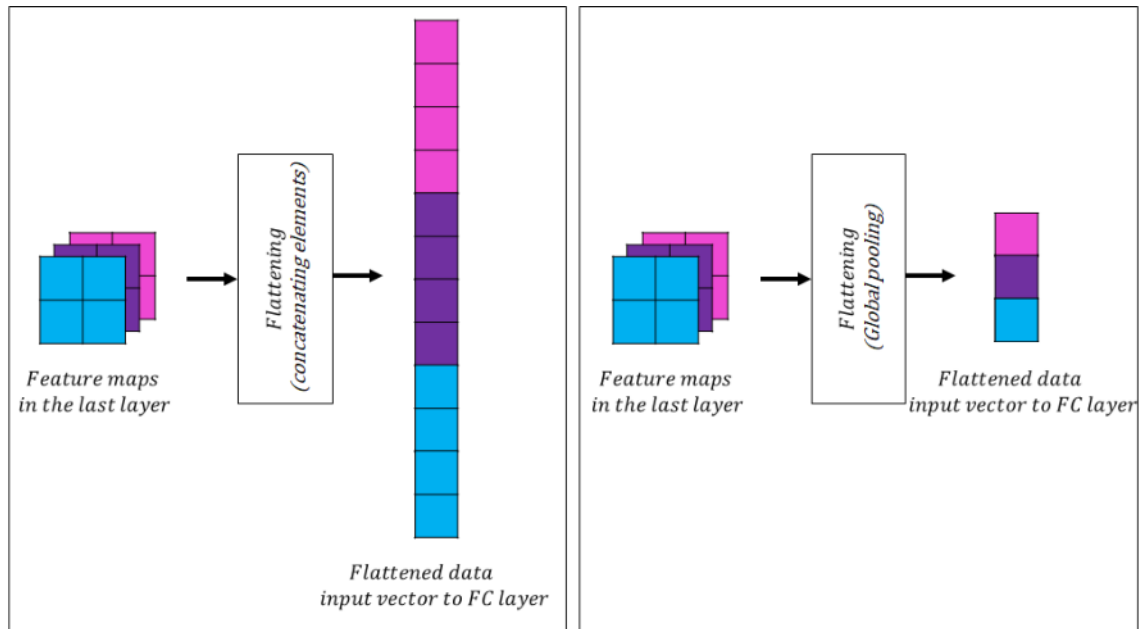


Figure 2.10: Example of the two standard types of flattening operation in a CNN.

Normalization layers

Normalizing data is a good practice in ML, and DL is not the exception. Normalizing the data ensures that all the data is on the same scale, which aids during training. Recapitulating, the output of one layer is the input to the next. Hence, every layer has its own input. Then, normalizing the data at each layer improves the training efficiency [1, 73, 75, 79]. Therefore, layers of deep networks usually have a normalization element. Figure 2.6 shows an example of an FC layer with a batch normalization phase at the beginning of the layer.

The placement of the normalization operation in a layer is still a matter of debate [1]. Many researchers assert that from a statistical point of view, the normalization should be after the activation function; this makes sense because you do not want the bias from the negative values that the activation function will set to zero. However, the study that first proposed implementing normalization per layer suggests placing it before the activation function [79], and many practitioners follow this guidance.

There are different types of normalization approaches. Nowadays, the stan-

standard normalization element is batch normalization. To analyze batch normalization, first, let us define the term batch. Recapitulating once again, ML models are trained using a training dataset. During training in ANNs, on a single inference iteration, a group of samples from the training set (a batch) passes through the network at the same time. One of the advantages of today's computational resources is the capability of parallel computing. So, batch normalization refers to the operation of normalizing the data in one training batch, as described by

$$y^{BN} = \widehat{X}^{l_\phi} = \frac{X^{l_\phi} - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \quad (2.49)$$

where X^{l_ϕ} is the input to the batch normalization operation in layer l_ϕ , μ_β is the mean of the batch of data X^{l_ϕ} , σ_β is the standard deviation of the batch of data X^{l_ϕ} , and ϵ is a small positive scalar used to prevent dividing by zero. Batch normalization also operates during regular inference, that is after the network was trained. In that situation, the values of μ_β and σ_β use the mean values of the training data.

Batch normalization benefits training by significantly reducing the number of required training epochs and stabilizing the trajectory of the optimization procedure, making it more robust to changes in its configuration [1, 75, 79] (hyperparameters, which are further discussed in Sections 2.3.2 and 2.4.2.). Most of the CNN architectures explored in this thesis use batch normalization inside their layer, as depicted in Figure 2.6.

Regularization elements

Generally, ANNs are trained on the tasks that they will perform. Section 2.2.3 explains that during the training of an ANN, the error \mathcal{E} is computed using the training data (thus, training error). However, it is impossible to train the model using all the specific samples of the target domain. The regularization terms help improve the network's generalization and prevent the model from adjusting particularly to the training set, which is a subset of the target domain. This issue is referred to as overfitting or memorization.

Estimating the performance of ML models outside the training samples is fundamental. For this purpose, the training dataset is often split into at least two

partitions: training and testing data. The regularization elements, or regularizers, comprise all the methods and techniques intended to reduce the error using the testing data (test error), regardless of the impact on the training error [1, 9]. Hence, the regularization elements' goal is to maintain or improve the model generalization or generalizability, which refers to the capacity of a model to perform the task it was trained for in new unseen data.

Regularization elements can take part in the learning algorithm and the model itself. Different types of specialized modules can be inserted as part of the layers of an ANN. In this regard, contemporary ANNs, as the ones studied in this thesis, customarily include a dropout phase within their layers [1, 7, 72]. Dropout is a filter that sets to zero randomly the activation of neurons in a layer during one training iteration to prevent the co-adaptation of specific learned representations that would lead to memorization of the training data [75, 80]. Figure 2.11 shows an example of how dropout works. Generally, the neurons of one layer connect to all the neurons in the next layer, and the dropout unit serves as a filter that cuts (by setting it to zero the activations) randomly some connections during training.

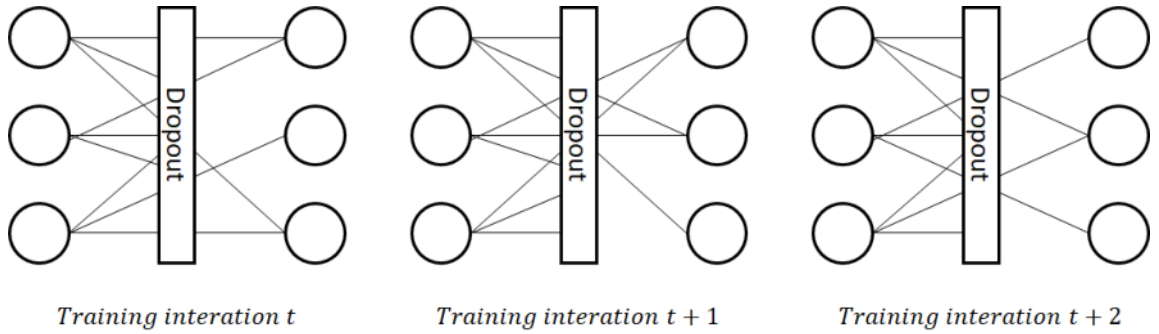


Figure 2.11: Example of the dropout operation in an ANN during three training iterations.

2.3 Learning procedure

The training of neural networks involves an optimization procedure that consists of finding the network parameters θ that minimize the cost function $\mathcal{E}(\widehat{y}(x, \theta), y)$. So, in the context of ANNs, the learning procedure encapsulates all the applied

methods that directly or indirectly serve to find the model parameters θ in such a way that the model improves at the target task.

As stated in Section 2.1, there are different types of learning. Currently, CNNs can use multiple techniques from different learning types. However, the most prominent approach is stochastic gradient-descent optimization, which is a supervised learning family of methods that use the gradients of the cost function with respect to the parameters to update the values of the parameters to decrease the measured error iteratively. All the learning algorithms used within this thesis to train CNNs make use of such type of methods.

Sections 2.2.3 through 2.2.4 examine how to estimate the needed gradients $\nabla_{\theta}(\mathcal{E})$ for the gradient-based optimization procedure, assuming that the error $\mathcal{E}(\widehat{y}(x, \theta), y)$ and the gradients of the error with respect to the outputs $\nabla_{\widehat{y}}(\mathcal{E})$ are given. Now, let us analyze how to estimate the error during training and its gradients.

2.3.1 Loss function

The cost function is also commonly known as the loss function, and in ANNs is more frequently so. The loss function has to contain a way to evaluate the network's performance in the training set and also typically includes some regularization elements.

Cross-entropy is the current default loss function for training DL architectures. In the study case of this thesis, the models perform single-label multi-class classification with K different labels. Then, the network's output is a vector of logits $\widehat{y} \in \mathbb{R}^K$, and the training targets are encoded using a codification scheme called one-hot encoding. Each training target is coded into a vector of binary values $y \in \{0, 1\}^K$, where a value of 1 in the k -th element of the vector denotes that the sample belongs to the k -th class, and a value of 0 indicates the opposite. Since this thesis deals with the single-label classification task, the target vectors have 1-norm $\|y\|_1 = \sum_{k=1}^K y_k = 1$. The softmax cross-entropy \mathcal{E}^{C-ent} loss is computed as

$$\mathcal{E}^{C-ent}(\widehat{y}, y) = - \sum_{k=1}^K y_k \widehat{y}_k + \log \sum_{k=1}^K e^{\widehat{y}_k} \quad (2.50)$$

Backpropagation requires $\nabla_{\widehat{y}}(\mathcal{E})$ at the last layer. If the cross-entropy loss described in Eq. 2.50 is the employed loss function, then

$$\nabla_{\widehat{y}}(\mathcal{E}^{C-ent}) = -\frac{\partial}{\partial \widehat{y}} \left(-\sum_{k=1}^K y_k \widehat{y}_k + \log \sum_{k=1}^K e^{\widehat{y}_k} \right) = -y + \frac{e^{\widehat{y}}}{\log \sum_{k=1}^K e^{\widehat{y}_k}} \quad (2.51)$$

2.3.2 Optimizer

Here, it is necessary to describe another ML key concept: hyperparameter, which comprises any parameter of an ML algorithm that defines the nature and operation of the model and learning procedure. Therefore, any ML model has two types of parameters. The trainable parameters that are part of the model and characterize the mapping function are denoted simply as parameters and the hyperparameters, which are design choices to construct and train the algorithm.

The optimizer of a learning procedure deals with finding only the parameters unless otherwise stated.

Let us analyze the classification task as an optimization problem. The purpose is to obtain a mapping function of the domain space onto the target space $h: \mathcal{X} \rightarrow \mathcal{Y}$, that is characterized by the optimal set of parameter parameters θ , such that it minimizes an arbitrary loss function \mathcal{E} using the training data [81], such that:

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^N \mathcal{E}(y^n, \widehat{y}^n) \quad (2.52)$$

where N is the total number of training samples and y^n, \widehat{y}^n are the target vectors and the logits output of the n -th training sample, respectively. In Eq. 2.52, the loss is a function of the logits \widehat{y} and the target vector y . Yet, the logits are a mapping of the input x (the images' pixels for the image classification task) using the hypothesis function h characterized by θ . Then, Eq. 2.52 can be expressed as:

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^N \mathcal{E}(y^n, h(x^n, \theta)) \quad (2.53)$$

where x^n represents the inputs of the n -th training sample. The optimizer is the

algorithm that attempts to find the set of parameters θ that, when applied to the hypothesis function h , reaches the condition defined in (2.53).

Optimization algorithms are the core of ML models since they allow them to learn structures from data. There are numerous approaches to finding model parameters that minimize the loss function, such as random search and heuristic models. However, the number of parameters in DL models can go from a few millions in compact, efficient models to hundreds of billions, such as in large language models (LLMs). This feature makes many numerical and heuristic optimization approaches unsuitable for training DL algorithms. However, since the parameter space is differentiable, gradient-descent optimization is applicable for this task and has become the learning procedures' linchpin for DL models.

Gradient descent is an iterative optimization approach that searches over the optimization surface that, in the context of training ANNs, is the loss landscape. Figure 2.12 illustrates a low-dimension example of the loss landscape. An important consideration is that the loss landscape shape is unknown, and it can not be actually seen. Consequently, the understanding of the optimization surface relies on indirect measurements of the loss landscape. The training of ANNs makes the assumption of convexity for the loss landscape, even though it is known not to be the case. Hence, gradient-descent algorithms can effectively train ANNs but only guarantee suboptimal sets of parameters θ .

The convergence regions of gradient-descent algorithms depend on several aspects, such as the number of training iterations, initialization values, etcetera. All these conditions are hyperparameters. Careful tuning of hyperparameters is consequential in finding better local optimal zones in the loss landscape.

Gradient-descent can be divided into two major categories: standard or vanilla gradient-descent and stochastic gradient-descent (SGD). The difference is that vanilla gradient-descent updates the parameters' values after computing the gradients using all the available information. Thus, one optimization iteration (or step) uses all the training data (so it is a training epoch). On the other hand, SGD uses random sub-sets (batches) of the training data to update the parameters' values. A training epoch consists of several optimization iterations (consider one training epoch once every sample in the training set is used to update the parameters' values.). It is stochastic because it takes many pseudo-random steps in the

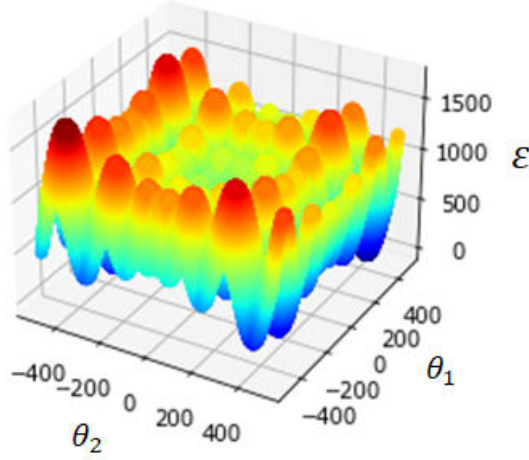


Figure 2.12: Low-dimensionality example of loss landscape of a model with only two parameters θ_1 and θ_2 . The valleys are local minima. Gradient-descent algorithms lead to local minimum points in the landscape. The depicted function is the Schwefel function and is shown only for illustrative purposes.

loss landscape during one training epoch.

SGD is more efficient when training ANNs due to the size of training datasets and also because the randomness in the optimization algorithm prevents it from getting stuck in sub-optimal local minimal zones for early convergence. The learning algorithms used to train the CNNs in this thesis are based on SGD.

Stochastic gradient descent

The SGD is an optimizer extensively used for training DL models. It is a first-order gradient-based algorithm. The SGD estimates the new values of the parameters θ' for random batches of training data β using the gradient of the loss function with respect to the parameters and a hyperparameter called learning rate α that controls the size of the changes on every optimization step

$$\theta' = \theta - \alpha \frac{\partial \mathcal{E}(y^\beta, \hat{y}^\beta)}{\partial \theta} \quad (2.54)$$

where θ' is the set of new parameters that becomes θ for the next optimization iteration, and y^β, \hat{y}^β are the targets and logits of the training batch β , respectively.

The SGD is computationally more efficient than vanilla gradient-descent. Nevertheless, it produces noise that makes the gradient oscillate. This oscillation can interfere with the algorithm’s convergence. Nonetheless, it also prevents it from getting stuck into local optimal, contrastively to algorithms that use deterministic gradients [81].

2.3.3 Transfer learning

DL models, including CNNs, for their size and number of parameters, demand massive datasets for training and considerable computing resources [7,73]. These conditions are an issue for applications with scarce data and conditioned computational requirements, limiting the scope of DL applications. However, transfer learning is a technique that aids in overcoming this noteworthy limitation [73,75].

The transfer learning foundation is that data from subspaces of a domain space have similar structures [75]. For instance, if a CNN model is trained for classifying images of cars, it learns to extract features from car images that are also useful for other types of images, such as the shape of tires being similar to the shape of a ball. This argument holds for abstract representations that can have no meaning to us but are relevant for data analysis.

Transfer learning refers to the practice of training models on large datasets similar to data of the target domain, then retraining (or fine-tuning) the model using the training dataset available for our target task. Therefore, the model parameters learned in one massive dataset are transferred to another similar domain and serve as initialization points for the new optimization process; hence, the name transfer learning [75].

Transfer learning is a common practice for training DL models where the data is scarce, and it has been proven effective in the particular field of medical image analysis [8, 22]. It has the potential to improve the generalizability, efficacy, and precision of DL models for medical image analysis [8]. Transfer learning is used in all the CNNs studied in this thesis for the above-stated reasons and since training contemporary CNN architectures from scratch is too expensive in terms of computational power and time for the scope of this research.

2.4 Hyperparameters

Section 2.3.2 introduced the concept of hyperparameter as any parameter of an ML algorithm that defines the nature and operation of the model and learning procedure. Thus, the word hyperparameter encompasses many different aspects of an ML system. From design choices such as the type of algorithm, in the case of ANNs, number of layers, neurons per layer, type of layers, in the case of CNNs, the number of filters in a conv layer, the shape of the kernels in each layer, and also other non-design choices that are consequential in the final model performance, such as the algorithm to train the network. Then, some choices come with their own hyperparameters. For instance, if SGD were to be used to train a model, a learning rate for the SGD algorithm would have to be chosen, or if a swarm algorithm were to be used, the number of individuals in the population and the topology of their interaction would have to be set, and so on.

The hyperparameter configuration defines an ML model, the space of parameters for the hypothesis function, and how the ML algorithm explores and exploits the parameter space. Let us categorize the hyperparameters into two classes: (1) Design hyperparameters, which define the type of model and its structure, and (2) learning hyperparameters, which, as the name suggests, define the learning procedure.

2.4.1 Design hyperparameters

There are several design choices when developing an ML system. Let us constrain to the specific case of CNNs. A CNN architecture is a particular arrangement of operators, including at least one conv layer. Several options, including the type of operators in a network and the specific order in which they interact, define a particular architecture. Let us analyze some quintessential characteristics that describe a CNN architecture.

Structural properties

When designing a CNN, one has to decide every aspect that defines the model topology. In DL, the number of layers denotes the network's depth. As for CNNs,

a layer's width refers to the number of channels in the layer. For consistency, when talking of FC layers, let the layer's width describe the number of neurons in it. The designer has to set the CNN depth and the layers' width. Besides, as explained in Section 2.2, every layer consists of different operations. Hence, the designer must define the type and order of operations in every layer. For instance, if the layer includes a batch normalization phase, and if it does, where the operation is to be performed.

Speaking of CNNs, as Section 2.2.2 states, the kernels in a CNN have an arbitrary shape, although they customarily have the same height and width of non-even size (for example, 3x3 or 5x5). Additionally, pooling operations downsample the output size of conv layers to an arbitrary size; the shape of pooling operations must be defined just as the kernels', and both kernels and pooling operators have other hyperparameters such as the stride size and the use of padding, which is the practice of adding zeros around the border of the inputs to perform the conv or pooling operation using all the elements of the original data.

Lastly, the designer has to set the connections between layers. For many years, CNNs operated as sequential neural networks. Hence, all the connections in a CNN feed the output of one layer only to the following one. This practice rapidly evolved, with studies showing that using skip or residual connections drastically improves the training efficiency (and the models' performance) [82, 83]. Then, other types of connection came, such as dense connections [84].

Activation functions

In theory, any non-linear function applied to the neuron's outputs would grant with non-linear behavior to an ANN. However, there are some desired aspects to an activation function. For starters, the function has to be continuously differentiable so it is compatible with the backpropagation of the gradients and allows the application of gradient-based optimizers for training [72]. In this regard, smooth and monolithic functions are best for the matter [72]. Finally, functions with a finite range and identity-like symmetric behavior near the origin are also preferred, considering the gradient-optimization process [72].

As Section 2.2.4 states, ReLU is the default activation function for CNNs. How-

ever, several other options exist. Many modern CNNs usually use other activation functions to improve different aspects of the networks. Let us cover the other activation functions that some of the studied architectures in this thesis employ.

A frequent activation function in modern architectures is the gaussian error linear unit (GELU) [85], and it is defined as

$$a_{GELU} = X_{GELU} \Phi(X_{GELU}) \quad (2.55)$$

where $\Phi(X_{GELU})$ is the standard Gaussian cumulative distribution function of X_{GELU} . So, GELU is approximate to

$$a_{GELU} \approx 0.5X_{GELU} \left[1 + \tanh \left(\sqrt{\frac{2}{\pi}} (X_{GELU} + 0.044715X_{GELU}^3) \right) \right] \quad (2.56)$$

From the same family of activation functions as GELU, the sigmoid linear unit (SeLU) activation function [85] is another common activation function of recent CNN architectures. The difference between GELU and SeLU is the cumulative distribution function it uses. In contrast to GELU, SeLU uses the logistic sigmoid function $\sigma(X_{SeLU})$. Hence, the SeLU activation function is

$$a_{SeLU} = X_{SeLU} \sigma(X_{SeLU}) \quad (2.57)$$

GELU and SeLU are like a smooth version of ReLU [85]. However, they differ from ReLU in that they are non-convex, non-monotonic, and non-linear in the positive domain [85]. These characteristics can help GELU and SeLU to better approximate complex functions than ReLU [85].

Another activation function used in modern CNNs is the hard-swish (h-swish) function [86],

$$a_{h-swish}(X_{h-swish}) = X_{h-swish} \frac{ReLU_6(X_{h-swish} + 3)}{6} \quad (2.58)$$

where

$$ReLU_6(X_{h-swish}) = \min(\max(0, X_{h-swish}), 6) \quad (2.59)$$

The h-swish activation function is very similar to the SeLU activation. But, it eliminates the logistic sigmoid function $\sigma(X)$ to alleviate the computational load of computing it. Hence, h-swish has many of the advantages of SeLU and is more efficient.

Figure 2.13 shows the GELU, SeLU, and h-swish activation functions. These are examples of some modern activation functions, but there are many others. The designer of a CNN has to decide the best activation function for their particular circumstances.

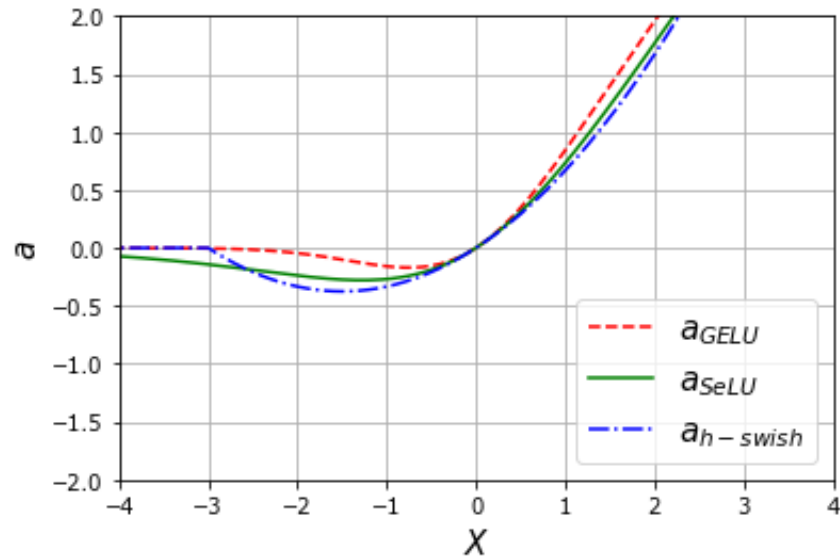


Figure 2.13: GELU, SeLU, and h-swish activation functions.

Specialized operators and modules

There are special types of operators and arrangements of these elements (modules) that CNNs can use. Let us briefly discuss the special modules that some of the CNN architectures studied in this thesis employ. For example, there are extraordinary types of convolutions like depth-wise convolution and point-wise convolution that decompose the standard convolution to make more efficient architectures [87,88], or group convolutions that can be used to generate a new dimension in CNNs called cardinality that refers to the number of transformations

in a layer [89]. By augmenting the cardinality of its layers, a CNN can improve its performance without augmenting its capacity (roughly speaking, number of parameters, depth, and width).

Also, some modules or blocks can be directly inserted into CNNs. Modules are specific arrangements of CNN elements that work in certain configurations to endow models of certain behaviors or capabilities, such as the inception module [90] that performs multi-scale feature extraction using kernels of different sizes and dimension-reduction operations, then concatenates the outputs of these operations to feed the next layer. Or attention modules, like Squeeze-and-Excitation blocks [91] that use global pooling and an MLP to add channel-wise attention to a CNN, or convolutional block attention module (CBAM) [92] that adds channel and spatial attention to a CNN layer in a way similar to the operation of Squeeze-and-Excitation blocks.

Here, a few special operators and modules are listed to illustrate that when creating a CNN model, there are many design options to consider. Even when choosing modules that are fixed arrangements of operators, the designer has to determine where to implement them in a model and how many times to use them. All of these are design hyperparameters.

2.4.2 Learning hyperparameters

The learning hyperparameters are all the choices that define the learning procedure. Speaking of ANNs, some of the fundamental learning hyperparameters include the selection of a proper cost function, optimizer for learning the parameters (as the endemic hyperparameters of the optimization algorithm), data augmentation techniques, regularization methods in the cost function, optimizer, and augmentation, the selection of stop criteria for the optimizer, etcetera.

Let us explain some prominent learning hyperparameters to keep under consideration when training a CNN model. These will become fundamental for the research presented in Chapter 3.

Learning rate

The learning rate is the size of the change in the parameters' values for every optimization iteration using a gradient descent algorithm [7]. Hence, the learning rate is closely related to the optimizer and is one of the most influential hyperparameters for ANN models [1,73].

The learning rate value can determine the convergence of the learning procedure. High learning rates can generate an unstable learning procedure by causing oscillation along the local optimal [73,75]. Whereas low learning rates can be inefficient at approaching the local optimal, drastically augmenting the needed time for training [73,75]. Thus, the recommended approach for training DL models is to use a dynamic learning rate, starting with higher values to accelerate convergence and decreasing the learning rate iteratively to reduce oscillating when getting closer to the local optimal.

There is a whole area of approaches to deliver dynamic learning rates. The techniques used for this purpose are called learning rate schedulers. Nevertheless, learning rate schedulers have their own hyperparameters, increasing the number of choices for developers to set when generating a model. Also, one model can deal with different learning rates, using a learning rate per layer, per neuron, or even per parameter during training [73,75]. Nowadays, it is common to use optimizers with different mechanisms for adaptive rate of change per parameter. This thesis uses this type of optimization algorithms, which are specified in each section. However, these optimizers still apply a nominal learning rate customarily delivered by a learning rate scheduler, and the details about their specialized mechanisms lay outside the scope of this thesis.

Optimizer

Optimization algorithms for training DL models have their own research agenda. Some of the challenges that optimizers have to overcome to train ANNs effectively are the networks' complexity, training data scarcity, and non-convex optimization surfaces.

SGD is still the foundation of many optimizers currently used for training DL models like Adam and Adamax [93]. As the optimizers get better at finding local

optimal, they also get more complex and frequently have hyperparameters that require tuning. For instance, SGD-based optimizers regularly use momentum factors to avoid early convergence issues, factors that need to be set by the user.

Some optimizers can work better for some architectures and datasets than others. Furthermore, the performance of an optimizer is dependent on the interactions between itself, its hyperparameters, the learning rate, the model, and the data [73]. DL developers have to choose the most appropriate optimizer for their system among the options available to train ANNs. Chapter 3 covers this matter.

Weight decay

Weight decay is also known as L2 regularization, and as the name suggests, it is a regularization technique. It is widely used in ML algorithms, including DL [7]. In ANNs, weight decay discourages parameters with high values by summing a factor that accounts for the L2 norm of the synaptic weights to the loss function. Then, the loss with weight decay is

$$\mathcal{E}_{WD} = \mathcal{E} + \lambda [\{\theta \setminus \mathcal{B}\}]^T [\{\theta \setminus \mathcal{B}\}] \quad (2.60)$$

where \mathcal{E} is an arbitrary loss function, λ is a scalar (and a hyperparameter) that controls the contribution of the L2 norm to the loss, θ is the set of the network trainable parameters, $\mathcal{B} \in \theta$ is the subset of biases, and $[\{\theta \setminus \mathcal{B}\}]$ is a vector containing all the trainable parameters of the network but the biases.

Data augmentation

Training DL models demands vast amounts of data. If there is not enough data available for training, the model will fail at learning the structure of the target domain. An option to deal with this common issue is creating new artificial data by transforming the training samples with expected variations of the data type while keeping the labels the same [75].

Data augmentation implicitly adds regularization to a model since it augments its generalizability by expanding the domain of training samples. Data augmentation uses sets of random transformations in the training input data. In the context

of images, some regularly used transformations are flipping the images, rotating them, cropping portions of them, altering the color channels, or even injecting noise into the data [7].

However, it is imperative to consider the features that are essential to our target data and task when performing transformations to the training data, as some transformations can alter these features, causing the algorithm not to learn them. A classic example of this is the assessment of images with textual information, where the orientation of the symbols is relevant to their meaning. Hence, the type of data augmentation to apply is another relevant hyperparameter for the generation of DL models.

Initialization values

The first step in training an ANN is initializing its parameters' values. This procedure is referred to as weight initialization. Properly doing so is a determinant factor for the convergence of the training process [94, 95]. The optimal initialization values are data-dependent, so it is unattainable to know them a priori [94]. Several techniques for parameter initialization of ANNs exist [94, 95]. Choosing a suitable one can accelerate the learning procedure and lead to better local optimal sets of parameters [95].

When using transfer learning, the initial parameter values are the ones of the pre-trained model. Still, usually, at least the last part of the networks (the classifier for classification models) has to be constructed from scratch. Thus, their parameters need an initialization technique. Three approaches exist for initialization parameters with unknown values: (1) random initialization, (2) data-driven initialization, and (3) hybrid paradigms [95].

Designing efficient weight initialization techniques is challenging since ANN optimization is yet to be understood [9]. One of the few properties that is well-known about it is that the initial parameter values have to break symmetry, so the units learn different things, a situation that motivates random weight initialization from Gaussian or uniform distributions [9]. The characteristics of the distribution to draw the parameters' initial values are common hyperparameters when training DL models.

All the CNNs employed within the research of this thesis use transfer learning. Therefore, the initialization values are the pre-trained parameters, which are specified in each section.

Training epochs

A training epoch refers to one complete iteration over all the training samples. As DL models use batches of data during training, one training epoch uses all the training batches. Training epochs are the default metric measure for training. Nevertheless, there is no deterministic method to define the required epochs to train a given ANN.

The developer has to determine the length of the training procedure. It can be training for a fixed number of epochs (a number that usually comes from the developer's own experience) or another type of automatic stop criterion, for instance, using some performance metric of the ANN to determine when training is not improving any longer the performance of the model. A good practice for this is keeping apart a subset of the training data to assess the performance of the model under training. This subset is called the validation set and is instrumental for hyperparameter tuning.

The methods that automatically stop the training process using some metric other than the number of epochs are known as stop criteria or early stopping. An early-stop method is also a form of implicit regularization since it can help detect overfitting and stop training, thus impeding the model from continuing to adjust only on the training dataset for the sake of generalization [1].

Loss function

The standard loss function for training DL models for image classification is the cross-entropy loss [1, 7, 96] as stated in Section 2.3.1. Nevertheless, recent evidence shows that the selection of the loss function can impact the performance of classification DL models [96]. Therefore, the choice of the loss function can also be considered as a hyperparameter that requires careful tuning from DL practitioners. This is further elaborated in Chapter 3.

2.5 Evaluation metrics for image classification

A central topic of classification systems is the proper evaluation of classification performance, which can be qualitative (using expert knowledge of the target domain) and quantitative (based on sampling methods) [6]. This thesis focuses on the quantitative evaluation of the performance and deals with the single-label multi-class classification task with K different classes. The output of the CNN model is a logit vector $\widehat{y} \in \mathbb{R}^K$. The classification algorithms in this thesis consider that every image belongs to the class with the highest positive value in \widehat{y} .

The following metrics are used to evaluate the classification performance of the studied algorithms: categorical accuracy (CAT-ACC), macro-specificity (Macro-SPEC), macro-precision (Macro-PREC), macro-recall (Macro-REC), macro-average F_1 score (F1), and Matthew's correlation coefficient (MCC):

$$\text{CAT-ACC} = \frac{\sum_{k=1}^K TP_k}{\rho} \quad (2.61)$$

$$\text{Macro-SPEC} = \frac{1}{K} \sum_{k=1}^K \frac{TN_k}{TP_k + FP_k} \quad (2.62)$$

$$\text{Macro-PREC} = \frac{1}{K} \sum_{k=1}^K \frac{TP_k}{TP_k + FP_k} \quad (2.63)$$

$$\text{Macro-REC} = \frac{1}{K} \sum_{k=1}^K \frac{TP_k}{TP_k + FN_k} \quad (2.64)$$

$$\text{Macro-F1} = 2 \times \frac{\text{Macro-PREC} \times \text{Macro-REC}}{\text{Macro-PREC} + \text{Macro-REC}} \quad (2.65)$$

$$\text{MCC} = \frac{\rho \times \sum_{k=1}^K TP_k - \mathcal{R}_1}{\sqrt{(\rho^2 - \mathcal{R}_2)(\rho^2 - \mathcal{R}_3)}} \quad (2.66a)$$

where

$$\mathcal{R}_1 = \sum_{k=1}^K (TP_k + FP_k) \times (TP_k + FN_k) \quad (2.66b)$$

$$\mathcal{R}_2 = \sum_{k=1}^K (TP_k + FP_k)^2 \quad (2.66c)$$

$$\mathcal{R}_3 = \sum_{k=1}^K (TP_k + FN_k)^2 \quad (2.66d)$$

Here, TP_k , TN_k , FP_k and, FN_k stand for *true positive*, *true negative*, *false positive* and, *false negative*, respectively, for a given class k , and ρ is the total number of predicted samples.

Accuracy is the most used metric to assess the performance of models for medical image classification in the literature [8]. Nonetheless, accuracy alone can be mischievous when using imbalanced datasets since it considers all the samples the same, disregarding the proportion of data [97]. Hence, classes with fewer samples have little impact on the overall accuracy performance. Data imbalance is endemic to the medical domain, so the MCC is elected as the principal performance indicator in the thesis since this metric is suitable for total unbalanced classification models [28, 97].

To assess the performance of a model for medical image classification, consider that in the industry, the standard for automatic detection systems is 0.85 for specificity and sensitivity (recall) [28]. For inference speed, the metric is the number of process images per second or frames per second (FPS). In the context of image analysis, models capable of processing at least 64 FPS can be considered capable of performing in real-time, a vital trait in clinical settings [28].

Chapter 3

Genetic algorithm-based hyperparameter optimization

Abstract

Machine learning has evolved rapidly during the last decade. Specifically, the neural networks field has thrived with the advent of the big data era and recent technological advances, allowing the implementation of deep learning techniques. Consequently, convolutional neural networks became the mainstream approach for the development of computer vision systems. Yet, the continuously growing complexity of these networks, as their several design choices (hyperparameters) and their intricate interactions that define the performance of these models in particular tasks, hinder the spreading of this technology to non-machine learning expert users. Automated machine learning offers a solution to this issue, relying on hyperparameter optimization algorithms for the automatic selection of these enigmatic variables. Genetic algorithms are suitable for non-convex, non-differentiable optimization tasks and are easy to implement and parallelize, all advantageous qualities for deep learning applications.

For these reasons, this chapter covers the creation of Gen-CNN: an automated machine learning framework based on a genetic algorithm that optimizes the selection of hyperparameters and automatically generates convolutional neural network models optimized for the classification of images in the domain of a given

dataset. Gen-CNN aims to facilitate the utilization of convolutional neural networks for real-world applications. Hence, it exclusively employs low computational complexity CNN architectures that can operate in ordinary electronic devices. The proposed framework is tested on three medical image datasets: KVASIR-v2, ISIC-2019, and BreakHis. The experimental results prove that Gen-CNN generates CNN models with classification performance comparable to or even better than state-of-the-art systems in the current literature with models of significantly less computational complexity. Gen-CNN code is available at Github.com/RogeGar/Gen-CNN

3.1 Introduction

The advent of AI in every aspect of modern life has created the expectation that it will soon become a core supporting element for medical practice [20, 20–22, 29, 98]. At the research level, there are cases of DL systems for medical image analysis with analog or superior performance than medical specialists [20, 22, 23, 25, 26, 30]. Yet, many challenges remain to overcome before these systems can meet the expectations and necessities of the medical community for their integration into actual practice [8, 29, 99, 100].

A current limitation of DL systems in medicine is the computational load that they convey, which is a constraint for deployment in clinical settings [7, 8, 29, 31]. Furthermore, the capability of performing in real-time is a desired characteristic for medical image analysis systems [25, 28, 99, 101] that many state-of-the-art architectures can not deliver [31].

The continued demand to keep improving the DL models' performance in terms of accuracy usually leads to the creation of models with increased complexity to expand the models' capacity and translate that into better performance [102]. However, large-scale models' computational needs hamper their viability for resource-constrained applications [102–104]. Also, increasing the size and complexity of DL models can generate other issues, such as over-fitting [29, 105], exploding gradients, and degradation problems [29, 103]. Besides, more complex models usually lack interpretation methods, which are essential for medical applications [8].

The selection of the ML paradigm and configuration of their design and training hyperparameters define the performance of ML models [106–109]. Effectively setting all of these options is burdensome and problematic, particularly for inexperienced AI practitioners [106, 109, 110]. In addition, the optimal selection of model and hyperparameters is dataset-dependent [107, 111]. Previous studies have demonstrated that a proper configuration of hyperparameters is decisive for the final classification performance of a DL model in the medical field [8].

Designing custom models (manually) has been effective for different CV applications in the past [112], but the broadening of design choices hinders finding optimal models [16, 106–111]. Thus, creating DL models for medical image analysis demands both time and expert ML knowledge to generate optimal models considering the particular dataset and necessities of the field [109]. This situation can delay the development of these systems and dissuade non-AI experts (including medical doctors) from following this research path [109]. However, the physicians' collaboration is crucial for this field's progress [28, 29, 99–101, 113, 114].

AutoML comes as a solution for these circumstances. AutoML enables the creation of ML applications for non-expert ML users, improves the performance of models, and accelerates ML research [106, 107]. HPO is the nuts and bolts for hyperparameter selection automation, so AutoML frameworks can effectively generate ML models to solve practical problems [106, 111].

HPO deals with non-convex, non-differentiable optimization problems with continuous, discrete, categorical, and conditional variables, reasons why regular optimization techniques are inadequate for it [109, 111]. Therefore, HPO systems take optimization approaches that can operate under these conditions, such as decision-theoretic approaches, bayesian optimization (BO) models, multi-fidelity optimization techniques, and metaheuristics algorithms [107, 108, 111, 115]. Nonetheless, the extensive and growing number of hyperparameters and complex search space have left many techniques, like BO models, impractical to use [106]. Parallel computing can tackle these issues; thus, it has become a prized feature for optimization algorithms in the context of HPO in DL systems [106].

On account of the above statements, this chapter covers the development of an AutoML framework founded on a GA that performs HPO, Gen-CNN. GAs are a type of evolutionary scheme that can effectively find optimal solutions for many

ML problems, especially in large and complex search spaces when the objective function evaluation is expensive [107, 110, 115, 116]. Moreover, GAs do not require many constraints and are easy to implement and parallelize [107].

Gen-CNN delivers automatically generated CNN models for image classification using datasets as input. Figure 3.1 shows a diagram of the Gen-CNN operation. It uses pre-trained CNNs to deal with applications with scarce data. In contrast to other HPO frameworks, Gen-CNN targets the maximization of the validation MCC to improve the overall classification performance of the resulting model and counteract meta-overfitting. Additionally, it uses a low-compute regime for the GA’s evaluation phase to attenuate the computational load of the algorithm and accelerate the HPO, approach that has been successful in the connected area of neural architecture search (NAS).

3.2 Hyperparameter optimization

As Section 2.4 explains, the concept of hyperparameters covers all the choices that define an ML model and training procedure. Contrary to parameters, the quantitative measurement of the influence on the final models’ performance of the different hyperparameters is still a research open dilemma [106]. Ergo, HPO is a prominent researched AutoML subfield [108].

Generating effective ML models for practical applications involves the hyperparameter selection enterprise, for which manual hyperparameter tuning remains the prevalent practice, even at the research level [106, 110, 111], even though it is wasteful in terms of time and expert knowledge [106, 107, 109, 111, 112]. Nonetheless, manual tuning grows ineffective due to the increasing number of hyperparameters, their complex non-linear interactions, and protracted evaluation [106, 111]. Thus, research on HPO algorithms keeps on gaining importance [111].

Again, as Section 2.4 describes, there are design and training hyperparameters. NAS is a research area encompassing algorithms that focus on design hyperparameters of DL models, and their purpose is automatically creating architectures [82, 102, 117–119]. NAS has proliferated in the last decade and has outperformed some of the best human-design architectures on several tasks, including

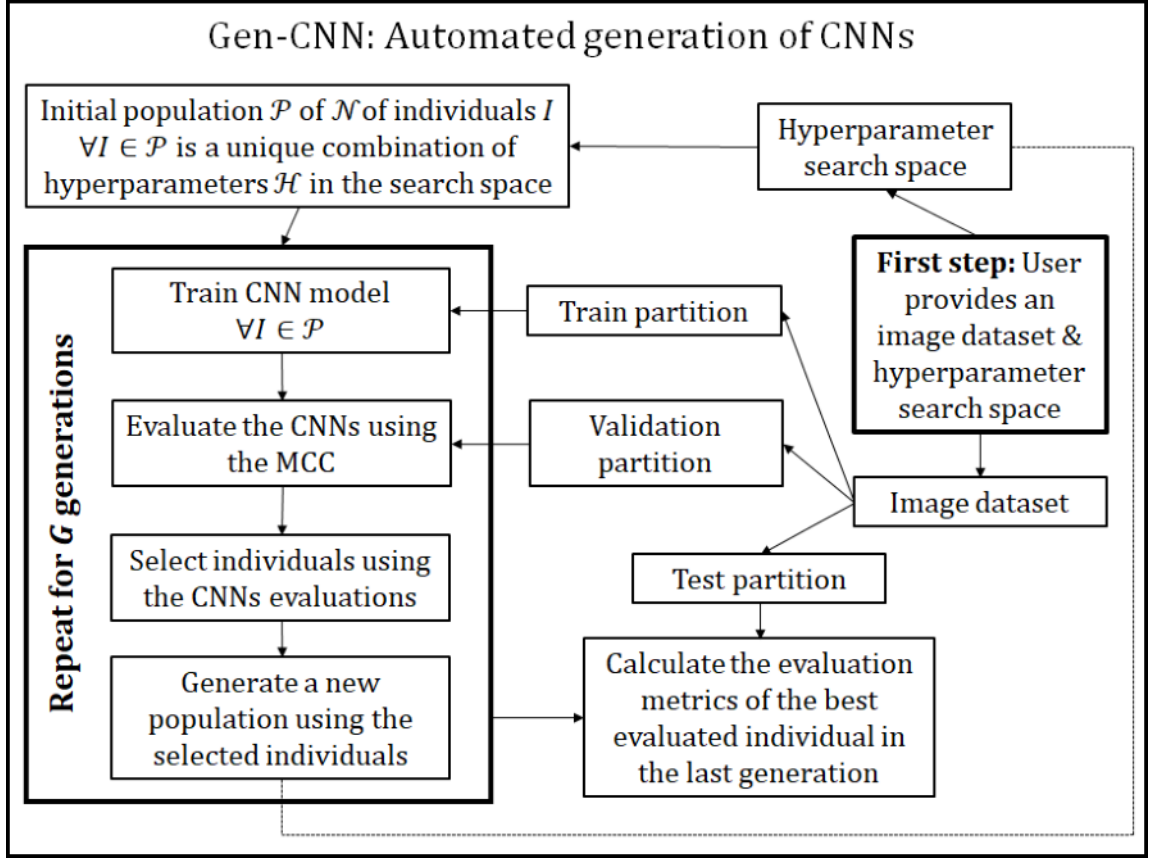


Figure 3.1: Diagram of Gen-CNN, the developed framework for the automatic generation of CNNs using a GA for HPO.

image classification [86, 102, 112, 117, 118].

NAS overlaps with HPO and can be seen as a subfield of it [117]. However, they differ in their search spaces' nature, so they tend to use different techniques [117]. HPO operates under a search space of numerical and categorical variables whose domains are treated independently [82, 117]. On the other hand, NAS optimizes the topology of the architecture, which can be represented as an acyclic graph, so its search space is usually discrete and can be represented with a graph since it is a hierarchical structure of conditional hyperparameters [112, 117, 118].

HPO systems are usually based on multi-fidelity optimization techniques, BO models, decision-theoretic approaches, and metaheuristic algorithms since these techniques can deliver results for non-convex and non-differentiable optimization

surfaces [107,108,111].

From the decision theoretical methods, grid search (GS) and random search (RS) are techniques that find the best set of hyperparameters space by evaluating different sets inside a defined search space [107,111]. For GS, the user specifies a domain of values for each variable, and the algorithm uses the Cartesian product of the set values as possible solutions to evaluate. GS is easy to implement and parallelize [111]. Nevertheless, it does not exploit the information acquired during the evaluations and suffers from the curse of dimensionality [107,108,110,111] (the exponential growth of required evaluations with respect to the number of optimized values).

RS tries different sets of hyperparameters in the search space without exploiting the evaluated data, similar to GS. Yet, RS avoids the dimensionality curse since it performs a predefined number of evaluations independently of the number of optimized variables [107,110,111,115]. RS randomly chooses values within the search space rather than exploring it grid-wise, like GS [107,110]. Nonetheless, both RS and GS lack adaptive mechanisms to use the data from previous evaluations and are inefficient in large search spaces.

BO has been succesful in HPO [107,108,111]. Unlike GS and RS, BO utilizes the information obtained in the evaluations to adapt and propose new solutions [107]. Also, BO uses a surrogate model for the evaluation phase, which is more efficient than executing the actual objective function for every evaluation [111]. The surrogate model uses a probability distribution from the data acquired in the evaluations [107,108,110]. Then, an acquisition function suggests new hyperparameter sets in a trade-off between the search space exploration and the exploitation of the formerly evaluated regions [108,111]. BO systems use different types of surrogate models, such as random forests, Gaussian processes, and tree Parzen estimators [107,111].

Metaheuristic algorithms mimic biological behavior for searching and are common in optimization systems since they are generally suitable for non-convex, non-continuous, and non-smooth optimization problems [111]. Hence, they are an appropriate approach for HPO [107,111]. Metaheuristics such as particle swarm algorithms and GAs have been successful for HPO [107,111].

3.2.1 Genetic algorithms for hyperparameter optimization

GAs are evolutionary algorithms that use the operators of crossover and mutation on previously evaluated solutions to search in a given space. GAs are noise-robust, can perform multi-objective optimization, are easy to parallelize, and manage mix search spaces, reasons why they are often employed as global optimizers [107, 108].

Additionally, GAs have the upper hand in time complexity regarding other optimization methods for HPO, such as BO with Gaussian Process, Differential Evolution, Covariance Matrix Adaptation-Evolutionary Strategy, and hybrid Bayesian optimization with GA or Differential Evolution, all of which have time complexity is of $O(n^3)$ [107]. While GAs have an asymptotic run time of $O(n^2)$ [107].

The literature on GAs for HPO is rich. Particularly, GAs have had success in the NAS branch, those are algorithms that focus on design hyperparameters of DL models [44, 110, 118, 120–123]. On the other hand, the literature on GAs applied for HPO on training hyperparameters is quite limited. Let us discuss the significant existing studies.

Vincent et al. [107] performed an exhaustive analysis of different HPO methods for training hyperparameters of DL models. Their study included GAs and hybrid techniques incorporating GAs. For the optimization process, they considered only numerical hyperparameters: learning rate, batch size, training epochs, and momentum. The CNN architecture was manually selected from off-the-shelf models for the experiments.

Other studies have explored HPO in training hyperparameters using transfer learning. Li et al. [116] optimized the layers to fine-tune on a MobileNetV2 [124] for different datasets using a GA. Ozcan et al. [125] experimented with a GA and swarm algorithms to optimize the training epochs, batch size, learning rate, weight decay, momentum, and data shuffling during training using transfer learning on an AlexNet [126].

As for AutoML systems that use GAs to generate CNN models for image classification targeting the specific dataset, there is CNN-GA [127], which employs a NAS algorithm called Genetic CNN [122] for the automatic generation of the DL models. Yet, this framework is centered on the design hyperparameters and uses

fixed standard training hyperparameters.

Finally, Han et al. [115] performed HPO on both training and design hyperparameters using a GA. They optimized the learning rate, momentum, batch size, kernels' size, number of kernels, and number of neurons. However, their algorithm requires the user to predefine the network structure, and it was only tested on toy datasets in the generation of toy models just to prove the efficacy of GAs in HPO.

Contrarily to all existing work, the system developed in this thesis, Gen-CNN, centers on solving the generation of practical models by utilizing a GA that optimizes categorical (CNN architecture, loss function, and optimizer) and numerical hyperparameters (learning rate, weight decay), uniquely considering the CNN architecture as a hyperparameter, selecting the optimal off-the-shelf architecture within the computational requirements of the user. Also, to the best of the author's knowledge, it is the first HPO system that considers the loss function as another training hyperparameter since different loss functions can lead to different (and better) outcomes [96].

In Gen-CNN, the optimizer (gradient-based for training) focuses on minimizing the loss in the training partition, and the GA deals with maximizing the validation MCC. So, the two optimization procedures operate in parallel with different targets on different data partitions. This behavior is introduced to address the critical and often overlooked issue of meta-overfitting of HPO systems [108, 128], which refers to overfitting to any of the data partitions or selecting apparently-optimal hyperparameter well evaluated for the stochasticity in the training procedure. Hence, in Gen-CNN the prevailing hyperparameters are the consistently good ones during the evolution of the population.

Also, using the MCC as the target metric regards the inherent imbalance of real-world datasets, helping Gen-CNN to generate models that perform well across all the classes of the input dataset.

3.3 Methodology

3.3.1 Search space

Defining the search space is fundamental for any optimization procedure since it contains all the possible solutions for the problem. In HPO, doing so is particularly challenging and consequential in the performance of the system [108]. When the hyperparameter space is too large, the curse of dimensionality desolates many optimization approaches, and other methods are ineffective since there is no efficient trade-off between exploration and exploitation. Whereas if the search space is small, the resolution of the HPO might not meet the expectations [108, 110]. Therefore, an HPO algorithm has to find as few hyperparameters as possible as long as the final performance of the model is acceptable [108].

The development of Gen-CNN was sequential. Early versions of the framework [129, 130] consider other hyperparameters that were left behind by the above argument. Let us describe the considered hyperparameters for the final version of Gen-CNN.

CNN architecture

The goal is to automatically generate practical models that meet the requirements for deployment in real-world applications, particularly for medical image analysis. As discussed in Section 1.1.4, the fundamental needs are high accuracy, computational efficiency, capability to learn with scarce data, and interpretability. All of these attributes can be fulfilled by CNN models. Even though other types of architectures, namely Transformers, can have better classification performance under certain circumstances, research has shown that CNN models can equal or even outperform Transformer-based state-of-the-art architectures with appropriate architectures and learning configuration [17]. Therefore, Gen-CNN constrains to CNN models. With the HPO, Gen-CNN boosts the classification performance of off-the-shelf architectures by finding the optimal configuration of their learning hyperparameters.

Some architectures can adapt better to some datasets, and the GA performs HPO to a specific dataset, finding the best-suited architecture for the dataset

under the computational requirements of the user. For efficiency, the search space of Gen-CNN in this thesis consists of off-the-shelf CNNs with the highest reported accuracy in classification tasks and low floating point operations (FLOP)s regime. Architectures capable of performing on a mobile regime (with a FLOP complexity of up to $\sim 600MFLOPs$ [16]) from a family of CNNs are included when available. Table 3.1 shows the architectures within the search space of Gen-CNN for this chapter.

Table 3.1: CNN architectures in the search space for chapter 3.

CNN family	Architecture	FLOPs
ConvNeXts [17]	ConvNext-Tiny	4.5 G
EfficientNetV2 [131]	EfficientNet-V2-Small	8.8 G
RegNet [16]	RegNet-Y-400MF	400 M
	RegNet-X-400MF	400M
MobileNetV3 [86]	MobileNet-V3-Small	59 M
	MobileNet-V3-Large	225 M
ShuffleNetV2 [132]	Shufflenet-v2-x1_0	146 M
	Shufflenet-v2-x1_5	299 M
	Shufflenet-v2-x2_0	591 M

FLOPs: Floating Point Operations.

Learning rate

Defined as the step size of the parameter updating [7], this hyperparameter is closely related to the optimizer since each one utilizes it and has different standard values for it, picked by their developers to provide an acceptable behavior for a wide range of applications. Therefore, Gen-CNN introduces and optimizes a hyperparameter a that scales the predefined learning rate for the selected optimizer such that

$$\alpha = a \cdot \alpha_{optimizer} \quad (3.1)$$

where α is the learning rate to use during training, a is the scaling factor optimized using the GA, and $\alpha_{optimizer}$ is the default learning rate for the selected optimizer. Considering that Gen-CNN uses transfer learning in pretrained models

and that the learning rate for fine-tuning is usually lower than the standard recommended values, the search space of the scaling factor is set to $a \in [0.00001, 1]$.

Weight decay

Regularization is an indispensable element for any DL model. Hence, regularization hyperparameters are considered the second most influential hyperparameter type for a DL model, only after the learning rate [1]. Weight decay is comprehensively employed in the training of ML models [7]. As explained in Section 2.4.2, it is a regularization that penalizes the weights' high values in a neural network, as shown in Eq. 2.60.

The contribution of the L2 norm to the loss λ is the hyperparameter optimized by Gen-CNN. There is no deterministic way to define a good value for this hyperparameter. So, the range of values used in the respective papers of the CNN architectures in the search space is set as the search space for λ . The lowest value for this hyperparameter is used in [17] and is 1×10^{-8} . The upper bound is set to 0.1 since values this high have been effective in multiclass classification tasks [133]. Hence, the search space is $\lambda \in [1 \times 10^{-9}, 0.1]$.

Loss function

The softmax cross-entropy loss is the default loss function for image classification problems using DL [1, 7, 96]. Nevertheless, softmax cross-entropy alone might not be the best option for imbalanced datasets [7]. Additionally, there is evidence that different loss functions lead to different classification performances with the same architectures and datasets [96].

Some cross-entropy variants introduce regularization elements to the loss functions, such as label smoothing that includes a little noise into the target values, which serves to reduce the overconfidence of the model on its predictions [134]. In fact, this helps reduce the effect of miss-labeled samples on the training dataset and allows the convergence of the loss function by eliminating the everlasting variance between the labels (with hard values of 0 or 1) and the softmax output, whose values only approximate 0 and 1 [9].

Other variants that add some elements to the vanilla cross-entropy loss are

logit penalty and logit normalization. The logit penalty works similarly to weight decay. It adds up the logits L2 norm into the loss function, which penalizes large values of the output logits, constraining the logits to small values [96]. On the other hand, logit normalization restrains the effect of the outputs' magnitude on the optimizer by normalizing the logits, encouraging the logits to have the same directions as the training targets without promoting an increase in magnitude, which can cause overconfidence problems [135]. Finally, sigmoid cross-entropy is the standard loss function for training multi-label classification models. Yet, using this loss in single-label multi-class classification problems (the ones that Gen-CNN is designed for) can improve the classification performance by allowing mutually inclusive predictions over the classes in the dataset [96, 136].

The proper selection of a loss function for a specific dataset, architecture, and training configuration is unclear: it is a research topic all alone. However, the above-described loss functions have proven to be effective in improving the generalization of DL models [9, 96, 134, 135]. Hence, the softmax cross-entropy loss function and the four above-described variants (label smoothing, logit penalty, logit normalization, and sigmoid cross-entropy) comprehend the search space for the loss function hyperparameter in the optimization process using the GA. For the specific equations for these losses, please refer to [129] or [96].

Optimizer

Gradient-based optimization is the heart of neural network training. Yet, there are various approaches to this type of technique, and some algorithms can work better for particular architectures or datasets than others [137]. The search space of optimizers comprises the optimizers that were effective in training the CNNs within the search space in their respective research papers: Adam, SGD, Adamax, and RMSprop.

3.3.2 Genetic algorithm

The GA mimics the principle of evolution. The best-adapted individuals in a population have more chances of having offspring to pass their genes, preserving their characteristics in the next generation [110, 116]. In a GA, the individuals are

solutions or points in the optimization surface, and the genes are the coded information of the values to optimize [107, 110]. The first population is generated with a random function. Then, the next generations are the result of combining the information of individuals in the previous population (crossover) and introducing new random information (mutation). Crossover exploits the information from previous evaluations by binding the search to well-evaluated zones, and mutation promotes the exploration of new zones in the search space to avoid early convergence in sub-optimal regions. Iteratively, every generation improves its overall performance until convergence, and the individual with the highest evaluation is considered the global optimum [110, 111].

In the GA, a population \mathcal{P} comprises a given quantity \mathcal{N} of individuals, and every individual I is a set of genes $I = \{\iota_1, \iota_2, \dots, \iota_J\}$. In the study case, J is the number of hyperparameters to optimize, and every individual denotes a unique combination of hyperparameters \mathcal{H} in the search space.

The information encoding strategy is fundamental in any computational application, and the problem domain determines the most effective approach [138]. In the particular case of Gen-CNN, the genes encode continuous and categorical variables. Thus, real-valued genes are the best option to represent the full problem domain. Therefore, the possible hyperparameter values are encoded into the interval $[0, 1]$ since the normalization of genes improves the overall performance of GAs, especially when using crossover and mutation mechanisms [139] (as Gen-CNN does). Figure 3.2 shows the encoding of hyperparameter values into the $[0, 1]$ interval, and Figure 3.3 shows an example of what a population represents.

A CNN model is a hypothesis function $h : x \rightarrow \widehat{y}$ that maps the inputs x (image's pixels) into a predictions vector \widehat{y} . θ are the trainable parameters of the model, and $h_{\mathcal{H}}$ is the hypothesis function represented by the model trained using the configuration of hyperparameters \mathcal{H} . Therefore, a population has \mathcal{N} different hypothesis $h_{\mathcal{H}_i}$, each of them with their own set of parameters θ_i , with $i \in \{1, 2, \dots, \mathcal{N}\}$, whose outputs are

$$\widehat{y}_i = h_{\mathcal{H}^i}(x, \theta_i) \quad (3.2)$$

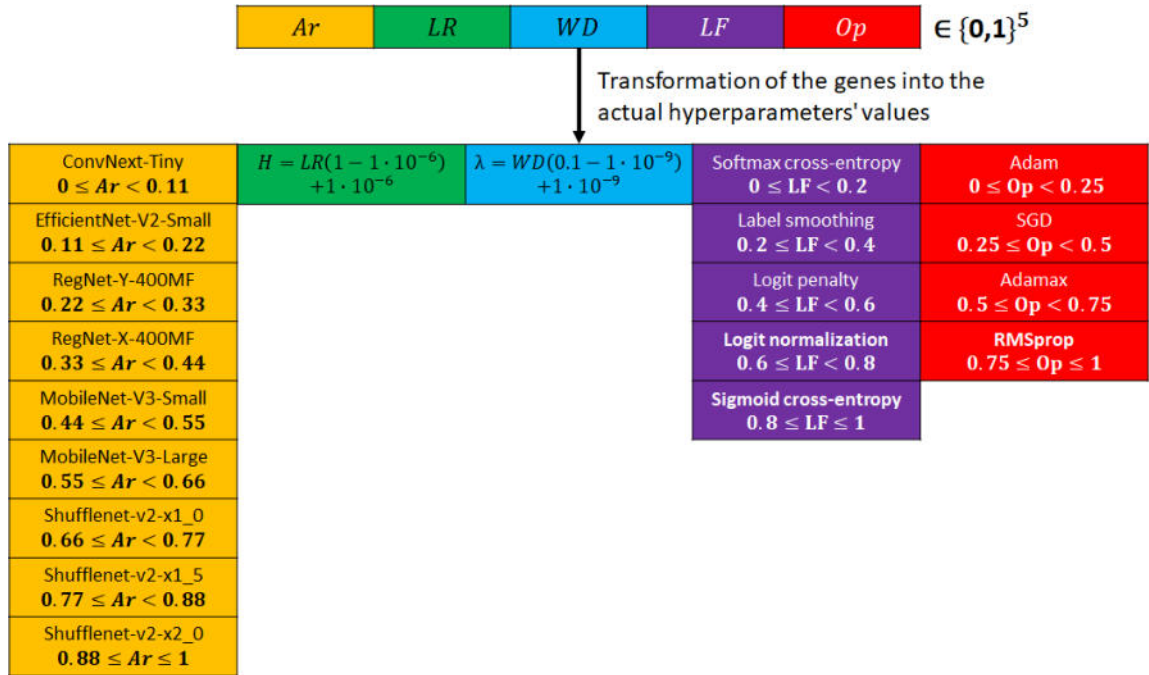


Figure 3.2: The individuals are composed of 5 genes $\{Ar, LR, WD, LF, Op\} \in \{0,1\}^5$, that stand for the hyperparameters of CNN architecture, learning rate scaling factor, weight decay factor, loss function, and optimizer, respectively. The genes' values are transformed into the hyperparameters' values as shown for each gene.

Evaluation of the individuals

Optimization algorithms have an evaluation phase to acquire the information to keep on the optimization. For GAs, the customary evaluation method is to execute the target function. For Gen-CNN, that is training the models described by the individuals in the population and estimating their performance with the elected metric (validation MCC). Evaluation is the most expensive stage in terms of time and computational resources, and many times poses a limitation for population-based optimization algorithms. In the context of Gen-CNN, training as many different DL models can be unattainable in a realistic time frame, making the whole system impractical. To solve this matter, Gen-CNN applies a technique that has been successful in the related field of NAS to save resources in the evaluation phase [16], which consists of training the candidate models in a low-compute, low-epoch regime. Specifically, training for the evaluation phase uses reduced

Population \mathcal{P}									
Individual 1									
RegNet-Y-400MF	0.001	0.026	Label smoothing	Adam					
Individual 2									
EfficientNet-V2-Small	0.068	0.031	Logit penalty	Adamax					
\vdots									
Individual \mathcal{N}									
ConvNext-Tiny	0.045	0.072	Cross-entropy	SGD					
<table border="1"> <tr> <td>CNN architecture</td><td>Learning rate scaler</td><td>Weight decay</td><td>Loss function</td><td>Optimizer</td></tr> </table>					CNN architecture	Learning rate scaler	Weight decay	Loss function	Optimizer
CNN architecture	Learning rate scaler	Weight decay	Loss function	Optimizer					

Figure 3.3: Illustration of a population of \mathcal{N} individuals where every individual is a different hyperparameter combination for a CNN model.

image sizes (112×112) and a few training epochs (3). This scheme delivers low-fidelity prototypes of the actual models but provides enough information to the HPO algorithm to map the optimization surface.

As stated before, the core of the individuals' evaluations is the validation MCC (MCC^{val}), which for an arbitrary i -th individual I in the population is defined as

$$MCC^{val}(I_i) = MCC(\mathcal{T}^{val}, \widehat{y}_i^{val}) = MCC(\mathcal{T}^{val}, h_{\mathcal{H}^i}(x_{val}, \theta_i)) \quad (3.3)$$

$\forall i \in \{1, 2, \dots, \mathcal{N}\}$, where $MCC(\cdot)$ is a function that applies Eq. 2.66 to the argument vectors, \mathcal{T}^{val} are the labels of the validation dataset, and x_{val} are the inputs (images) of the validation dataset.

As a final step, the evaluations of a population \mathcal{P} are normalized. So, the final evaluation Ev_i of an arbitrary i -th individual in the population is

$$Ev_i = \frac{MCC^{val}(I_i) - MCC_{min}^{val}(\mathcal{P})}{MCC_{max}^{val}(\mathcal{P}) - MCC_{min}^{val}(\mathcal{P})} \quad (3.4)$$

$\forall i \in \{1, 2, \dots, \mathcal{N}\}$, where $MCC_{min}^{val}(\mathcal{P})$ and $MCC_{max}^{val}(\mathcal{P})$ are the maximum and minimum MCC^{val} evoked by individuals of the population \mathcal{P} , respectively.

Selection

Using the evaluations, Gen-CNN selects individuals in a population to pass their information to the next generation. To that end, it employs a tournament mechanism in which every individual competes against another random individual in the population, and the one with the highest evaluation gets selected. Therefore, all the individuals have at least a chance to get into the selected population, but the ones with the highest evaluations have more probability of doing so. Also, one individual can get selected more than once. Thus, best-fitted individuals are more likely to pass their genes to the next generation. Algorithm 1 shows the pseudocode of the selection process¹.

Algorithm 1 Selection mechanism of the GA. Ev represents the evaluations of the population \mathcal{P} . Ev_i is the evaluation of the i th individual in \mathcal{P} , and $Rand(i)$ stands for a random index of an individual in \mathcal{P} . So, $Rand(i) \in \{1, 2, \dots, \mathcal{N}\}$. All the random values are drawn from a uniform distribution.

```
INPUT  $\mathcal{P}, Ev$ 
OUTPUT Selected population  $\mathcal{PS}$ 
FOR ALL  $i \in \mathcal{P}$ 
   $Opponent \leftarrow Rand(i) \in \{1, 2, \dots, \mathcal{N}\}$ 
  IF  $Ev_i > Ev_{Opponent}$ 
    Add  $I_i$  to  $\mathcal{PS}$ 
  ELSE
    Add  $I_{Opponent}$  to  $\mathcal{PS}$ 
  END IF
END FOR
```

Generation

The offspring of the selected individuals are the next generation of solutions to evaluate. The operators of crossover and mutation generate new individuals using the genes of the selected individuals. Any number of parents can be used to create a new individual. The GA of Gen-CNN uses two parents to generate a single individual. Each individual of the selected population mates with another

¹Note. In this thesis, when referring to random values in the GA, all of them are drawn from a uniform distribution.

random individual in the same population to have offspring. Algorithm 2 shows the pseudocode of the generation mechanism.

Algorithm 2 Generation mechanism of the GA. C and M are fixed hyperparameters that represent the crossover and mutation probabilities, and $|C| = |M|$. All individuals have the same C and M . C_i , M_i are the probabilities of mutation and crossover for the gene i . Ev^{PS} represents the evaluations of the selected population \mathcal{PS} . $Ev_{P_1}^{PS}$, $Ev_{P_2}^{PS}$ are the evaluations of the parents P_1 and P_2 , respectively. $Rand(i)$ stands for a random index of an individual in \mathcal{PS} . $Rand([0, 1])$ is a random value in the interval $[0, 1]$. All the random values are drawn from a uniform distribution.

```

INPUT  $\mathcal{PS}, Ev^{PS}, M, C$ 
OUTPUT New population  $\mathcal{P}$ 

FOR ALL  $I \in \mathcal{PS}$ 
   $P_1 \leftarrow I$ 
   $P_2 \leftarrow I_{Rand(i)} \in \mathcal{PS}$ 
   $Offspring \leftarrow P_1$ 
  FOR ALL  $i \in Offspring$ 
    IF  $Rand([0, 1]) < C_i$ 
       $i_{Offspring} = \frac{i_{P_1}Ev_{P_1}^{PS} + i_{P_2}Ev_{P_2}^{PS}}{Ev_{P_1}^{PS} + Ev_{P_2}^{PS}}$ 
    END IF
    IF  $Rand([0, 1]) < M_i$ 
       $i_{Offspring} \leftarrow Rand([0, 1])$ 
    END IF
  END FOR
  Add  $Offspring$  to the new population  $\mathcal{P}$ 
END FOR
Replace the last  $I \in \mathcal{P}$  for the best-evaluated  $I \in \mathcal{PS}$ 

```

Crossover combines the genetic information of the parents. There are different mechanisms for doing so. The most common practice is using the arithmetic mean of the genes. Gen-CNN employs the weighted arithmetic mean as a function of the evaluation of the parents. In that way, the parent with the highest evaluation has more influence on the genes of the resulting offspring. Thus, a given gene $i_{Offspring}$ of an offspring individual is given by

$$i_{Offspring} = \frac{i_{P_1}Ev_{P_1} + i_{P_2}Ev_{P_2}}{Ev_{P_1} + Ev_{P_2}} \quad (3.5)$$

where l_{p1} , l_{p2} are the genes, and Ev_{p1} , Ev_{p2} are the evaluations of the parents 1 and 2, respectively.

As for mutation, it introduces new information to the offspring genes within the search space, regardless of the parents' genes. Mutation allows the GA to keep exploring the search space and avoid premature convergence or getting stuck in suboptimal local regions. Considering that the individuals' genes are coded and normalized, then the mutation of a given gene $l_{Offspring}$ of an offspring individual is given by

$$l_{Offspring} = Rand([0, 1]) \quad (3.6)$$

where $Rand([0, 1])$ is a random value in the interval $[0, 1]$. Finally, the generation mechanism employs an elitism policy where the best-evaluated individual in the selected generation is introduced as the last element of the next generation to aid the stability of the optimization process.

3.3.3 Training resulting model

At the end of the HPO, Gen-CNN trains a model with the hyperparameters of the best-evaluated individual through the GA performance. This final model is trained in a higher compute and epochs regime since the individuals in the GA are low-fidelity prototypes. The conditions to generate the final models are training for 50 epochs with the images resized to 448×448 using the Cosine Annealing learning rate scheduler.

3.4 Results

3.4.1 Datasets

The Sign Language Digits dataset [140] is used to test the convergence of the HPO technique employed by Gen-CNN. The motivation for electing this dataset for this purpose is that for testing the convergence of heuristic algorithms, the system has to be executed several times to assess the convergence statistically, and doing so with large datasets would require a lot of time and computation. Hence, a small

dataset (with a few thousand samples) is suitable for this type of experiment. Additionally, the literature contains results of other studies using different HPO approaches on this dataset, which serves to compare results.

The Sign Language Digits dataset has 2,062 RGB images of 218 different people signaling the digits of the American Sign Language. It is a balanced dataset with minimal variations in the number of samples per class. Figure 3.4 shows an image of each class in the Sign Language Digits dataset.

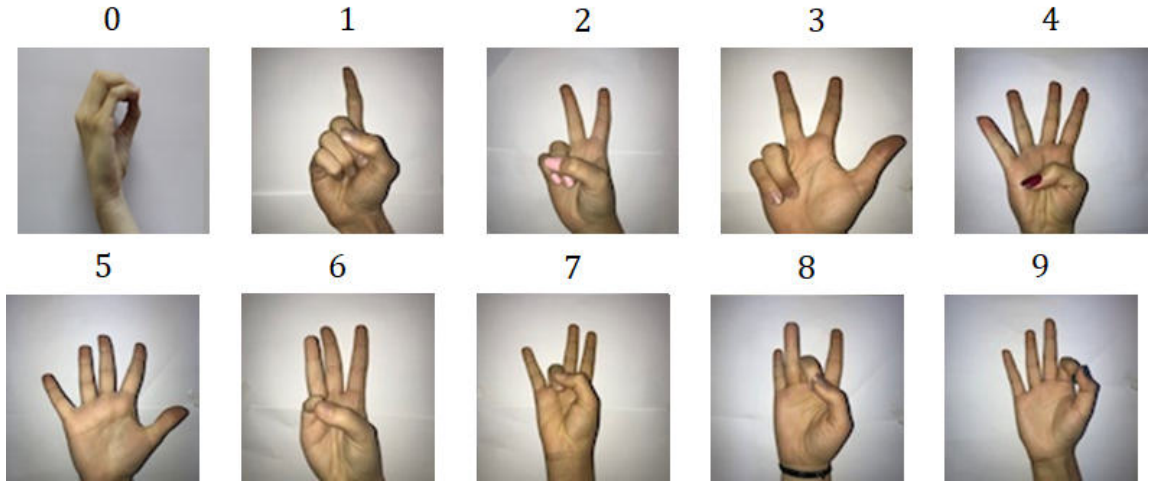


Figure 3.4: Example images of every class in the Sign Language Digits dataset [140].

The Gen-CNN framework is compatible with any type of image data. However, this thesis aims at creating practical models for medical image classification. Thus, the test subjects for the developed framework are three medical image datasets with different types of images from distinct specialties: (1) KVASIR-v2 [141], (2) ISIC-2019 training data (consisting of the BCN-20000 Dataset [142], HAM10000 Dataset [143], and MSK Dataset [144]), and (3) BreakHis-v1 [145].

The KVASIR dataset contains 8,000 images of eight different gastrointestinal findings in endoscopic images. The classes are different types of anatomical landmarks (Normal cecum, Normal pylorus, and Normal z-line), pathological findings (Esophagitis, Polyps, and Ulcerative colitis), and endoscopic procedures (Dyed lifted polyps and Dyed resection margins). Figure 3.5 shows a sample of every class in the KVASIR-v2 dataset. The KVASIR dataset is balanced. Hence, it con-

tains 1,000 images of each class.

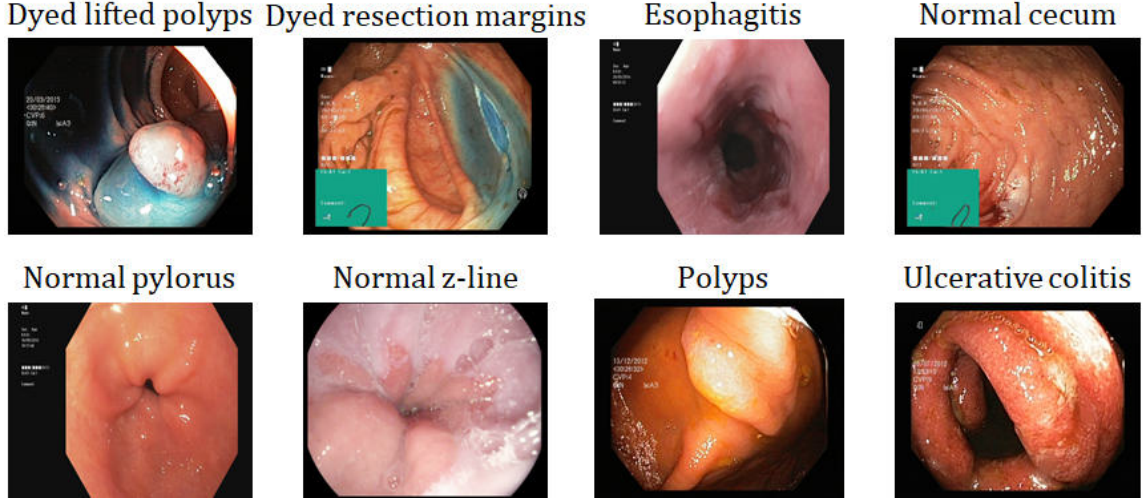


Figure 3.5: Example images of every class in the KVASIR-v2 dataset [141].

The ISIC-2019 dataset consists of 25,331 dermoscopic images that belong to one of eight different categories: Melanoma, Melanocytic nevus, Basal cell carcinoma, Actinic keratosis, Benign keratosis, Dermatofibroma, Vascular lesion, and Squamous cell carcinoma. The ISIC-2019 is an imbalanced dataset, so the number of samples for each class varies significantly. Figure 3.6 shows a sample of every class in the ISICS-2019 dataset.

The Breast Cancer Histopathological Image Classification (BreakHis) dataset comprises 7,909 microscopic images of breast tumor tissue. The images were collected at different magnifying factors (40X, 100X, 200X, and 400X) and can be categorized into two principal classes: Malignant tumors and benign. Further, each sample in the dataset also belongs to one of eight classes. There are four benign breast tumors' subclasses: adenosis (A), fibroadenoma (F), phyllodes tumor (PT), and tubular adenoma (TA); and four malignant tumors: carcinoma (DC), lobular carcinoma (LC), mucinous carcinoma (MC) and papillary carcinoma (PC). The proposed framework is employed to generate a model to classify the images of this dataset into eight different subclasses, regardless of the images' magnifying factor. The BreakHis dataset is imbalanced, hence the number of samples for class (and subclass) differs. Figure 3.7 shows a sample of every class in the BreakHis

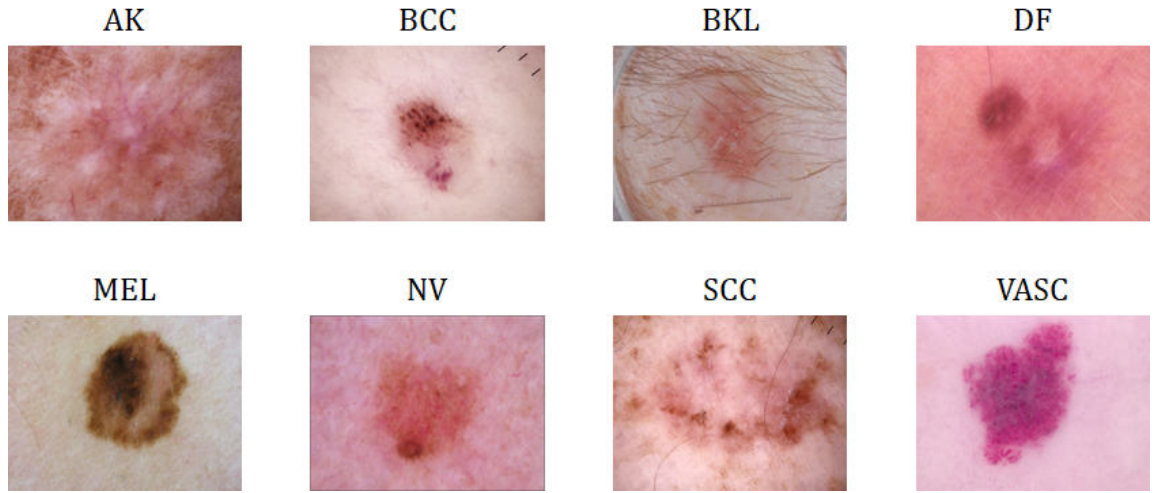


Figure 3.6: Example images of every class in the ISIC-2019 dataset [142–144].

dataset, regardless of their magnifying factors (40X, 100X, 200X, and 400X).

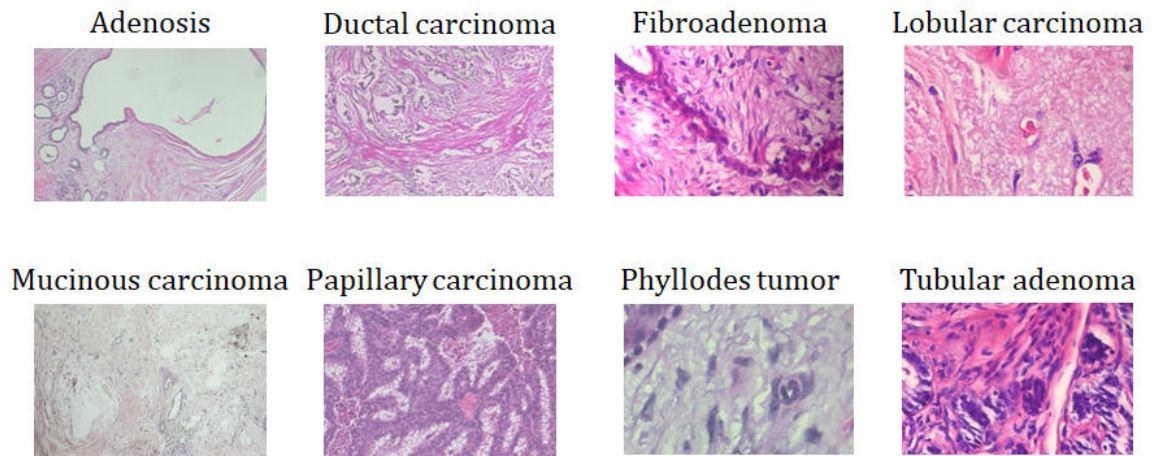


Figure 3.7: Example images of every class in the BreakHis dataset [145] (ignoring their magnifying factors).

Every dataset is split into three partitions for training, validation, and testing. Each of these partitions contains 60%, 20%, and 20% of the data, respectively. Table 3.2 shows the number of samples in each partition for all the datasets employed in the experiments.

Table 3.2: Number of samples in every dataset and partition for the experiments.

Dataset	Class	Train (60%)	Valid (20%)	Test (20%)
Sign Language Digits	0	123	41	41
	1	124	41	41
	2	124	41	41
	3	124	41	41
	4	124	42	41
	5	124	42	41
	6	124	42	41
	7	124	41	41
	8	125	42	41
	9	122	41	41
Total	2,062	1,238	414	410
KVASIR v2	Dyed lifted polyps	600	200	200
	Dyed resection margins	600	200	200
	Esophagitis	600	200	200
	Normal cecum	600	200	200
	Normal pylorus	600	200	200
	Normal Z-line	600	200	200
	Polyps	600	200	200
	Ulcerative colitis	600	200	200
Total	8,000	4,800	1,600	1,600
ISIC-2019	AK (Actinic keratosis)	520	174	173
	BCC (Basal cell carcinoma)	1,994	664	665
	BKL (Benign keratosis)	1,574	525	525
	DF (Dermatofibroma)	143	48	48
	MEL (Melanoma)	2,713	904	905
	NV (Melanocytic Nevus)	7,725	2,575	2,575
	SCC (Squamous cell carcinoma)	377	126	125
	VASC (Vascular lesion)	152	51	50
Total	25,331	15,198	5,067	5,066
BreakHis	Adenosis	266	89	88
	Ductal carcinoma	2,070	690	691
	Fibroadenoma	608	203	203
	Lobular carcinoma	376	126	124
	Mucinous carcinoma	475	158	159
	Papillary carcinoma	337	112	111
	Phyllodes tumor	271	91	91
	Tubular adenoma	342	114	114
Total	7,909	4,745	1,583	1,581

Data augmentation

During training, every batch of data goes through a transformation phase where the techniques of random crop, random rotation (of up to 90°), random vertical flip (50% of occurrence probability), random horizontal flip (50% of occurrence probability), and color jitter transformation serve as augmentation techniques.

3.4.2 Experimental setting

All the experiments used the same software and hardware conditions: AMD Ryzen 5 3400G CPU, one NVIDIA GeForce GTX 1660 Ti GPU, 16 GB RAM, and 476 GB system memory. All the algorithms were implemented in Python 3.8.16, using the environment Spyder 5.4.2. Pytorch 1.13.1 and CUDA 11.6 are the frameworks for neural networks and optimizers (Adam, SGD, Adamax, and RMSprop). The optimizers used their default values in Pytorch for their specific hyperparameters.

3.4.3 Testing the convergence of Gen-CNN

The convergence of heuristic algorithms, such as GAs, is not deterministic. Therefore, heuristic algorithms can reach different solutions for the same problem under the same conditions. Yet, a good optimization platform should deliver reliable results (with similar performance), regardless of the approach. The guidance to test the convergence of heuristic algorithms is executing them on the same task and under the same conditions for 30 repetitions to assess the similarity of results [125]. This experiment for Gen-CNN was carried out using the Sign Language dataset. Table 3.3 displays the results of the experiment and other HPO algorithms in the literature using the same dataset and number of evaluations during the optimization (results from [125]). It is important to remark that the studies in Table 3.3 also deal with hyperparameter optimization, but their optimization target differs from the Gen-CNN's target since they are minimizing the error rate on the validation dataset (D_{val}), so their algorithms' target is

$$\min_{\mathcal{H}} [1 - \text{CAT-ACC}(h_{\mathcal{H}}(D_{val}))] \quad (3.7)$$

whereas Gen-CNN's target is

$$\max_{\mathcal{H}} [\text{MCC}(h_{\mathcal{H}}(\mathcal{D}_{val}))] \quad (3.8)$$

In the optimization targets (3.7) and (3.8), $h_{\mathcal{H}}$ is the mapping function denoted by the CNN model configured by the i -th set of hyperparameters \mathcal{H} in the population, as defined in previous sections. Although the other algorithms in Table 3.3 are directly maximizing the CAT-ACC (by minimizing the error rate), whereas Gen-CNN maximizes the MCC, Gen-CNN attains a higher mean CAT-ACC, which suggests that the selection of MCC as the optimization target is effective in enhancing the overall classification performance of the generated models.

Table 3.3: Performance comparison of the proposed approach to other HPO algorithms using the Sign Language Digits dataset [140]. Mean and standard deviation of the resulting models' Cat-ACC after 30 repetitions of the optimization algorithms. All the algorithms used the same number of evaluations during the optimization cycle.

Optimization algorithm	Optimized hyperparameters	CAT-ACC Mean (STD)
Genetic algorithm [125]	Training epochs, batch size, learning rate, weight decay, data shuffling, momentum	0.9830 (± 0.0075)
Particle swarm optimizer [125]	Training epochs, batch size, learning rate, weight decay, data shuffling, momentum	0.9816 (± 0.0105)
Artificial bee colony algorithm [125]	Training epochs, batch size, learning rate, weight decay, data shuffling, momentum	0.9840 (± 0.0081)
Gen-CNN (proposed approach)	CNN architecture, learning rate, weight decay, loss function, and optimizer	0.9918 (± 0.0138)

3.4.4 Assessing Gen-CNN against state-of-the-art hand-design models for medical image classification

To test the performance of the models automatically generated using Gen-CNN, these are compared to state-of-the-art hand-design models in the three medical image datasets. Gen-CNN uses the hyperparameters and training conditions that Table 3.4 shows for all the experiments. The resulting models are compared against the models with the highest reported classification metrics in the current

literature for each dataset.

Table 3.4: Fixed Gen-CNN’s hyperparameters and final training conditions for the experiments with the KVASIR-v2, ISIC-2019, and BreakHis datasets. The crossover and mutation probabilities are normalized, and the values are for the genes of CNN architecture, loss function, optimizer, learning rate, and weight decay, respectively.

Hyperparameter	Value
For Gen-CNN	
Generations	25
Individuals	50
Training epochs for each individual	3
Image size	112×112
Evaluation function	MCC (validation)
Crossover probability	[0, 0, 0, 0.9, 0.9]
Mutation Probability	[0.4, 0.4, 0.4, 0.2, 0.2]
For training the model with the resulting optimal hyperparameters	
Epochs	50
Image size	448×448
Learning rate scheduler	Cosine Annealing

The best-evaluated individual during the whole optimization process is considered the optimal solution. Table 3.5 shows the optimal hyperparameters set found using Gen-CNN for the three tested datasets. Then, the final model is trained with the optimal hyperparameters found (on a single HPO implementation) and using the conditions that Table 3.4 shows.

The classification performance metrics using the test partition were calculated for the comparison of the automatically generated models using Gen-CNN and the state-of-the-art hand-design models in the literature. Figures 3.8, 3.9, and 3.10 show the confusion matrix for the KVASIR-v2, ISIC-2019, and BreakHis datasets, respectively. Tables 3.6, 3.7, and 3.8 show the classification performance metrics of the resulting models using Gen-CNN and the hand-design models with the highest classification metrics in the current literature.

Table 3.5: Optimal hyperparameters found using a single run of Gen-CNN for the three medical image datasets.

Optimal hyperparameters	KVASIR-v2	ISIC-2019	BreakHis
Architecture	EfficientNet-V2-Small	ConvNext-Tiny	ConvNext-Tiny
Loss function	Label smoothing	Cross-entropy	Label smoothing
Optimizer	Adam	Adam	Adam
H	0.0399	0.0381	0.0541
WD	0.1835	0.0541	0.1771
Optimal generation	25	8	13
Validation MCC ★	0.9157	0.7668	0.8117

★ The validation MCC is of the low-fidelity prototype model used for the evaluation phase in the HPO.

KVASIR-v2

	PREDICTED								
TRUE	191	7	0	0	0	0	2	0	Dyed lifted polyps
	8	192	0	0	0	0	0	0	Dyed resection margins
	0	0	163	0	0	37	0	0	Esophagitis
	0	0	0	195	0	0	2	3	Normal cecum
	0	0	0	0	199	0	0	1	Normal pylorus
	0	0	18	0	0	182	0	0	Normal z-line
	1	0	0	2	4	0	189	4	Polyps
	0	0	0	3	1	0	1	195	Ulcerative colitis
	Dyed lifted polyps	Dyed resection margins	Esophagitis	Normal cecum	Normal pylorus	Normal z-line	Polyps	Ulcerative colitis	

Figure 3.8: Confusion matrix in the test partition of the KVASIR-v2 dataset of the resulting model using a single run of Gen-CNN.

3.5 Discussion and conclusions

Nowadays, several CNN architectures capable of classifying images with outstanding performance exist. However, some architectures adapt better to some

ISIC-2019									
PREDICTED									
TRUE	119	18	16	1	9	3	7	0	AK
	5	635	5	1	6	10	3	0	BCC
	14	12	436	1	20	33	9	0	BKL
	0	3	0	34	4	5	1	1	DF
	7	21	16	1	721	130	7	2	MEL
	3	13	16	0	61	2477	3	2	NV
	7	11	3	1	4	3	96	0	SCC
	0	0	0	0	1	0	0	49	VASC
	AK	BCC	BKL	DF	MEL	NV	SCC	VASC	

Figure 3.9: Confusion matrix in the test partition of the ISIC-2019 dataset of the resulting model using a single run of Gen-CNN.

BreakHis										
		PREDICTED								
TRUE	Adenosis	89	0	0	0	0	0	0	0	Adenosis
	Ductal carcinoma	0	665	0	24	2	0	0	0	Ductal carcinoma
	Fibroadenoma	2	1	197	0	0	0	3	0	Fibroadenoma
	Lobular carcinoma	0	17	0	106	1	0	0	0	Lobular carcinoma
	Mucinous carcinoma	0	3	0	2	154	0	0	0	Mucinous carcinoma
	Papillary carcinoma	0	2	0	0	1	106	0	2	Papillary carcinoma
	Phyllodes tumor	0	2	2	0	0	0	86	1	Phyllodes tumor
	Tubular adenoma	0	0	0	0	1	0	0	113	Tubular adenoma
		Adenosis	Ductal carcinoma	Fibroadenoma	Lobular carcinoma	Mucinous carcinoma	Papillary carcinoma	Phyllodes tumor	Tubular adenoma	

Figure 3.10: Confusion matrix in the test partition of the BreakHis dataset of the resulting model using a single run of Gen-CNN.

datasets, and the complex non-linear interactions between all their hyperparameters, the dataset, and architecture define the model’s final performance [24]. This situation raises the challenge of finding the best-performing architecture and hy-

Table 3.6: Comparison of the test classification performance metrics on the KVASIR-v2 dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.

Reference (Year)	Approach	Model GFLOPs	CAT- ACC	F1	MCC
Escobar et al. [146] (2021)	Transfer learning using VGG-16 and fine-tuning at different depths	≈ 15	0.9300	0.9308	0.9201
Mohapatra et al. [147] (2021)	Image-preprocessing with a 2D discrete wavelet transform and a custom CNN-based two-level classification	No data	0.9398	0.9391	0.9310
Gunasekaran et al. [148] (2023)	GIT-Net: Weighted average ensemble of DenseNet201 + InceptionV3 + ResNet50	≈ 12.63	0.9400	0.9409	0.9315
Obayya et al. [149] (2023)	Image-preprocessing with MF, CapsNet + Class-Attention Layer + Deep Belief Network Extreme Learning Machine + HPO with Modified SSA	No data	0.9300	0.9308	0.9201
Proposed approach (2024)	Gen-CNN Resulting model: EfficientNet-V2-Small	≈ 8.8	0.9413	0.9417	0.9330

2D: Two-Dimensional, MF: Median-Filtering, HPO: Hyperparameter Optimization, Salp Swarm Algorithm.

perparameters to tackle a specific task. Additionally, other characteristics of models, besides the classification performance, are prominent for creating practical systems, such as the computational complexity [132]. Frequently, the computational resources available and latency requirements are constraints in real-world scenarios [44, 132].

Robust and efficient optimization techniques can offer a solution for creating DL systems for real-world applications since they can attend to two of the current necessities of the field, which are automating the creation of ML models [112] and developing high-performance models with a computational complexity that satisfies the computational capabilities of smart-electronics and mobile devices [44].

Table 3.7: Comparison of the test classification performance metrics on the ISIC-2019 dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.

Reference (Year)	Approach	Model GFLOPs	CAT- ACC	F1	MCC
Kassem et al. [150] (2020)	GoogLeNet + multi-class SVM	≈ 2	0.7681	0.7409	0.6997
Pacheco et al. [151] (2020)	PNASNet + SENet + VGG-19 + Average outputs	≈ 25.5	0.8962	0.8989	0.8481
Iqbal et al. [152] (2021)	CSLNet: Custom CNN	No data	0.8926	0.8749	0.8545
Olayah et al. [153] (2023)	AlexNet + GoogLeNet + VGG16 + ANN	≈ 17.72	0.9606	0.8885	0.9427
Proposed approach (2024)	Gen-CNN Resulting model: ConvNext-Tiny	≈ 4.5	0.9015	0.8484	0.8542

ANN: Artificial Neural Network, SVM: Support Vector Machine.

This chapter covers the development and validation of Gen-CNN, a new framework that uses a to find and generate optimal and low-complexity CNN models. The results presented in Section 3.4.3 prove that the models generated using the novel Gen-CNN framework surpass, in terms of classification performance, models generated by using other heuristic HPO algorithms. Distinctively, the results displayed in Tables 3.6, 3.7, and 3.8 show that Gen-CNN automatically generates CNN models that are competitive with the published state-of-the-art hand-design custom models for real-world application datasets (specifically for medical image classification). Furthermore, and more importantly, the models generated with Gen-CNN demand significantly fewer computational resources than the compared hand-design custom models since, during the carried experiments, the search space of the developed algorithm considers only CNN architectures with a low FLOPs complexity (see Table 3.1). Another advantage of using GAs is that their operation is intuitive for any user, which can help to build trust in the

Table 3.8: Comparison of the test classification performance metrics on the BreakHis dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.

Reference (Year)	Approach	Model GFLOPs	CAT- ACC	F1	MCC
Umer et al. [154] (2022)	6B-Net: Custom CNN + ResNet50 + feature selection with PSO and ACS + ensemble subspace KNN classifier	No data	0.9016	0.8889	0.8696
Tummala et al. [155] (2022)	Swin-Transformer Tiny + small + Base + Large + Average output	≈ 28.6	0.9353	0.9284	0.9151
Clement et al. [156] (2023)	ResNet50 + ResNet18 + DenseNet201 + EfficientNetb0 + One-versus-one SVM	≈ 10.39	0.9777	0.9795	0.9709
Proposed approach (2024)	Gen-CNN Resulting model: ConvNext-Tiny	≈ 4.5	0.9583	0.9557	0.9452

PSO: Particle Swarm Optimizer, ACS: Ant Colony System, KNN: K-Nearest Neighbors, SVM: Support Vector Machine.

system.

Tables 3.6, 3.7, and 3.8 show the models with the highest reported classification metrics that use the exact same data sets as the presented experiments. Thus, these tables do not include studies that used subsets of the KVASIR, ISIC-2019, and BreakHis datasets or metadata to aid in the classification task. Additionally, all the displayed metrics were calculated as described in Section 2.5. So, the studies were considered for comparison only if the authors provided the information to calculate these metrics. Chiefly, the confusion matrix gives enough information to estimate any relevant classification performance metric [97].

For the KVASIR dataset, the model obtained with Gen-CNN achieves the highest classification metrics amongst the papers evaluated in Table 3.6 whilst using the lowest computational load in terms of FLOPs. As for the ISICS-2019 and BreakHis datasets, the generated models with Gen-CNN are the second highest in classification accuracy compared to the other recent research papers featured in Tables 3.7 and 3.8 and use less than half of FLOPs than the models with the high-

est classification metrics. Thus, Gen-CNN generates models with a classification performance comparable to the state-of-the-art with a much lower computational complexity, which is a prominent feature for many AI applications.

All the models displayed in Tables 3.6, 3.7, and 3.8, besides the ones generated using Gen-CNN, were custom-made to solve the dataset in question, required expert knowledge and time to be generated, and are the result of trial-and-error design processes. Additionally, the majority uses ensemble models, which combine different neural networks' architectures and other ML models. Using ensemble models augments the computational complexity of the resulting system. Also, many authors design their own CNN to tackle the classification of a given dataset. This method can be effective but time-consuming. In contrast, the models generated with Gen-CNN were automatically created using a single run of the proposed framework, proving that Gen-CNN enables the application of efficient and effective CNN models to non-ML-experts and hastens the development of such models.

Locating studies to compare experimental results was more problematic than anticipated. The application of DL to pretty much any task is a current hot research topic. The literature on DL models for the classification of images has proliferated in the past few years. Nonetheless, several studies suffer from relevant flaws in their results delivery. For instance, many studies use only one classification metric to evaluate the performance of their models, namely accuracy. However, after exhaustively reviewing the results, many of these studies present the micro-accuracy in multi-classification tasks. As discussed before, accuracy is not a wise option for imbalanced datasets. Furthermore, micro-accuracy is mischieving in the multi-class classification task since even if a system misclassifies a sample, micro-accuracy considers one false negative, one false positive, and $K - 2$ true negatives, a situation that evidently makes the micro-accuracy grow, which can lead to misinterpretations of the model's performance. Providing other more robust classification metrics, such as MCC and F1, along with the confusion matrix is fundamental to exchanging information to assess models' performance.

A primary constraint when using HPO is meta-overfitting (also referred to as overtuning, oversearching, or simply overfitting) [108, 128]. That is when, after several evaluations of different hyperparameter configurations, the selection of

the best configurations can be biased by the stochasticity of the many inner processes of the ML models or by overfitting to a dataset partition [108]. However, this effect is still to be further studied and lacks countermeasures in the literature (like regularization techniques for training ML models) [108]. The experimental results of this chapter suggest that the use of the validation partition MCC as the optimization target in the HPO helps mitigate this effect. The training optimization (that uses the training partition, a gradient-based optimizer, and its goal is to minimize the loss function) and the HPO (that uses the validation partition, the GA, and its goal is to maximize the MCC) affect each other’s results. Nevertheless, they do not have any control nor communication with one another, so they operate in a push-and-pull fashion. The experimental results presented in this chapter support the hypothesis that this method promotes the overall system generalization.

As described in Section 3.3.1, the selection of the hyperparameters to optimize is crucial for the performance of the overall HPO algorithm and resulting models. Some interesting hyperparameters to keep under consideration for future work are the augmentation techniques and layers to fine-tune. Previous work, like Autoaugment [157], has demonstrated that optimizing the data augmentation techniques per dataset and model can significantly improve the final model’s performance. As for the fine-tuning layers, several studies have addressed the question of whether it is better to fine-tune the whole network or just a part of it when using transfer learning, and the evidence seems to indicate that it depends on the source and target datasets and that layer-wise fine-tuning delivers better results, depending on the architecture [158–161].

To the best of the author’s knowledge, Gen-CNN is the only framework that considers the architecture and loss functions as hyperparameters during the HPO procedure. As the experimental results show, assigning the selection of the architecture to the GA automizes a procedure that is hard even for ML-expert users. In the presented experiments, the Gen-CNN search space was constrained to CNN architectures with relatively low FLOP complexity to show that an optimal hyperparameter configuration allows even small (in terms of FLOPs) off-the-shelf CNNs to be competitive against state-of-the-art, custom hand-made, and ensemble models with much higher computational demand. This is meaningful for the

rapidly growing community of AI users that require AI solutions capable of performing in real-world scenarios.

Whether a CNN model can perform in real time depends on the embedded system characteristics. However, models with up to 600 MFLOPs are said to be able to perform on mobile devices [16], which is a good indicator of the models' efficiency. The experimental results demonstrated that Gen-CNN can generate CNN models within that FLOP regime and extraordinary classification performance, which is a step towards the automatic generation of models for medical image classification for real-time inference. Gen-CNN users can modify the hyperparameters search space following their needs and preferences. For instance, when the computational load is not a limitation during model deployment, Gen-CNN search space can include architectures with a higher level of complexity, or if the final model's application is on a mobile device, then the search space can be defined only by architectures capable of running on mobile devices.

Considering that the evaluation of the candidate solutions is the most expensive phase of population-based algorithms, such as GAs, Gen-CNN uses transfer learning and a low-compute regime to accelerate its operation. This technique creates low-fidelity prototypes of the models to assess their suitability for the task and has been effective in the related field of NAS. The convergence of the developed framework was analyzed in the recommended regime for heuristic optimization techniques, achieving higher mean categorical accuracy than other HPO algorithms reported in the current literature. The optimization target of Gen-CNN is the maximization of the MCC on the validation data since the MCC is a robust and interpretable metric that reflects the performance of classification algorithms on the whole confusion matrix. Gen-CNN achieved better classification performance than other HPO algorithms that directly maximize the accuracy, which suggests that the maximization of MCC is effective in improving the overall classification performance of models. However, Gen-CNN could have other optimization targets, following the needs and preferences of its users. In addition, multi-objective optimization has potential for future research. Another interesting future research would be introducing model fusion techniques into the AutoML algorithm.

Gen-CNN is a free library available at <https://github.com/RogeGar/Gen-CNN>.

Chapter 4

Artificial astrocytes to complement artificial neural networks

Abstract

Artificial neural networks find their inspiration in the operation of the human brain. Deep learning, a machine learning paradigm that uses several layers of arranged artificial neural networks, has prospered thanks to modern technological advances in hardware and the increasing availability of data. However, standard deep learning systems are still based on connectionist models of neurons from several decades ago. Neuroscience has a different perspective and a deeper understanding of the human brain's information processing compared to when these connectionist neuron models originated. Thus, research that attempts to bring closer the current deep learning systems and biological neural networks is gaining momentum as a means to improve this technology.

Glial cells occupy more than half of the volume of the human brain, especially astrocytes, which are crucial for different adaptive processes related to neuroplasticity, key in learning and memory formation. Consequently, artificial astrocytes are an emerging research topic. Nevertheless, research attempting to introduce astrocyte-like behavior into current convolutional neural networks is quasi-inexistent. Consequently, this chapter covers the research and development of a new artificial astrocyte unit created to address this gap in the current liter-

ature. The unit is inspired by previous research on neuron-glia networks and acts as a learning mechanism that imitates the multi-time-scale neuroplasticity of the brain. The proposed artificial astrocyte is compatible with the existing convolutional neural network architectures, introduces no additional parameters into the models, and is only active during training. Experimental results show that, depending on the hyperparameter configuration, astrocytes can improve the models’ classification performance. Additionally, there are cases where astrocytes set the parameters of some convolutional filters to zero without affecting the classification performance and producing similar representations to the same architecture trained under the same conditions without astrocytes, suggesting that artificial astrocytes could work as a pruning method, trimming the unnecessary information.

4.1 Introduction

Computer science has historically drawn inspiration from biology [1, 9, 24, 73, 162]. Specifically, neuroscience and neuro-inspired computing have co-evolved over time [162]. The last couple decades was fructiferous for both disciplines. The field of neuro-inspired computing progressed rapidly with the consolidation of DL [1, 7]. Meanwhile, neuroscience broke old paradigms and discovered new mechanisms involved in learning and the brain’s information processing [163–165]. The present circumstances of both disciplines offer a perfect stage for collaboration to achieve significant advancements at an unprecedented scale [162].

Despite the success of current DL models, their operation foundation is quite simple and differs qualitatively from the brain’s core information-processing mechanisms [162]. Introducing these types of mechanisms to the existing DL paradigm represents, at the very least, an intriguing research opportunity. In this sense, astrocytes have been noteworthy recently since they play a vital role in brain function [53, 165], especially in the experience-dependent structural synaptic adaptations that rule learning and memory [52, 64, 164, 165]. Several studies have pointed out the potential of complementing the existing ANNs paradigm with astrocyte-like mechanisms [51–53, 166, 167]. Thus, research on artificial astro-

cytes is a relatively new and promising field in DL.

The current DL paradigm uses ANNs that belong to the connectionist systems archetype. Despite the existence of numerous biophysical astrocyte models, the modeling of these cells from a connectionist perspective is at the emergence phase [64,166]. The term neuron-glia network (NGN) was coined in 2011 by Porto-Pazos et al. [60] to refer to connectionist systems using ANNs and artificial astrocytes. Since then, different approaches for implementing astrocyte-like behavior into ANNs have emerged with distinct goals and success grades. There is evidence showing that astrocytes can improve the classification efficacy of MLP [60, 61, 168–170].

Most of the research done in the field of artificial astrocytes on connectionist ANNs utilizes MLPs with a few layers. Therefore, the potential of granting CNNs with astrocyte-like mechanisms remains extensively unexplored since just a few studies have attended to the application of artificial astrocytes into CNNs [171–173]. The scarce current studies on this topic take astrocytes as inspiration to develop agents that provide the CNNs with a specific attribute, such as a short-term memory unit [172], a neurons’ connections optimizer for structure learning [173], or an emulator of the self-attention transformer’s mechanisms [53,174].

This chapter tackles the gap in the current literature on the artificial astrocyte properties in standard CNNs to improve their performance by covering the development of an artificial astrocyte based on previous work on NGNs [60] that resembles its biological counterpart’s properties and interactions with neurons. The developed artificial astrocyte modulates the convolutional filters’ parameters by their activation rates, modifying the parameters at a different rate than the mainstream training procedure (gradient descent optimization), resembling the multi-time-scale dynamical process of the brain’s neuroplasticity.

4.2 Artificial Neuron-Glia Networks

The work by Ikuta et.al. [63] is the first antecedent of introducing glial-cell properties into ANNs. They developed a network of inter-communicated artificial glial units that communicated with the hidden layers of an MLP, delivering chaotic

oscillations into the integration of the neuron inputs, improving the learning efficiency and classification performance [63].

Ikuta et.al. continued their research. In another study, the impulses generated by the glial network depended on the neurons' outputs [175]. If a neuron reaches a given excitation level, the associated glial unit responds with an impulse that propagates through the glial network [175]. This approach has better results than their prior work. They further investigated this impulse glia network and found the correlation of some hyperparameters of this network with the learning performance of the MLP [168]. Additionally, they proved that the impulse glia network conducts the MLP to a better generalization capability [168].

The same research team continued to improve their model. In further research, they connected the hidden layer neurons of an MLP one-to-one to the glial units of the glial network [169, 170]. In this work, the glial units' output takes positive or negative values depending on the associated neuron's output [169, 170]. The denominated MLP with positive and negative pulse glial chain had better classification and learning performance than the previously discussed approaches [169, 170]. Also, this team proposed introducing a signal that controls the learning of different groups of neurons during training [176], achieving a slight enhancement in the learning efficiency.

The next step for Ikuta et al. was introducing the glial network into a four-layer MLP [177]. In this case, the glial units connected neurons of different layers [177]. The goal was to reproduce the ability of glial cells to communicate spacially with neurons in distant regions of the brain [177]. Experiments using this approach demonstrated that this type of communication could enhance the network's learning capability and classification performance [177]. Nevertheless, the network performance depends on non-learnable parameters of the glial network [177].

Following the trend of using algorithms inspired by the role of astrocytes in the learning process, a different research team proposed a network called Self Organizing Neuro-Glial Network, SONG-NET [178, 179]. This network uses an MLP and self-organizing Kohonen maps to replicate the effect of astrocytes in the modulation of synaptic efficacy during learning [178, 179]. Experiments proved that SONG-NET is up to 12 times faster at learning than an MLP using the back-

propagation algorithm [178, 179].

On a different but related research direction, Porto-Pazos et al. [60] recognized the opportunity of introducing units that resemble the role of glial cells in ANN and proposed a new paradigm: artificial neuron-glia networks: NGN. They developed a computational unit inspired by the astrocytes' function and paired each artificial neuron with an artificial astrocyte, which regulates the synaptic weights of the associated neuron according to the neural activity. The synaptic changes induced by the astrocyte last for an arbitrary number of iterations and do not consider the spatial spread of the astrocyte signal to other neurons or communication between astrocytes. This team tested their proposed NGNs on classification problems against ANN. They used MLPs with a different number of layers and neurons for the ANN, and for the NGNs, they used the same architectures with the addition of the artificial astrocytes. For the training of the networks, they used a GA that changed the synaptic weights according to the mean square error (MSE) of the network using the training set, and for the NGNs, they manually tuned the artificial astrocytes' hyperparameters.

Experimentally, Porto-Pazos et.al. [60] concluded that the NGNs' classification performance was better than that of the ANNs and that the improvement was due to artificial astrocytes. Also, they noticed that the network performance improvement by artificial astrocytes increases as the network complexity increases and that the relative performance improvement depends on the tested problem.

In consecutively research studies, Alverellos et.al. [61] used the artificial astrocyte units presented in [60] and studied the classification performance of NGNs with different neuron-glia algorithms. They tried six different neuron-glia algorithms that attempted to imitate the observed behavior of astrocytes in the nervous system of living organisms. In this work, the authors also trained their networks using a GA. They found that the NGNs have a better classification performance than the ANN regardless of the employed neuron-glia algorithm. However, they concluded that the neuron-glia algorithm that evokes the best classification performance depends on the classification problem. In subsequent work, Pastur-Romay et.al. [62] changed the artificial astrocytes' ability in this kind of network from modulating the synaptic weights to directly modulating the neuronal outputs. With this approach, Pastur-Romay et.al. [62] achieved significantly

better classification performance than an MLP without artificial astrocytes.

Gergel and Farkaš [166] inspired by the previous works to develop an NGN based on an MLP with astrocytes in the hidden layers like the ones proposed by Ikuta et.al. [175] but to modulate the synaptic efficacy like Porto-Pazos et.al. [60]. They introduced different astrocytic mechanisms with learning parameters compatible with the backpropagation algorithm and a few hyperparameters [166]. They concluded that the efficacy of the NGN depended on the classification problem and that the correct tuning of the astrocytes hyperparameters should be further investigated [166].

Gergel and Farkaš continued their research [64] by implementing an Echo State Network with artificial astrocytes for classification tasks [64]. They found out that this type of network with fixed weights for astrocytes presented little to nothing enhancement compared to the same network without astrocytes [64]. However, by introducing Hebbian learning for the astrocytes' weights, they achieved a significantly better classification performance [64].

4.3 Methodology

4.3.1 Artificial astrocyte

The development of an artificial astrocyte compatible with the current CNNs architectures and learning algorithms is the first step to assessing the effects of providing CNNs with mechanisms similar to those of the biological astrocytes. The antecedents of NGNs [60–62] that use MLPs are the foundation to do so since they already devised connectionist models of these cells that are effective in improving the networks learning efficiency and classification performance.

Extending previous work on NGN, an artificial astrocyte compatible is developed and implemented on off-the-shelf CNNs. The operation principle of the artificial astrocyte is the recording of the neurons' activations to make non-supervised adaptations to their parameters by their firing rates in a given period. Thus arises the necessity for a time metric during the training of the networks. Given that the backpropagation algorithm is the central element of the training procedure, the artificial astrocytes consider one iteration (defined time metric) as

one forward passing of a training batch.

To understand how the artificial astrocytes record and modulate the parameters inside of a CNN, it is helpful to think of a neuron as a set of weights (ignoring the bias term). Each neuron has i inputs (hence i synaptic weights) and a single output. The astrocyte modulates all the synaptic weights of its associated neurons by their outputs during the defined iteration number. In ANNs, all the neurons in a layer have the same number of inputs i , and each neuron produces a single output. Consequently, a layer with an arbitrary number z of neurons has a set of $i \times z$ synaptic weights. An artificial astrocyte covers all the neurons in a given layer. Then, the astrocyte unit has z counters, one for each neuron, and modulates uniformly all the i synaptic weights of a neuron. For convolutional layers in CNNs, the astrocytes operate identically; But, with a set of convolutional weights (the kernel parameters) per each input channel, modulating all the parameters in the kernel. The developed artificial astrocyte works for linear and convolutional layers. However, it is implemented only in the convolutional layers in this thesis.

Algorithm 3 shows the forward function of a convolutional layer with the implementation of the developed artificial astrocyte. Note that the astrocytes disable gradient computation before making any changes to the layer parameters. Thus, the astrocytes are compatible with gradient-descent training algorithms and work in parallel with them, making changes in the parameters between optimization steps.

The implemented artificial astrocyte has four key hyperparameters: (1) the neuron-glia power connection (**NG**), which defines the number of iterations that the astrocytes keep a record of the neuron activations before they modulate the synaptic weights and reset the record. (2) The astrocytic sensitivity (**AS**), which sets the boundaries (positive and negative) that will trigger a synaptic modulation from the astrocytes. (3 and 4) The increase and decrease factors for astrocytic modulation of synaptic weights (**Str** for highly-active neurons and **Weak** for scarcely-active neurons). Note that we use **Str** to identify the neural connection as ‘strong’ since it reaches the defined astrocytic sensitivity upper bound, and **Weak** to identify the neural connection as ‘weak’ since it reaches the defined astrocytic sensitivity lower bound, not to symbolize that the astrocytes strengthen or weaken the synaptic weight. Whether **Str** and **Weak** strengthen or weaken

Algorithm 3 Forward function of a convolutional layer l with the implemented artificial astrocyte. $l(\circ)$ indicates that the argument \circ passes through the convolutional filters of layer l . *Training* is a bool variable that indicates whether the network is in training mode or not, and *no_grad* indicates disabling gradient computation. *Iteration* counts the forwarding of batches and is initialized in 0. $\mathcal{A}_l \in \mathbb{R}^z$ is the astrocyte unit of the layer l , where z is the number of out channels of layer l , and all its values are initialized in 0. $\text{TanH}(\circ)$ is the hyperbolic tangent activation function, $\text{MaxPool2d}(\circ)$ is the 2D max pooling function, $\Sigma(\circ)$ is the summation operator. \mathcal{W}^l is the set of all trainable parameters of l such that $\mathcal{W}^l = \{W_j^l \mid \forall j \in \{1, 2, \dots, z\}\}$, and W^j is the set of trainable parameters of the j -th convolutional filter. Finally, **NG**, **AS**, **Str**, and **Weak**, are hyperparameter of the artificial astrocyte.

```

INPUT Data:  $\mathcal{D}$ 
OUTPUT Layer output:  $x$ 
 $x \leftarrow l(\mathcal{D})$ 
IF Training
  WITH no_grad()
     $\text{Iteration} \leftarrow \text{Iteration} + 1$ 
     $\mathcal{A}_l \leftarrow \mathcal{A}_l + \text{MaxPool2d}(\text{TanH}(\Sigma(x)))$ 
    IF  $K \geq \mathbf{NG}$ 
       $a \leftarrow 0$ 
      FOR ALL  $W^l \in \mathcal{W}^l$ 
        IF  $\mathcal{A}_l[a] > \mathbf{AS}$ 
           $W^l \leftarrow W^l \times \mathbf{Str}$ 
        END IF
        IF  $\mathcal{A}_l[a] < -\mathbf{AS}$ 
           $W^l \leftarrow W^l \times \mathbf{Weak}$ 
        END IF
       $a \leftarrow a + 1$ 
    END FOR
     $\text{Iteration} \leftarrow 0$ 
     $\mathcal{A}_l \leftarrow \mathcal{A}_l \times 0$ 
  END IF

```

the associated synaptic weight is defined by their set values. Also, the astrocytic sensitivity is symmetric.

4.3.2 Analysis of the CNNs

CAT-ACC, F1, and MCC are the employed metrics for the evaluation of the classification performance of the models. Additionally, projection weighted canonical correlation analysis (PWCCA) [180] is used to examine how the networks produce different representations (activations) due to the influence of the artificial astrocytes, using different configurations for the four principal hyperparameters of the astrocytes: **NG**, **AS**, **Str**, and **Weak**. PWCCA is a technique created to determine the similarity between sets of neuron activations. PWCCA is invariant to affine transformations since it is a canonical correlation analysis (CCA)-based method. Moreover, it assigns importance weights to the vectors of the CCA to contrast between components with relevant information and the noise associated with deep neural networks. The weighted average of the individual similarity scores measures more precisely the similarity between sets of neuron activations. PWCCA delivers similarity scores in the interval $[0, 1]$, where identical sets of neurons' activations evoke a score of 1, and the lesser the score, the larger the differences between the sets.

This research aims to define the effects of introducing astrocyte-like mechanisms into the current CNN paradigm. Thus, the developed artificial astrocyte is implemented in three off-the-shelf CNN architectures: RegNet-Y-400MF [16], MobileNet-V3-Large [86], and Shufflenet-v2-x1.5 [132]. These architectures are selected because of their training speed and fewer parameters when compared to other architectures. Due to the massive amount of experiments to characterize the behavior of the CNNs under different hyperparameter configurations of the astrocytes, the experiments are limited to these three architectures by the computational resource constraints. Also, all the experiments are carried out using transfer learning on pretrained versions of the architectures.

From this point, CNN models with artificial astrocytes are referred to as CNGNs since the implemented artificial astrocytes are based on NGNs.

4.4 Results

4.4.1 Datasets

The experiments employ the benchmarking datasets CIFAR-10 and CIFAR-100 [181], which contain small images (32×32 pixels) of common objects such as transportation means and animals. The train data of both datasets are split into train and validation partitions in a proportion of 90% and 10%, respectively. The test data is used only for testing. Additionally, two medical image datasets are used to assess the performance of the CNGNs in the medical image classification task: (1) the KAVSIR-v2 [141] and (2) the BreKHis-v1 [145] datasets. Both datasets are split into 60%, 20%, and 20% for training, validation, and testing, respectively. For the BreKHis-v1 dataset, the models are trained to classify its images into its eight different subclasses (adenosis, fibroadenoma, phyllodes tumor, tubular adenoma, carcinoma, lobular carcinoma, mucinous carcinoma, and papillary carcinoma), regardless of the images’ magnifying factor.

For all the experiments, every training batch goes through a transformation phase where the techniques of random crop, random rotation (of up to 90°), random vertical flip (50% probability), random horizontal flip (50% probability), color jitter transformation, and Cutmix or Mixup serve as augmentation techniques.

4.4.2 Experiments

The implemented artificial astrocyte (Algorithm 3) has four hyperparameters: (1) the neuron-glia power connection (**NG**) represents the number of iterations that the astrocytes record the activation frequency of the associated neurons, that is how many times a neuron has an output different from zero over an arbitrary number of data batches. (2) The astrocytic sensitivity (**AS**) delimits the boundaries of activation frequencies at which the synaptic modulation occurs. (3 and 4) The modulation factors (**Str** and **Weak**) are the hyperparameters that define whether the astrocytes increase or decrease (and to what extent) the synaptic weights per their firing frequency. Thus, they characterize the effect that the activation frequency has on the learning process, whereas the astrocytic sensitivity

and the neuron-glia power connection control the rate of change in the parameters' weights. Therefore, both of the modulation factors have to be analyzed first to dissect the effect that the artificial astrocyte can have on the learning process.

Accordingly, the hyperparameters values for **Str** and **Weak** are tested in a grid-like fashion, using the CIFAR-10 and CIFAR-100 datasets in the interval $[0.5, 1.5]$ with steps of 0.25. All the other hyperparameters have fixed values, as Table 4.1 shows, including the neuron-glia power connection (**NG**) and the astrocytic sensitivity (**AS**). This experiment is referred to as Exp 1.

Table 4.1: Hyperparameters for the experiments of artificial astrocytes.

Hyperparameter	Value
Epochs	5 in Exp 1, 10 in Exp 2 & 3
Batch size	256
Images size	224×224
Optimizer	Adam
Learning rate	0.0001
Learning rate scheduler	None in Exp1, cosine annealing in Exp 2 & 3
Loss function	Label smoothing
Weight decay	0.01
Neuron-glia power connection	4
Astrocytic sensitivity	± 2

The above-described procedure is performed three times under the same conditions. Figures 4.1 and 4.2 show the mean validation CAT-ACC of the three tested architectures under the different hyperparameter configurations for the CIFAR-10 and CIFAR-100 datasets, respectively. In the grid, the configuration **Str**=1, **Weak**=1 serves as a reference of the astrocyte's effect since the astrocytes do not modulate the synaptic weights (both modulation factors are one), so the CNGNs behave like regular CNNs without astrocytes. This configuration is referred to as the control case. Also, during this chapter, the term accuracy refers specifically to the CAT-ACC.

For easy assessment of the experiment results, each configuration mean accuracy is presented by architecture and training epoch in heat maps in Figures 4.1 and 4.2, where the values with higher accuracy (than the average per heat map)

appear in red and the configurations with lower accuracy in blue.

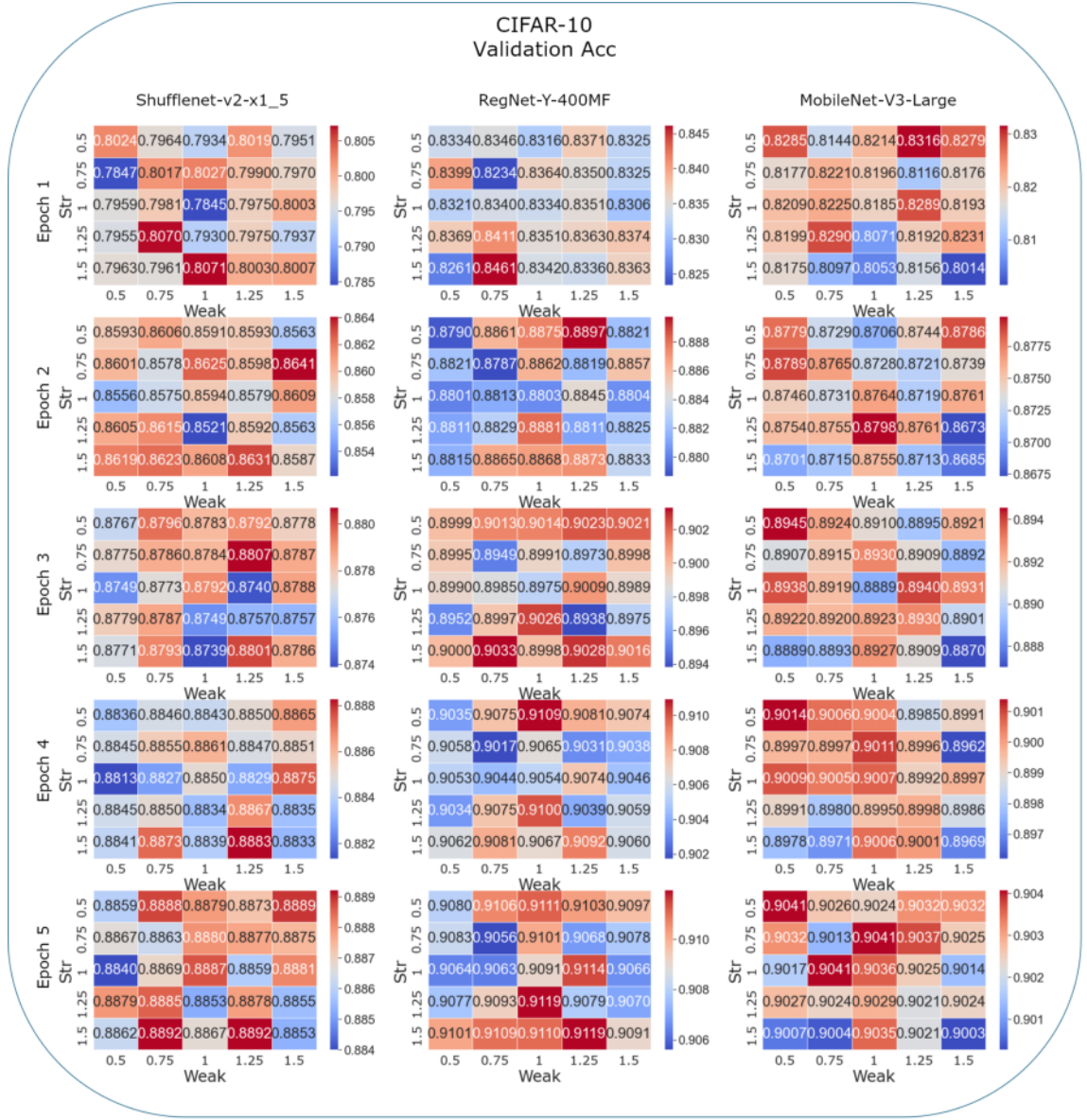


Figure 4.1: Classification performance of the three tested architectures after training under the conditions described in Table 4.1 (Exp 1) using CIFAR-10. Mean validation CAT-ACC for three repetitions per hyperparameter configuration in the grid. [Best viewed in color]

The control case does not have the highest mean validation accuracy for any tested architecture in any training epoch, as Figures 4.1 and 4.2 show. Hence,

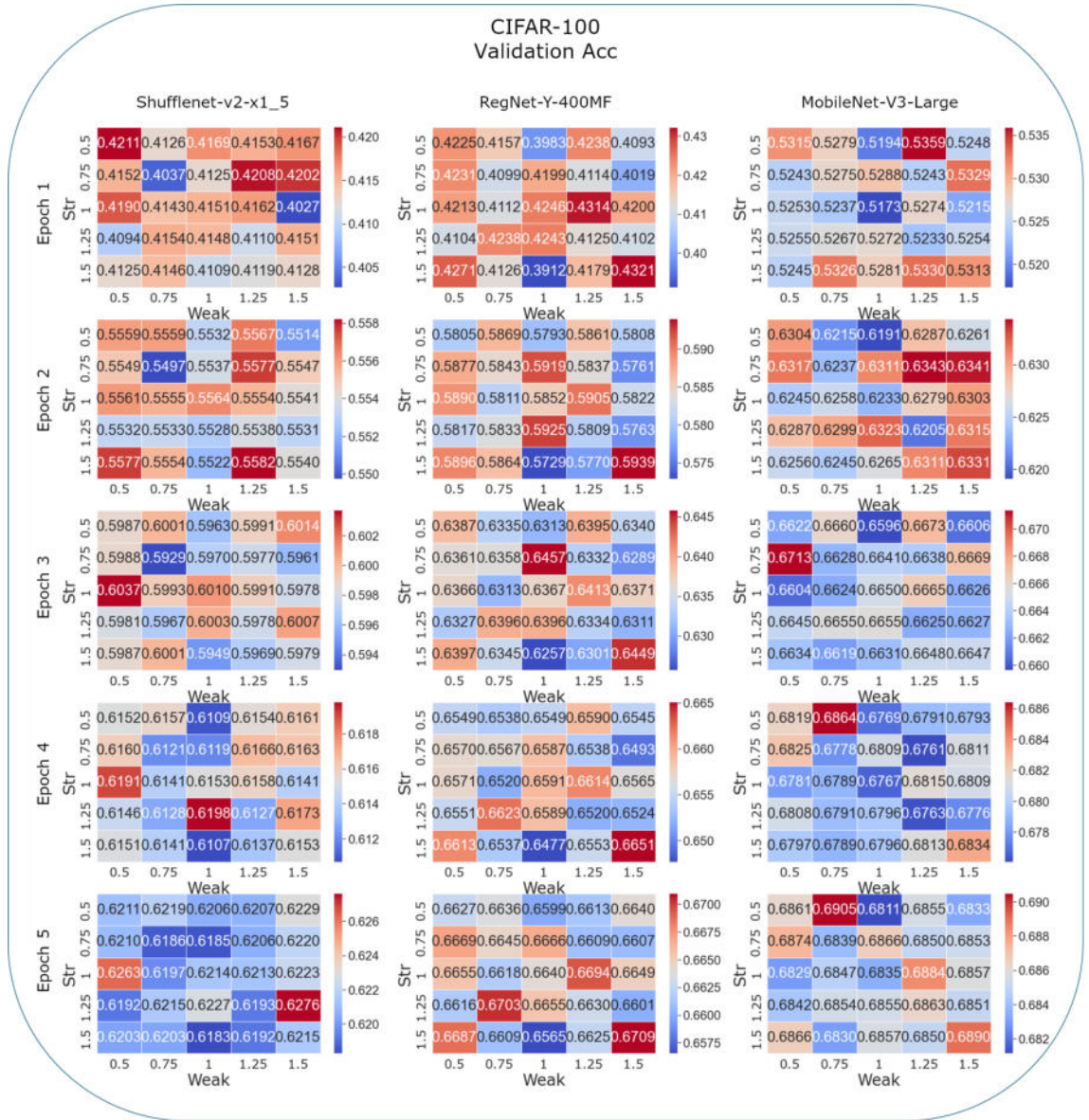


Figure 4.2: Classification performance of the three tested architectures after training under the conditions described in Table 4.1 (Exp 1) using CIFAR-100. Mean validation CAT-ACC for three repetitions per hyperparameter configuration in the grid. [Best viewed in color]

considering a regular CNN with no astrocytic modulation, there are hyperparameter configurations of the astrocytes that lead to better validation accuracy for the three tested architectures along the five training epochs in both benchmarking

datasets. However, the gain in accuracy is marginal by the fifth epoch, regardless of the hyperparameter configuration. These results suggest that (1) the models converge to similar parameters even with the non-supervised aggressive modulation of the astrocytes using their neuron activations firing rates. Additionally, (2) the optimal hyperparameter configuration seems to depend on the architecture, dataset, and training epoch since the data exhibited in Figures 4.1 and 4.2 reveals no apparent patterns in the classification accuracy regarding the astrocytes' hyperparameter configuration. These two observations are further explored experimentally (Exp 2 and Exp 3).

A new experiment (Exp 2) is designed in light of the previous results. This experiment consists of optimizing the astrocyte modulation factors along ten training epochs using grid-search in the following mode: At the start, the control coordinate is the center of the grid (1,1). From there, the algorithm trains the architectures using nine different configurations, varying the two modulation factors by ± 0.25 one at a time. Then, the set of learnable parameters of the model with the highest validation accuracy becomes the starting point for the next training epoch, and its modulation factors are the new center of the grid for the next training epoch. This experiment is carried out with the CIFAR-10 and CIFAR-100 datasets and the two medical image datasets, KVASIR-v2 and BreakHis-v1. These last two are included to assess the astrocytes' potential to boost the models' performance in the medical image classification task.

Training a given model under the same conditions can lead to small variations in classification performance due to the random processes involved in training (such as dropout and augmentation techniques). To discard any improvement in the models' performance by this effect when assessing the astrocytes' contribution to training, a control model (with no astrocytes) is trained using the same number of iterations. Hence, the same architectures (with no astrocytes) are trained nine times per training epoch, and the set of learnable parameters with the highest validation accuracy passes for the next epoch.

Table 4.2 shows the classification metrics of the CNGNs and control CNNs during the grid-search optimization experiment. The Δ column shows the percentage change in the test accuracy of the CNGNs with respect to the same architectures without astrocytes, using the optimal astrocytes' hyperparameter config-

urations found using the described grid-search. For the CIFAR-10 and CIFAR-100 datasets, the change in accuracy is trivial (less than 0.5% in most cases). However, in both medical image datasets, the astrocytes had a greater effect on the models' accuracy. The case of the MobileNet-V3-Large architecture is particularly noteworthy since it is the one whose accuracy changed the most among the experiments, with a significant improvement due to the astrocytes. Tables 4.3-4.6 shows the found optimal configurations of the astrocyte modulation factors for each dataset and architecture.

A third experiment (Exp 3) is carried out. It encompasses training the CNGNs using the optimal astrocyte's hyperparameters for three repetitions (configurations shown in Tables 4.3-4.6) and analyzing the mean test classification performance and the activations evoked in the last convolutional layer of the tested architectures. Once again, an analogous CNN model for each CNGN (same architecture and training conditions) is trained to serve as a reference. Table 4.7 shows the mean test accuracy of the CNNs and CNGNs. Anew, the change in test accuracy on the CIFAR-10 and CIFAR-100 is minimal, and the larger changes in test accuracy occur on the medical image datasets. By analyzing the results per architecture, it can be inferred that the astrocytes influence the MobileNet-V3-Large architecture the most. Yet, in the particular case of RegNet-Y-400MF in the BreaKHis-v1 dataset, the astrocytes significantly improved the architecture mean test accuracy (by 1.31%). Figure 4.3 shows the evolution during training with the optimal modulation factors of the mean test accuracy of the CNGNs in the four tested datasets.

Besides the CAT-ACC, the F1 and MCC of the models are analyzed. These two metrics are better than the accuracy for detecting unbalanced behavior in the models' performance (if the models are better at classifying some classes than others). This is not a usual issue in balanced benchmarking datasets such as CIFAR-10 and CIFAR-100 but is a common challenge in real-world applications datasets, which are intrinsically unbalanced, such as BreaKHis-v1. Moreover, these metrics serve to evaluate if the astrocytes promote learning features that are convenient for only specific classes since they perform non-supervised modulations in the models' parameters depending on their neurons' firing rates. Table 4.8 shows the mean F1 and MCC of the CNNs and CNGNs of Exp 3.

Table 4.2: Classification CAT-ACC of the CNNs and CNGNs with the highest validation CAT-ACC during the grid-search optimization. Table 4.1 (Exp 2) shows the training conditions. Δ is the percentage change in test accuracy of the CNGNs with respect to the CNNs in the same dataset and training conditions. The highest test accuracy between the CNN and CNGN is in bold case.

CIFAR-10					
	CNN		CNGN		Δ
Architecture	Valid	Test	Valid	Test	Test
Shufflenet-v2-x1_5	0.9186	0.9115	0.9184	0.9138	+0.25%
MobileNet-V3-Large	0.9226	0.9220	0.9198	0.9179	-0.44%
RegNet-Y-400MF	0.9310	0.9267	0.9302	0.9226	-0.44%
CIFAR-100					
	CNN		CNGN		Δ
Architecture	Valid	Test	Valid	Test	Test
Shufflenet-v2-x1_5	0.693	0.6867	0.6998	0.6883	+0.23%
MobileNet-V3-Large	0.7296	0.7312	0.7288	0.7317	+0.07%
RegNet-Y-400MF	0.741	0.7278	0.7288	0.7217	-0.84%
KVASIR-v2					
	CNN		CNGN		Δ
Architecture	Valid	Test	Valid	Test	Test
Shufflenet-v2-x1_5	0.7694	0.8913	0.7500	0.8963	+0.56%
MobileNet-V3-Large	0.7750	0.8844	0.7938	0.9119	+3.11%
RegNet-Y-400MF	0.7713	0.9038	0.7525	0.9069	+0.34%
BreakHis-v1					
	CNN		CNGN		Δ
Architecture	Valid	Test	Valid	Test	Test
Shufflenet-v2-x1_5	0.6195	0.6694	0.6188	0.6643	-0.75%
MobileNet-V3-Large	0.6340	0.7124	0.6340	0.7257	+1.86%
RegNet-Y-400MF	0.6106	0.7174	0.6201	0.7105	-0.96%

Analyzing the F1 and MCC displayed in Table 4.8, both the CNGN and CNN models, these two metrics do not present a significant change concerning the CAT-ACC in Table 4.7 for the balanced datasets CIFAR-10, CIFAR-100, and KVASIR-v2, but they do for the unbalanced dataset BreakHis-v1. However, since the CNNs and CNGNs models present similar F1 and MCC and similar differences regarding their respective CAT-ACCs, the unbalanced classification behavior is caused by the grid-search optimization and not the astrocytes effect. The optimization

Table 4.3: Optimal **Str** & **Weak** per epoch found using grid-search (Exp 2) on the CIFAR-10 dataset.

CIFAR-10						
Arquitect- ure	Shufflenet-v2-x1.5		MobileNetV3Large		RegNet-Y-400MF	
Epoch	Str	Weak	Str	Weak	Str	Weak
1	1	0.75	1.25	0.75	0.75	0.75
2	1.25	0.75	1	0.75	1	1
3	1.5	0.5	0.75	0.75	1	1
4	1.5	0.5	0.5	0.75	0.75	0.75
5	1.25	0.75	0.5	1	0.75	0.75
6	1.25	0.5	0.25	1	0.5	0.75
7	1.25	0.5	0	1	0.75	0.5
8	1	0.5	0	1	0.5	0.75
9	0.75	0.75	0	0.75	0.25	0.5
10	0.75	1	0.25	1	0.25	0.5

Table 4.4: Optimal **Str** & **Weak** per epoch found using grid-search (Exp 2) on the CIFAR-100 dataset.

CIFAR-100						
Arquitect- ure	Shufflenet-v2-x1.5		MobileNetV3Large		RegNet-Y-400MF	
Epoch	Str	Weak	Str	Weak	Str	Weak
1	1.25	1.25	1.25	1.25	0.75	0.75
2	1.25	1	1.25	1	0.75	0.75
3	1	1.25	1	1.25	1	0.75
4	1	1.5	0.75	1.25	0.75	1
5	1.25	1.25	0.75	1.5	0.5	1
6	1.5	1.25	0.5	1.75	0.25	0.75
7	1.5	1	0.75	1.75	0	1
8	1.75	0.75	0.5	1.75	0.25	1.25
9	1.5	0.5	0.75	1.75	0.5	1
10	1.5	0.5	0.5	1.5	0.25	1

procedure maximized the CAT-ACC. Given that the BreakHis-v1 dataset is highly unbalanced, the found optimal hyperparameter configuration surely promotes the correct classification of the dominant classes and neglects the classes with fewer samples. Furthermore, the F1 and MCC of the MobileNet-V3-Large archi-

Table 4.5: Optimal **Str** & **Weak** per epoch found using grid-search (Exp 2) on the KVASIR-v2 dataset.

Arquitect- ture	KVASIR-v2					
	Shufflenet-v2-x1.5		MobileNetV3Large		RegNet-Y-400MF	
Epoch	Str	Weak	Str	Weak	Str	Weak
1	1.25	1	1.25	1.25	0.75	0.75
2	1	1	1.5	1.5	0.5	1
3	0.75	1	1.25	1.25	0.75	1
4	0.5	1.25	1	1	0.5	1.25
5	0.75	1.5	1	1.25	0.5	1.25
6	0.75	1.25	0.75	1.5	0.5	1.25
7	1	1.25	1	1.25	0.25	1
8	0.75	1	0.75	1.5	0.5	1
9	0.75	0.75	1	1.75	0.75	1
10	1	0.5	1.25	1.5	0.5	1.25

Table 4.6: Optimal **Str** & **Weak** per epoch found using grid-search (Exp 2) on the BreakHis-v1 dataset.

Arquitect- ture	BreakHis-v1					
	Shufflenet-v2-x1.5		MobileNetV3Large		RegNet-Y-400MF	
Epoch	Str	Weak	Str	Weak	Str	Weak
1	0.75	1.25	0.75	0.75	0.75	1
2	0.5	1.25	1	1	1	1.25
3	0.5	1.5	1	1	1	1.5
4	0.75	1.75	0.75	0.75	0.75	1.75
5	0.75	1.75	0.75	0.75	1	1.75
6	0.5	2	0.5	0.75	1.25	1.5
7	0.25	2.25	0.25	0.5	1	1.25
8	0	2.25	0.5	0.25	1.25	1
9	-0.25	2.5	0.25	0.25	1	1.25
10	-0.25	2.75	0.5	0.25	1	1.5

architecture confirm that the astrocytes significantly improve the classification performance of this architecture on the KVASIR-v2 dataset.

Finally, PWCCA is employed to estimate the similarity score of the produced features of the corresponding CNGNs and CNNs, specifically at the last convo-

Table 4.7: Mean test classification CAT-ACC of three training repetitions using the optimal hyperparameter configurations found with the grid search. Table 4.1 (Exp 3) shows the training conditions. Δ is the mean percentage change in test accuracy of the CNGNs with respect to the CNNs in the same dataset and training conditions. The highest mean test accuracy between the CNN and CNGN is in bold case.

CIFAR-10								
Shufflenet-v2-x1.5			MobileNet-V3-Large			RegNet-Y-400MF		
CNN	CNGN	Δ	CNN	CNGN	Δ	CNN	CNGN	Δ
0.9103	0.9138	+0.38%	0.9202	0.9189	-0.14%	0.9247	0.9230	-0.18%
CIFAR-100								
Shufflenet-v2-x1.5			MobileNet-V3-Large			RegNet-Y-400MF		
CNN	CNGN	Δ	CNN	CNGN	Δ	CNN	CNGN	Δ
0.6869	0.6863	-0.09%	0.7255	0.7290	+0.48%	0.7255	0.7256	+0.02%
KVASIR-v2								
Shufflenet-v2-x1.5			MobileNet-V3-Large			RegNet-Y-400MF		
CNN	CNGN	Δ	CNN	CNGN	Δ	CNN	CNGN	Δ
0.8969	0.8921	-0.53%	0.8908	0.9119	+2.36%	0.9035	0.9000	-0.39%
BreakHis-v1								
Shufflenet-v2-x1.5			MobileNet-V3-Large			RegNet-Y-400MF		
CNN	CNGN	Δ	CNN	CNGN	Δ	CNN	CNGN	Δ
0.6721	0.6692	-0.43%	0.7187	0.7210	+0.34%	0.7086	0.7179	+1.31%

lutional layer of the architectures, using the test data at different training stages. This allows examining how the learned representations of the CNGNs and CNNs changed concerning one another during training. Figure 4.4 displays the similarity score of the activations at the last convolutional layer of the CNN and CNGN using the test data evolves along training for the different architectures and datasets. The similarity score is the estimated PWCCA of the two pairs of models' activations sets in the three training repetitions of Exp 3.

The tested CNNs and CNGNs have the same architectures with the same pre-trained parameters. Thus, if we were to estimate the similarity scores of their activations before training, we would get a value of 1 (or very close to it) because they evoke the same activations using the same data. Analyzing the similarity scores after (and during) training helps us understand how the astrocytes affect the learned representations for a given architecture.

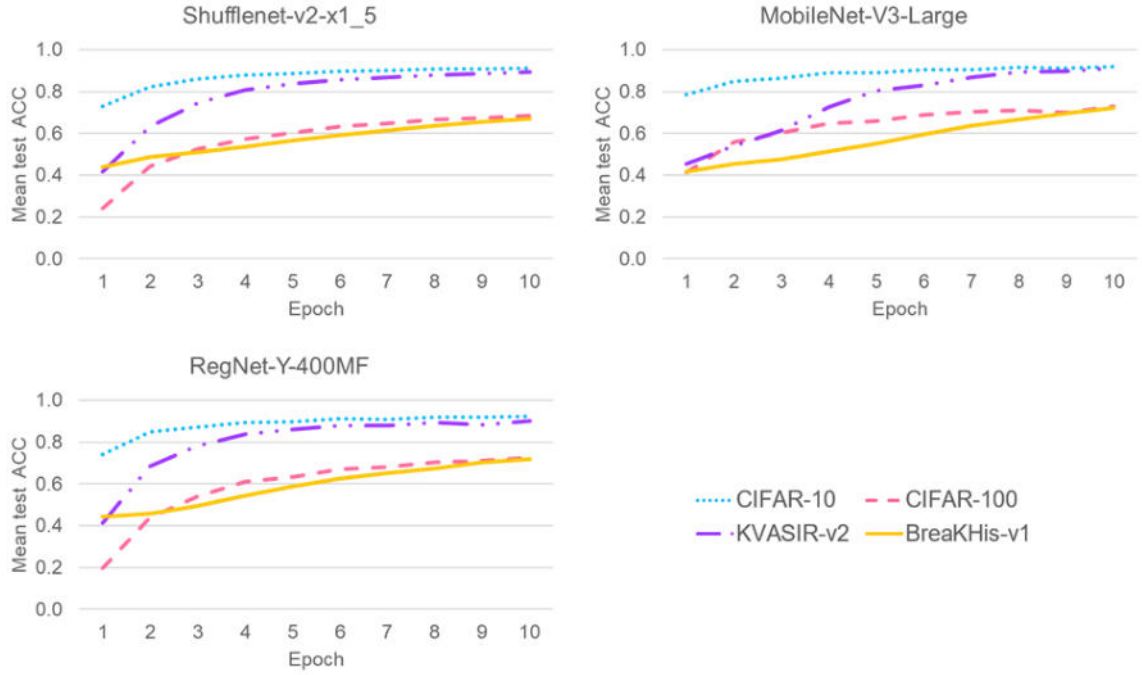


Figure 4.3: Mean test accuracy of the CNGNs during training with the optimal modulation factors in the four tested datasets.

As Figure 4.4 shows, there is a pattern in the similarity following the datasets for the three architectures. For CIFAR-10 and CIFAR-100, the similarity shows little change during training. The similarity is higher in the medical image datasets than in the CIFAR datasets and decreases as the training progresses. Analyzing the similarity per architecture, RegNet-Y-400MF is the architecture that presents the lowest mean similarity, indicating that the astrocytes make this architecture learn more different representations than with the other two tested architectures.

4.5 Discussion and conclusions

Designing and implementing astrocyte-like mechanisms in CNNs is a novel research direction. The few studies in the current literature on this topic have different approaches, spanning the creation of new generalization techniques [171], structure learning systems [173], and short-term memory units [172], all inspired by the astrocytes' role in the biological nervous system. This chapter covers the

Table 4.8: Mean test F1 and MCC of three training repetitions using the optimal hyperparameter configurations found with the grid search. Table 4.1 (Exp 3) shows the training conditions. Δ is the mean percentage change in test MCC of the CNGNs with respect to the CNNs in the same dataset and training conditions. The mean highest test F1 and MCC between the CNN and CNGN is in bold case.

CIFAR-10					
	CNN		CNGN		Δ
Architecture	F1	MCC	F1	MCC	MCC
Shufflenet-v2-x1_5	0.9098	0.9	0.9121	0.9026	+0.28%
MobileNet-V3-Large	0.921	0.9114	0.9187	0.91	-0.15%
RegNet-Y-400MF	0.9245	0.9163	0.9229	0.9145	-0.19%
CIFAR-100					
	CNN		CNGN		Δ
Architecture	F1	Mcc	F1	MCC	MCC
Shufflenet-v2-x1_5	0.6831	0.6839	0.6825	0.6832	-0.1%
MobileNet-V3-Large	0.7287	0.7286	0.7253	0.7254	-0.44%
RegNet-Y-400MF	0.7230	0.7228	0.7237	0.7230	+0.02%
KVASIR-v2					
	CNN		CNGN		Δ
Architecture	F1	Mcc	F1	MCC	MCC
Shufflenet-v2-x1_5	0.8967	0.8824	0.8917	0.8771	-0.60%
MobileNet-V3-Large	0.8859	0.8793	0.9108	0.9005	+2.41%
RegNet-Y-400MF	0.9032	0.8902	0.8994	0.8863	-0.42%
BreakHis-v1					
	CNN		CNGN		Δ
Architecture	F1	Mcc	F1	MCC	MCC
Shufflenet-v2-x1_5	0.5122	0.5507	0.5011	0.5455	-0.96%
MobileNet-V3-Large	0.6178	0.6041	0.6175	0.6208	+0.46%
RegNet-Y-400MF	0.6036	0.6041	0.6072	0.617	+2.14%

development and implementation of an artificial astrocyte that, in contrast to the existing literature, is based on previous work on NGN that mimics the behavior of glial cells in ANNs and has been effective in enhancing the classification performance of MLPs [60–62, 166]. The ideas of NGNs were extended to adapt to the current CNN mainstream models for image analysis. Moreover, we have established the impact of the astrocytes on the learning process by analyzing the CNNs’ learned representations during and after training by making use of the

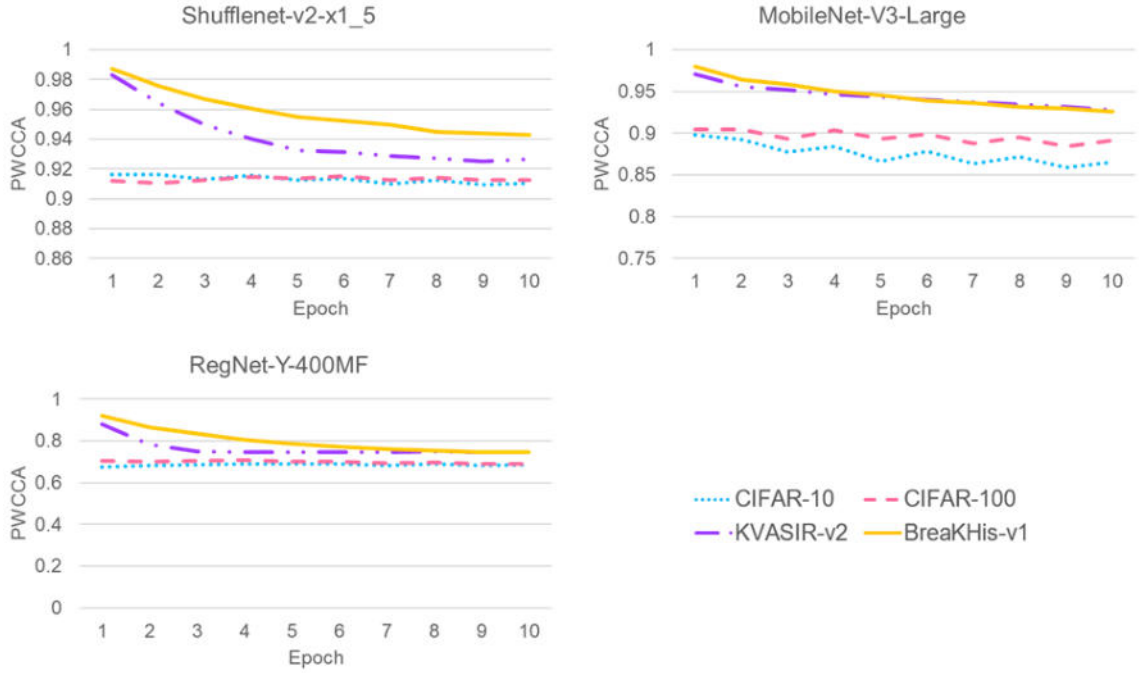


Figure 4.4: Similarity score (mean of the three training repetitions of Exp 3) between the activations at the last convolutional layer of the CNNs and CNGNs during training.[Best viewed in color]

PWCCA technique.

The comprehension of neural networks is a work in progress. The neural networks' internal representations analysis has gained importance in the research community. Different similarity measures have been used to this end [180, 182–184]. Centered kernel alignment (CKA) is frequently preferred over CCA-based methods since it overcomes some limitations of this type of analysis [183]. However, both CKA and CCA methods have disadvantages [184]: CKA fails at detecting differences outside the ten principal components of the representations, whilst CCA is sensitive to the noise of the random initialization of the neural networks' weights, and it needs that the number of data points exceeds the width of the analyzed representations. The limitations of CCA were not meaningful for the presented study since all the neurons' weights had the same initial values during the performed experiments (pretrained networks and transfer learning were used in all cases), and the representations' width was always significantly smaller than

the number of data points in the study. Therefore, PWCCA was selected for the analysis since it outperforms other CCA-based methods [184].

The common ground of almost all studies related to artificial astrocytes is the addition of the concept of time. This is pertinent since the current learning paradigm for neural networks lacks such a notion. Recent neuroscience findings indicate that the learning mechanisms of the biological brain work at different rates and levels. That is to say, there are distinct mechanisms for neuroplasticity. For example, the structural plasticity of the brain –that is, the changes in the network of brain cell interconnections– is the result of a series of entangled processes happening at different time scales. Hence, it is intuitive that we can improve the existing learning paradigm for ANNs by complementing it with other mechanisms working at different rates. Considering that astrocytes interact at a slower rate than neurons in the nervous system, the developed artificial astrocyte use a single change in the inner weights of the network (an optimizer step) to measure the working rate of the neurons, and the astrocytes modulate the neuron’s weights every few iterations of neuronal activity.

The proposed artificial astrocyte is compatible with the existing CNNs. Furthermore, it only engages during training and introduces no additional parameters to the models. The experimental results suggest that the astrocytes can affect the classification performance of the models during the first training epochs, but the effect decreases as the training progresses. Yet, in the particular case of the MobileNet-V3-Large architecture, the astrocytes improve the architecture’s classification accuracy on the KVASIR-v2 dataset by a margin of 2.36% with the training regime employed during the carried experiments.

The optimal configurations of the astrocyte modulation factors found using the grid search are architecture- and dataset-dependent. Furthermore, the found configurations are intriguing (see Tables 4.3-4.6). For example, for epochs 7 through 9, the optimal **Str** is zero for the MobileNet-V3-Large architecture on the CIFAR-10 dataset. That indicates that the astrocytes set to zero the parameters of the convolutional filters that reach the positive Astrocytic sensitivity in the defined number of iterations (Neuron-Glia power connection). Analyzing the mean similarity score of the CNNs and CNGNs neuron activations for CIFAR-10 during epochs 7 through 9, the similarity score does not significantly change. The PWCCA tech-

nique is intended to disregard the noise inside the models’ representations. Perhaps all the convolutional filters whose parameters the astrocytes set to zero in these epochs (with $\mathbf{Str} = \mathbf{0}$) produce no relevant information. Hence, there is no evident change in the similarity of the useful information in the activations and no drop in the test accuracy (see Figure 4.3).

By examining the optimal configurations for the RegNet-Y-400MF architecture, it is clear that the modulation factors are more alike than for the other architectures. Consequently, the astrocytes in the RegNet-Y-400MF seem to compensate for a sub-optimal learning rate for this architecture. The astrocyte hyperparameters are entangled with all the other hyperparameters. An interesting experiment would be to optimize jointly other relevant hyperparameters. As for the MobileNet-V3-Large, the **Weak** hyperparameter tends to have a higher value than the **Str** as the training advances. Again, this behavior could be that the astrocytes decrease the contribution of convolutional filters that provide only noise and empower the ones that provide relevant information for the task.

It is noteworthy that even with the aggressive astrocytes’ modulation of the convolutional filters’ parameters, the models converge towards almost identical classification metrics, and they produce similar neuron activations (as the similarity analysis suggests), even in the extreme cases when the **Str** took values of zero. A possible explanation is that the astrocytes move the convergence point along a flat zone, cutting unnecessary information and acting like a pruning technique. If this is the case, then the artificial astrocytes can be a key to optimizing models for inference by reducing their size in parameters (and FLOPs), making them more efficient for embedded systems. However, more research in this direction is needed to draw solid conclusions.

The consistent increment in the classification performance induced by the astrocytes in the MobileNet-V3-Large architecture on the KVASIR-v2 dataset could indicate that astrocytes improved the model’s generalization. Thus, the developed artificial astrocyte could act as a regularization technique. Interestingly, another research [171] linked the ideas of artificial astrocytes and dropout, a classical regularization technique. Past studies [185, 186] found that dropout can improve a model’s generalization only to the point of the dropout probability and pointed to the need for better regularization techniques that overcome this limitation.

The presented artificial astrocyte unit is based on previous work on NGNs that attempt to imitate the interactions of astrocytes with neurons. It is fascinating how apparently different ideas in research seem to connect. Perhaps the developed method can be thought of as an evolution of dropout since it modulates the network's weights by using their units' activation rates, discouraging the model's reliance on single directions. To test this hypothesis, more research on the impact of the astrocytes on other properties of CNNs using generalization measures and visualization techniques is needed.

In a nutshell, this chapter has analyzed the impact of introducing astrocyte-like units on mainstream CNNs. The experimental results indicate that the astrocytes may improve the classification performance of the mainstream CNNs and help reduce the overparameterized condition of models using the inner information produced during training to prune units that produce irrelevant information for the performed task. However, the carried-out analysis is limited by the scarce precedents of this type of research, the number of experiments, and experimental conditions, such as using the same training scheme for all the architectures and datasets. This work can be a stepping stone for new studies following this research pathway. It is imperative to keep dissecting the many unknowns of ANNs by keeping pure research going and not getting carried away by trending technological advances, overlooking the proper study of the current paradigms.

Chapter 5

Hybrid learning with model fusion and hyperparameter optimization

This chapter presents a final approach that connects all the previous work presented in this thesis. However brief, it serves as a prologue for a new and promising research direction that seems to be the element that will bridge the state-of-the-art deep learning methods and real-world applications. Let this chapter be an epilogue of this thesis research, delivering key ideas for the future and cementing the new generation of intelligent systems for medical image classification with the delivery of a novel hybrid learning strategy that empowers efficient convolutional neural network models to reach unprecedented classification performance while maintaining the computational requirements low.

Ensemble learning involves combining multiple models to improve classification performance. However, as deep learning models have become more complex, ensembles have become impractical due to the high computational resources they require. On the other hand, model fusion aims to merge different models into a single one. Weight averaging is a method that combines the parameters of neural networks to create a new parameter set that generalizes better. This method has been particularly effective in recent years with deep learning models of similar architectures, especially when the models are fine-tuned using different hyperparameters. In light of this, a new hybrid learning strategy called Gen-Soup has been proposed. Gen-Soup optimizes hyperparameters using a genetic algo-

rithm and uses weight averaging to combine the parameters in the populations of solutions. This causes the population to evolve by simultaneously learning hyperparameters and parameters. Additionally, artificial astrocytes are utilized to add variance to the parameters without affecting the convergence region in the loss landscape, allowing the populations to reside in different zones of optimal regions, which is a desirable condition for weight averaging in deep learning models. Experimental results validate the effectiveness of this approach.

5.1 Introduction

Ensemble learning is a prevailing practice in ML that combines the outputs of various ML models to overcome the limitations of a single model and improve the final system’s performance [43, 44]. However, ensemble learning bears a higher computational cost from storing multiple models and running all of them at inference operation [43–45]. This burden increases with the size and complexity of the ensembled models [44] and can be a limitation for applications with low-latency requirements [43] (as medical image classification can be).

Deep model fusion has gained attention for its potential to save resources in DL applications practice [44], by replacing several DL networks in an ensemble with a single one. Model fusion generates a single network by merging various DL models. Additionally, the resulting model is less prone to overfitting and is more robust by reducing the individual models’ bias and noise [44], which are two of the three inevitable error sources in any ML model, as explained in Section 2.1.

Regarding deep neural networks, when trained with gradient-based algorithms, the solutions (parameters) usually converge to points proximate to the boundary of a flat region in the loss landscape [44]. Fusing different solutions can approach the central zone of a flat optimal, leading to a better generalization solution [44]. Hence, fusing models’ parameters instead of just combining their outputs can have advantages in both computational efficiency and classification performance. Moreover, the combination of fine-tuning with model fusion can reduce training costs and address the requirements of specific tasks or applications of DL [44].

Recent evidence shows that fusing DL models with similar architectures, fine-

tuned with different hyperparameters, leads to better classification performance [43,45], especially in data with a distribution shift (distinct from the training data but from the same domain) [49, 50, 187]. Thus, these models generalize better outside the scope of the training data.

The depicted advantages of fused models are more than enough reason to explore this research direction for medical image classification systems. Consequently, using the work presented in Chapters 3 and 4, a new framework based on model fusion is conceived: Gen-Soup. Gen-CNN (presented in Chapter 3) already finds optimal hyperparameters with small variations for a given dataset, and model fusion is compatible with the populations of solutions in the GA of Gen-CNN. Moreover, the gene that represents the architecture type serves to identify similar models that can be fused using weight averaging [43,45,49], a simple yet effective fusing technique for similar models. Additionally, weight averaging thrives when uses parameters close to an optimal region in the loss landscape but with some variations [49]. The artificial astrocyte unit developed in Chapter 4 can deliver the diversity in the parameters trained with similar and pseudo-optimal hyperparameters.

Note: by the nomenclature of the model fusion field, in this chapter, the terms of model parameters and weights are used interchangeably.

5.2 Methodology

5.2.1 Genetic algorithm

The Gen-CNN's GA presented in Chapter 3 also serves as the foundation for this new framework. Besides the five genes that represent the hyperparameters optimized by Gen-CNN (architecture, learning rate scaling factor, weight decay factor, loss function, and optimizer), two more genes are included in the individuals that represent the solutions of the GA. These two new genes represent the Str and Weak artificial astrocyte hyperparameters. Figure 3.2 shows the encoding of hyperparameter values into the $[0,1]$ interval.

The GA works just the same as previously explained in Section 3.3.2, with the addition of an elitism strategy that preserves the best solution from one gen-

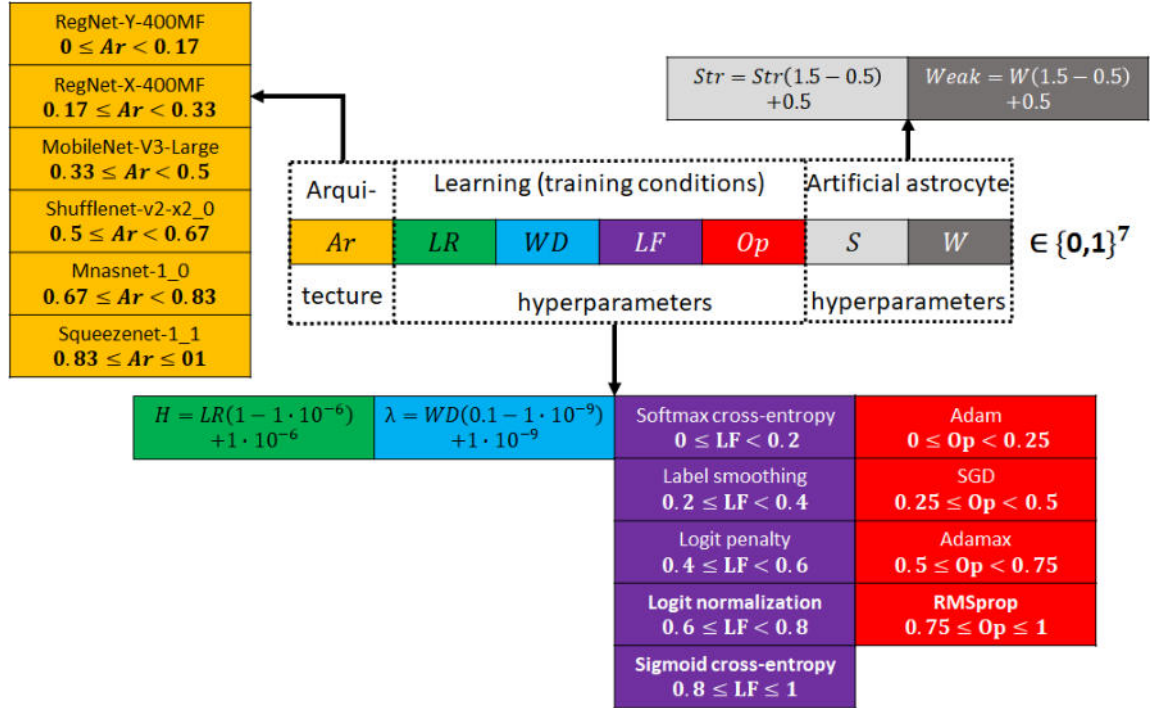


Figure 5.1: Encoding scheme of the individuals into 7 real-valued genes, each with magnitude of one, $\{Ar, LR, WD, LF, Op, S, W\} \in \{0,1\}^7$, that stand for the hyperparameters of CNN architecture, learning rate scaling factor, weight decay factor, loss function, optimizer, artificial astrocytes' Str factor, and artificial astrocytes' Weak factor, respectively. The genes' values are transformed into the hyperparameters' values as shown for each gene.

eration to the next. Hence, the best-evaluated individual of the generation gets directly into the next one.

5.2.2 Hybrid learning with weight averaging

In Gen-CNN, the GA served to find the optimal hyperparameter configuration for the dataset, effectively using low-fidelity prototypes during the evaluation phase for computational efficiency. In this chapter, the GA serves as a means to train the population of models to fuse at the same time as the hyperparameter optimization is performed, constituting a learning strategy that simultaneously learns hyperparameters and parameters by using the evaluation phase as the training step of the candidate models.

The GA initiates with a large population of models with different hyperparameters. Each model is trained for a single epoch and evaluated using the validation data (as described in Section 3.3.2). Then, the best-fitted individuals are selected with a tournament mechanism (as described in Section 3.3.2), and a new generation with fewer individuals is generated using the selected models. The new generation models inherit the hyperparameter configuration characteristics from their parents, as in Chapter 3. But, additionally, they receive the learned parameters of the best-fitted of their parents. Thus, each generation learns for one epoch and passes the learned parameters to the next generation, executing a sequential learning of parameters along the evolution of the population.

In order to inherit the learned parameters from its best-evaluated parent, the offspring also receives the architecture gene from it. This behavior, along with the progressive shrinking of the population size, provokes that a single architecture gene prevails over the others in the latter generations. From an evolutionary perspective, we could think of the different architectures as competing species in an environment with limited resources. The other (learning) hyperparameter genes define behavior tendencies that determine how the individuals explore their environment to find the resources (best zones in the loss-landscape), and the received learned parameters are inherited knowledge about the environment, which the new individuals try to improve to survive and pass their genes.

Iteratively averaging the weights of models with the same architecture at dif-

ferent training times can reduce the resulting model's variance as long as the averaged models have similar training trajectories [44]. The model fusion element is introduced precisely in the inheritance of the learned parameters. If the two parents have the same architecture, the learned parameters that are passed to the offspring are a weighted average of the set of parameters of both parents. The set of inherited parameters is created using the same mechanism as the crossover of hyperparameter genes, with each parent's contribution to the offspring determined by their evaluations. Thus, the parameters of the offspring $\theta_{Offspring}$ are given by

$$\theta_{Offspring} = \begin{cases} \theta_{P_{BE}} & \text{if } P_1^{Ar} \neq P_2^{Ar} \\ \frac{\theta_{P_1} Ev_{P_1} + \theta_{P_2} Ev_{P_2}}{Ev_{P_1} + Ev_{P_2}} & \text{if } P_1^{Ar} = P_2^{Ar} \end{cases} \quad (5.1)$$

where $\theta_{P_{BE}}$ is the set of parameters of the best-evaluated parent, P_1^{Ar} and P_2^{Ar} are the architecture types of the first and second parent, respectively. θ_{P_1} and θ_{P_2} are the sets of parameters of the first and second parent. And, Ev_{P_1} and Ev_{P_2} are the evaluations of the first and second parent.

Gen-CNN uses low-fidelity prototypes during the evaluation phase of the hyperparameter optimization to tackle the efficiency issue of population-based optimization algorithms. Then, using the found global optimal set of hyperparameters, the final model is trained with a higher computational compute regime that uses the training images in a much larger size than in the low-fidelity prototypes. Using larger image sizes improves the classification performance of DL models but increases the computational load exponentially. Gen-Soup takes inspiration from Gen-CNN to improve the algorithm efficiency and maintain the advantage of using larger image sizes.

Since the proposed hybrid learning approach uses more numerous populations in the early generations than in the latter, using a big image size from the beginning would pose a significant computational load. Consequently, Gen-Soup scales the image size with the passing of generations to improve the computational efficiency of the algorithm. The early generations use small image sizes to explore the hyperparameter search space and approach optimal zones in the loss landscape. Ultimately, the latter generations, composed of much fewer individ-

uals, take advantage of both the increased image size and the optimal hyperparameter configurations found at that point to exploit the optimal loss-landscape regions.

5.3 Results

5.3.1 Datasets

Two of the medical image datasets used in the previous chapters, KVASIR-v2 [141], and BreKHis-v1 [145], are once again employed to assess the suitability of this hybrid learning approach for the development of medical image classification systems. These two datasets are the choice for the following reasons:

1. Efficiency. Since these experiments are to prove the operability and usefulness of the proposed approach without seeking final results or a comprehensive assessment of the algorithm behavior, it is better to test it with datasets that are not too numerous so the experiments do not take too much time to complete. These two datasets have enough images to effectively fine-tune DL models and are much scarcer than standard benchmarking datasets, such as CIFAR-10 [181], or other medical image datasets, such as ISIC-2019 [142–144].
2. Effective representation of the target domain. The objective of this thesis is to develop image classification systems suitable for medical images. The KVASIR-v2 has images from the gastrointestinal tract captured with a camera during endoscopies, whereas BreKHis-v1 contains microscopic images of breast tumor tissue collected at different magnifying factors (40X, 100X, 200X, and 400X). Therefore, these two datasets consist of medical images from distinct medical specialties, from unlike body parts at very dissimilar scales, captured with different devices. Thus, they serve to represent a portion of the vast diversity of the medical image domain.
3. Contrast between balanced and unbalanced datasets. This thesis frequently discusses the challenge of the natural imbalance in medical imaging sam-

ples. Furthermore, the experimental results in Chapter 4 on artificial astrocytes suggest that some hyperparameter configurations of these units can negatively impact the classification performance of models in unbalanced datasets. Consequently, experiments are conducted using a balanced dataset (KVASIR-v2) and a highly unbalanced one (BreakHis-v1) to assess how the proposed hybrid learning approach is affected by dataset imbalance and whether the hyperparameter optimization of the artificial astrocytes can address the previously observed behavior of CNGNs in unbalanced datasets.

5.3.2 Experiments

Experimental setting

All the experiments used the same software and hardware conditions: AMD Ryzen 5 3400G CPU, one NVIDIA GeForce GTX 1660 Ti GPU, 16 GB RAM, and 476 GB system memory. All the algorithms were implemented in Python 3.8.18, using the environment Spyder 5.4.3. Pytorch 2.2.1 and CUDA 11.8 are the frameworks for neural networks and optimizers. The optimizers used their default values in Pytorch for their specific hyperparameters.

The hyperparameters for Gen-Soup used in the experiments are displayed in Table 5.1

Search space

The search space of the optimized hyperparameters used for the experiments is a combination of the search spaces defined in previous Chapters. The learning (training) hyperparameters have the same search space as in Gen-CNN in Chapter 3, and the astrocytes hyperparameters have the search space defined in Chapter 4. As for the architecture, for this batch of experiments, the search space is defined only with architectures capable of performing on mobile devices (with a FLOP complexity of up to ~ 600 MFLOPs [16]) for efficiency of the experiments. Table 5.2 shows the search space of the optimized hyperparameters using Gen-Soup.

Table 5.1: Hyperparameters of Gen-Soup for the experiments. The crossover and mutation probabilities appear in order for the genes of CNN architecture, learning rate scaling factor, weight decay factor, loss function, optimizer, artificial astrocytes’ Str factor, and artificial astrocytes’ Weak factor, respectively.

Hyperparameter	Value
Generations	Exp1 = 20 & Exp2 = 35
Training epochs per generation	1
Training epochs at the same image size	1
Training epochs at max image size	10
Image sizes	From 224×224 to 302×302
Learning rate scheduler	Exp1 = Cosine annealing Exp2 = None
Initial population size	50 individuals
Reduction of population size per epoch	5 individuals
Minimum population size	10 individuals
Evaluation function	MCC (validation)
Crossover probability	[0, 0, 0, 0.9, 0.9, 0.9, 0.9]
Mutation probability	[0, 0.25, 0.25, 0.25, 0.25, 0.15, 0.15]

Table 5.2: Search space of the hyperparameters optimized by Gen-Soup.

Hyperparameter	Search space
Architecture	RegNet-Y-400MF, RegNet-X-400MF, MobileNet-V3-Large, Shufflenet-v2-x2_0, Mnasnet-1_0, Squeezenet-1_1’
Learning rate scaling factor	$[1 \times 10^{-5}, 1]$
Weight decay	$[1 \times 10^{-9}, 0.1]$
Loss function	Cross-entropy, label smoothing, logit penalty, logit normalization, sigmoid cross-entropy
Optimizer	Adam, AdamW, Adamax, Adagrad, ASGD, LBFGS, NAdam, RAdam, RMSprop, Rprop, SGD
Astrocytes’ Str	[0.5, 1.5]
Astrocytes’ Weak	[0.5, 1.5]

5.3.3 Data augmentation

For all the experiments, every training batch goes through a transformation phase where the techniques of random crop, random rotation (of up to 90°), random vertical flip (50% probability), random horizontal flip (50% probability), color jitter transformation, and Cutmix or Mixup serve as augmentation techniques.

Results

This chapter aims to present the hybrid learning algorithm and validate its effectiveness. Thus, it is a proof of concept of the hybrid learning framework based on hyperparameter optimization and model fusion. Therefore, two experiments are performed with the elected datasets to assess the viability of this new research direction. For all the Gen-Soup hyperparameters, there exists an antecedent to guide its configuration. Thus, these two puzzling hyperparameters have different values in the two performed experiments precisely to evaluate their impact on the algorithm.

These two hyperparameters are the number of training epochs and the employment of a learning rate scheduler. For almost all of the Gen-Soup hyperparameters, the Gen-CNN hyperparameters used in Chapter 3 serve as guidance. Gen-CNN trained the final model using the hyperparameters set found by the GA using a standard number of epochs and a learning rate scheduler for the type of model and dataset. However, Gen-Soup trains the final model along with the hyperparameter optimization procedure. Thus, the number of generations and training epochs per generation determine the number of training epochs of the resulting model. For simplicity, let us set the number of training epochs per generation to one. Therefore, a generation is equivalent to a training epoch. Also, the employment of a learning rate scheduler is a good practice. However, the learning rate changes each generation (epoch) due to the hyperparameter optimization, and the astrocytes perform an additional non-supervised modulation of the networks' parameters that also change each generation by the hyperparameter optimization. Thus, there is no way of anticipating how all of these different modulations of the parameters interact with each other.

For the evaluation of the impact of artificial astrocytes on the proposed learn-

ing framework, each experiment applies the Gen-Soup algorithm to the datasets in two ways: once with astrocytes and once without. These two experiments utilize different numbers of generations, with one using the same learning rate scheduler as Gen-CNN and the other using no learning rate scheduler. Table 5.1 shows all the Gen-Soup hyperparameters used for both experiments. Let us designate the experiment that uses learning rate scheduler as Exp1, and the other as Exp2. In the experiments, when the astrocytes are not included, the individuals have five hyperparameters, just like the experiments using Gen-CNN in Chapter 3. On the other hand, when the astrocytes are utilized, the individuals have seven hyperparameters. Experiments using CNGNs (CNNs with astrocytes) have individuals composed in the way that Figure 5.1 shows. Experiments that do not include artificial astrocytes remove the *Artificial astrocyte’s hyperparameters* genes from their genomes.

Figure 5.2 shows the evaluation (MCC in the validation partition) of the best-evaluated individual per generation in Exp1. The models seemingly converge to similar classification performance regardless of the astrocytes in both datasets. Table 5.3 shows the classification performance metrics of the optimal individuals (the ones with the highest evaluation during the operation of Gen-Soup) in Exp1. The Shufflenet-v2-x2 architecture prevailed in all cases but the KVASIR-v2 dataset without astrocytes where RegNet-Y-400MF prevailed, which is also the case that achieved the best classification metrics for Exp1 in the KVASIR-v2 dataset. For the BreakHis dataset, the case with astrocytes had a slight enhancement in classification metrics with respect to its counterpart without astrocytes.

Figure 5.3 shows the evaluation of the best-evaluated individual per generation in Exp 2. As with Exp 1, the models converge to similar classification performance in both datasets. In contrast with Exp1, the evaluations’ behavior for KVASIR-v2 is quite unstable. Table 5.4 shows the classification performance metrics of the optimal individuals in Exp2. Once again, the Shufflenet-v2-x2 and RegNet-Y-400MF were the prevailing architectures, this time an equal amount of cases each. For Exp2, the cases that used artificial astrocytes have the highest classification metrics for both datasets.

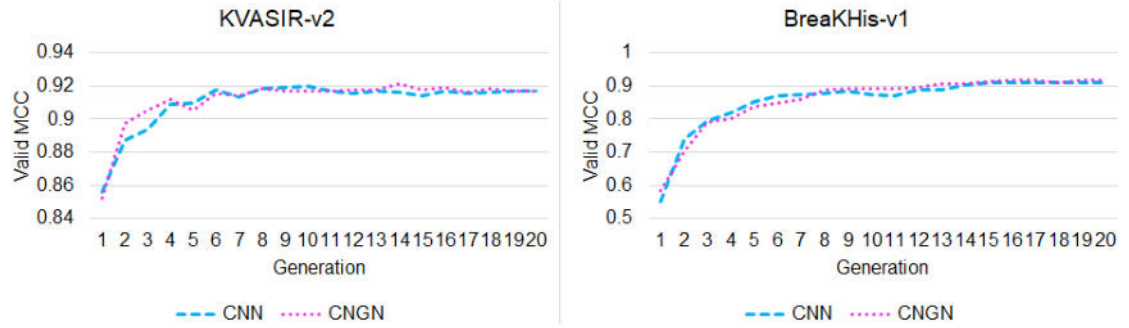


Figure 5.2: Validation MCC of the best-evaluated individual in each generation of Gen-Soup in Exp 1.

Table 5.3: Classification metrics of resulting models in Exp 1.

KVASIR-v2				
Artificial astrocytes	Prevailing architecture	Test CAT-ACC	Test F1	Test MCC
No	RegNet-Y-400MF	0.9287	0.9285	0.9187
Yes	Shufflenet-v2-x2_0	0.9237	0.9237	0.9131
BreakHis-v1				
Artificial astrocytes	Prevailing architecture	Test CAT-ACC	Test F1	Test MCC
No	Shufflenet-v2-x2_0	0.9279	0.9172	0.9051
Yes	Shufflenet-v2-x2_0	0.9336	0.9256	0.9129

5.4 Discussion and conclusions

The proposed hybrid learning algorithm is fully operational. These experiments, which are intended to be a proof-of-concept, were successful in demonstrating the effectiveness of the method since all the tested cases achieved competent classification performance (can see Tables 3.6 and 3.8 for references of the state-off-the-art models' performance in these datasets) despite the reduced training setting in terms of the number of total training epochs, image sizes, and the lack of precursors to guide the selection of many of the fundamental hyperparameters of Gen-Soup.

With the experiments' efficiency in mind, only low-FLOPs complexity architectures were included in the search space. Despite the simplicity of the archi-

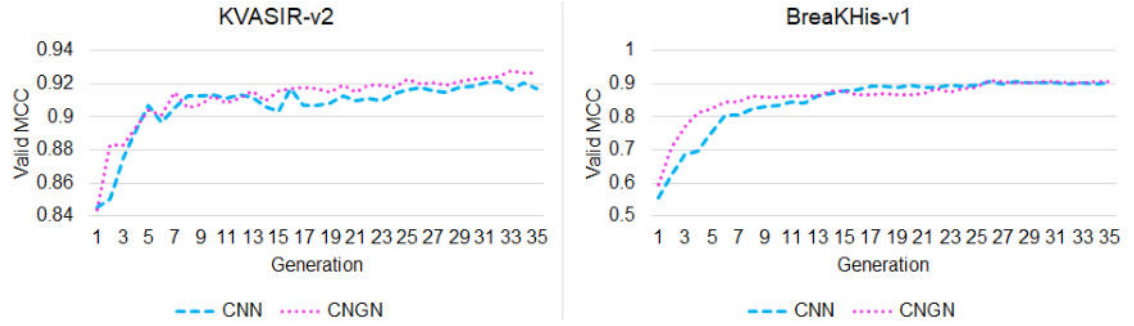


Figure 5.3: Validation MCC of the best-evaluated individual in each generation of Gen-Soup in Exp 2.

Table 5.4: Classification metrics of resulting models in Exp 2.

KVASIR-v2				
Artificial astrocytes	Prevailing architecture	Test CAT-ACC	Test F1	Test MCC
No	Shufflenet-v2-x2_0	0.9387	0.9387	0.9301
Yes	RegNet-Y-400MF	0.9418	0.9419	0.9335
BreakHis-v1				
Artificial astrocytes	Prevailing architecture	Test CAT-ACC	Test F1	Test MCC
No	Shufflenet-v2-x2_0	0.9140	0.8956	0.8866
Yes	RegNet-Y-400MF	0.9292	0.9162	0.9066

tectures when compared to the state-of-the-art architectures, they achieved outstanding classification performance using the proposed hybrid learning method. The results presented in Table 5.4 for the KVASIR-v2 dataset are, to the best of the author’s knowledge, the highest reported in the current literature. Notably, these results were obtained using relatively simple architectures with up to approximately 21 times fewer FLOPs than their closest competitors in this dataset (can see FLOPs of competitor models in Table 3.6).

The Shufflenet-v2-x2 and RegNet-Y-400MF architectures prevailed against all others in the experiments. However, the architectures consistently changed with the experiments’ conditions, with only de Shufflenet-v2-x2 prevailing always for the BreakHis-v1 dataset with no astrocytes case. These results imply that these two architectures adapt better than the other in the search space for the consid-

ered hyperparameters of the Gen-Soup algorithm. However, the optimal architecture continues to seem dependent on the dataset and all the other hyperparameters' settings. This observation justifies that HPO algorithms are fundamental to obtaining optimal results in DL applications.

For the KVASIR dataset, the usage of a learning rate scheduler appears to be consequential in the stability of the models' evaluations stability for the hyperparameter optimization phase, as the differences between Figure 5.2 and Figure 5.3 suggest. Perhaps that is the case for other datasets. However, both cases (with and without astrocytes) achieved better classification performance in Exp2 than in Exp 1 for this dataset. Yet, it can also be because of the increased number of training generations. Nevertheless, for the BreKHis-v1 dataset, the opposite occurs. Despite the increased number of training epochs in Exp2, this experiment had lower classification metrics than Exp1. These results imply that the optimal usage of a learning rate scheduler can be dataset-dependent. In future work, the usage or not of a learning rate scheduler can be considered as a hyperparameter for the HPO. However, the question of whether it would be an independent hyperparameter of a conditional hyperparameter of the learning rate scaling factor needs a deeper analysis.

Even though the initial experiments were successful, there is still much to learn about the behavior of Gen-Soup. The hybrid learning algorithm being proposed has several hyperparameters that require further investigation. For example, while scaling the image size over the generations appears to be effective, it can bias the evaluations of early generations, consequently influencing the prevailing architecture. This could result in a suboptimal architecture prevailing due to this method, ultimately impacting the algorithm's overall performance. Moreover, the learning hyperparameters used in the experiments are the same as in Chapter 3 due to their proven efficacy in shaping the models' performance. However, the considered artificial astrocyte's hyperparameters are the same as in Chapter 4 for the lack of a deeper understanding of the artificial astrocytes that require further investigation all alone.

The model fusion technique implemented in Gen-Soup is based on the model soups approach [45], from which it derives its name. Model soups involve combining models' parameters with the same architecture but different hyperparam-

eters, such as learning rate, weight decay, training epochs, data augmentation, mixup, and label smoothing. Many of these hyperparameters are within the scope of the HPO process of Gen-CNN. Therefore, the methods of model soups and Gen-CNN are compatible, and their combination originated the idea of hybrid learning that Gen-Soup performs.

The population parameter averaging (PAPA) [43] method is a weighted average approach that combines the parameters of models trained on the same data but with different augmentation techniques. At regular intervals during training epochs, the individual model parameters are adjusted towards the mean weights of a population of models. PAPA achieves this by averaging the models' weights using the exponential moving average, which is a hyperparameter that regulates the linear interpolation of the averaged weights. In contrast, Gen-Soup averages the parameters of pairs of models (parents) using the weighted average by the parent's evaluations that the proposed GA also uses in the crossover of genes.

The experimental results of the PAPA approach demonstrated that weight averaging is effective when employed with population-based methods [43], which reinforced the idea of combining the population-based HPO of the GA in Gen-CNN with weight averaging to construct the Gen-Soup algorithm.

Weight average is the most efficient model fusion technique, and, for similar models, it can effectively approximate the optimal point of the region in the parameter space where the loss function value is low [44]. The hypothesis behind Gen-Soup is that the GA drives the models to similar trajectories in the loss landscape since the prevailing genes define how the models engage with the optimization surface in training (the loss landscape). Thus, as the GA converges, the models should be near each other in the loss landscape and weight averaging their parameters aids them to reach the center of optimal flat regions.

An interesting future research is using mode connectivity methods to assess the loss-landscape of similar individuals in the population. For instance, creating a generation technique that uses the parameter space of the low-loss manifold connecting two individuals. Also, using the pipeline quantification of mode connectivity as an evaluation metric for populations of solutions to find optimal regions in the landscape using the local optimal points that are the individuals.

Chapter 6

Discussion and conclusions

6.1 Hyperparameter optimization

Throughout this thesis, the selection of different hyperparameters has been continuously proven to be fundamental in determining the classification performance of models. Moreover, the experimental results in chapters 3 and 5 show that, with hyperparameter optimization, models with significantly less computational complexity can match or surpass the classification performance of ensemble and state-of-the-art models. Such is the case of the two models based on the ShuffleNet-v2-x2_0 architecture in Table 5.4, which have only 591 MFLOPs and almost the same classification performance as the ensemble model with approximately 12.63 GFLOPs (≈ 21 times more than the ShuffleNet-v2-x2_0 models) in Table 3.6.

Even more impressive is the fact that the ShuffleNet-v2-x2_0 model in Table 5.4, trained with the hybrid training scheme (using the artificial astrocytes) proposed in Chapter 5, is the model with the highest classification metrics for the KVASIR-v2 dataset in the current literature to the best knowledge of the author. Even though the experiments conducted in Chapter 5 serve as a proof of concept for the proposed hybrid learning approach and were not intended to achieve the best possible results. Let us recall that the artificial astrocytes do not introduce more parameters to the networks nor extend their capacity and are present exclusively in the training phase. The models are the same off-the-shelf architectures that have been available in many DL libraries for years. The proposed approaches

only guide the networks to optimal solutions without introducing anything new into them.

The results of this thesis demonstrate that optimizing a few pivotal learning hyperparameters is sufficient to develop competent image classification models. These results add to the building of knowledge about the hyperparameters of CNNs. It is worth mentioning that, to the best of this author’s knowledge, the research performed in this thesis is the first precedent of introducing the pretrained CNN architecture as a learning hyperparameter, allowing the HPO algorithm to choose the best fitting architecture for the dataset from a pool of off-the-shelf CNNs. Besides, it was proven that the low-fidelity prototypes commonly used in the related field of NAS are effective surrogate models for the evaluation of CNNs under different hyperparameter configurations, hastening the evaluation phase of the GA, which is the most resource-intensive and time-consuming step of population-based optimization algorithms.

The novelty and expectations of DL have caused the unstoppable development of more complex architectures and learning methods, increasing the number of hyperparameters and intricacy in their interactions at an unexpected rate. Therefore, hyperparameter optimization will be crucial for the future of DL. Although HPO algorithms for DL are still in their early stages, they have demonstrated remarkably good results, as evidenced in this thesis. Nevertheless, there is much that remains unknown. For instance, the effect of meta-overfitting is a crucial limitation of HPO that remains widely ignored and lacks of proven and efficient countermeasures. In this thesis, the maximization of the validation MCC was set as the optimization target for the HPO algorithms to reduce the meta-overfitting effect with seemingly positive results. However, an in-depth analysis of this measure is needed to assess its effectiveness. Another important topic that requires more attention is the design of optimization targets for HPO in DL. Almost all of the existing methods, including the ones proposed in this thesis, maximize the performance in a partition of the training dataset, which has been successful up to this point but seems quite simplistic and lacks relevant research that supports it as the better choice.

Many opportunities exist to enhance HPO techniques for DL. These opportunities originate from the unresolved mysteries that CNNs, and ANNs in gen-

eral, still have. For the progress of DL, it is crucial to conduct research that examines the behavior of CNNs and quantifies their properties. HPO for DL can benefit from using methods that effectively quantify the network's characteristics. Currently, there is ongoing research in this area that has shown promising results, such as the class-selectivity index measure and linear mode connectivity approaches. Until our understanding of the behavior of CNNs improves, we can continue to rely on metaheuristic methods, such as GAs, which have demonstrated effectiveness and efficiency in the field of DL.

6.2 Neuro-inspired models

ANNS are ultimately inspired by the human brain. However, in recent years, this fundamental connection has been overlooked. The contemporary success of DL can be attributed to the advancements in technology and the availability of massive amounts of data. Despite this, it is important to acknowledge the origins of DL. Most research in the past decade has focused on improving model performance at the expense of increased complexity. While this approach has been effective, it has also introduced other issues. More significantly, there is a scarcity of research that genuinely examines these systems. Much remains unknown about DL models, and the field is evolving at a rapid pace, often neglecting the necessary analysis of these systems. In this thesis, a simple model resembling astrocytes was developed to explore the potential impact of the mechanisms of these cells on standard CNNs, with enigmatic findings.

The developed artificial astrocyte modulates the network parameters by the firing rate of their neurons in a non-supervised fashion. The several experiments performed to characterize the effect of this parameter modulation mechanism show that the CNNs converge to similar classification performance despite the aggressive and non-supervised modulation by the astrocytes, suggesting that ANNs have the capacity to autoregulate using their inner signals. This opens the door to a whole new family of approaches. In the author's opinion, astrocytes appear to function as a regulatory element within neural networks. They could serve as a crucial factor in enhancing trust in ANNs, which are often seen as black boxes due to their opaque nature.

The experiments performed with the artificial astrocyte are limited in different aspects. First, only two of the four hyperparameters of the artificial astrocyte model were investigated. Additionally, other hyperparameters influence the artificial astrocyte behavior, such as the batch size in training. An in-depth analysis of these is pending. Second, the analysis of the effect on the networks is limited by the employed metrics. A common issue for the proper study of neural networks is the lack of reliable and straightforward methods to measure their characteristics. The PWCCA metric was useful for assessing the similarity in evoked activations between networks, and the classification metrics always shed light on the performance of the models in the target task. However, more analysis using other approaches is needed to characterize the behavior of the proposed CNGNs paradigm.

The results in Chapter 4 (comparing the accuracy in Table 4.7 and Table 4.8 for the BreKHis-v1 dataset) showed that the astrocytes were detrimental to the classification accuracy of models trained with unbalanced datasets. However, it was hypothesized that it was due to the grid-search optimization of their hyperparameters that targeted the classification performance, which is not the best choice for imbalanced datasets (hence the selection of the MCC for the developed HPO algorithms). The results of Chapter 4 confirm this hypothesis since the astrocytes guided the models to outperform the CNNs (networks without astrocytes) in the BreKHis-v1 dataset (Tables 5.3 & 5.4), which is unbalanced. This also reinforces the fact that the MCC is a good optimization target for HPO algorithms since it is a well-rounded and robust metric that represents the classification performance in a more reliable way than other metrics, such as accuracy.

This thesis results on artificial astrocytes are limited by the previously discussed factors. Additionally, the proposed artificial astrocyte unit is a simplistic interpretation of the role of astrocytes in the human brain. Even so, the experimental results show that the artificial astrocytes indeed contribute to the models' learning process. An intriguing phenomenon that remains answerless and has to be further studied is when astrocytes set to zero the parameters of whole kernels without influencing the models' performance and evoking similar activation to the CNNs trained under the same conditions (without astrocytes).

Finally, this thesis explored neuro-inspired units based on astrocytes to im-

prove the learning paradigm of neural networks. However, the area of neuro-inspired computing is immense and full of possibilities. The fields of AI and neuroscience have always found support in one another, but an important part of the current research on DL is too focused on technological developments, overlooks fundamental research that is always needed, and neglects other opportunity areas. It is important to raise the voice about this issue and support new and original research.

6.3 Model fusion

New methods are emerging to address the issue of the computational efficiency of DL models. It is important to handle this matter and not get carried away by impressive results at an experimental level. The field of AI has historically suffered from big expectations founded on early results that failed to deliver practical solutions. Model fusion apparently will play a key role in the future, given that ensemble learning has become impractical for contemporary DL models.

Together, model fusion and HPO can handle the generation of practical models with outstanding classification performance for non-expert ML users. Facilitating the application of this technology is fundamental to allowing its employment in other areas and hastening the research of applied DL.

The proposed hybrid learning method has a surprisingly effective performance. The experiments conducted in Chapter 5 were only intended to show the operability of the algorithm and set the base for future research. Nevertheless, the resulting models have an outstanding classification performance despite their FLOP complexity. The results displayed in Table 5.4 for the KVASIR dataset are the highest in the current literature for the best knowledge of this author. Achieving this with an architecture that is theoretically capable of operating on mobile devices is a prominent result that demonstrates that the current DL models have the capacity to offer the solution expected from them. We only need to find a way of utilizing them properly.

The results obtained with Gen-Soup evidence that we are going in the correct direction to fulfill the expectations about practical AI solutions. However, there is much work to do. The results of Gen-Soup are still preliminary, and an in-depth

analysis of the algorithm behavior is still pending.

6.4 Ethical implications

DL algorithms are powerful tools but still data-driven, which bear many issues. This type of technology thrives in good part because of the Big Data era. However, having massive amounts of data does not guarantee that the data captures the real world. In healthcare, DL systems can be uncertainty-biased as a consequence of the under-representation of marginalized groups [11] or present bias due to the data being endogamic [8, 25]. Also, the data carries the predispositions of the people that generated it [8]. For instance, a study found that an algorithm was categorizing unequally the risk levels of black and white people despite their illness signs, affecting millions of patients [11, 188]. In this regard, this author believes that DL systems targeted to clinical applications should undergo a comprehensive validation process, using data from the facilities where the systems are to be used whenever possible. Additionally, explainable AI will play a major role in building trust in these systems.

Also, DL systems are prone to error, as any other system. However, the lack of proper guidelines and laws regarding the application of AI in medicine for the novelty of such systems poses a potential liability for medical practitioners [189]. Moreover, datasets of medical images can convey privacy and ethical issues [8]. Sensitive patient information is prone to abuse or unauthorized utilization [8]. Thus, protecting patient privacy and confidential information is crucial while creating medical datasets [8]. These situations lay outside the scope of research and engineering, but are still key factor for the spreading of these type of technology.

6.5 Conclusions

The results of this thesis show that the existing DL architectures can offer solutions to the contemporary challenges of AI. Extending the capacity of models and designing more complex architectures has been successful in improving the classification performance but poses many other challenges for real-world appli-

cations, hindering the spreading of this technology to other areas, such as medical image classification. The research presented in this thesis demonstrates that there are other, more efficient ways of improving classification performance without compromising the scalability and practicalness of the classification models.

Several contributions were made throughout this thesis. In the HPO of DL systems regard, a few influential hyperparameters were found to be sufficient to shape the classification performance of models trained using transfer learning. Additionally, a GA with real-valued genes capable of optimizing numerical and categorical hyperparameters was proposed and validated using the standard criteria for heuristic models in Chapter 3. The proposed GA counteracts the effect of meta-overfitting, a limitation for HPO in DL, by using the MCC in the validation data as an optimization target. This countermeasure was successful throughout the thesis, but a formal analysis is pending to completely validate its effectiveness.

Chapter 3 culminates with the development of an AutoML framework that automatically generates optimized CNN models for image classification. Gen-CNN is capable of generating medical image classification models with low computational complexity (in terms of FLOPs) with classification performance comparable to or even better than the state-of-the-art, custom-made, and ensemble models in the current literature.

Contributions to the field of neuro-inspired AI were made in Chapter 4. A novel artificial astrocyte that mimics the multi-time scale neuroplasticity mechanisms of the human brain is proposed. The proposed artificial astrocyte regulates the models' parameters by the firing rates of its neurons. Additionally, the proposed artificial unit is compatible with the current deep learning architectures, adds no parameters, and is active only in the learning phase of the model. This chapter premieres the paradigm of CNGNs by introducing the proposed artificial astrocytes into standard CNNs. Experimental results show that the artificial astrocytes act as a regulatory element of the internal signal of the network and can improve the models' performance, depending on their hyperparameter configuration. All these results aid in filling the gap in the current literature about artificial astrocytes.

Finally, Gen-Soup, a novel hybrid learning approach that integrates all the previous work, is presented in Chapter 5 with promising preliminary results.

This method also utilizes model fusion, which is gaining momentum as a means to combine DL models in a more effective way than ensemble learning does. Ensemble learning is effective in enhancing classification performance but is impractical. Model fusion techniques are an emerging and effective way of combining the attributes of models into a practical single network. The fields of HPO and model fusion appear to be cornerstones that will bridge the sophisticated DL approaches and real-world applications. Gen-Soup is a disruptive new method that combines HPO and model fusion to achieve the simultaneous learning of parameters and hyperparameters. To the best of this author's knowledge, the preliminary results using Gen-Soup outperform all other approaches in the current literature for the classification of images in the medical image dataset KVASIR-v2.

Bibliography

- [1] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PYIM-AGESEARCH, 2017.
- [2] Pooja Balaram and Jon Kaas. Current research on the organization and function of the visual system in primates. *Eye and Brain*, page 1, September 2014.
- [3] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.
- [4] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall Upper Saddle River, N.J., Upper Saddle River, N.J., 2nd ed edition, 2002.
- [5] H. Hu, Z. Zhang, Z. Xie, and S. Lin. Local relation networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3463–3472, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society.
- [6] Wenyi Lin, Kyle Hasenstab, Guilherme Moura Cunha, and Armin Schwartzman. Comparison of handcrafted features and convolutional neural networks for liver mr image adequacy assessment. *Scientific Reports*, 10(1):20336, Nov 2020.
- [7] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts,

- cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, Mar 2021.
- [8] Mengfang Li, Yuanyuan Jiang, Yanzhou Zhang, and Haisheng Zhu. Medical image analysis using deep learning algorithms. *Frontiers in Public Health*, 11, 2023.
 - [9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
 - [10] Warren S. McCulloch and Walter Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*, page 15–27. MIT Press, Cambridge, MA, USA, 1988.
 - [11] Thomas P. Quinn, Stephan Jacobs, Manisha Senadeera, Vuong Le, and Simon Coghlan. The three ghosts of medical ai: Can the black-box present deliver? *Artificial Intelligence in Medicine*, 124:102158, 2022.
 - [12] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.
 - [13] Marvin Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, USA, 1969.
 - [14] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.
 - [15] Qianying Liu, Chaitanya Kaul, Christos Anagnostopoulos, Roderick Murray-Smith, and Fani Deligianni. Optimizing vision transformers for medical image segmentation. *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2022.
 - [16] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *2020 IEEE/CVF Conference*

- on *Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020.
- [17] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.
 - [18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
 - [19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
 - [20] Abhimanyu S. Ahuja. The impact of artificial intelligence in medicine on the future role of the physician. *PeerJ*, 7, 2019.
 - [21] Thomas M. Maddox, John S. Rumsfeld, and Philip R. O. Payne. Questions for Artificial Intelligence in Health Care. *JAMA*, 321(1):31–32, 01 2019.
 - [22] Hiroshi Fujita. Ai-based computer-aided diagnosis (ai-cad): the latest review to read first. *Radiological Physics and Technology*, 13(1):6–19, Mar 2020.
 - [23] C. Krittanawong. The rise of artificial intelligence and the uncertain future for physicians. *European Journal of Internal Medicine*, 48:e13–e14, 2018.
 - [24] D. R. Sarvamangala and Raghavendra V. Kulkarni. Convolutional neural networks in medical image understanding: a survey. *Evolutionary Intelligence*, 15(1):1–22, Mar 2022.
 - [25] Chang Seok Bang, Jae Jun Lee, and Gwang Ho Baik. Artificial intelligence for the prediction of helicobacter pylori infection in endoscopic images: Systematic review and meta-analysis of diagnostic test accuracy. *J Med Internet Res*, 22(9):e21983, Sep 2020.

- [26] Nicolin Hainc, Christian Federau, Bram Stieltjes, Maria Blatow, Andrea Bink, and Christoph Stippich. The bright, artificial intelligence-augmented future of neuroimaging reading. *Frontiers in Neurology*, 8, 2017.
- [27] Edward H. Shortliffe. Artificial intelligence in medicine: Weighing the accomplishments, hype, and promise. *Yearb Med Inform*, 28(01):257–262, Aug 2019.
- [28] Debesh Jha, Sharib Ali, Steven Hicks, Vajira Thambawita, Hanna Borgli, Pia H. Smedsrud, Thomas de Lange, Konstantin Pogorelov, Xiaowei Wang, Philipp Harzig, Minh-Triet Tran, Wenhua Meng, Trung-Hieu Hoang, Danielle Dias, Tobey H. Ko, Taruna Agrawal, Olga Ostroukhova, Zeshan Khan, Muhammad Atif Tahir, Yang Liu, Yuan Chang, Mathias Kirkerød, Dag Johansen, Mathias Lux, Håvard D. Johansen, Michael A. Riegler, and Pål Halvorsen. A comprehensive analysis of classification methods in gastrointestinal endoscopy imaging. *Medical Image Analysis*, 70:102007, 2021.
- [29] Mohamed Abd-ElRahman Abdou. Literature review: efficient deep neural networks techniques for medical image analysis. *Neural Computing and Applications*, 34:5791 – 5812, 2022.
- [30] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, Feb 2017.
- [31] Thomas de Lange, P. Halvorsen, and M. Riegler. Methodology to develop machine learning algorithms to improve performance in gastrointestinal endoscopy. *World Journal of Gastroenterology*, 24:5057 – 5062, 2018.
- [32] Minsik Hong, Jerzy W. Rozenblit, and Allan J. Hamilton. Simulation-based surgical training systems in laparoscopic surgery: a current review. *Virtual Reality*, 25(2):491–510, Jun 2021.
- [33] Zachary C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, June 2018.

- [34] Mauricio Reyes, Raphael Meier, Sérgio Pereira, Carlos A. Silva, Fried-Michael Dahlweid, Hendrik von Tengg-Kobligk, Ronald M. Summers, and Roland Wiest. On the interpretability of artificial intelligence in radiology: Challenges and opportunities. *Radiology: Artificial Intelligence*, 2(3):e190043, 2020. PMID: 32510054.
- [35] Francesco Pinto, Philip H. S. Torr, and Puneet K. Dokania. An impartial take to the cnn vs transformer robustness contest. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 466–480, Cham, 2022. Springer Nature Switzerland.
- [36] José Maurício, Inês Domingues, and Jorge Bernardino. Comparing vision transformers and convolutional neural networks for image classification: A literature review. *Applied Sciences*, 13(9), 2023.
- [37] Kangrui Lu, Yuanrun Xu, and Yige Yang. Comparison of the potential between transformer and cnn in image classification. In *ICMLCA 2021; 2nd International Conference on Machine Learning and Computer Application*, pages 1–6, 2021.
- [38] Muhammad Muzammal Naseer, Kanchana Ranasinghe, Salman H Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23296–23308. Curran Associates, Inc., 2021.
- [39] Yutong Bai, Jieru Mei, Alan L Yuille, and Cihang Xie. Are transformers more robust than cnns? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 26831–26843. Curran Associates, Inc., 2021.
- [40] Luca Deiner, Bernhard Stimpel, Anil Yuce, Samaneh Abbasi-Sureshjani, Simon Schönenberger, Paolo Ocampo, Konstanty Korski, and Fabien Gaire.

A comparative study between vision transformers and cnns in digital pathology, 2022.

- [41] Annarita Fanizzi, Federico Fadda, Maria Colomba Comes, Samantha Bove, Annamaria Catino, Erika Di Benedetto, Angelo Milella, Michele Montrone, Annalisa Nardone, Clara Soranno, Alessandro Rizzo, Deniz Can Guven, Domenico Galetta, and Raffaella Massafra. Comparison between vision transformers and convolutional neural networks to predict non-small lung cancer recurrence. *Scientific Reports*, 13(1):20605, Nov 2023.
- [42] Zhendong Liu, Shuwei Qian, Changhong Xia, and Chongjun Wang. Are transformer-based models more robust than cnn-based models? *Neural Networks*, 172:106091, 2024.
- [43] Alexia Jolicoeur-Martineau, Emy Gervais, Kilian Fatras, Yan Zhang, and Simon Lacoste-Julien. Population parameter averaging (papa), 2023.
- [44] Weishi Li, Yong Peng, Miao Zhang, Liang Ding, Han Hu, and Li Shen. Deep model fusion: A survey, 2023.
- [45] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR, 17–23 Jul 2022.
- [46] Ammar Mohammed and Rania Kora. A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, 35(2):757–774, 2023.
- [47] Manojee Roy and Ujjwala Baruah. Enhancing medical image classification through controlled diversity in ensemble learning. *Engineering Applications of Artificial Intelligence*, 133:108138, 2024.

- [48] Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. An analysis on ensemble learning optimized medical image classification with deep convolutional neural networks. *IEEE Access*, 10:66467–66480, 2022.
- [49] Alexandre Rame, Matthieu Kirchmeyer, Thibaud Rahier, Alain Rakotomamonjy, Patrick Gallinari, and Matthieu Cord. Diverse weight averaging for out-of-distribution generalization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 10821–10836. Curran Associates, Inc., 2022.
- [50] Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 71095–71134. Curran Associates, Inc., 2023.
- [51] Fredric Narcross. Artificial nervous systems; a new paradigm for artificial intelligence. *Patterns*, 2(6), Jun 2021.
- [52] Lucas Antón Pastur-Romay, Francisco Cedrón, Alejandro Pazos, and Ana Belén Porto-Pazos. Deep artificial neural networks and neuromorphic chips for big data analysis: Pharmaceutical and bioinformatics applications. *International Journal of Molecular Sciences*, 17(8), 2016.
- [53] Md Zesun Ahmed Mia, Malyaban Bal, and Abhronil Sengupta. Delving deeper into astromorphic transformers, 2024.
- [54] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12, 2018.
- [55] Amar Shrestha. *Inference And Learning In Spiking Neural Networks For Neuromorphic Systems*. PhD thesis, Syracuse University, 2021.

- [56] Ruthvik Vaila, Chen Hao, Mehrpouyan Hani, and Saeed Reza, Kheradpisheh. *Deep Convolutional Spiking Neural Networks for Image Classification*. PhD thesis, USA, 2021. AAI28320379.
- [57] Spyridon Chavlis and Panayiota Poirazi. Drawing inspiration from biological dendrites to empower artificial neural networks. *Current Opinion in Neurobiology*, 70:1–10, 2021. Computational Neuroscience.
- [58] Toviah Moldwin, Menachem Kalmenson, and Idan Segev. The gradient clusteron: A model neuron that learns to solve classification tasks via dendritic nonlinearities, structural plasticity, and gradient descent. *PLOS Computational Biology*, 17(5):1–29, 05 2021.
- [59] Mariana-Iuliana Georgescu, Radu Tudor Ionescu, Nicolae-Cătălin Ristea, and Nicu Sebe. Nonlinear neurons with human-like apical dendrite activations. *Applied Intelligence*, 53(21):25984–26007, Nov 2023.
- [60] Ana B Porto-Pazos, Noha Veiguela, Pablo Mesejo, Marta Navarrete, Alberto Alvarellos, Oscar Ibáñez, Alejandro Pazos, and Alfonso Araque. Artificial astrocytes improve neural network performance. *PloS one*, 6(4):e19109, 2011.
- [61] Alberto Alvarellos, Alejandro Pazos, and Ana Porto-Pazos. Computational models of neuron-astrocyte interactions lead to improved efficacy in the performance of neural networks. *Computational and mathematical methods in medicine*, 2012:476324, 05 2012.
- [62] Lucas Anton Pastur-Romay, Francisco Cedrón, and Ana B. Porto-Pazos. Artificial astrocytic modulation of neuron’s output. In Hannu Eskola, Outi Väisänen, Jari Viik, and Jari Hyttinen, editors, *EMBECE & NBC 2017*, pages 912–915, Singapore, 2018. Springer Singapore.
- [63] Chihiro Ikuta, Yoko Uwate, and Yoshifumi Nishio. Chaos glial network connected to multi-layer perceptron for solving two-spiral problem. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1360–1363, 2010.

- [64] Peter Gergel' and Igor Farkaš. Echo state networks with artificial astrocytes and hebbian connections. In Ignacio Rojas, Gonzalo Joya, and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 457–466, Cham, 2019. Springer International Publishing.
- [65] Crina Grosan and Ajith Abraham. *Hybrid Intelligent Systems*, pages 423–450. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [66] Eka Miranda, Mediana Aryuni, and E. Irwansyah. A survey of medical image classification techniques. In *2016 International Conference on Information Management and Technology (ICIMTech)*, pages 56–61, 2016.
- [67] Mina Rajabi, Saeed Hossani, and Fatemeh Dehghani. A literature review on current approaches and applications of fuzzy expert systems. *ArXiv*, abs/1909.08794, 2019.
- [68] Haneet Kour, Jatinder Manhas, and Vinod Sharma. Usage and implementation of neuro-fuzzy systems for classification and prediction in the diagnosis of different types of medical disorders: a decade review. *Artificial Intelligence Review*, 53(7):4651–4706, Oct 2020.
- [69] Xing Wu, Cheng Chen, Mingyu Zhong, and Jianjia Wang. Hal: Hybrid active learning for efficient labeling in medical domain. *Neurocomputing*, 456:563–572, 2021.
- [70] D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing*, 28(5):823–870, 2007.
- [71] Bas H.M. van der Velden, Hugo J. Kuijf, Kenneth G.A. Gilhuijs, and Max A. Viergever. Explainable artificial intelligence (XAI) in deep learning-based medical image analysis. *Medical Image Analysis*, 79:102470, jul 2022.
- [72] N. Ketkar and E. Santana. *Deep Learning with Python*. O'Reilly, 2016.
- [73] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 1st edition, 2017.

- [74] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.
- [75] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.
- [76] Benjamin R. Mitchell. *The spatial inductive bias of deep learning*. Phd thesis, Johns Hopkins University, Baltimore, Maryland, March 2017. Available at <http://jhir.library.jhu.edu/handle/1774.2/40864>.
- [77] Yun-Hao Cao and Jianxin Wu. A random cnn sees objects: One inductive bias of cnn and its applications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):194–202, Jun. 2022.
- [78] Stéphane d’Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: improving vision transformers with soft convolutional inductive biases*. *Journal of Statistical Mechanics: Theory and Experiment*, 2022(11):114005, nov 2022.
- [79] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
- [80] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.
- [81] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective, 2019.
- [82] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
- [83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe,

- and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing.
- [84] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.
- [85] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). 2016.
- [86] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.
- [87] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2016.
- [88] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.
- [89] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2016.
- [90] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2014.

- [91] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.
- [92] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018.
- [93] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [94] G. Thimm and E. Fiesler. Neural network initialization. In José Mira and Francisco Sandoval, editors, *From Natural to Artificial Neural Computation*, pages 535–542, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [95] Meenal V. Narkhede, Prashant P. Bartakke, and Mukul S. Sutaone. A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, 55(1):291–322, Jan 2022.
- [96] Simon Kornblith, Honglak Lee, Ting Chen, and Mohammad Norouzi. What’s in a loss function for image classification? *ArXiv*, abs/2010.16402, 2020.
- [97] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *ArXiv*, abs/2008.05756, 2020.
- [98] Luciano M. Prevedello, Safwan S. Halabi, George Shih, Carol C. Wu, Marc D. Kohli, Falgun H. Chokshi, Bradley James Erickson, Jayashree Kalpathy-Cramer, Katherine P. Andriole, and Adam E. Flanders. Challenges related to artificial intelligence research in medical imaging and the importance of image analysis competitions. *Radiology. Artificial intelligence*, 1 1:e180031, 2019.
- [99] Juri Yanase and Evangelos Triantaphyllou. The seven key challenges for the future of computer-aided diagnosis in medicine. *International Journal of Medical Informatics*, 129:413–422, 2019.

- [100] Tyler M. Berzin, Sravanthi Parasa, Michael B. Wallace, Seth A. Gross, Alessandro Repici, and Prateek Sharma. Position statement on priorities for artificial intelligence in gi endoscopy: a report by the asge task force. *Gastrointestinal Endoscopy*, 92(4):951–959, 2020.
- [101] Sravanthi Parasa, Michael Wallace, Ulas Bagci, Mark Antonino, Tyler Berzin, Michael Byrne, Haydar Celik, Keyvan Farahani, Martin Golding, Seth Gross, Vafa Jamali, Paulo Mendonca, Yuichi Mori, Andrew Ninh, Alessandro Repici, Douglas Rex, Kris Skrinak, Shyam J. Thakkar, Jeanin E. van Hooft, John Vargo, Honggang Yu, Ziyue Xu, and Prateek Sharma. Proceedings from the first global artificial intelligence in gastroenterology and endoscopy summit. *Gastrointestinal Endoscopy*, 92(4):938–945.e1, 2020.
- [102] Siyi Li, Yanan Sun, Gary G. Yen, and Mengjie Zhang. Automatic design of convolutional neural network architectures under resource constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3832–3846, 2023.
- [103] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. NIPS’16, page 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc.
- [104] Boyu Zhang, Azadeh Davoodi, and Yu Hen Hu. Chapr: Efficient inference of cnns via channel pruning. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6, 2020.
- [105] Mohammad-Ali Maleki, Alireza Nabipour-Meybodi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. An energy-efficient inference method in convolutional neural networks based on dynamic adjustment of the pruning level. *ACM Transactions on Design Automation of Electronic Systems*, 26:1–20, 08 2021.
- [106] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *ArXiv*, abs/2003.05689, 2020.

- [107] Amala Mary Vincent and P. Jidesh. An improved hyperparameter optimization framework for automl systems using evolutionary algorithms. *Scientific Reports*, 13(1):4737, Mar 2023.
- [108] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.
- [109] Gang Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5, 2016.
- [110] Xueli Xiao, Ming Yan, Sunitha Basodi, Chunyan Ji, and Yi Pan. Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm, 2020.
- [111] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
- [112] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.
- [113] Anthony D. Yao, Derrick L. Cheng, Ian Pan, and Felipe Kitamura. Deep learning in neuroradiology: A systematic review of current algorithms and approaches for the new wave of imaging technology. *Radiology: Artificial Intelligence*, 2(2):e190026, 2020. PMID: 33937816.
- [114] Gursimran S. Kochhar, Neil M. Carleton, and Shyam Thakkar. Assessing perspectives on artificial intelligence applications to gastroenterology. *Gastrointestinal Endoscopy*, 93(4):971–975.e2, 2021.
- [115] Ji-Hoon Han, Dong-Jin Choi, Sang-Uk Park, and Sun-Ki Hong. Hyperparameter optimization using a genetic algorithm considering verification

time in a convolutional neural network. *Journal of Electrical Engineering & Technology*, 15(2):721–726, Mar 2020.

- [116] Chen Li, JinZhe Jiang, YaQian Zhao, RenGang Li, EnDong Wang, Xin Zhang, and Kun Zhao. Genetic algorithm based hyper-parameters optimization for transfer convolutional neural network, 2021.
- [117] Colin White, Mahmoud Safari, Rhea Sanjay Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *ArXiv*, abs/2301.08727, 2023.
- [118] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, page 419–427, New York, NY, USA, 2019. Association for Computing Machinery.
- [119] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.
- [120] Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, MLHPC '15*, New York, NY, USA, 2015. Association for Computing Machinery.
- [121] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3924–3928, 2017.
- [122] Lingxi Xie and Alan Yuille. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1388–1397, 2017.

- [123] Mohammad Hassan Tayarani Najaran. A genetic programming-based convolutional deep learning algorithm for identifying covid-19 cases via x-ray images. *Artificial Intelligence in Medicine*, 142:102571, 2023.
- [124] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [125] Tayyip Ozcan and Alper Basturk. Transfer learning-based convolutional neural networks with heuristic optimization for hand gesture recognition. *Neural Computing and Applications*, 31(12):8955–8970, Dec 2019.
- [126] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [127] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, September 2020.
- [128] Ngoc Tran, Jean-Guy Schneider, Ingo Weber, and A.K. Qin. Hyperparameter optimization in classification: To-do or not-to-do. *Pattern Recognition*, 103:107245, 2020.
- [129] Rogelio García-Aguirre, Luis Torres-Treviño, Eva María Navarro-López, and José Alberto González-González. Automatic generation of optimized convolutional neural networks for medical image classification using a genetic algorithm. 2022.
- [130] Rogelio García-Aguirre, Luis Torres-Treviño, Eva María Navarro-López, and José Alberto González-González. Towards an interpretable model for automatic classification of endoscopy images. In Obdulia Pichardo Lagunas, Juan Martínez-Miranda, and Bella Martínez Seis, editors, *Advances in*

Computational Intelligence, pages 297–307, Cham, 2022. Springer Nature Switzerland.

- [131] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, 2021.
- [132] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 122–138, Cham, 2018. Springer International Publishing.
- [133] Dipam Vasani. This thing called Weight Decay — towardsdatascience.com. <https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab>. [Accessed 25-10-2024].
- [134] Chang-Bin Zhang, Peng-Tao Jiang, Qibin Hou, Yunchao Wei, Qi Han, Zhen Li, and Ming-Ming Cheng. Delving deep into label smoothing. *IEEE Transactions on Image Processing*, 30:5984–5996, 2021.
- [135] Hongxin Wei, Renchunzi Xie, Hao Cheng, Lei Feng, Bo An, and Yixuan Li. Mitigating neural network overconfidence with logit normalization, 2022.
- [136] Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet?, 2020.
- [137] Ramaprasad Poojary and Akul Pai. Comparative study of model optimization techniques in fine-tuned cnn models. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–4, 2019.
- [138] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, Feb 2021.

- [139] Sung-Soon Choi and Byung-Ro Moon. Normalization in genetic algorithms. In Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence David Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell Standish, Graham Kendall, Stewart Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitch A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasha Jonoska, and Julian Miller, editors, *Genetic and Evolutionary Computation — GECCO 2003*, pages 862–873. Springer Berlin Heidelberg, 2003.
- [140] Arda Mavi. A new dataset and proposed convolutional neural network architecture for classification of american sign language digits, 2021.
- [141] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, Michael Riegler, and Pål Halvorsen. Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection. In *Proceedings of the 8th ACM on Multimedia Systems Conference, MMSys'17*, pages 164–169, New York, NY, USA, 2017. ACM.
- [142] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Cristina Carrera, Alicia Barreiro, Allan C. Halpern, Susana Puig, and Josep Malvehy. Bcn20000: Dermoscopic lesions in the wild, 2019.
- [143] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, 5(1):180161, Aug 2018.
- [144] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, and Allan Halpern. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic), 2018.

- [145] Fabio A. Spanhol, Luiz S. Oliveira, Caroline Petitjean, and Laurent Heutte. A dataset for breast cancer histopathological image classification. *IEEE Transactions on Biomedical Engineering*, 63(7):1455–1462, 2016.
- [146] Jessica Escobar, Karen Sanchez, Carlos Hinojosa, Henry Arguello, and Sergio Castillo. Accurate deep learning-based gastrointestinal disease classification via transfer learning strategy. In *2021 XXIII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, pages 1–5, 2021.
- [147] Subhashree Mohapatra, Janmenjoy Nayak, Manohar Mishra, Girish Kumar Pati, Bignaraj Naik, and Tripti Swarnkar. Wavelet transform and deep convolutional neural network-based smart healthcare system for gastrointestinal disease detection. *Interdisciplinary Sciences: Computational Life Sciences*, 13(2):212–228, Jun 2021.
- [148] Hemalatha Gunasekaran, Krishnamoorthi Ramalakshmi, Deepa Kanmani Swaminathan, Andrew J, and Manuel Mazzara. Git-net: An ensemble deep learning-based gi tract classification of endoscopic images. *Bioengineering*, 10(7), 2023.
- [149] Marwa Obayya, Fahd N. Al-Wesabi, Mashael Maashi, Abdullah Mohamed, Manar Ahmed Hamza, Suhanda Drar, Ishfaq Yaseen, and Mohamed Ibrahim Alsaid. Modified salp swarm algorithm with deep learning based gastrointestinal tract disease classification on endoscopic images. *IEEE Access*, 11:25959–25967, 2023.
- [150] Mohamed A. Kassem, Khalid M. Hosny, and Mohamed M. Fouad. Skin lesions classification into eight classes for isic 2019 using deep convolutional neural network and transfer learning. *IEEE Access*, 8:114822–114832, 2020.
- [151] Andre G. C. Pacheco, Abder-Rahman Ali, and Thomas Trappenberg. Skin cancer detection based on deep learning and entropy to detect outlier samples, 2020.
- [152] Imran Iqbal, Muhammad Younus, Khuram Walayat, Mohib Ullah Kakar, and Jinwen Ma. Automated multi-class classification of skin lesions

through deep convolutional neural network with dermoscopic images. *Computerized Medical Imaging and Graphics*, 88:101843, 2021.

- [153] Fekry Olayah, Ebrahim Mohammed Senan, Ibrahim Abdulrab Ahmed, and Bakri Awaji. Ai techniques of dermoscopy image analysis for the early detection of skin lesions based on combined cnn features. *Diagnostics*, 13(7), 2023.
- [154] Muhammad Junaid Umer, Muhammad Sharif, Seifedine Kadry, and Abdullah Alharbi. Multi-class classification of breast cancer using 6b-net with deep feature fusion and selection method. *Journal of Personalized Medicine*, 12(5), 2022.
- [155] Sudhakar Tummala, Jungeun Kim, and Seifedine Kadry. Breast-net: Multi-class classification of breast cancer from histopathological images using ensemble of swin transformers. *Mathematics*, 10(21), 2022.
- [156] David Clement, Emmanuel Agu, Muhammad A. Suleiman, John Obayemi, Steve Adeshina, and Wole Soboyejo. Multi-class breast cancer histopathological image classification using multi-scale pooled image feature representation (mpifr) and one-versus-one support vector machines. *Applied Sciences*, 13(1), 2023.
- [157] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data, 2019.
- [158] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016.
- [159] S.H. Shabbeer Basha, Sravan Kumar Vinakota, Viswanath Pulabaigari, Snehasis Mukherjee, and Shiv Ram Dubey. Autotune: Automatically tuning convolutional neural networks for improved transfer learning. *Neural Networks*, 133:112–122, 2021.

- [160] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5):1285–1298, 2016.
- [161] Rune Johan Borgli, Håkon Kvale Stensland, Michael Alexander Riegler, and Pål Halvorsen. Automatic hyperparameter optimization for transfer learning on medical image datasets using bayesian optimization. In *2019 13th International Symposium on Medical Information and Communication Technology (ISMICT)*, pages 1–6, 2019.
- [162] David Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current biology : CB*, 24:R921–R929, 09 2014.
- [163] Andrea Volterra and Jacopo Meldolesi. Astrocytes, from brain glue to communication elements: the revolution continues. *Nature Reviews Neuroscience*, 6(8):626–640, Aug 2005.
- [164] Cagla Eroglu and Ben A. Barres. Regulation of synaptic connectivity by glia. *Nature*, 468(7321):223–231, Nov 2010.
- [165] John J. Wade, Liam J. McDaid, Jim Harkin, Vincenzo Crunelli, and J. A. Scott Kelso. Bidirectional coupling between astrocytes and neurons mediates learning and dynamic coordination in the brain: A multiple modeling approach. *PLOS ONE*, 6(12):1–24, 12 2011.
- [166] Peter Gergel’ and Igor Farkaš. Investigating the role of astrocyte units in a feedforward neural network. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 73–83, Cham, 2018. Springer International Publishing.
- [167] Tiina Manninen, Jugoslava Aćimović, and Marja-Leena Linne. Analysis of network models with neuron-astrocyte interactions. *Neuroinformatics*, 21(2):375–406, Apr 2023.

- [168] Chihiro Ikuta, Yoko Uwate, and Yoshifumi Nishio. Performance and features of multi-layer perceptron with impulse glial network. In *The 2011 International Joint Conference on Neural Networks*, pages 2536–2541, 2011.
- [169] Chihiro Ikuta, Yoko Uwate, and Yoshifumi Nishio. Multi-layer perceptron with positive and negative pulse glial chain for solving two-spirals problem. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2012.
- [170] Chihiro Ikuta, Yoko Uwate, Yoshifumi Nishio, and Guoan Yang. Multi-layer perceptron with pulse glial chain. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E99.A(3):742–755, 2016.
- [171] Chihiro Ikuta and Isao Goto. Convolutional neural network with glial dropout for solving fashion-mnist. *IEICE Proceeding Series*, 74:247–250, 11 2020.
- [172] Ilya A. Zimin, Victor B. Kazantsev, and Sergey V. Stasenko. Artificial neural network model with astrocyte-driven short-term memory. *Biomimetics*, 8(5), 2023.
- [173] M. Han, L. Pan, and X. Liu. Astronet: When astrocyte meets artificial neural network. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20258–20268, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society.
- [174] Leo Kozachkov, Ksenia V. Kastanenka, and Dmitry Krotov. Building transformers from neurons and astrocytes. *Proceedings of the National Academy of Sciences*, 120(34):e2219150120, 2023.
- [175] Chihiro Ikuta. Multi-layer perceptron with impulse glial network. 2010.
- [176] Chihiro Ikuta, Yoko Uwate, Yoshifumi Nishio, and Guoan Yang. Multi-layer perceptron including glial pulse and switching between learning and non-learning. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2107–2110, 2013.

- [177] Chihiro Ikuta, Yoko Uwate, and Yoshifumi Nishio. Investigation of four-layer multi-layer perceptron with glia connections of hidden-layer neurons. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2013.
- [178] Hajer Landolsi and Kirmene Marzouki. Self organizing neuro-glial network, song-net. In *2014 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*, pages 85–91, 2014.
- [179] Kirmene Marzouki. Neuro-glial interaction: Song-net. In Sabri Arik, Tingwen Huang, Weng Kin Lai, and Qingshan Liu, editors, *Neural Information Processing*, pages 619–626, Cham, 2015. Springer International Publishing.
- [180] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Neural Information Processing Systems*, 2018.
- [181] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.
- [182] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6078–6087, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [183] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3519–3529. PMLR, 09–15 Jun 2019.
- [184] Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. Grounding representation similarity with statistical testing, 2024.

- [185] Ari S. Morcos, David G. T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization, 2018.
- [186] Matthew L. Leavitt and Ari Morcos. Selectivity considered harmful: evaluating the causal impact of class selectivity in dnns, 2020.
- [187] Alexandre Ramé, Kartik Ahuja, Jianyu Zhang, Matthieu Cord, Léon Bottou, and David Lopez-Paz. Model ratatouille: recycling diverse models for out-of-distribution generalization. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org, 2023.
- [188] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mul-lainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.
- [189] II Price, W. Nicholson, Sara Gerke, and I. Glenn Cohen. Potential Liability for Physicians Using Artificial Intelligence. *JAMA*, 322(18):1765–1766, 11 2019.

List of Figures

2.1	Representation of a three-dimensional matrix of depth C , height H , and width W	40
2.2	Example of the convolution operation $(*)$ using two matrices. $X * \mathbf{K} = Y$ (X in white convolves with \mathbf{K} in blue and produces Y in yellow).	41
2.3	Trainable parameters of a convolutional layer with l_W^ϕ kernels of height \mathbf{K}_H and width \mathbf{K}_W and takes an input with X_C channels. In the figure, the superscript denotes the kernel number, and the subscripts index over the kernel's depth, height, and width, respectively. In practice, every neuron has only one associated bias. Hence, all the elements in a bias's two-dimensional matrix are the same.	43
2.4	Schematic of the information flow in a standard ANN.	46
2.5	Example of the full-convolution operation $(*)^{full}$ using two matrices. $X *^{full} K = Y$ (X in white fully-convolves with K in blue and produces Y in yellow).	53
2.6	Example of the composition of an arbitrary FC layer l_ϕ^{FC} . A layer can have different operations and in different order.	54
2.7	ReLU, TanH, and Sigmoid activation functions.	56
2.8	Example of the composition of a basic conv layer.	59
2.9	Example of commonly used pooling functions. The pooling operation operates channel-wise and slides with an arbitrary stride over the input channels. The size of the pooling receptive field (width and height of the striding window) is also arbitrary.	59
2.10	Example of the two standard types of flattening operation in a CNN.	61

2.11	Example of the dropout operation in an ANN during three training iterations.	63
2.12	Low-dimensionality example of loss landscape of a model with only two parameters θ_1 and θ_2 . The valleys are local minima. Gradient-descent algorithms lead to local minimum points in the landscape. The depicted function is the Schwefel function and is shown only for illustrative purposes.	67
2.13	GELU, SeLU, and h-swich activation functions.	72
3.1	Diagram of Gen-CNN, the developed framework for the automatic generation of CNNs using a GA for HPO.	84
3.2	The individuals are composed of 5 genes $\{Ar, LR, WD, LF, Op\} \in \{0,1\}^5$, that stand for the hyperparameters of CNN architecture, learning rate scaling factor, weight decay factor, loss function, and optimizer, respectively. The genes' values are transformed into the hyperparameters' values as shown for each gene.	93
3.3	Illustration of a population of \mathcal{N} individuals where every individual is a different hyperparameter combination for a CNN model. . .	94
3.4	Example images of every class in the Sign Language Digits dataset [140].	98
3.5	Example images of every class in the KVASIR-v2 dataset [141]. . . .	99
3.6	Example images of every class in the ISIC-2019 dataset [142–144]. .	100
3.7	Example images of every class in the BreakHis dataset [145] (ignoring their magnifying factors).	100
3.8	Confusion matrix in the test partition of the KVASIR-v2 dataset of the resulting model using a single run of Gen-CNN.	105
3.9	Confusion matrix in the test partition of the ISIC-2019 dataset of the resulting model using a single run of Gen-CNN.	106
3.10	Confusion matrix in the test partition of the BreakHis dataset of the resulting model using a single run of Gen-CNN.	106

4.1	Classification performance of the three tested architectures after training under the conditions described in Table 4.1 (Exp 1) using CIFAR-10. Mean validation CAT-ACC for three repetitions per hyperparameter configuration in the grid. [Best viewed in color]	124
4.2	Classification performance of the three tested architectures after training under the conditions described in Table 4.1 (Exp 1) using CIFAR-100. Mean validation CAT-ACC for three repetitions per hyperparameter configuration in the grid. [Best viewed in color] . .	125
4.3	Mean test accuracy of the CNGNs during training with the optimal modulation factors in the four tested datasets.	132
4.4	Similarity score (mean of the three training repetitions of Exp 3) between the activations at the last convolutional layer of the CNNs and CNGNs during training.[Best viewed in color]	134
5.1	Encoding scheme of the individuals into 7 real-valued genes, each with magnitude of one, $\{Ar, LR, WD, LF, Op, S, W\} \in \{0, 1\}^7$, that stand for the hyperparameters of CNN architecture, learning rate scaling factor, weight decay factor, loss function, optimizer, artificial astrocytes' Str factor, and artificial astrocytes' Weak factor, respectively. The genes' values are transformed into the hyperparameters' values as shown for each gene.	141
5.2	Validation MCC of the best-evaluated individual in each generation of Gen-Soup in Exp 1.	149
5.3	Validation MCC of the best-evaluated individual in each generation of Gen-Soup in Exp 2.	150

List of Tables

3.1	CNN architectures in the search space for chapter 3.	89
3.2	Number of samples in every dataset and partition for the experiments.	101
3.3	Performance comparison of the proposed approach to other HPO algorithms using the Sign Language Digits dataset [140]. Mean and standard deviation of the resulting models' Cat-ACC after 30 repetitions of the optimization algorithms. All the algorithms used the same number of evaluations during the optimization cycle.	103
3.4	Fixed Gen-CNN's hyperparameters and final training conditions for the experiments with the KVASIR-v2, ISIC-2019, and BreakHis datasets. The crossover and mutation probabilities are normalized, and the values are for the genes of CNN architecture, loss function, optimizer, learning rate, and weight decay, respectively.	104
3.5	Optimal hyperparameters found using a single run of Gen-CNN for the three medical image datasets.	105
3.6	Comparison of the test classification performance metrics on the KVASIR-v2 dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.	107
3.7	Comparison of the test classification performance metrics on the ISIC-2019 dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.	108
3.8	Comparison of the test classification performance metrics on the BreakHis dataset of the proposed approach and the current best-evaluated models in the literature for this dataset.	109

4.1	Hyperparameters for the experiments of artificial astrocytes.	123
4.2	Classification CAT-ACC of the CNNs and CNGNs with the highest validation CAT-ACC during the grid-search optimization. Table 4.1 (Exp 2) shows the training conditions. Δ is the percentage change in test accuracy of the CNGNs with respect to the CNNs in the same dataset and training conditions. The highest test accuracy between the CNN and CNGN is in bold case.	128
4.3	Optimal Str & Weak per epoch found using grid-search (Exp 2) on the CIFAR-10 dataset.	129
4.4	Optimal Str & Weak per epoch found using grid-search (Exp 2) on the CIFAR-100 dataset.	129
4.5	Optimal Str & Weak per epoch found using grid-search (Exp 2) on the KVASIR-v2 dataset.	130
4.6	Optimal Str & Weak per epoch found using grid-search (Exp 2) on the BreakHis-v1 dataset.	130
4.7	Mean test classification CAT-ACC of three training repetitions using the optimal hyperparameter configurations found with the grid search. Table 4.1 (Exp 3) shows the training conditions. Δ is the mean percentage change in test accuracy of the CNGNs with respect to the CNNs in the same dataset and training conditions. The highest mean test accuracy between the CNN and CNGN is in bold case.	131
4.8	Mean test F1 and MCC of three training repetitions using the optimal hyperparameter configurations found with the grid search. Table 4.1 (Exp 3) shows the training conditions. Δ is the mean percentage change in test MCC of the CNGNs with respect to the CNNs in the same dataset and training conditions. The mean highest test F1 and MCC between the CNN and CNGN is in bold case.	133

5.1 Hyperparameters of Gen-Soup for the experiments. The crossover and mutation probabilities appear in order for the genes of CNN architecture, learning rate scaling factor, weight decay factor, loss function, optimizer, artificial astrocytes' Str factor, and artificial astrocytes' Weak factor, respectively. 146

5.2 Search space of the hyperparameters optimized by Gen-Soup. . . . 146

5.3 Classification metrics of resulting models in Exp 1. 149

5.4 Classification metrics of resulting models in Exp 2. 150