NOVA IMS Information Management School

DOCTORAL PROGRAMME

Information Management

Specialization in Decision Support Systems

Deep Learning Techniques for Medical Image Classification

Ibrahem Hamdy Abdelhamid Kandel

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor in Information Management

May 2021

NOVA Information Management School
Universidade Nova de Lisboa

Deep Learning Techniques for Medical Image Classification

Supervised by:

Professor Doctor Mauro Castelli
NOVA Information Management School
Universidade Nova de Lisboa
Lisbon, Portugal

2021

Abstract

In recent years, artificial intelligence (AI) has been applied in many fields to address complex and critical real-world tasks. Deep learning rises as a subfield of AI, where artificial neural networks (ANN) are used to map complicated functions, which can be challenging even for experienced users. One of the ANN variants is called convolutional neural network (CNN), which has shown great potential in image processing by providing state-of-the-art results for many significant image processing challenges. The medical field can significantly benefit from AI usage, especially in the medical image classification domain. In this doctoral dissertation, we applied different AI techniques to analyze medical images and to give the physicians a second opinion or reduce the time and effort needed for the image classification. Initially, we reviewed several studies that were published to discuss the transfer learning of CNNs. Afterward, we studied different hyperparameters that need to be optimized for CNNs to be trained accurately. Lastly, we proposed a novel CNN architecture to help in the classification of histopathology images.

Keywords

Image classification; Convolutional neural networks; Transfer learning; Deep learning; Medical images.

Original Contributions

Published articles:

- Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review

Kandel, I.; Castelli, M. Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review. *Applied Sciences* 2020, *10*, 2021.
https://doi.org/10.3390/app10062021

- Musculoskeletal Images Classification for Detection of Fractures Using Transfer Learning

Kandel I, Castelli M, Popovič A. Musculoskeletal Images Classification for Detection of Fractures Using Transfer Learning. *Journal of Imaging*. 2020; 6(11):127.
https://doi.org/10.3390/jimaging6110127

- A Novel Architecture to Classify Histopathology Images Using Convolutional Neural Networks

Kandel, I.; Castelli, M. A Novel Architecture to Classify Histopathology Images Using Convolutional Neural Networks. *Applied Sciences* 2020, *10*, 2929.
https://doi.org/10.3390/app10082929

- The Effect of Batch Size on the Generalizability of the Convolutional Neural Networks on a Histopathology Dataset

I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," ICT Express, 2020.
https://doi.org/10.1016/j.icte.2020.04.010

- How Deeply to Fine-Tune a Convolutional Neural Network: A Case Study Using a Histopathology Dataset

Kandel, I.; Castelli, M. How Deeply to Fine-Tune a Convolutional Neural Network: A Case Study Using a Histopathology Dataset. *Applied Sciences* 2020, *10*, 3359.
https://doi.org/10.3390/app10103359

- Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images

Kandel I, Castelli M, Popovič A. Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *Journal of Imaging*. 2020; 6(9):92.
https://doi.org/10.3390/jimaging6090092

Acknowledgments

ALLAH. Praise is to ALLAH, the Almighty, the creator of everything. Thank you, ALLAH, for the guidance, the help, and the mercy. Alhamdu-Lillah.

To my mother, who taught me how to hold the pen. To my father, who gave me the pen. To my sisters, who encouraged me to use the pen. To my teachers and professors who gave me materials to make use of the pen. To my managers, who taught me how to make a living using the pen.

To my supervisor, prof. Mauro, thank you so much for your help, guidance, support, confidence, and encouragement. This thesis would not be completed without you.

To my wife Amal, who encouraged me throughout the whole period of this Ph.D. Thank you so much, my hope, my only hope.

It was hard; it was challenging; it was inspiring.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Networks
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
LeakyReLU	Leaky Rectified linear unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
DR	Diabetic Retinopathy

Chapter 1 - Introduction

1.1. Research Background and Motivation

Since 1895, with the invention of the X-ray cathode tube by Wilhelm Rontgen, medical images play a crucial role in inpatient treatment. However, the shortage of workforce, the time required to reach a decision, and the need for a second opinion are factors that significantly impact the process. In 1956, artificial intelligence was formally introduced, and since then, many attempts were made to make use of artificial intelligence in the medical field. One of the earliest attempts to incorporate AI in the medical field was proposed in the late 1960s (Clancey & Shortliffe, 1984). One of the first devices to use AI was proposed in the 1990s (Lindsay et al., 1993).

A recent breakthrough in the artificial intelligence field is machine learning. In machine learning, an algorithm can be developed to extract image features automatically. When the algorithm used is a neural network with more than one hidden layer, it is referred to as deep learning. Deep learning can be implemented in the image classification domain. In particular, a feed-forward convolutional neural network (CNN) can be used to classify images automatically. Making the CNN able to classify images is called the training phase; in the training phase, CNN's weights are adjusted to suit the image dataset under study. The main point of CNN is that it can map important features of images that can be used to classify those images without CNN being explicitly programmed to do so.

Image classification, which is defined as grouping images into successive predefined labels, plays a vital role in many areas, like the medical field. Deep learning algorithms can detect important features of images without any manual feature engineering, which can be thought of as using autonomous algorithms that can learn by themselves how to differentiate between distinct image classes. One of the earliest attempts to construct an automatic image classifier that could learn how to distinguish between classes using CNN was introduced by LeCun (1989), who was inspired by the work of Fukushima et al. (1980) and Hubel and Wiesel (1977) and was named convolutional neural networks (CNN). Still, this attempt was limited because of the size of datasets and the computational power available at that time. In 2012, Krizhevsky et al. (2012) introduced their CNN architecture named AlexNet and won first place in the ILSVRC competition, with an error rate of 16% compared to the second-place winner's 25%. Since then, CNN has become a state-of-the-art image classifier.

Training the deep network weights from scratch requires a substantial amount of time and massive datasets (hundreds of thousands of images). These requirements make deep learning algorithms very challenging in medical images where, typically, only a limited number of images are available. A lot of time and experience are required to annotate medical images. To get over the dataset's size, many studies were introduced to help solve this problem. The primary technique used is transfer learning (Shin et al., 2016; Kandel & Castelli, 2020).

In the context of deep learning, transfer learning is a technique that exploits the usage of features that were learned by a network over a given problem to solve a different challenge in the same domain.

Transfer learning has many advantages. First, it saves computational time because it uses the already available information from the last training process instead of training a new model from scratch. Second, it extends the knowledge it acquired from previous models, and third, transfer learning is beneficial when the size of the new training dataset is small. Transfer learning promises valuable contributions to the fields of computer vision, audio classification, and natural language processing. There have been many attempts to automatize the image classification task to facilitate the process of image classification or make it more accurate.

1.2. Research Goals

This thesis aims to investigate the usage of deep learning techniques in the medical field, specifically in medical image classification. The implementation of deep learning in the medical domain can help the physicians:

- Either in giving a second opinion or giving assistance to the physicians in classifying medical images.
- To facilitate the process of image classification and to make it more accurate.

1.3. Research Contributions

To achieve the research goals, we considered three different approaches for two medical fields. The first approach was to review the latest papers published to investigate the usage of transfer learning techniques for DR. The second approach was to investigate the role of four essential hyperparameters that need to be tuned to train a CNN correctly. The third approach was to introduce a novel CNN architecture to classify the histopathology dataset. Our contributions are:

1.3.1. Reviewing transfer learning techniques for diabetic retinopathy image classification.

The third chapter of this thesis reviews research papers that focus on DR classification by using transfer learning to present the best existing methods to address this problem. This review can help future researchers to find out existing transfer learning methods to address the DR classification task and to show their differences in terms of performance.

1.3.2. Studying the effect of using transfer learning.

In the fourth chapter of this thesis, we investigated the effect of using transfer learning compared to training the CNN from scratch. We used a large X-Ray images dataset for our experiments.

1.3.3. Proposing a new CNN model to classify histopathology images.

In the fifth chapter of this thesis, we propose a novel CNN architecture to classify histopathology images. The proposed model consists of 15 convolution layers and two fully connected layers. A comparison between different activation functions was performed to detect the most efficient one, considering two different optimizers.

1.3.4. Studying the effect of batch size on CNN performance.

In the sixth chapter of this thesis, the effect of batch size on the performance of convolutional neural networks and the impact of learning rates were studied for image classification.

1.3.5. Studying the effect of fine-tuning each block to determine the optimum depth.

In the seventh chapter of this thesis, we studied the effect of block-wise fine-tuning of CNNs on a histopathology dataset to determine how deep to fine-tune a CNN to obtain the best results.

1.3.6. The impact of different optimizers on the performance of CNNs.

In the eighth chapter of this thesis, we assessed the performance of six different optimizers with three different learning rates. We used a large histopathology dataset to conduct our experiments.

Chapter 2 - Image Classification Using Convolutional Neural Networks

2.1. Convolution Neural Networks

After CNN's success in the ILSVRC challenge, it becomes the de-facto algorithm for image classification. The difference between CNN and any other neural network is the presence of the convolution layer. The importance of the convolution layer is that it decreases the number of connections required. Also, it takes into consideration the spatial and temporal information of images. The convolution layer works by applying a window size called kernel that convolves the image to detect essential features that can be used to classify the image. Equation (1) shows the convolution operation for colored images:

$$O[i,j] = F(u,v) * I(i,j) = \sum_u \sum_v \sum_{c\in\{R,G,B\}} F_c(u,v) \odot I_c(i+u, j+v) \qquad (1)$$

where $I(.)$ is the input image, c is the color channels, $F(.)$ is the kernel, and $O(i,j)$ is the output pixel in the (i,j) position.

A CNN consists of two main types of layers: primary layers and secondary layers, where the primary layers are the mandatory layers that constitute the CNN, like Convolution layers, pooling layers, and fully connected layers. The secondary layers are the layers used to increase CNN's performance, like dropout layers and normalization layers.

2.2. Activation Functions

Activation functions are essential for neural networks because they will make the network learn nonlinear relationships. The earliest activation functions used were sigmoid and Tanh functions. These saturated activation functions were prevalent due to their computational simplicity, and they indeed achieved outstanding results. Nevertheless, there was a need for non-saturated activation functions that can make the neural networks more robust. According to Xu et al. (B. Xu, Wang, Chen, & Li, 2015), the non-saturated activation functions can help make the neural networks more robust by reducing the vanishing gradient problem that is happening to feed-forward neural networks because of the use of the backpropagation. Nair et al. (Nair & E. Hinton, 2010) introduced one of the first non-saturated activation functions, called the ReLU function. Since then, many non-saturated activation functions have been introduced. Below is a brief description of the different activation functions used in this paper. Different activation functions and their derivatives are shown in Figure 1.

2.2.1. Sigmoid Activation Function

One of the earliest activation functions used. As shown in equation (3), the sigmoid function's output ranges between $[0,1]$. The sigmoid function is considered a saturated function because of its bounds. The equation of the sigmoid function is shown in equation (2). The gradient of the sigmoid function that is used in backpropagation is shown in equation (4).

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

$$0 \leq sigmoid(x) \leq 1 \quad (3)$$

$$\frac{d(sigmoid(x))}{dx} = sigmoid(x)(1 - sigmoid(x)) = \frac{e^{-x}}{(1 + e^{-x})^{-2}} \quad (4)$$

where x is the input to the activation function.

2.2.2. Tanh Activation Function

Tanh functions are commonly used in neural networks. As shown in equation (6), the Tanh activation function ranges between [-1,+1]. The equation of the Tanh function is shown in equation (5). The gradient of the Tanh function that is used in backpropagation is shown in equation (7).

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (5)$$

$$-1 \leq Tanh(x) \leq 1 \quad (6)$$

$$\frac{d(tanh\,(x))}{dx} = 1 - Tanh^{2}(x) = 1 - \frac{(e^{x} - e^{-x})^{2}}{(e^{x} + e^{-x})^{2}} \quad (7)$$

2.2.3. ReLU Activation Function

Rectified linear unit (ReLU) was introduced by Nair et al. (2010) to make the neural networks more robust by learning highly complicated nonlinear functions. It saturates for negative values. ReLU form is shown in equation (8), and its gradient is shown in equation (10). Equation (9) shows the ReLU range.

$$ReLU(x) = \begin{cases} x \; if \; x > 0 \\ 0 \; if \; x < 0 \end{cases} \quad (8)$$

$$0 \leq ReLU(x) \leq \infty \quad (9)$$

$$\frac{d(ReLU(x))}{dx} = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x < 0 \end{cases} \quad (10)$$

2.2.4. LeakyReLU Activation Function

Leaky Rectified Linear Unit (LeakyReLU) was introduced by Maas et al. (Maas, Hannun, & Ng, 2013) to add a new hyperparameter α to the ReLU activation function which will allow a small negative slope. The point of adding the negative slope is to fix the dying neuron problem caused by using the ReLU activation function (B. Xu et al., 2015). The default α value is 0.01. LeakyReLU form is shown in equation (11), and its gradient is shown in equation (13). Equation (12) shows the LeakyReLU range.

$$LeakyReLU(x) = \begin{cases} x \; if \; x > 0 \\ \alpha x \; if \; x < 0 \end{cases} \quad (11)$$

$$-\infty \leq LeakyReLU(x) \leq \infty \quad (12)$$

$$\frac{d\big(LeakyReLU(x)\big)}{dx} = \begin{cases} 1 \; if \; x > 0 \\ \alpha \; if \; x < 0 \end{cases} \quad (13)$$

2.2.5. ELU Activation Function

The exponential linear unit was introduced by Clevert et al. (2016). It is similar to the ReLU activation function on the positive side but different on the negative side, which is not constant as LeakyReLU but is instead a logarithmic curve. The α hyperparameter controls the network saturation on the negative side. The high negative values introduced by the ELU activation function can help in normalizing the activations. The default α value is 1. ELU form is shown in equation (14), and its gradient is shown in equation (16). Equation (15) shows the LeakyReLU range.

$$elu(x) = \begin{cases} x \; if \; x > 0 \\ \alpha(e^x - 1) \; if \; x \leq 0 \end{cases} \quad (14)$$

$$-\alpha \leq elu(x) \leq \infty \quad (15)$$

$$\frac{d(elu(x))}{dx} = \begin{cases} 1 \; if \; x > 0 \\ \alpha(e^x - 1) + \alpha \; if \; x \leq 0 \end{cases} \quad (16)$$

2.2.6. SELU Activation Function

Scaled exponential linear units were introduced by Klambauer et al. (2017). The SELU is a modified version of the ELU activation function, where a new constant λ is introduced. SELU normalizes both the positive and negative sides. As being proposed by Klambauer et al. (2017), the default values are $\alpha = 1.67326$ and $\lambda = 1.0507$. SELU form is shown in equation (17) and its gradient is shown in equation (19). Equation (18) shows the SELU range.

$$SELU(x) = \begin{cases} \lambda x \; if \; x \geq 0 \\ \lambda\alpha(e^x - 1) \; if \; x < 0 \end{cases} \quad (17)$$

$$-\lambda\alpha \leq SELU(x) \leq \infty \quad (18)$$

$$\frac{d(SELU(x))}{dx} = \begin{cases} \lambda \; if \; x \geq 0 \\ \lambda\alpha(e^x - 1) + \lambda\alpha \; if \; x < 0 \end{cases} \quad (19)$$

2.3. Optimization

Given a $Dataset = \{(x_i, y_i)\}_{i=1}^{N}$, where (x_i, y_i) are set of pairs, x_i is the $i - th$ example, and y_i is its label, and with a finite cardinality N. The optimization process is to find the optimal model parameters θ that will yield the lowest error possible for the loss function. The loss function L can be formally defined as equation (20):

$$L(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} l_i(f(x_i; \theta), y_i) \quad (20)$$

In binary classification problems, the loss function used is the binary cross-entropy, which can be formally defined as equation (21):

$$L_{BCE}(\hat{y}_i, y_i) = -\frac{1}{N} \sum\nolimits_{i=1}^{N} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \quad (21)$$

where $\hat{y}_i$ is the predicted label.

An optimization algorithm is needed to achieve the minimum of the loss function. A generic optimization algorithm is shown in equation (22):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta) \qquad (22)$$

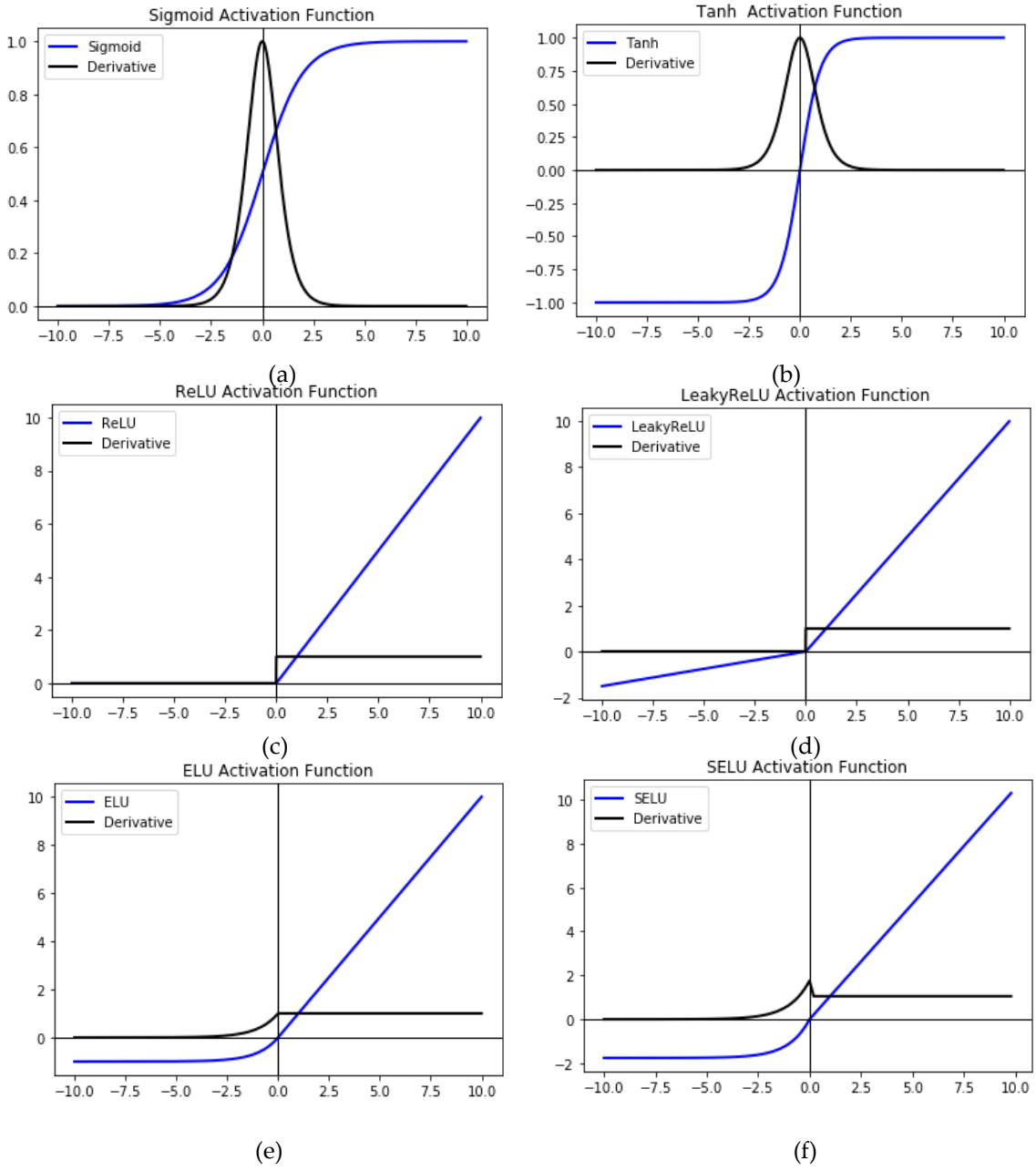


Figure 1. A plot of the activation functions and their derivatives: (a) Sigmoid activation function; (b) Tanh activation function; (c) ReLU activation function; (d) LeakyReLU activation function; (e) ELU activation functions; (f) SELU activation function.

2.3.1. RMSProp Optimizer

RMSProp optimizer was introduced by Hinton (Hinton, 2012) to address the problem of the monotonically decreasing learning rate of the Adagrad optimizer. The network parameters are updated based on the following equations:

$$G = \nabla_\theta L(\theta) \qquad (23)$$

$$E[G^2]_t = \lambda E[G^2]_{t-1} + (1-\lambda) G_t^2 \qquad (24)$$

$$\theta_t^i = \theta_{t-1}^i - \frac{\eta}{\sqrt{E[G^2]_t + \epsilon}} \cdot \nabla_\theta L(\theta_t^i) \qquad (25)$$

where λ is used to select the amount of information needed from the previous update. $E[G^2]_t$ The running average of the squared gradients has been used to avoid the monotonically decreasing gradients of the AdaGrad optimizer. Furthermore, η is the learning rate.

2.3.2. Adam Optimizer

Adam optimizer was introduced by Kingma et al. (2014) to combine the benefits of using both the momentum method and the RMSProp method. The network parameters are updated based on the following equations:

$$\theta_t^i = \theta_{t-1}^i - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t \qquad (26)$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad (27)$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad (28)$$

$$m_t = \beta_1 m_{t-1} + (1+\beta_1) G \qquad (29)$$

$$v_t = \beta_2 v_{t-1} + (1+\beta_2)[G]^2 \qquad (30)$$

$$G = \nabla_\theta L(\theta) \qquad (31)$$

Where η is the learning rate and the hyperparameter β_i is used to select the amount of information needed from the previous update, where $\beta_i \in [0,1]$. m_t is the first moment and v_t is the second moment. The authors noticed that by initializing m_t and v_t from zero, will lead to a zero-biased problem, and so the authors suggested using the corrected moments: $\hat{m}_t$ and $\hat{v}_t$.

Chapter 3. Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review[1]

Abstract: Diabetic retinopathy (DR) is a dangerous eye condition that affects diabetic patients. Without early detection, it can affect the retina and may eventually cause permanent blindness. The initial diagnosis of DR is crucial for its treatment. However, the determination of DR is a challenging process that requires an experienced ophthalmologist. A breakthrough in artificial intelligence called deep learning can give the ophthalmologist a second opinion regarding DR's classification using an autonomous classifier. To accurately train a deep learning model to classify DR, an enormous number of images are required, which is a significant limitation in the DR domain. Transfer learning is a technique that can help in overcoming the scarcity of images. The main idea exploited by transfer learning is that a deep learning architecture, previously trained on non-medical images, can be fine-tuned to suit the DR dataset. This paper reviews research papers that focus on DR classification by using transfer learning to present the best existing methods to address this problem. This review can help future researchers find out existing transfer learning methods to address the DR classification task and show their performance differences.

Keywords: Diabetic retinopathy; deep learning; convolutional neural networks; transfer learning

---

3.1. Introduction

Diabetes mellitus (DM) is a chronic, metabolic, clinically heterogeneous disorder in which prevalence has been increasing steadily worldwide (Chen et al., 2012). It is estimated that 366 million people had DM in 2011; by 2030, this will have risen to 552 million (Cho et al., 2018). DM is characterized by persistent hyperglycemia, which may be due to impaired insulin secretion, resistance to insulin's peripheral actions, or both, which eventually leads to pancreatic beta-cell failure (Okur et al., 2017). People living with DM are more vulnerable to various forms of both short- and long-term complications due to metabolic aberrations that can cause damage to various organ systems, leading to the development of disabling and life-threatening health complications, the most prominent of which are microvascular (retinopathy, nephropathy, and neuropathy) and macrovascular complications (Lotfy et al., 2015).

Diabetic retinopathy (DR) is one of the most common microvascular complications caused by DM. It happens when the retina's blood vessels are affected by high blood levels (Gupta & Chhikara, 2018). DR can create some irreversible complications that can lead to blindness in many cases. The number of patients that suffer from DR was estimated at 126.6 million in 2010, and this number is expected to grow to 191 million by 2030 (Zheng et al., 2012). More than 2.6% of blindness worldwide happens because of DR (Bourne et al., 2013). This percentage corresponds to a significant number of persons whose quality of life is severely affected. Though the early diagnosis of DR can help prevent blindness (Vashist et al., 2011), this is challenging. More in detail, the main challenge of early-

[1] This chapter has been published in MDPI journal as Kandel, I.; Castelli, M. Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review. *Applied Sciences* 2020, *10*, 2021. https://doi.org/10.3390/app10062021

detected DR is the workforce that is needed to examine the retina images to detect DR (Abramoff et al., 2010) because diabetic patients must be assessed by an ophthalmologist at least once a year to detect the early signs of DR. Therefore, a reliable detection technology is needed to assist health care personnel in analyzing DR. According to Wilkinson et al. (Wilkinson et al., 2003), DR can be classified into five grades: grade 0 is normal with no sign of DR, grade 1 means the presence of mild DR, grade 2 means moderate, grade 3 means severe, and, finally, grade 4 is defined by new vessel proliferation, where risks of vision loss include bleeding into the vitreous and tractional retinal detachment. Figure 2 shows the different grades of DR.

Figure 2. Random samples of different grades of diabetic retinopathy. (a) grade 0, (b) grade 1, (c) grade 2, (d) grade 3, and (e) grade 4.

Deep learning belongs to the broad family of machine learning methods (Mitchell, 1997). Unlike traditional neural networks-based classifiers, deep learning builds classifiers with many hidden layers, aiming to identify the salient low-level features of an image (Goodfellow et al., 2016). In the context of deep learning, transfer learning is a technique that exploits the usage of features that were learned by a network over a given problem to solve a different challenge in the same domain. Transfer learning has many advantages. First, it saves computational time because, instead of training a new model from scratch, it uses the already available information from the last training process. Second, it extends the knowledge it acquired from previous models, and third, transfer learning is beneficial when the size of the new training dataset is small. Transfer learning promises valuable contributions to the fields of computer vision, audio classification, and natural language processing.

There have been many attempts to automatize the image classification task to facilitate the process or make it more accurate. One of the earliest attempts was the convolutional neural network (CNN), which LeCun et al. (1989) introduced for the image classification task.

In 2012, thanks to Krizhevsky et al. (2012), CNN became the most popular technique for addressing the image classification problem. The authors achieved state-of-the-art performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition (Russakovsky et al., 2015), outperforming other commonly used machine learning techniques. CNN can be used in image classification and natural language processing (Collobert & Weston, 2008; Dos Santos & Gatti de Bayser, 2014; Kalchbrenner et al., 2014) and time series analysis (Wang et al., 2017; Zhao et al., 2017). In all of these cases, training the deep network weights from scratch requires a substantial amount of time and massive datasets (hundreds of thousands of images). These requirements make

deep learning algorithms very challenging in the context of medical images where, typically, only a limited number of images are available. A lot of time and experience are required to annotate medical images. That is where transfer learning can play a significant role: It allows for the use of a pre-trained architecture that was previously fitted to images of the same domain.

Thus, transfer learning is particularly suitable for addressing the DR classification domain, where there is a lack of images to accurately train a CNN from scratch.

Several studies have been done to classify DR by using CNN, either by using transfer learning or by introducing novel architectures (Amin et al., 2016; Ishtiaq et al., 2018; Nørgaard & Grauslund, 2018; Padhy et al., 2019; Paranjpe, 2014), but to the best of our knowledge, there have not been any reviews that survey the existing transfer learning techniques to classify DR images. To answer this call, in this paper, we discuss state-of-the-art DR image classification models that use the transfer learning of deep CNNs. Moreover, we discuss some essential open questions to apply transfer learning in the DR domain better.

More in detail, we discuss state-of-the-art models and techniques that were published from 2015 to mid-2019. We used the following descriptors: "diabetic retinopathy," "convolutional neural networks," "transfer learning," and "image classification" to cover the primary studies that address the classification of DR images by using transfer learning. These keywords were entered into the most popular academic databases, namely Scopus and PubMed.

Two filters were used to produce the results: The first filter excluded any paper that was not about DR, which reduced the results from 172 papers to 31 papers; the second filter excluded any paper that was not about transfer learning, which resulted in 18 papers that were about transfer learning applied to DR.

This paper is organized as follows: Section II gives an overview of a CNN structure. Section III discusses various CNN architectures that are commonly used in transfer learning. Section IV provides a brief description of the primary DR datasets that are available for public use. Section V provides a review of papers on the usage of transfer learning in classifying DR. Section VI presents the discussion, while Section VII presents open research questions. Finally, Section VIII concludes the paper.

3.2. Convolutional Neural Networks and Transfer Learning

CNN layers can be classified into two categories: primary layers and secondary layers. The primary layers are the main layers used in the CNN and consist of convolution layers, activation layers, pooling layers, flatten layers, and dense layers. Secondary layers are optional layers that can be added to make CNN more robust against overfitting and increase its generalizability. They include dropout layers, batch normalization layers, and regularization layers. Figure 3 shows a CNN structure.

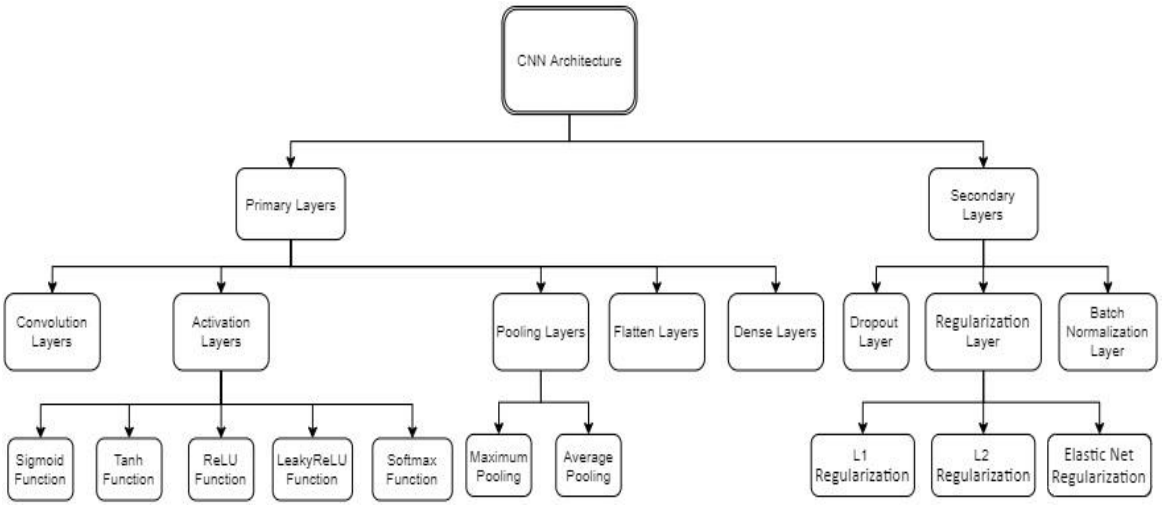


Figure 3. Convolutional neural network (CNN) structure.

3.2.1. Convolution Layers

The first and most important layer in CNN is the convolution layer, which can automatically extract the image features without defining these features manually. The convolution layer can be described mathematically by:

$$f(g(t)) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\mathcal{T}) g(t-\mathcal{T})\, d\mathcal{T} \tag{1}$$

The convolution is integral to the pointwise multiplication of two functions after one of them has been reversed and shifted (Dumoulin & Visin, 2016). From Equation (1), the $g(.)$ function is the filter that is used. It is then reversed and slides along to the $f(.)$ function, where $f(.)$ is the input function. The area of the intersection between the two functions, $g(.)$ and $f(.)$, is the convolution value. In a CNN, the filters are not reversed but instead used as-is. The filter used, $g(.)$, can be expressed as a grid of order n. Usually, the numbers inside the filter are initialized randomly, and then these numbers are learned during the network's training process. The result of the pointwise multiplication between the filter $g(.)$ and the input function $f(.)$ is saved in a new matrix called the output feature map. Figure 4 represents the differences between the convolution, the filter, and the output feature map.

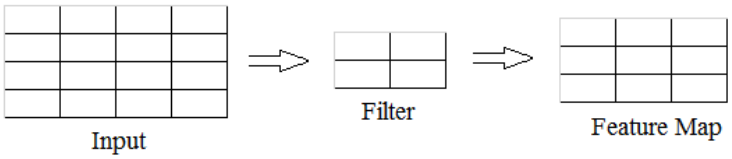


Figure 4. Convolution, filter, and feature map

The steps that are performed by the filter function over the input function define the stride parameter. The stride can be formally defined as the amount by which the filter function $g(.)$ moves at each step over the input function $f(.)$. Usually, after the convolution operation, the output feature

map will have smaller dimensions than the input function. One can rely on padding, a technique that adds zeroes around the input signal to maintain the original size, maintain the output map's dimensions, and prevent it from shrinking. Padding can be defined as the number of zeros added to the input function to control the output feature map's spatial size throughout a network, especially deep networks. Figure 5 represents an input function with zero-padding. The convolution operation output depends on the input size, the used filter size, the used stride, and the padding. The output feature map size is calculated as follows:

$$Output\ Height = \frac{H - K + S + P}{S}\ , Output\ Width = \frac{W - K + S + P}{S}$$

Where filter size is $K \times K$, input dimension is $H \times W$, the stride is S, and padding is P.

0	0	0	0	0	0
0					0
0					0
0					0
0					0
0	0	0	0	0	0

Figure 5. Zero padding input.

3.2.2. Activation Layers

Activation layers, nonlinear layers that usually follow the convolution layers, play an essential role as a selection criterion that decides whether a selected neuron will fire. The input of the activation layer is a real number that is transferred by applying a nonlinear function. The activation layer is vital because it allows the network to learn nonlinear mappings to make it more robust against complex functions. The most common activation layers used in CNNs are sigmoid, Tanh, ReLU, LeakyReLU, and softmax. The activation layers can be classified into saturated activation layers and non-saturated activation layers. If the output of the activation layer ranges between finite boundaries, it is classified as saturated; otherwise, it is considered a non-saturated activation function if it tends to infinite. The non-saturated activation functions have many advantages compared to saturated activation layers. For instance, the non-saturated layers can significantly help in the exploding/vanishing gradient problem of the backpropagation algorithm (Xu et al., 2015), one of the main issues when training a CNN. Different activation functions are shown in Figure 6.

3.2.2.1. Sigmoid Function

A saturated activation layer, which is a different form of a logistic function where the input is a real number and the output is a number in the range of [0,1], can be defined by

$$f(x) = \frac{1}{1 + e^{-x}}$$
$$f(x) \in (0,1) \tag{2}$$

3.2.2.2. Tanh Activation Function

The hyperbolic tangent function is a saturated activation layer commonly used when a negative gradient is essential. It outputs a number in the range of [-1, +1]. The following formula defines it:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
$$f(x) \in (-1,1) \tag{3}$$

3.2.2.3. ReLU Activation Function

The rectified linear activation layer (Nair & Hinton, 2010) is considered one of CNN's most important activation layers. It is a non-saturated activation function that is mainly used to remove any negative values. It is advantageous in CNN because it eliminates any negative gradients when the threshold is at zero.

$$f(x) = \max(0, x) \tag{4}$$

3.2.2.4. LeakyReLU Activation Function

A leaky rectified linear activation layer (Maas et al., 2013) is a non-saturated activation function that allows some negative gradients to pass. It is used to reduce the effect of the negative gradients by factor α.

$$f(x) = \begin{cases} x, x > 0 \\ \alpha x, x < 0 \end{cases} \tag{5}$$

3.2.2.5. Softmax Activation Function

Softmax is an activation layer usually at the end of a network, and it produces a discrete probability distribution vector.

$$P(y = j|X) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{X^T w_k}} \tag{6}$$

where X is the input vector and w_i is the predicted probability of $y = j$.

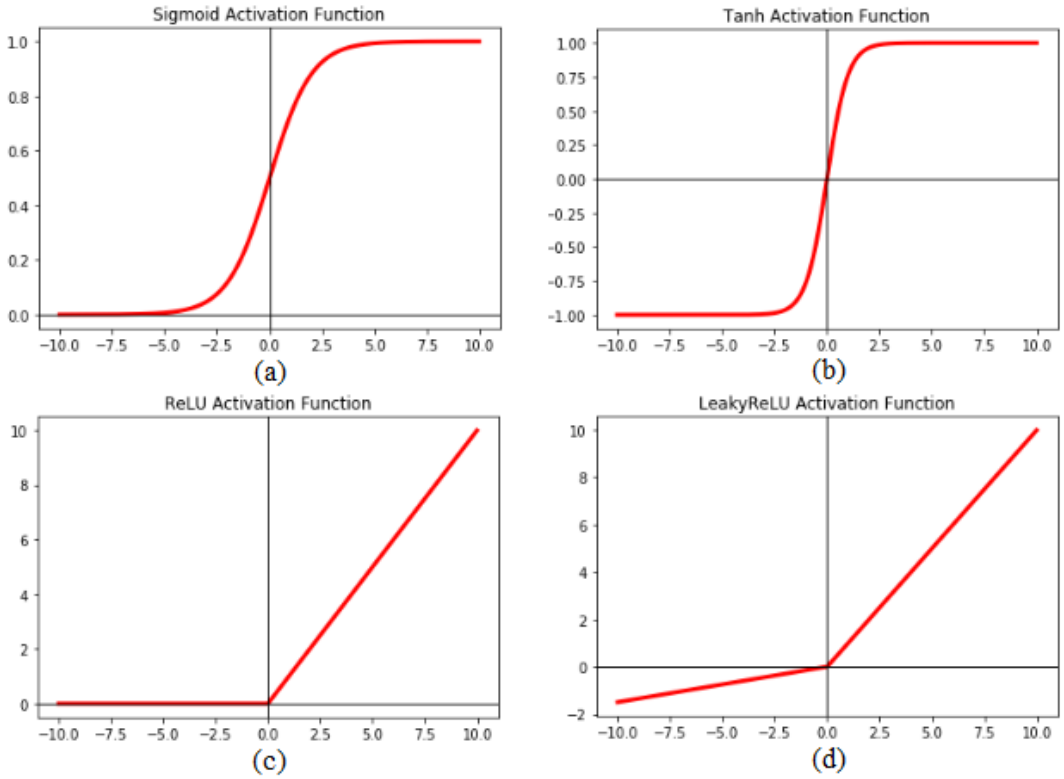


Figure 6. Plot of different activation functions: (a) Sigmoid activation function; (b) Tanh activation function; (c) ReLU activation function; and (d) LeakyReLU activation function. The x-axis represents the input x and the y-axis represents the output function.

3.2.3. Pooling Layers

Pooling layers are usually between consecutive convolution layers to progressively reduce the representation's spatial size to reduce the number of parameters and computation in a network. A pooling layer reduces the convolution layer's output feature map by extracting necessary pixels and removing noise. In this work, we assumed that the measurements were not noisy, and if this were not the case, a de-noising procedure would be necessary (Ouahabi, 2013). Additionally, a pooling layer is used to strengthen network spatial invariance (Scherer et al., 2010). The two main parameters of the pooling layers are the filter size and stride. The two main types of pooling layers are the maximum pooling layer and the average pooling layer.

3.2.3.1. Maximum Pooling

The pooling layer slides the filter over the previous convolution layer's output feature map and keeps each grid's maximum value.

$$f_{MP}(X) = max_{i,j}(i,j) \tag{7}$$

3.2.3.2. Average Pooling

The pooling layer slides the filter over the previous convolution layer's output feature map and takes the grid's average.

$$f_{AP}(X) = \frac{1}{n+m}\sum_{i=1}^{n}\sum_{j=1}^{m} X(i,j) \quad (8)$$

3.2.4. Flattening Layers

The output of the pooling layer is flattened to a $1D$ vector because the subsequent dense layers can only receive $1D$ vectors. A flattening layer can be seen in Figure 7. The dimensionality of the resulting vector is given by:

$$Dim_{Flat} = Dim_{img} * Dim_{img} * num_{color}$$

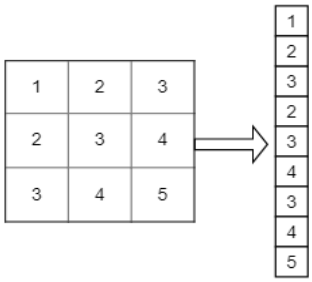


Figure 7. Flattening 2D feature maps to 1D vector.

3.2.5. Dense Layers

Dense layers, also known as fully connected layers, are usually placed at the end of a network, and they receive the output of the feature extraction layers as the input. The dense layer's main purpose is to consider all the features extracted from the previous layers and use them to classify the original image. At the end of the network, a softmax or sigmoid function is applied to output the target probability.

3.2.6. Dropout Layer

A dropout layer is a regularization layer that was first introduced by (Srivastava et al., 2014). It can be applied to any layer in the network. During network training, some neurons are disabled with a predefined dropout-rate probability P. It can be thought of as bagging for neural networks.

3.2.7. Regularization Layers

Complex models with large weights usually have low generalizability since they can learn noise instead of learning the true model patterns (Chollet, 2017a). Under the assumption that models with small weights have better generalizability than those with large weights, regularization functions are commonly used to limit overfitting. Regularization works by adding a penalty term to the loss function to avoid large weights being used by the model (James et al., 2014). The main idea of regularization is to eliminate the weights that do not contribute to the model accuracy by shrinking them to zero. Three types of regularization have been introduced in the literature: L1, L2, and elastic nets. The main differences between these regularizations lie in the penalty terms.

3.2.7.1. $L1$ regularization

$L1$ regularization constrains the weights to zero by adding the sum of the absolute values of the weights to the loss function. It can push some weights to be exactly zero and so can be thought of as a feature extractor. The magnitude of the penalty is determined by α, so the larger the value of α, the higher the constraint to the weights, usually $0 \leq \alpha \leq 1$. $L1$ regularization can be formally defined as:

$$f(w) = \sum_{i=1}^{n} (y_i - \sum_{j=1}^{m} x_{ij} w_j)^2 + \alpha \sum_{j=1}^{m} |w_j| \quad (9)$$

where n is the number of training examples, m denotes the number of weights, w_j is the weight at j neuron, y_i is the label, and α is the regularization factor.

3.2.7.2. $L2$ regularization

$L2$ regularization decreases large weights by adding the sum of the squares of the weights to the loss function. The magnitude of the penalty is determined by α, so the larger the value of α, the higher the constraint to the weights, usually $0 \leq \alpha \leq 1$. $L2$ regularization can be formally defined as:

$$f(w) = \sum_{i=1}^{n} (y_i - \sum_{j=1}^{m} x_{ij} w_j)^2 + \alpha \sum_{j=1}^{m} w_j^2 \quad (10)$$

where n is the number of training examples, m denotes the number of weights, w_j is the weight at j neuron, y_i is the label, and α is the regularization factor.

3.2.7.3. Elastic Net regularization

To overcome both techniques' shortcomings, an elastic net was introduced, as it linearly combines both regularization techniques to benefit from both techniques at once. Elastic net can be defined as:

$$f(w) = \frac{\sum_{i=1}^{n} (y_i - \sum_{j=1}^{m} x_{ij} w_j)^2}{2n} + \alpha \left(\frac{1-\gamma}{2} \sum_{j=1}^{m} w_j^2 + \gamma \sum_{j=1}^{m} |w_j| \right) \quad (11)$$

where n is the number of training examples, m denotes the number of weights, w_j is the weight at j neuron, y_i is the label, α is the regularization factor, and γ is the mixing parameter between the ridge ($\gamma = 0$) and the lasso ($\gamma = 1$). By combining both $L1$ and $L2$, the strength of each term can be tuned by α.

3.2.8. Batch Normalization Layers

Batch normalization can speed up the network's training and increase its robustness against overfitting (Ioffe & Szegedy, 2015). It reduces the network covariance shift (Santurkar et al., 2018). Additionally, batch normalization adds noise to each layer to increase its robustness. It works by normalizing each layer's inputs by subtracting the batch mean and dividing by the batch standard deviation.

$$\mu_B = \frac{1}{n_B}\sum_{i=1}^{n_B} x^{(i)} \ is\ the\ mini-batch\ mean \tag{12}$$

$$\sigma_B^2 = \frac{1}{n_B}\sum_{i=1}^{n_B} (x^{(i)} - \mu_B)^2 \ is\ the\ mini-batch\ variance \tag{13}$$

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \ is\ the\ sample\ normalization \tag{14}$$

$$Z^{(i)} = \gamma * \hat{x}^{(i)} + \beta \ is\ the\ batch\ normalization \tag{15}$$

where μ_B is the mini-batch B mean, σ_B is the mini-batch B standard deviation, n_B is the number of instances in the mini-batch, $\hat{x}^{(i)}$ is the zero-centered and normalized input, for instance, i, γ is the scaling parameter for the layer, β is the shifting parameter (offset) for the layer, ϵ is a tiny number to avoid division by zero (typically 10^{-5}; it is called a smoothing term), and $Z^{(i)}$ is the output of the BN operations (it is a scaled and shifted version of the inputs). Thus, in total, four parameters must be learned for each batch-normalized layer: $\gamma\ (scale), \beta\ (offset), \mu\ (mean)$ and $\sigma\ (standard\ deviation)$.

3.2.9. Transfer learning

Transfer learning is a deep learning technique that is used to rapidly and accurately train a CNN in which its weights are not initialized from scratch. Instead, they are imported from another CNN that was trained on a larger dataset. The most popular set of weights used for transfer learning is from the ImageNet dataset (Deng et al., 2009). Several CNN architectures have been trained on the ImageNet dataset and have achieved high accuracy. These weights can be used to classify another completely different dataset instead of randomly initializing the weights from scratch. There are four strategies in transfer learning. The first strategy is to remove the original fully connected layers that act as classifiers, freeze the entire network weights, use the CNN pre-trained layers as feature extraction, and then add a classifier layer such as a fully connected layer or another machine learning classifier, like a support vector machine. The second strategy is to remove the original fully connected layers, fine-tune the entire network weights by using a minimal learning rate (LR), and add a new classifier layer that suits the new task. The third strategy is to remove the fully connected layers, fine-tune only the top layers while keeping the bottom layers frozen, and then add a new classifier layer that suits the new task. Many researchers have suggested that the bottom layers only detect generic features such as edges and circles, while the top layers detect more dataset-specific features.

For this reason, many authors recommend only fine-tuning the top layers (Kornblith et al., 2018; Shin et al., 2016; Yosinski et al., 2014). The fourth strategy is to use a state-of-the-art architecture and start training it from scratch by using only the architecture proven to work on different challenging datasets. A generic CNN model architecture can be seen in Figure 8.

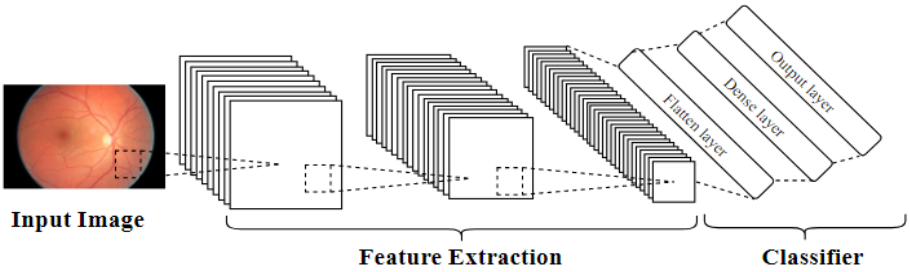


Figure 8. Convolutional neural network architecture.

3.3. CNN Architectures

In this section, the main CNN architectures used in transfer learning are reviewed. According to (Alom et al., 2018), the rise of deep learning in image classification started in 2012 with the introduction of AlexNet (Krizhevsky et al., 2012), which introduced the ReLU activation layer as well. The usage of a CNN in image classification increased its accuracy and eliminated each image's need to feature-engineer. After AlexNet, many architectures—namely VGG16, VGG19, ResNet, GoogLeNet, DenseNet, and Xception—were introduced with more features to classify images efficaciously.

3.3.1. VGG Network Architecture

In 2014, researchers at Oxford's Visual Geometry Group introduced two novel architectures named VGG16 (Simonyan & Zisserman, 2014) and VGG19 (Simonyan & Zisserman, 2014). VGG16 achieved a top-five accuracy rate of 91.90% in the ImageNet competition in 2014. The VGG16 architecture has 138,355,752 parameters, five convolution blocks, and three dense layers. Each block contains some convolutional layers and then a max pool layer to decrease the block output size and remove the noise. The first two blocks have two convolutional layers each, and the last three blocks have three convolutional layers each. The size of the kernel that is used throughout this network has a stride of 1. After the five blocks, a flatten layer was added to convert the 3D vector of the blocks to a 1D vector to be inserted into the fully connected layers. The first two fully connected layers have 4096 neurons, and the last fully connected layer has 1000 neurons. After the fully connected layers, a softmax layer is inserted, and this is used to ensure that the probability summation of the output is one. The main difference between VGG16 and VGG19 is that VGG19 has 19 convolution layers instead of 16 convolution layers. The number of parameters increases from 138,357,544 to 143,667,240 because of additional layers. The authors argued that these additional layers make the architecture more robust and can learn more complex architectures.

This network's main benefit is its sequential blocks, where the sequential convolutional layers that are inserted after each other allow for a reduction of the amount of spatial information needed. This network's main drawback is that the authors specify more weights for the classifier portion and not for

the feature extraction portion. This considerably increases the number of parameters. The network's ImageNet weights are available in the Keras package.

3.3.2. ResNet Network Architecture

ResNet, which stands for residual network, was introduced by He et al. (2016) in 2015 and achieved first place in the 2015 ImageNet competition with a top-five accuracy rate of 94.29%. It has a total of 25,000,000 parameters. Compared to other architectures, ResNet is a very deep network that can reach up to 152 layers. It has a unique connection called the residual connection, which is a connection that is applied between the convolutional layers and then passed to the ReLU activation layer. The residual connection makes sure that during backpropagation, the weights learned from the previous layers do not vanish. Three versions (which differ in the number of layers) of this network have been introduced, namely ResNet50, ResNet101, and ResNet152. The main benefit of this network is the use of residual connections, making it possible to use many layers.

Moreover, increasing the network's depth (instead of widening it) results in fewer extra parameters. This network's main drawbacks are the summation in each residual block, which makes the filter size the same. Additionally, this network requires large datasets to be properly trained, thus resulting in a computationally expensive training phase. The network's ImageNet weights are available in the Keras package.

3.3.3. GoogLeNet Network Architecture

In 2014, Google researchers introduced a novel architecture called the GoogLeNet network (Szegedy et al., 2015), which is also known as IncpetionV1 architecture. The authors won the ImageNet competition (Russakovsky et al., 2015) with a top 5 accuracy rate of 92.2%. After the success of InceptionV1, the authors introduced other versions like InceptionV2 and InceptionV3. The main idea of GoogLeNet architecture is to use multiple convolution layers in the same block to go not only deeper but wider and to capture different features of the images; these blocks are referred to as Inception blocks. The most popular GoogLeNet architectures are the InceptionV1 and InceptionV3 architectures. In the InceptionV1 inception blocks, six convolution layers are used, while in the InceptionV3 inception blocks, seven convolution layers are used. In the remainder of the paper, just like in the literature, the InceptionV1 architecture is referred to as the GoogLeNet architecture. The main benefit of this network is the presence of an inception module, which allows the network to capture different aspect ratios of the same image by using the convolution layers in parallel. The main drawback of this network is the computational effort that is needed to train it because the layers are deep and wide. The InceptionV3's ImageNet weights are available in the Keras package. An InceptionV1 block and InceptionV3 block are shown in Figure 9.

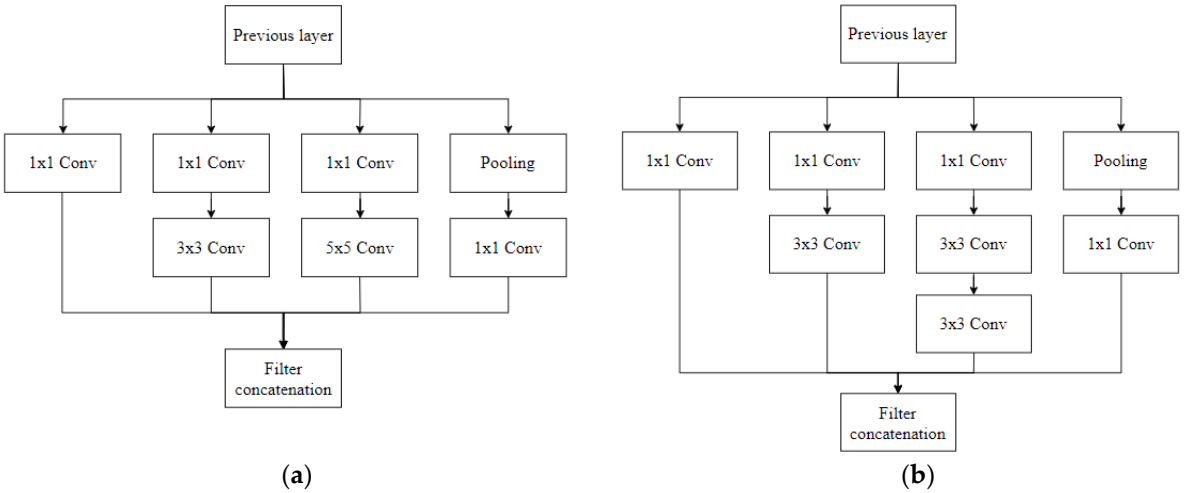


Figure 9. Inception blocks. (a) InceptionV1. (b) InceptionV3.

3.3.4. AlexNet Network Architecture

AlexNet architecture (Krizhevsky et al., 2012) was the first CNN network to participate in the ImageNet challenge in 2012. It achieved an accuracy rate of 84.6%, which outperformed all the previous shallow algorithms used in image classification. Since then, CNNs have become the state-of-the-art algorithm in image classification. The AlexNet architecture has 60,000,000 parameters, five convolution layers, and three dense layers. The two novel introductions in AlexNet were using the ReLU activation function (instead of the sigmoid activation function) and the usage of dropout to overcome the overfitting that this deep architecture can cause. This network's main advantage relies on the fact that the training process is computationally efficient compared with the other networks that have been taken into account. On the other hand, the network is not deep enough to capture complicated features from images.

3.3.5. DenseNet Network Architecture

DenseNet architecture (Huang et al., 2017) stands for densely connected convolutional networks. It was inspired by ResNet, but instead of the residual connections, the authors proposed the use of dense blocks. The dense block consists of sequentially placed convolution layers, like VGG, but each layer has a connection to all the subsequent layers. The main idea is for each convolution layer to receive the information from all the previous layers. DenseNet has 8,062,504 parameters and achieved a 93.34% top 5 accuracy rate on the ILSVCR challenge. This network's main advantage is the presence of connections between all layers, which reduces the information loss between layers (especially the deep layers). The main drawbacks are the following: The training phase is computationally expensive, and it requires very large datasets to achieve satisfactory performance. The network's ImageNet weights are available in the Keras package.

3.3.6. Xception Network Architecture

The Xception (which stands for extreme inception) network was introduced by Chollet (2017), and it was inspired by the InceptionV3 architecture. The main idea exploited by the Xception architecture is to replace the inception module with depthwise separable convolution, followed by a pointwise separable convolution. This network is 71 layers deep, and it has 22.9 million parameters. The Xception network achieved a 94.50% top 5 accuracy rate on the ILSVCR challenge. This network's main advantage is that it has a deep architecture but with a small number of parameters, thus making it computationally efficient compared to other deep networks. The main drawback is that this network requires very large datasets to be able to train all its parameters.

Table 1. shows a summary of the proposed networks with their number of parameters and their accuracy over the ImageNet dataset. The accuracy is calculated by dividing the correctly classified observations over the total number of observations. The $top-k$ accuracy is the accuracy of the architecture over predicted labels $\hat{y}$, where the top 5 accuracy represents the accuracy over 5 classes accuracy, and the top 1 accuracy represents the accuracy for a single-class classification. When $k = 5$, the accuracy is measured by taking into account if the label y is present in the top 5 predicted labels $\hat{y}$, while if $k = 1$, $top-k$ is the de-facto accuracy measure. The $top-k$ accuracy measure was used here because the ILSVRC challenge had 1000 classes.

Table 1. Top 5 accuracy, top 1 accuracy, and the number of parameters of AlexNet, VGG, Inception, and ResNet in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenge.

Architecture	Number of Parameters	Top 5 Accuracy	Top 1 Accuracy
AlexNet	62,378,344	84.60%	63.30%
VGG16	138,357,544	91.90%	74.40%
GoogLeNet	23,000,000	92.2%	74.80%
ResNet-152	25,000,000	94.29%	78.57%
DenseNet	8,062,504	93.34%	76.39%
Xception	22,910,480	94.50%	79.00%

3.4. DR Datasets

Several DR datasets were made publicly available to allow researchers to develop algorithms that can classify DR. A brief description of these datasets is given in this section. DR Datasets descriptions are shown in Table 2.

Table 2. Diabetic retinopathy (DR) datasets description.

Dataset	Size	Grades
Kaggle	88,000	5
Messidor	1200	4
DR1	1014	2
E-ophtha	463	2
STARE	397	14

3.4.1. Kaggle Dataset

The Kaggle DR (Cuadros & Bresnick, 2009) dataset is considered one of the most important datasets for DR because it includes more than 88,000 publicly available images captured using different cameras at different angle dimensions. This dataset is divided into 40% for training and 60% for testing, and various cameras took the images. Therefore, different levels of quality appear in this dataset. The annotation of this dataset is a five-class annotation, as proposed by Wilkinson et al. (2003). The dataset suffers from imbalance, as the rare DR levels (3 and 4) cover less than 5% of the dataset.

3.4.2. Messidor Dataset

Messidor (Abramoff et al., 2013; Decencière et al., 2014) is a publicly available dataset that consists of 1200 DR images. This dataset, like the Kaggle dataset, was acquired by using different cameras and settings, and it was built by collecting images from three different hospitals in France. This dataset is more balanced than Kaggle's because each class is distributed uniformly. The DR grades are divided into four grades.

3.4.3. DR1 Dataset

DR1 (Pires et al., 2014) is a publicly available dataset that was provided by the Federal University of Sao Paulo, Brazil. The dataset contains 1014 images with 68% normal images and 32% DR images. All the images were captured by using the same camera.

3.4.4. E-ophtha Dataset

The E-ophtha dataset (Decencière et al., 2013) is a publicly available dataset that contains two main subsets of images. The E-ophtha_Ex dataset has the objective of detecting exudates in fundus images. This dataset has 82 images split into 47 fundus images with exudates and 35 images without exudates. The other dataset is the E-ophtha_MA, and the objective is to detect microaneurysms in fundus images. This dataset contains 381 images divided into 148 images with aneurysms and 233 without arterial swelling.

3.4.5. STARE Dataset

The STARE dataset (Goldbaum et al., 1990) is a publicly available dataset that contains 400 images that were captured by using the same camera. It has 397 fundus images divided into 14 retina-related diseases.

3.5. Paper review

This section discusses the selected papers based on different aspects like the architecture used, the target dataset used, the optimizer used, and the LR used, the performance of the architecture after

transfer learning, the fine-tuning process employed, and, finally, the validation process whenever applicable.

In transfer learning, a set of weights that were learned from an image dataset can be used to classify another image dataset. The deep layers are generic and can be used to extract salient features that are suitable for classifying any image. This aspect is why many authors have tried to use transfer learning in detecting DR. For instance, Gulshan et al. (2016) used InceptionV3 architecture to classify DR into two grades: DR or No DR. The dataset that the authors considered contained 128,175 images. The reported results on two test datasets, with sizes 9963 and 1748, had sensitivities of 97.5% and 96.1%, respectively. Masood et al. (2017) used the Kaggle dataset to assess the performance of the InceptionV3 model to classify DR into five grades. The authors chose 4000 images and cropped them to 500 pixels. The authors used accuracy to assess the model's performance, which was reported as 48.8%.

Li et al. (2017) discussed using transfer learning for detecting DR by comparing different network architectures, including AlexNet, VGG-S, VGG16, and VGG19, to two datasets: the Messidor and DR1 datasets. Three transfer learning techniques were analyzed: fine-tuning the entire networks, fine-tuning the networks layer-wise, and, finally, freezing the entire network's weights and applying SVM as a classification layer. The authors used a stochastic gradient descent for the optimizer, and the images were pre-classified as either DR or No DR to pose a binary classification problem. The accuracy measure used was the AUC of the ROC curve. The highest AUC achieved was obtained by fine-tuning the entire network, while the second-best performance was achieved by fine-tuning layer-wise. The VGG-S architecture obtained the highest AUC that was achieved for the Messidor dataset with an AUC of 98.34%. For the DR1 dataset, an AUC of 97.86% was obtained by using the same network.

Mohammadian et al. (2017) compared the InceptionV3 and Xception architectures to classify DR into two grades, DR or No DR, by using the Kaggle dataset. The authors used the whole dataset of 35126 images, with 20% of the images being used to test the algorithm’s performance over unseen data. The authors fine-tuned the two architectures' last two blocks and compared two optimizers with different LRs: stochastic gradient descent and Adam. The authors augmented the images by horizontally and vertically flipping the images or by shifting and rotating the images to increase the model's robustness. The authors used the accuracy measure to assess the performance of the architectures. The reported results were 87.12% for the InceptionV3 architecture and 74.49% for Xception.

Takahashi et al. (2017) trained a modified GoogLeNet architecture by using a private dataset. They used 9443 images to train the model and 496 to test it. They cropped the images to 1272×1272 pixels, and they considered a four-class classification scheme. The reported accuracy was 81%, and the kappa score was equal to 0.74. Choi et al. (2017) investigated the impact of transfer learning on

the STARE dataset (Goldbaum et al., 1990). They used image augmentation techniques to increase the dataset's size to 10,000 images, with ten retina disorder categories, including DR. The authors opted for the pre-trained VGG19 and AlexNet architectures. An ensemble was created to increase the network accuracy, and K-fold validation with k = 5 was used to validate the results. The authors obtained the highest accuracy by using VGG19 architecture with random forest (RF) as a classifier.

Wang et al. (2018) investigated transfer learning techniques by using three network architectures: AlexNet, VGG16, and InceptionNetV3. The authors used 166 images from the Kaggle dataset to tune the algorithms. The authors opted for the five-stage classification approach instead of the binary classification approach that other authors have used for this specific dataset. Additionally, they employed a stochastic gradient descent optimizer with Nesterov momentum to accelerate the convergence to the minimum. The authors cropped the images for each architecture to 227×227 for AlexNet, 224×224 for VGG16, and 299×299 for InceptionV3. They used the network's accuracy as the evaluation metric, and they used $K - fold\ validation$ with $K = 5$ to cross-validate the results. The best-reported accuracy was 63.2% for the InceptionV3 architecture. Hazim et al. (2018) used 580 images from the Messidor dataset to test the transfer learning of AlexNet. They opted for a two-class classification, and they cropped the images to 227×227. They achieved an 88.3% accuracy on the test set, which consisted of 290 images.

Lam et al. (2018) considered the sliding windows algorithm, where small patches from the original images are used to train the CNN. These patches contain the important features of each image, such as the presence of exudates or microaneurysms. The authors used the Kaggle dataset to extract these patches. They extracted 1324 patches from 243 images and split these patches into training and testing datasets. They tested the proposed algorithm by using the E-Optha dataset, which contained 195 images. They used GoogLeNet architecture to train the model with an input size of 128×128. The authors considered a multi-class classification task with five DR grades. They resized the test images to 2048×2048 and normalized the pixels to test the model. Subsequently, the trained model crossed over the test image to produce a heat map with a probability score for every one of the five grades. The authors compared five pre-trained architectures (AlexNet, VGG16, GoogLeNet, ResNet, and InceptionV3) for binary classification and multi-class classification. The best performing architecture was InceptionV3, with a multi-class accuracy of 96% and a binary-class accuracy of 98%.

Lam et al. (2018) trained a CNN by using transfer learning of the AlexNet, VGG16, and GoogLeNet models, and they utilized Kaggle two-class output. The authors reported that GoogLeNet achieved the highest sensitivity of 95% and specificity of 96%. The authors tried to utilize the multi-class Kaggle dataset, but they stated that a CNN could not learn mild class sensitivity. The authors achieved decent results for detecting mild grades when using the Messidor dataset. Wan et al. (2018) compared the difference between transfer learning and learning from scratch. The authors used four

CNN architectures, namely AlexNet, ResNet, GoogLeNet, and VGG. The authors performed their experiments on the full Kaggle dataset, and they used the AUC of the ROC curve, accuracy, sensitivity, and specificity as evaluation criteria. The authors reported that transfer learning did significantly increase CNN's performance, with VGG-S producing the highest AUC.

Xu et al. (2018) studied the difference between the performance of DenseNet with and without fine-tuning. The authors examined their method on a private dataset with 10,000 images and five grades. The authors used image augmentation to increase the dataset's size and balance the dataset between different classes. The final dataset contained 20,000 images that were distributed equivalently between the five classes. The authors used a stochastic gradient descent (SGD) as an optimizer with an LR of 0.1 for training from scratch and an LR of 0.01 for fine-tuning the network. The authors reported that transfer learning increased the accuracy of the model used.

Tsighe et al. (2019) investigated the usage of the InceptionV3 architecture to detect DR in the Kaggle dataset. The authors chose 2500 images and cropped them to 300×300 to train the model, and 5000 images were used to test the model. The authors pre-classified the images as either DR or No DR to make it a binary classification task. They employed a stochastic gradient descent as an optimizer, with an LR of 0.0005, to fine-tune the neural network. The reported result was a 90.9% accuracy and a 3.94% loss. Chen et al. (2019) considered the pre-trained InceptionV3 architecture to classify DR on 7023 images of the Kaggle dataset. The authors adopted a five-stage classification approach with the quadratic weighted kappa as an accuracy measure. The images were cropped to 229×229, and a stochastic gradient descent was used as an optimizer. Image augmentation was used with an early stop for 15 iterations to overcome the overfitting of the network. The reported Kappa score was 0.64, with an accuracy of 80%.

Zeng et al. (2019) proposed a novel Siamese-like architecture where the left and right fundus images were classified together. Siamese neural networks are networks with two parallel neural networks, and each of these networks takes different inputs. The authors used the Kaggle dataset with 28,104 training images split between right and left eyes and 7024 to test the architecture. They used the pre-trained InceptionV3 network on the ImageNet dataset. The authors examined the five-stage classification, as proposed by Wilkinson et al. (2003), and opted to use a binary class classification. They used Adam as an optimizer, quadratic weighted kappa as the accuracy measure for the multiclass classification, and the AUC of the ROC for the binary classification. All the layers of InceptionV3 were fine-tuned, and the images were cropped to 229×229. The authors augmented images by randomly flipping them horizontally and randomly applying a geometric transformation to increase the dataset's size and control overfitting. They normalized all images from [0,255] to [-1,+1]. They reported the kappa result as 0.829 for the multiclass classification and an AUC of 95.1% for the binary classification.

Zhang et al. (2019) used a private dataset with 13,767 images to propose a model called DeepDR, which uses deep learning based on transfer learning models to detect DR. The model consists of three stages: identification, grading, and reporting. The identification stage is a binary classification model to predict the presence of DR. If DR exists, then the image is graded by using the grading stage of the four stages of DR; the last stage reports the result of the model. The authors used InceptionV3, Xception, and InceptionResNetV2 for feature extraction in the identification system. Moreover, they added a global average pooling layer to normalize the feature extractor's output, and they subsequently added four dense layers with sizes 1024, 512, 256, and 128, respectively. A dropout layer between the dense layers, with a probability of 50%, was employed to limit overfitting. Due to its speed of convergence, the authors opted for the LeakyReLU activation function with a β of 0.2 and, in the end, a softmax layer to sum up the probabilities to 100%. The authors used ResNet50, DenseNet169, and DenseNet201 for feature extraction in the grading system. They then added a global average pooling layer and four dense layers with sizes 2048, 1024, 512, and 256, respectively. They employed a dropout layer between the dense layers with a probability of 50%, LeakyReLU, as the activation function for all the dense layers with β of 0.2 and, in the end, the softmax layer. The authors averaged the outputs of the three models' softmax layer to decrease the variance of the model output. The identification model achieved a sensitivity of 97.5% and a specificity of 97.7%, while the grading model reached 98.1% for sensitivity and 98.9% for specificity.

Yip et al. (2019) explored three CNN architectures, namely VGG, ResNet, and an ensemble of both architectures. The authors experimented with using a private dataset with three DR classes and with 148,266 images divided into 51.5% to train and 48.5% to validate the model. Three measures were used to assess the model's quality, namely AUC, sensitivity, and specificity. The authors reported that transfer learning increased model accuracy. Gao et al. (2019) used a private dataset with 4476 images with four classes. The authors cut the original images into four $300 * 300$ partitions that were the input of four InceptionV3 networks, and then they concatenated the results to a single layer. The original fully connected layers were removed, and only a softmax layer was used. The Adam optimizer was employed to fine-tune the InceptionV3 networks. The authors compared their method against ResNet18, ResNet101, VGG19, and InceptionV3. The reported results showed that their model achieved higher accuracy than the other models. Table 3 shows the list of the reviewed papers that applied transfer learning to classify DR.

Table 3. Studies applying transfer learning.

Study	Architecture	Number of Classes	Dataset	Dataset Size	Performance Measure	Results
Gulshan et al. (2016)	InceptionV3	2 classes	Private	128,175	Sensitivity	97.5%
Masood et al. (2017)	InceptionV3	5 classes	Kaggle	4000	Accuracy	48.8%

Li et al. (2017)	AlexNet	2 classes	Messidor	1200	AUC	77.27%*
	VGG-S	-	DR1	1014	-	**98.34%***
	VGG16	-	-	-	-	74.37%*
	VGG19	-	-	-	-	68.69%*
Mohammadian et al. (2017)	InceptionV3	2 classes	Kaggle	35,126	Accuracy	**87.12**%
	Xception					74.49%
Takahashi et al. (2017)	GoogLeNet	4 classes	Private	9443	Accuracy	81%
					Kappa	0.74
Choi et al. (2017)	VGG19	10 classes	STARE	10,000	AUC	**90.3**%*
	AlexNet					81.6%
Wang et al. (2018)	AlexNet	5 classes	Kaggle	166	Accuracy	37.43%
	VGG16					50.03%
	InceptionV3					**63.23%**
Hazim et al. (2018)	AlexNet	2 classes	Messidor	580	Accuracy	88.3%
Lam et al. (2018a)	AlexNet	-	Kaggle	1050	Accuracy	79%
	VGG16	-	e-ophtha	274	-	90%
	GoogLeNet	2 classes	-	-	-	**98%**
	ResNet	-	-	-	-	95%
	InceptionV3	-	-	-	-	**98%**
Lam et al. (2018b)	AlexNet	2 classes	Kaggle	35,000	Sensitivity	95%*
	GoogleNet	-	Messidor	1200	Specificity	96%*
	VGG16	-	-	-	-	-
Wan et al. (2018)	AlexNet	5 classes	Kaggle	35,126	AUC	93.42%
	VGG-S					**97.86%**
	VGG16					96.16%
	VGG19					96.84%
	GoogLeNet					92.72%
	ResNet					93.65%
Xu et al. (2018)	DenseNet	5 classes	Private	20,000	Error rate	17.48%*
Tsighe et al. (2019)	InceptionV3	2 classes	Kaggle	2500	Accuracy	90.9%
					Loss	3.94%
Chen et al. (2019)	InceptionV3	5 classes	Kaggle	7023	Kappa	0.64
					Accuracy	80%
Zeng et al. (2019a)	InceptionV3	2 classes 5 classes	Kaggle	28,104	Kappa AUC	0.829 95.1%
Zhang et al. (2019)	ResNet	4 classes	Private	13,767	Sensitivity	98.1%*
	DenseNet				Specificity	98.9%*
Yip et al. (2019)	VGG16	3 classes	Private	148,266	AUC	95.8%*
	ResNet					**99.4%***
Gao et al. (2019)	Inception@4	4 classes	Private	4476	Accuracy	**88.72%**
	InceptionV3					88.35%
	ResNet18					87.61%
	ResNet101					87.26%
	VGG19					85.50%

*The results from Li et al. (2017) are the results of fine-tuning the entire networks by using the Messidor dataset. The results from Zhang et al. (2019) are the results of the grading model. The results of Lam et al. (2018) are GoogLeNet architecture results for the two-class Kaggle dataset. The results from Yip et al. (2019) are the results of the vision-threatening DR. The results shown from Xu et al.

(2018) are the results of using transfer learning with 24 kernels. The VGG19 results shown from Choi et al. (2017a) are the VGG19 with transfer learning and RF as a classifier.

3.6. Discussion

This study reviewed recent studies that implemented transfer learning in classifying diabetic retinopathy images. These studies were extracted from two databases (PubMed and Scopus), and after applying two filters, 18 studies were selected. The selected papers were analyzed based on six aspects: the architecture used, the target dataset used, the optimizer used, the LR used, the performance of the architecture after transfer learning, the fine-tuning process used, and, finally, the validation process that was applied. In this section, we discuss the main findings of this analysis.

3.6.1. Architectures used

In the reviewed articles, many state-of-the-art architectures were used to classify DR. Among them, InceptionV3 was the most commonly used, followed by the AlexNet and VGG16 architectures. The choice of the architectures did not depend on the size of the dataset. In studies (Choi et al., 2017a; Gao et al., 2019; Lam et al., 2018a; Lam et al., 2018b; Li et al., 2017; Mohammadian et al., 2017; Wan et al., 2018; Wang et al., 2018; Yip et al., 2019; Zhang et al., 2019), the authors compared different architectures to determine the best performing one. In studies (Gao et al., 2019; Lam et al., 2018; Mohammadian et al., 2017; Wan et al., 2018; Wang et al., 2018), the authors compared InceptionV3 architecture to other networks, and InceptionV3 achieved the best performance in all the studies except for (Wan et al., 2018). The AlexNet architecture achieved the lowest performance in the following studies: (Choi et al., 2017a; Lam et al., 2018a; Lam et al., 2018b; Wan et al., 2018; Wang et al., 2018). The high performance of InceptionV3 may be attributed to the inception module used. This module can capture different aspect ratios in the same image, which was shown to be very useful in DR images. The low performance of AlexNet could have been caused by the fact that it only uses five convolution layers. This number is not sufficient to accurately classify challenging images, like DR. A summary of the architectures used is shown in Table 4.

Table 4. Architectures analysis.

Architecture	Count
InceptionV3	9
AlexNet	7
VGG16	6
VGG19	3
VGG-S	2
Xception	2
DenseNet	2
ResNet	5
GoogleNet	4

3.6.2. The datasets used

In the reviewed papers, the most commonly used public datasets were the Kaggle dataset due to its availability and its size, followed by the Messidor dataset. Many private datasets were used as well in studies (Gulshan et al., 2016; Takahashi et al., 2017; Xu et al., 2018; Yip et al., 2019; Zhang et al., 2019). Many researchers like Tsighe et al. (2019), Li et al. (2017), Mohammadian et al. (2017), Hazim et al. (2018), Lam et al. (2018a), and Lam et al. (2018b) considered a binary classification task due to the lack of a sufficient number of images for some of the classes. In particular, the lack of severe cases images plays a vital role because too few images are available for training the network. An important factor that affected the performance of the classifier was the size of the datasets. It played a significant role in classification performance, especially when using an algorithm like CNN. The second important factor was the number of classes of each dataset, with the binary classification outperforming the multiclass classification. This can be attributed to the unbalance of the datasets and to the difficulty (for some of the models used) in distinguishing among more than two classes. This difficulty was caused by the low number of examples of a given class, as well as by the quality of the images. A summary of the datasets used is shown in Table 5.

Table 5. Datasets analysis.

Dataset	Count
Kaggle	9
Messidor	3
Private	6
e-ophtha	1
DR1	1
STARE	1

3.6.3. The optimizers used

The optimizer's main task during network training is to update the weights to reduce the value of the loss function. The optimizer can significantly impact the convergence of the training process, especially for transfer learning, as pointed out by Mohammadian et al. (2017) and Lam et al. (2018). Four optimizers were mainly reported by the authors, namely SGD for studies (Choi et al., 2017b; Gulshan et al., 2016; Hazim et al., 2018; Lam et al., 2018a; Lam et al., 2018b; Li et al., 2017; Masood et al., 2017; Mohammadian et al., 2017; Takahashi et al., 2017; Tsighe, 2019; Wan et al., 2018; Wang et al., 2018; Xu et al., 2018). SGD with momentum for studies (Choi et al., 2017b; Mohammadian et al., 2017; Wang et al., 2018), Adam for studies (Gao et al., 2019; Mohammadian et al., 2017; Zeng et al., 2019b) and RMSProp in (Zhang et al., 2019). The stochastic gradient descent optimizer (SGD) allows for a faster training process than the traditional gradient descent because it only considers, at each iteration, a subset of the training set. Thus, it generally achieves faster iterations in trade for a (slightly) lower convergence rate. SGD with momentum (SGDM) can be used instead of SGD.

Adding the momentum and thus determining the next update of the weights based on a linear combination of the gradient and the previous update prevents the training process from showing oscillatory behavior. This should result in faster and accurate convergence. The RMSProp optimizer, a member of the adaptive gradient group, was introduced to overcome the problem of determining the learning rate's initial value, which is now learned during the training process. The Adam optimizer was introduced to combine the benefits from both the SGDM and the RMSProp optimizer.

The LR chosen by the authors was very low to avoid losing the original weights of the layers, and it ranged from 1×10^{-2} to 1×10^{-5}. Wang et al. (2018) used LR = 1×10^{-4} for AlexNet and VGG16, as well as LR=1×10^{-3} for InceptionV3. Tsighe et al. (2019) used LR = 5×10^{-4}. Mohammadian et al. (2017) used LR = 1×10^{-4}. Zhang et al. (W. Zhang et al., 2019) used LR = 2×10^{-4}. Lam et al. (2018) used LR = 2×10^{-3}. Xu et al. (2018) used LR = 0.01. Choi et al. (2017a) used LR = 1×10^{-6}. Wan et al. (2018) used LR = [0.1-0.0001]. Gao et al. (2019) used LR = 1×10^{-5}. Not all the authors reported the optimizer that they used, or the LR used.

The optimizer choice and the learning rate can play a vital role in network performance and convergence time, especially when using transfer learning. Optimizers like SGD and SGDM can take a longer time to reach convergence, while the RMSProp optimizer can take a shorter time but might not reach the same performance as SGD and SGDM. The Adam optimizer can reach the performance of SGD and SGDM while taking a shorter time, like RMSProp. The learning rate is significant because the choice of a high learning rate can completely change the pre-trained weights, thus deteriorating the network's performance. On the other hand, with a low learning rate value, the network weights will be adjusted to the new dataset without completely change the original weights. A summary of the optimizers that were found in the reviewed papers is shown in Table 6.

Table 6. Optimizer analysis.

Optimizer	Count
Stochastic gradient descent optimizer (SGD)	5
Stochastic gradient descent optimizer with momentum (SGDM)	3
Adaptive Moment Estimation (Adam)	3
Root Mean Square Propagation (RMSProp)	1

3.6.4. The performance difference by applying transfer learning

The suitability of transfer learning for DR image classification can only be assessed by comparing the architecture that was trained from scratch to its fine-tuned version. Masood et al. (2017) reported that the network accuracy increased from 37.6% to 48.8% by using transfer learning on the InceptionV3 architecture that was trained on the Kaggle dataset. Wan et al. (2018) confirmed the effect of transfer learning on six state-of-the-art architectures that use a full-size Kaggle dataset.

The authors reported that the accuracy increased significantly by using transfer learning, and they also observed that using transfer learning significantly decreased overfitting. Xu et al. (2018) reported that DenseNet architecture's accuracy significantly increased by using transfer learning with a private dataset.

From the results obtained by the previously mentioned studies, we can conclude that transfer learning can significantly contribute to the classification of DR. The DR images are very challenging to classify, and, usually, the DR datasets only have a limited number of images. For this reason, the use of transfer learning is particularly suitable for achieving high accuracies instead of training the networks from scratch.

3.6.5. The fine-tuning technique

Fine-tuning the entire network was the most commonly used method for transfer learning in the reviewed papers. Some novel approaches were introduced, like the Siamese network presented by Zeng et al. (2019), where two networks were used in parallel. Li et al. (2017) compared three different transfer learning techniques: fine-tuning the networks, fine-tuning networks layer-wise, and feature extraction. The highest AUC was achieved by fine-tuning the entire networks. Zeng et al. (2019) reported that they fine-tuned the entire InceptionV3 to suit the Kaggle dataset. Mohammadian et al. (2017) compared the fine-tuning of the last two layers against fine-tuning the last four layers and feature extraction. They confirmed that fine-tuning the last two layers achieved the highest performance for InceptionV3. Lam et al. (2018) froze the weights of AlexNet and GoogLeNet architectures and employed feature extraction. Not all the authors reported the method that they used to fine-tune their architecture, while others stated that they fine-tuned the network without explicitly stating how.

3.6.6. Performance validation

Two main methods are commonly used to validate model performance, namely k-fold validation and splitting the dataset into training and test sets. Depending on the dataset's size, some authors opted to use the test split method (usually with an 80%/20% split). In contrast, other authors used k-fold validation, mainly if the target dataset was small in size. Wang et al. (2018) used k-fold with $k = 5$ to validate their results, taking into account that they only had 166 images in their dataset. Li et al. (2017) used k-fold with $k = 5$, and the sizes of the datasets used were 1200 and 1014. Zeng et al. (2019a) and Mohammadian et al. (2017) used 20% of their dataset to validate their dataset results, which was a full-size Kaggle dataset with a size of 35,128 images. Lam et al. (2018) validated their results by using different test datasets.

3.7. Open questions

This section discusses various challenges that the researchers have not addressed in the previous literature about using transfer learning for DR classification. Further research is needed to improve the performance of the networks and to explore other powerful techniques. Some challenges that deserve further investigation are listed below.

3.7.1. The effect of layer-wise fine-tuning instead of full fine-tuning on DR image classification

One of the main questions of applying transfer learning to DR is how deep to fine-tune the network, considering the size of the DR dataset and the architecture used. This question still needs further studies to understand each layer's effect on the network's performance and determine how deep to fine-tune a CNN. Full fine-tuning can be very computationally expensive, as it requires much time, and it may not always guarantee to converge better than top-layer fine-tuning.

3.7.2. The effect of the optimizer used and the learning rate used in DR image classification

For DR datasets, the optimizer that is used can have a significant impact on the performance of the network and the time needed for convergence. The choice of initial LR is still a very debatable area, especially in fine-tuning. Two questions to be answered are the following: does it depend on the DR dataset's size or not? Do we need different LR for full fine-tuning, for top-layer fine-tuning, and feature extraction?

3.7.3. The effect of the batch size used in DR image classification

The impact of batch size on the fine-tuning process still needs to be investigated in detail because this can have a huge impact on the network’s performance. Additionally, its relationship with the size of the DR dataset and the architecture used deserves further analysis.

3.7.4. The effect of choosing another dataset than ImageNet

ImageNet is the de-facto database for transfer learning because it is trained on millions of images with thousands of classes. What is the effect if the ImageNet was substituted with another large dataset to perform transfer learning for DR datasets? Currently, there is no medical image dataset that can play the same role as the ImageNet dataset. Thus, the medical community's effort would be fundamental to build a vast dataset that can be used to train different architectures designed to address the DR classification task.

3.7.5. The effect of image augmentation

Is image augmentation needed in DR classification? The geometric transformation of DR images, like rotation and transformation, can distort them and mask essential features that the algorithm can use to output the predicted grade. Additionally, the usage of image augmentation with transfer learning for DR needs further investigation because image augmentation was mainly

introduced to mitigate the effect of small datasets. However, transfer learning is used for the same reason.

3.8. Conclusion

The computer-assisted detection of medical images is a recently emerging artificial intelligence application that can save time, money, and the workforce. The main challenge of using CNN in medical image classification is the size of the training dataset, which is typically limited since an experienced doctor must annotate each image and, sometimes, even resort to a second opinion to classify some difficult images. Transfer learning can be a viable option considering its suitability when a limited number of training observations are available to address the image classification task. Thus, transfer learning can play an essential role in the medical field. Complex and deep architectures are being developed to solve tasks related to computer vision. These architectures can be successfully applied to solve the challenges in the field of medical images.

This paper reviewed CNN-based techniques for classifying DR images. Though many novel architectures have been proposed to solve DR classification, the current paper only focused on transfer learning-based methods and how transfer learning can be applied to classify DR images.

Chapter 4. Musculoskeletal Images Classification for Detection of Fractures Using Transfer Learning[2]

Abstract: The classification of musculoskeletal images can be very challenging, mainly when it is being done in the emergency room, where a decision must be made rapidly. The computer vision domain has gained increasing attention in recent years due to its achievements in image classification. The convolutional neural network (CNN) is one of the latest computer vision algorithms that achieved state-of-the-art results. A CNN requires an enormous number of images to be adequately trained, and these are always scarce in the medical field. Transfer learning is a technique that is being used to train the CNN by using fewer images. In this paper, we study the appropriate method to classify musculoskeletal images by transfer learning and by training from scratch. We applied six state-of-the-art architectures and compared their performance with transfer learning and with a network trained from scratch. From our results, transfer learning did increase the model performance significantly, and, additionally, it made the model less prone to overfitting.

4.1. Introduction

Bone fractures are among the most common conditions that are treated in emergency rooms (CDC, 2017). Bone fractures represent a severe condition that could result from an accident or a disease like osteoporosis. The fractures can lead to permanent damage or even death in severe cases. The most common way of detecting bone fractures is by investigating an X-ray image of the suspected organ. Reading an X-ray is a complex task, especially in emergency rooms, where the patient is usually in severe pain, and the fractures are not always visible to doctors. Musculoskeletal images are a subspecialty of radiology, which includes several techniques like X-ray, Computed Tomography (CT), and Magnetic Resonance Imaging (MRI), among others. For detecting fractures, the most commonly used method is the musculoskeletal X-ray image (Tanzi et al., 2020). This process involves the radiologists, who are the doctors responsible for classifying the musculoskeletal images. The emergency physicians, who are the doctors present in the emergency room where any patient with a sudden injury is admitted once arrived at the hospital. Emergency physicians are not very experienced in reading X-ray images like radiologists, and they are prone to errors and misclassifications (Hallas & Ellingsen, 2006; Moonen et al., 2017). Image-classification software can help emergency physicians accurately and rapidly diagnose a fracture (Lindsey et al., 2018), especially in emergency rooms. A second opinion is much needed and, usually, is not available.

Deep learning is a recent breakthrough in the field of artificial intelligence, and it has demonstrated its potential in learning and prioritizing essential features of a given dataset without being explicitly programmed to do so. The autonomous behavior of deep learning makes it particularly suitable in the field of computer vision. The area of computer vision includes several tasks, like image segmentation, image detection, and image classification. Deep learning was successfully applied in many computer vision tasks, like in retinal image segmentation (Almubarak

[2] This chapter has been published in MDPI journal as: Kandel, I.; Castelli, M.; Popovič, A. Musculoskeletal Images Classification for Detection of Fractures Using Transfer Learning. *J. Imaging* 2020, *6*, 127. https://doi.org/10.3390/jimaging6110127

et al., 2020), histopathology image classification (Kandel & Castelli, 2020), and MRI image classification (Farooq et al., 2017), among others.

Focusing on image classification, in 2012, Krizhevsky et al. (2012) proposed a convolutional neural network CNN-based model, and they won a very popular image classification challenge called ILSVRC. Afterward, CNNs gained popularity in the area of computer vision, and it is nowadays considered the state-of-the-art technique for image classification. The process of training a classifier is time-consuming and requires large datasets to be correctly trained. In the medical field, there is always a scarcity of images that can be used to train a classifier, mainly due to the regulations implemented in the medical field. Transfer learning is a technique that is usually used to train CNNs, when there are not enough images available or when obtaining new images is particularly difficult. Transfer learning is about training a CNN to classify large non-medical datasets and then use the weights of such a CNN as a starting point for classifying other target images, in our case, X-ray images.

Several studies addressed the classification of musculoskeletal images using deep learning techniques. Rajpurkar et al. (2017) introduced a novel dataset called MURA dataset that contains 40,005 musculoskeletal images. The authors used DenseNet169 CNN to compare the performance of the CNN against three radiologists. The model achieved an acceptable performance compared to the predictions of the radiologists. Chada (2019) investigated the performance of three state-of-the-art CNNs, namely DenseNet169, DenseNet201, and InceptionResNetV2, on the MURA dataset. The author fine-tuned the three architectures using Adam optimizer with a learning rate of 0.0001. Fifty epochs were used with a batch size of eight images to train the model. The author reported that DenseNet201 achieved the best performance for the humerus images, with a Kappa score of 0.764, and InceptionResNetV2 achieved the best performance for the finger images, with a Kappa score of 0.555.

To demonstrate the importance of deep learning in the emergency room for fracture detections, Lindsey et al. (2018) investigated the usage of CNNs to detect wrist fractures. Subsequently, they measured the radiologists' performance of detecting fractures with and without the help of CNN. The authors reported that, by using a CNN, the performance of the radiologists increased significantly. Kitamura et al. (2019) studied the possibility of detecting ankle fractures with CNNs, using InceptionV3, ResNet, and Xception networks for their experiments. The authors trained a CNN from scratch without any transfer learning, and they used a private dataset and an ensemble of the three architectures and reported an accuracy of 81%.

In this paper, we are extending the work of Rajpurkar et al. (2017) and Chada (2019) by investigating the usage of transfer learning of a CNN to classify X-ray images to detect bone fractures. To do so, we used six state-of-the-art CNN architectures that were previously trained on the ImageNet dataset (an extensive non-medical dataset). To the best of our knowledge, this is the first paper that

performs a rigorous investigation on the use of transfer learning in the context of musculoskeletal image classification. More in detail, we investigate the following:

1. The effect of transfer learning on image classification performance. To do that, we compare the performance of six CNN architectures that were trained on ImageNet to classify fracture images. Then, we train the same datasets with the same networks but without the ImageNet weights.
2. The best classifier that achieves the best results on the musculoskeletal images.
3. The effect of the fully connected layers on the performance of the network. To do that, two fully connected layers were added after each network, and then we recorded their performance. Subsequently, the layers are removed, and the performance of the networks is recorded as well.

The paper is organized as follows: In Section 2, we present the methodology used. In Section 3, we present the results achieved by training the MURA dataset on the considered CNNs. In Section 4, we present a discussion about the results obtained, and we compare them to other state-of-the-art results. In Section 5, we conclude the paper by summarizing the main findings of this work.

4.2. Methodology

In this section, we briefly describe the main methods used in this paper.

4.2.1. Convolutional Neural Networks and Transfer Learning

A convolutional neural network is a feed-forward neural network with at least one convolution layer. A convolution layer is a hidden neural network layer with a convolution operation, where the convolution operation is a mathematical operation used to make use of the spatial information presented in images. Training a CNN requires a significant amount of images, which is one of the most severe limitations in deep learning. In particular, deep learning has an extreme dependence on massive training data compared to traditional machine learning methods. It needs a large amount of data to understand the latent patterns of data. Unfortunately, there are problems in which insufficient training data are an inescapable issue. This may happen in domains in which obtaining new observations are either expensive, time-consuming, or impossible. In these situations, transfer learning provides a suitable way for training a CNN. More in detail, transfer learning is a technique used in the deep learning field to make use of knowledge that different domains can share. According to Pan and Yang (2010), transfer learning can be defined as improving the predictive function, $f_T(.)$ by using the knowledge acquired from the source domain, D_S, into the target domain, D_T. Transfer learning relaxes the hypothesis that the training data must be independent and identically distributed with the test data. This allows us to use transfer learning for training CNNs in a given domain and to use them to subsequently address a problem in which data scarcity is a significant limitation. For

more details on transfer learning, the interested reader is referred to Pan and Yang (Pan & Yang, 2010).

4.2.2. State-of-the-Art Architectures

Many CNNs were introduced to participate in the ILSVRC challenge. In this section, we present different CNNs that are considered in this study.

4.2.2.1. VGG

Simonyan et al. (2014) introduced the VGG network in 2014. The VGG was implemented in many variations like VGG16 and VGG19, which only differ in the number of convolution layers used in each. In this paper, we use VGG19 because it is the largest one, and it usually produces better performance than VGG16. VGG19 consists of 19 convolution layers and one dense layer with 4096 neurons to classify the ImageNet images. For more information, the reader is referred to the corresponding paper (Simonyan & Zisserman, 2014).

4.2.2.2. Xception

Chollet (2017b) introduced a novel architecture called Xception (extreme inception), where the author replaced the conventional convolutional layers with depthwise separable convolutional layers. These modified layers decreased the network parameters without decreasing their capacity, which yielded a robust network with fewer parameters and fewer computational resources needed for training. For more information, the reader is referred to the corresponding paper (Francois Chollet, 2017b).

4.2.2.3. ResNet

ResNet architecture was introduced by He et al. (2015) in 2015. ResNet was developed by exploiting the concept of residual connections. The authors introduced the concept of a residual connection to minimize the effect of the vanishing gradient. The ResNet architecture comes in many variants. In this paper, we use the ResNet50 network, which contains 50 layers. For more information, the reader is referred to the corresponding paper (He et al., 2015).

4.2.2.4. GoogLeNet

GoogLeNet architecture was introduced by Szegedy et al. (2015) in 2015. The authors proposed a novel idea called the inception module, which takes the aspect ratio of each image into account. There are many variants for GoogLeNet architecture, and we use the InceptionV3 network. For more information, the reader is referred to the corresponding paper (Szegedy et al., 2015).

4.2.2.5. InceptionResNet

Längkvist et al. (2014) created a novel architecture called InceptionResNet, where the authors combined the inception module idea from GoogLeNet architecture (Szegedy et al., 2015) with the residual idea from ResNet architecture (He et al., 2015). InceptionResNet is more computationally efficient than both ResNet and Inception architectures and achieved higher results than both on the ImageNet dataset. For more information, the reader is referred to the corresponding paper (Längkvist et al., 2014).

4.2.2.6. DenseNet

Huang et al. (2017) introduced a novel architecture called DenseNet. In this architecture, the convolution blocks are densely connected to each other, and the convolution blocks are concatenated to each other instead of being added like in the ResNet network. For more information, the reader is referred to the corresponding paper (Huang et al., 2017).

4.2.3. Evaluation Metrics

Two evaluation metrics are being used to assess the performance of each network. Accuracy and Kappa. Below, we briefly summarize each metric:

4.2.3.1. Accuracy

This metric quantifies how accurate the classifier is. It is calculated as the number of correctly classified data points divided by the total number of data points. The formula is shown in Equation (1).

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN} \tag{1}$$

where TP stands for true positive, TN stands for true negative, FP stands for false positive, and FN stands for false negative. In the context of this study, images without fractures belong to the negative class, whereas images with a bone fracture belong to the positive class.

4.2.3.2. Kappa

This is an evaluation metric that is usually used to consider the probability of selecting by chance, especially in cases of unbalanced datasets, and it was introduced by Cohen (1960). The upper limit of the Kappa metric is 1, which means that the classifier classified everything correctly. At the same time, the lower bound can go below zero, which indicates the classifier is just classifying by luck. The Kappa formula is presented in Equation (2).

$$Kappa = \frac{Agreement_{Observed} - Agreement_{Expected}}{1 - Agreement_{Expected}} \tag{2}$$

4.2.4. Statistical Analysis

Statistical analysis was performed to assess the statistical significance of the results. We considered a confidence interval with a 95% error rate (95% CI) and a hypothesis test. Two hypothesis tests can be used: ANOVA or Kruskal–Wallis test. The choice of the test mainly depends on the normality of the data under observation. The ANOVA test is a parametric test that assumes that the data have a normal distribution. The null hypothesis of the ANOVA test is that the considered samples have the same mean, and the alternative hypothesis is that the samples have a different mean.

The non-parametric test is the Kruskal–Wallis hypothesis test (Kruskal & Wallis, 1952). This test does not make any assumption on the normality of the data, and it compares the medians of different samples. The null and alternative hypotheses tested are the following:

$$H_0: The\ populations\ medians\ are\ all\ equal$$

$$H_1: The\ populations\ medians\ are\ not\ all\ equal$$

In this paper, we first tested the normality assumption by using the Shapiro–Wilk test. Considering that the test does not allow to reject the alternative hypothesis (i.e., data not normally distributed), the Kruskal–Wallis test was used to test the significance of the results obtained. To make the hypothesis test and to report means and significance errors, each setting was repeated 30 times using different seeds and different validation split. In this way, each approach's stability can be assessed by also mitigating the effect of lucky seeds (Schmidt et al., 2020).

4.2.5. Dataset

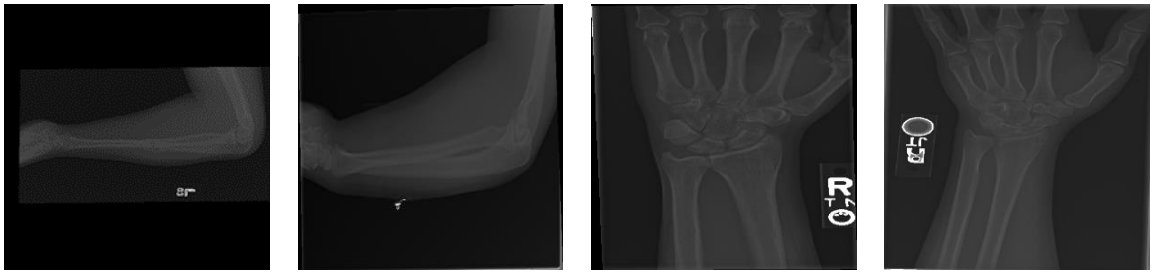
The dataset used in this paper is the publicly available MURA dataset (Rajpurkar et al., 2017). The dataset consists of seven different skeletal bones: elbow, finger, forearm, hand, humerus, shoulder, and wrist. Each category has a binary label, indicating if the image presents a broken bone or not. The dataset contains a total of 40,005 images. The authors of the dataset split it into training and test sets. The train set included 21,935 images without fractures (54.83% of the dataset) and 14,873 images with fractures (37.17% of the dataset), and the test set contained 1667 images without fractures (4.16% of the dataset) and 1530 images with fractures (3.84% of the dataset).

All in all, 92% of the dataset is used for training, and 8% of the dataset is used for testing the results. The summary of the dataset is presented in Table 7. A sample of the MURA dataset is presented in Figure 10.

Table 7. MURA dataset summary.

Category	Training dataset		Test dataset	
	Normal	Fractured	Normal	Fractured
Elbow	2925	2006	235	230
Finger	3138	1968	214	247
Hand	4059	1484	271	189

Humerus	673	599	148	140
Forearm	1164	661	150	151
Shoulder	4211	4168	285	278
Wrist	5765	3987	364	295
Total	21,935	14,873	1667	1530



Broken Forearm Normal Forearm Broken Wrist Normal Wrist

Figure 10. A sample of the MURA dataset.

4.3. Results

Throughout the experiments, all the hyperparameters were fixed. All the networks were either fine-tuned completely or trained from scratch. Adam optimizer (Kingma & Ba, 2014) was used in all the experiments. As noted by studies (Kingma & Ba, 2014; Schmidt et al., 2020), the learning rate should be low to avoid dramatically changing the original weights, so we set the learning rate to be 0.0001. All the images were resized to 96×96 pixels. Binary cross-entropy was used as the loss function because the images are binary classified. An early stopping criterion of 50 epochs would be used to stop the algorithms if no updates happened to the validation score. The batch size was selected to be 64, and the training dataset was split into 80% to train and 20% to validate the results during training. Four image augmentation techniques were used to increase the training dataset's size and make the network more robust against overfitting; the augmentation techniques used are horizontal and vertical flips, 180 rotations, and zooming.

Additionally, image augmentation is performed to balancing the number of images in the two target classes, thus achieving 50% of images without fractures and 50% of images with fractures in the training set. After the training, each network's performance was tested using the dataset that was supplied by the owner and creator of the dataset. The test dataset was not used during the training phase but only in the final testing phase. The hyperparameters used are presented in Table 8. In the following sections, Kappa is the metric considered for comparing the performance of the different architectures.

Table 8. The hyperparameters were used for all the experiments.

Framework	Keras with Python

Optimizer	Adam
Learning Rate	0.0001
Loss Function	Binary Cross-entropy
Early Stopping	50 epochs
Batch Size	64
Validation Split	20%
Image Augmentation	Horizontal flips
	Vertical flips
	180 rotations
	Zooming

4.3.1. Wrist Images Classification Results

Two main sets of experiments were performed: the first consists of adding two fully connected layers after each architecture to act as a classifier block. The second consists of adding only a sigmoid layer after the network. Both the results of the first set and the second set are presented in Table 9.

In the first set of experiments, the fine-tuned VGG19 network had a Kappa score of 0.5989, while the network that was trained from scratch had a score of 0.5476. For the Xception network, the transfer learning score was higher than the one trained from scratch by a large margin. The ResNet50 network performance improved significantly by using transfer learning rather than training it from scratch. This indicates that transfer learning is fundamental for this network, that it could not learn the features of the images from scratch. Both the fine-tuned InceptionV3, InceptionResNetV2, and DenseNet121 networks have a higher score than training them from scratch. Overall, fine-tuning the networks did yield better results than training the networks from scratch. The best performance for the first set of experiments was achieved by fine-tuning the DenseNet121 network.

In the second set of experiments, all the networks' performance increased by fine-tuning than by training from scratch. The ResNet network was the network with the highest difference between fine-tuning and training from scratch. Overall, the best performance for the second set of experiments was achieved by fine-tuning the Xception network. Comparing the first set of experiments to the second set, we see that the best performance for classifying wrist images was the fine-tuned DenseNet121 network with fully connected layers. The presence of fully connected layers did not have any noticeable increase in performance; however, it is worth noting that the ResNet network with fully connected layers did not converge when trained from scratch.

Table 9. Accuracy and Kappa scores of classifying wrist images with and without fully connected layers (±95% CI).

	With FC	**Without FC**

Network	Method	Mean Accuracy	Mean Kappa	Mean Accuracy	Mean Kappa
VGG19	TL	80.45% ± 1.26%	0.5989 ± 2.39%	80.63% ± 1.64%	0.6035 ± 3.33%
	Scratch	78.07% ± 0.94%	0.5476 ± 2.11%	79.89% ± 1.84%	0.5846 ± 3.74%
InceptionV3	TL	79.92% ± 2.07%	0.5886 ± 3.61%	79.94% ± 1.46%	0.5876 ± 2.77%
	Scratch	77.01% ± 2.98%	0.5241 ± 6.49%	77.59% ± 1.51%	0.5389 ± 3.03%
ResNet	TL	78.76% ± 0.88%	0.5647 ± 2.04%	80.85% ± 1.70%	0.6046 ± 3.63%
	Scratch	58.65% ± 8.70%	0.0836 ± 21.31%	70.99% ± 4.20%	0.4018 ± 8.27%
Xception	TL	80.93% ± 0.88%	0.6098 ± 1.69%	81.18% ± 0.47%	0.6133 ± 0.94%
	Scratch	77.44% ± 1.99%	0.5333 ± 4.41%	77.14% ± 1.86%	0.5318 ± 3.32%
DenseNet	TL	81.71% ± 0.94%	0.6245 ± 1.98%	78.76% ± 2.27%	0.5663 ± 4.29%
	Scratch	76.40% ± 2.30%	0.5083 ± 5.17%	76.68% ± 3.62%	0.5214 ± 7.25%
InceptionResNet	TL	80.1% ± 1.66%	0.5917 ± 3.32%	80.55% ± 1.06%	0.6010 ± 2.25%
	Scratch	77.77% ± 1.59%	0.5450 ± 2.85%	78.55% ± 1.70%	0.5580 ± 3.48%

4.3.2. Hand Images Classification Results

As done with the wrist images, two sets of experiments were performed. Both the results of the first set and the second set are presented in Table 10.

In the first set of experiments, for the VGG19 and the ResNet networks, fine-tuning the networks resulted in significantly higher performance than training the networks from scratch. The networks trained from scratch did not converge to an acceptable result. This fact highlights the importance of transfer learning for these networks that cannot learn the images' features from scratch. For the remaining networks, fine-tuning achieved significantly better performance than training the networks from scratch. Overall, all the fine-tuned networks achieved better results than by training from scratch. The best performance of the first set of experiments was obtained with the fine-tuned Xception network.

In the second set of experiments, the performance of all the networks increased by fine-tuning than by training from scratch. The ResNet network was the network with the highest difference between fine-tuning and training from scratch. Overall, the best network was the VGG19 network. Comparing the first set of experiments to the second set, we see that the best performance for classifying hand images was the fine-tuned Xception network with fully connected layers. The presence of fully connected layers did not significantly increase the performance; however, it is essential to point out that the VGG19 network with fully connected layers did not converge when it was trained from scratch.

Table 10. Accuracy and Kappa scores of classifying hand images with and without fully connected layers (±95% CI).

		With FC		Without FC	
Network	Method	Mean Accuracy	Mean Kappa	Mean Accuracy	Mean Kappa
VGG19	TL	70.11% ± 9.23%	0.3089 ± 25.42%	73.04% ± 1.27%	0.3960 ± 3.23%
	Scratch	58.91% ± 0%	0 ± 0%	63.22% ± 5.90%	0.1312 ± 15.90%
InceptionV3	TL	70.25% ± 1.34%	0.3261 ± 3.57%	72.10% ± 0.79%	0.3829 ± 2.15%
	Scratch	66.38% ± 3.80%	0.2382 ± 9.76%	66.56% ± 2.48%	0.2361 ± 6.07%
ResNet	TL	72.25% ± 1.25%	0.3754 ± 3.55%	71.12% ± 1.94%	0.3503 ± 5.44%
	Scratch	59.28% ± 0.93%	0.0103 ± 2.65%	62.10% ± 1.21%	0.0971 ± 3.26%
Xception	TL	75.36% ± 2.56%	0.4621 ± 6.27%	72.50% ± 2.0%	0.3778 ± 5.41%
	Scratch	66.74% ± 2.58%	0.2277 ± 7.20%	66.81% ± 3.60%	0.2334 ± 10.05%
DenseNet	TL	72.21% ± 1.69%	0.3746 ± 4.33%	70.22% ± 3.28%	0.3243 ± 9.03%
	Scratch	63.33% ± 1.16%	0.1308 ± 3.47%	62.79% ± 1.50%	0.1231 ± 4.39%
InceptionResNet	TL	71.96% ± 1.80%	0.3709 ± 4.33%	71.81% ± 1.91%	0.3670 ± 4.65%
	Scratch	68.48% ± 1.53%	0.2788 ± 4.36%	69.09% ± 1.04%	0.3071 ± 2.43%

4.3.3. Humerus Images Classification Results

For the humerus images, the results of both the first and second sets of experiments are presented in Table 11. In the first set of experiments, fine-tuning VGG19 architecture did not converge to any acceptable results, while training the VGG19 from scratch did yield higher performance. For the rest of the networks, fine-tuning did achieve better results than training the networks from scratch. The highest difference was between fine-tuning the ResNet network and training it from scratch. Overall, the best network in the first sets of experiments was the fine-tuned DenseNet network, with a Kappa score of 0.6260.

In the second set of experiments, fine-tuning did achieve better results for all the networks than training the networks from scratch. The best-achieved network was the VGG19 network, with a Kappa score of 0.6333. Comparing the first set of experiments to the second set, we see that the best performance for classifying humerus images was the fine-tuned VGG19 network without fully connected layers. Just as in the previous experiments, the fully connected layers' presence did not provide any significant performance improvement; however, fine-tuning the VGG19 with fully connected layers did not converge compared to fine-tuning the same network without any fully connected layers.

Table 11. Accuracy and Kappa scores of classifying humerus images with and without fully connected layers (±95% CI).

	With FC	**Without FC**

Network	Method	Mean Accuracy	Mean Kappa	Mean Accuracy	Mean Kappa
VGG19	TL	51.39% ± 0%	0 ± 0%	81.66% ± 2.74%	0.6333 ± 5.43%
	Scratch	62.04% ± 3.67%	0.239 ± 8.01%	69.44% ± 8.28%	0.3893 ± 16.77%
InceptionV3	TL	80.32% ± 2.74%	0.6070 ± 5.51%	80.56% ± 1.48%	0.6114 ± 2.94%
	Scratch	67.77% ± 3.12%	0.3603 ± 6.08%	64.06% ± 3.51%	0.2879 ± 6.81%
ResNet	TL	80.38% ± 2.55%	0.6084 ± 5.03%	78.18% ± 2.20%	0.5647 ± 4.31%
	Scratch	54.28% ± 7.46%	0.0849 ± 14.97%	65.63% ± 5.19%	0.3171 ± 10.26%
Xception	TL	80.03% ± 1.92%	0.6010 ± 3.81%	79.75% ± 1.67%	0.5942 ± 3.40%
	Scratch	66.55% ± 2.84%	0.3386 ± 5.46%	66.32% ± 4.72%	0.3334 ± 9.11%
DenseNet	TL	81.31% ± 1.88%	0.6260 ± 3.81%	77.84% ± 1.52%	0.5563 ± 3.16%
	Scratch	70.54% ± 5.85%	0.4134 ± 11.37%	71.93% ± 2.74%	0.4406 ± 5.35%
InceptionResNet	TL	78.41% ± 1.84%	0.5697 ± 3.61%	78.76% ± 2.56%	0.5761 ± 5.07%
	Scratch	65.34% ± 3.89%	0.3135 ± 7.61%	65.34% ± 4.37%	0.3139 ± 8.47%

4.3.4. Elbow Images Classification Results

For the elbow images, we performed the same two sets of experiments performed with the previously analyzed datasets. Both the results of the first set and the second set are presented in Table 12. In the first set of experiments, the fine-tuned VGG19 score was less than training the same network from scratch. For the rest of the networks, fine-tuning did achieve higher performance than training the networks from scratch. The ResNet network achieved the highest difference between fine-tuning and training from scratch. Overall, the best network was the fine-tuned DenseNet121, with a Kappa score of 0.6510.

In the second set of experiments, no fully connected layers were added. For all the networks, fine-tuning did achieve higher results than training from scratch. Overall, the best network was the fine-tuned Xception network, with a Kappa score of 0.6711. Comparing the first set of experiments to the second set, we see that the best performance for classifying elbow images was the fine-tuned Xception network without fully connected layers.

Table 12. Accuracy and Kappa scores of classifying elbow images with and without fully connected layers (±95% CI).

		With FC		Without FC	
Network	Method	Mean Accuracy	Mean Kappa	Mean Accuracy	Mean Kappa
VGG19	TL	71.36% ± 16.95%	0.4232 ± 13.01%	81.61% ± 1.56%	0.6316 ± 3.12%
	Scratch	75.81% ± 34.45%	0.5136 ± 26.45%	76.52% ± 13.45%	0.5279 ± 27.33%
InceptionV3	TL	81.72% ± 0.91%	0.6339 ± 1.84%	80.93% ± 2.3%	0.6180 ± 4.6%

	Scratch	77.96% ± 2.92%	0.5583 ± 5.84%	76.09% ± 4.29%	0.5208 ± 8.6%
ResNet	TL	81.79% ± 3.28%	0.6351 ± 6.61%	81.9% ± 2.22%	0.6374 ± 4.44%
	Scratch	56.20% ± 9.60%	0.1161 ± 19.68%	71.04% ± 4.02%	0.4191 ± 8.09%
Xception	TL	82.15% ± 1.21%	0.6425 ± 2.42%	83.58% ± 1.64%	0.6711 ± 3.31%
	Scratch	78.21% ± 2.83%	0.5631 ± 5.70%	78.49% ± 1.88%	0.5690 ± 3.75%
DenseNet	TL	82.58% ± 1.97%	0.6510 ± 3.92%	81.08% ± 2.23%	0.6208 ± 4.47%
	Scratch	75.38% ± 3.36%	0.5060 ± 6.82%	73.84% ± 4.70%	0.4754 ± 9.49%
InceptionResNet	TL	80.82% ± 0.61%	0.6159 ± 1.21%	80.47% ± 1.68%	0.6087 ± 3.37%
	Scratch	79.82% ± 1.45%	0.5955 ± 2.91%	78.49% ± 1.28%	0.5694 ± 2.58%

4.3.5. Finger Images Classification Results

As with the previous datasets, two main sets of experiments were performed. Both the results of the first set and the second set are presented in Table 13. In the first set of experiments, fine-tuning achieved better results than training the networks from scratch for all the networks. The best-achieved network was the fine-tuned VGG19, with a Kappa score of 0.4379. In the second set of experiments, fine-tuning produced better results than training from scratch for all the six networks. The best network was the fine-tuned InceptionResNet network, with a Kappa score of 0.4455. Comparing the first set of experiments to the second set, we see that the best performance for classifying finger images was the fine-tuned InceptionResNet network without fully connected layers. Moreover, in this case, the presence of the fully connected layers did not provide any significant advantage in terms of performance.

Table 13. Accuracy and Kappa scores of classifying finger images with and without fully connected layers (±95% CI).

		With FC		**Without FC**	
Network	**Method**	**Mean Accuracy**	**Mean Kappa**	**Mean Accuracy**	**Mean Kappa**
VGG19	TL	71.4% ± 1.84%	0.4379 ± 3.59%	68.44% ± 2.76%	0.3847 ± 5.10%
	Scratch	66.78% ± 2.83%	0.3505 ± 5.22%	66.16% ± 3.10%	0.3413 ± 5.26%
InceptionV3	TL	67.68% ± 2.15%	0.3686 ± 3.87%	68.55% ± 3.19%	0.3834 ± 5.97%
	Scratch	63.52% ± 2.66%	0.2911 ± 4.82%	63.88% ± 2.15%	0.2916 ± 4.09%
ResNet	TL	70.17% ± 1.16%	0.4129 ± 2.17%	69.02% ± 2.52%	0.3900 ± 4.44%
	Scratch	60.34% ± 8.07%	0.2341 ±13.90%	66.41% ± 2.98%	0.3431 ± 5.68%
Xception	TL	71.37% ± 2.43%	0.4369 ± 4.42%	70.64% ± 2.27%	0.4234 ± 4.28%
	Scratch	64.75% ± 2.79%	0.3109 ± 5.33%	64.57% ± 2.72%	0.3055 ± 5.31%
DenseNet	TL	66.78% ± 2.64%	0.3552 ± 4.66%	66.81% ± 1.92%	0.3512 ± 3.61%

	Scratch	62.18% ± 2.10%	0.2692 ± 3.60%	64.32% ± 3.16%	0.3051 ± 5.64%
InceptionResNet	TL	70.97% ± 2.05%	0.4294 ± 3.80%	71.8% ± 1.49%	0.4455 ± 2.88%
	Scratch	64.64% ± 3.07%	0.3112 ± 5.70%	65.29% ± 3.09%	0.3204 ± 5.72%

4.3.6. Forearm Images Classification Results

As with the previous datasets, two sets of experiments were performed on the forearm images dataset. Both the results of the first set and the second set are presented in Table 14. In the first set of experiments, fine-tuning all the networks produced better results than training from scratch. Training of the ResNet network from scratch did not yield any satisfactory results, which can imply that fine-tuning this network was crucial for obtaining a good result. The best network was the DenseNet121 network, with a Kappa score of 0.5851.

Moreover, in the second set of experiments, fine-tuning achieved better results than training from scratch. The best network was the fine-tuned ResNet network, with a Kappa score of 0.5673. Comparing the first set of experiments to the second set, we see that the best performance for classifying forearm images was the fine-tuned DenseNet network with fully connected layers. As observed in other datasets, the presence of the fully connected layers did not have any significant advantage in terms of performance.

Table 14. Accuracy and Kappa scores of classifying forearm images with and without fully connected layers (± 95% CI).

		With FC		**Without FC**	
Network	**Method**	**Mean Accuracy**	**Mean Kappa**	**Mean Accuracy**	**Mean Kappa**
VGG19	TL	77.02% ± 1.27%	0.5408 ± 2.54%	76.3% ± 2.16%	0.5264 ± 4.31%
	Scratch	64.29% ± 9.50%	0.2870 ± 18.91%	71.15% ± 6.41%	0.4237 ± 12.75%
InceptionV3	TL	76.52% ± 1.46%	0.5308 ± 2.91%	77.46% ± 1%	0.5496 ± 1.99%
	Scratch	64.84% ± 5.50%	0.2973 ± 11.01%	65.84% ± 4.95%	0.3171 ± 9.89%
ResNet	TL	74.7% ± 1.20%	0.4943 ± 2.38%	78.35% ± 3.46%	0.5673 ± 6.92%
	Scratch	50.11% ± 0.56%	0.0055 ± 1.11%	63.79% ± 3.58%	0.2764 ± 7.12%
Xception	TL	75.08% ± 1.84%	0.5022 ± 3.69%	76.08% ± 1.66%	0.5222 ± 3.32%
	Scratch	66.00% ± 2.54%	0.3204 ± 5.08%	65.73% ± 5.31%	0.3148 ± 10.64%
DenseNet	TL	79.24% ± 0.53%	0.5851 ± 1.05%	76.14% ± 2.32%	0.5232 ± 4.63%
	Scratch	68.77% ± 3.75%	0.3755 ± 7.51%	69.66% ± 3.12%	0.3935 ± 6.25%
InceptionResNet	TL	74.7% ± 2.64%	0.4945 ± 5.27%	74.86% ± 1.98%	0.4977 ± 3.95%
	Scratch	65.45% ± 3.36%	0.3096 ± 6.69%	69.1% ± 6.07%	0.3824 ± 12.12%

4.3.7. Shoulder Images Classification Results

In the first set of experiments, the VGG19 network did not converge to an acceptable result by using both methods. For the rest of the networks, fine-tuning the networks achieved better results than training the networks from scratch. The best network was the fine-tuned Xception network, with a Kappa score of 0.4543. Both the results of the first set and the second set are presented in Table 15. In the second set of experiments, training the ResNet network from scratch achieved slightly better results than fine-tuning. For the rest of the networks, fine-tuning achieved better results. The best network was the fine-tuned VGG19, with a Kappa score of 0.4502. Comparing the first set of experiments to the second set, we see that the best performance for classifying shoulder images was the fine-tuned Xception network with fully connected layers. The presence of the fully connected layers did not show any significant advantage in terms of performance. Anyhow, the VGG19 network with fully connected layers did not converge to any satisfactory result compared to the same network without any fully connected layers.

4.3.8. Kruskal–Wallis Results

We applied the Kruskal–Wallis test to assess the statistical significance of different settings. The Kruskal–Wallis test yielded a p-value < 0.05 for all the results, which indicates to reject the null hypothesis that the settings have the same median and to accept the alternative hypothesis that there is a statistically significant difference between different settings (transfer learning "with and without fully connected layers" vs. training from scratch "with and without fully connected layers").

Table 15. Accuracy and Kappa scores of classifying shoulder images with and without fully connected layers (±95% CI).

		With FC		**Without FC**	
Network	**Method**	**Mean Accuracy**	**Mean Kappa**	**Mean Accuracy**	**Mean Kappa**
VGG19	TL	50.62% ± 0%	0 ± 0%	72.53% ± 1.75%	0.4502 ± 3.45%
	Scratch	50.62% ± 0%	0 ± 0%	54.80% ± 10.73%	0.0855 ± 21.98%
InceptionV3	TL	69.60% ± 2.20%	0.3936 ± 4.37%	68.65% ± 2.32%	0.3748 ± 4.52%
	Scratch	66.46% ± 2.44	0.3306 ± 4.80%	65.28% ± 3.53%	0.3077 ± 6.86%
ResNet	TL	68.92% ± 1.48%	0.3801 ± 2.93%	67.29% ± 4.06%	0.3477 ± 8.03%
	Scratch	52.16% ± 4.30%	0.0330 ± 9.11%	67.91% ± 3.40%	0.3585 ± 6.72%
Xception	TL	72.68% ± 1.17%	0.4543 ± 2.33%	72.14% ± 1.78%	0.4440 ± 3.53%
	Scratch	67.67% ± 1.65%	0.3543 ± 3.23%	66.22% ± 3.82%	0.3258 ± 7.51%
DenseNet	TL	70.60% ± 1.79%	0.4130 ± 3.51%	70.37% ± 2.54%	0.4085 ± 5.04%
	Scratch	68.92% ± 1.61%	0.3768 ± 3.20%	65.45% ± 3.47%	0.3093 ± 6.74%
InceptionResNet	TL	70.57% ± 1.91%	0.4125 ± 3.77%	71.43% ± 2.18%	0.4295 ± 4.32%
	Scratch	66.79% ± 2.87%	0.3372 ± 5.63%	67.02% ± 3.86%	0.3419 ± 7.59%

4.4. Discussion

In this paper, we compared the performance of fine-tuning on six state-of-the-art CNNs to classify musculoskeletal images. Training a CNN network from scratch can be very challenging, especially in the case of data scarcity. Transfer learning can help solve this problem by initiating the weights with values learned from a large dataset instead of initializing the weights from scratch. Musculoskeletal images play a fundamental role in classifying fractures. However, these images are always challenging to be analyzed, and a second opinion is often required, which will not always be available, especially in the emergency room. As pointed out by Lindsey et al. (Lindsey et al., 2018), the presence of an image classifier in the emergency room can significantly increase physicians' performance in classifying fractures.

For the first research question about the effect of transfer learning, we noted that transfer learning produced better results than training the networks from scratch. For our second research question, the classifier that achieved the best result for wrist images was the fine-tuned DenseNet121 with fully connected layers; the classifier that achieved the best performance for elbow images was the fine-tuned Xception network without fully connected layers; for finger images, the best classifier was the fine-tuned InceptionResNetV2 network without fully connected layers; for forearm images, the best classifier was the fine-tuned DenseNet network with fully connected layers; for hand images, the best classifier was a fine-tuned Xception network with fully connected layers; the best classifier for humerus images was the fine-tuned VGG19 network without fully connected layers; finally, the best classifier for classifying the shoulder images was the fine-tuned Xception network with fully connected layers. A summary of the best CNNs is presented in Table 16. Concerning the third research question, the fully connected layers had a negative effect on the performance of the considered CNNs. In particular, in many cases, it decreased the performance of the network. Further research is needed to study, in more detail, the impact of fully connected layers, especially in the case of transfer learning.

Table 16. The best convolutional neural network (CNN) for each image category.

Fracture	CNN
Wrist	DenseNet
Elbow	Xception
Finger	InceptionResNetV2
Forearm	DenseNet121
Hand	Xception
Humerus	VGG19
Shoulder	Xception

The authors of the MURA dataset (Rajpurkar et al., 2017) assessed the performance of three radiologists on the dataset and compared their performance against the one of a CNN. In Table 17, we present their results, along with our best scores.

The first radiologist achieved the best score for classifying elbow images, and our score was comparable to other radiologists (Rajpurkar et al., 2017). For finger images, our score was higher than the three radiologists. For forearm images, our score was lower than the radiologists. For hand images, our score was the lowest. For humerus images, shoulder images, and wrist images, our score was lower than the radiologists. We still believe that the scores achieved in this paper are promising, keeping in mind that these scores came from off-the-shelf models that were not designed for medical images in the first place and that the images were resized to be 96×96 pixels due to hardware limitations. Nevertheless, additional efforts are needed to outperform the performance of experienced radiologists.

Table 17. Kappa scores of three radiologists reported in Reference (Rajpurkar et al., 2017) compared to our results.

Fracture	1st Radiologist	2nd Radiologist	3rd Radiologist	Our Score
Elbow	0.850	0.710	0.719	0.671
Finger	0.304	0.403	0.410	0.445
Forearm	0.796	0.802	0.798	0.585
Hand	0.661	0.927	0.789	0.462
Humerus	0.867	0.733	0.933	0.633
Shoulder	0.864	0.791	0.864	0.454
Wrist	0.791	0.931	0.931	0.625

On the other side of the spectrum, there is the study of Raghu et al. (2019), where the authors argued that transfer learning is not good enough for medical images and will be less accurate compared to training from scratch or compared to novel networks explicitly designed for the problem at hand. The authors studied the effect of transfer learning on two medical datasets, namely, retina images and chest X-ray images. The authors stated that designing a lightweight CNN can be more accurate than using transfer learning. In our study, we did not consider “small” CNN's trained from scratch. Thus, it is not possible to directly compare the results obtained to the ones presented in Raghu et al. (2019). Anyway, more studies are needed to better understand the effect of transfer learning on medical image classification.

4.5. Conclusion

In this paper, we investigated the effect of transfer learning on classifying musculoskeletal images. We find that out of the 168 results obtained that were performed by using six different CNN architectures and seven different bone types, transfer learning achieved better results than training a CNN from scratch. Only in 3 out of the 168 results did training from scratch achieve slightly better results than transfer learning. The weaker performance of the training-from-scratch approach could be related to the number of images in the considered dataset, as well as to the choice of the hyperparameters. In particular, the CNN's taken into account are characterized by the presence of a large number of trainable parameters (i.e., weights), and the number of images used to train these networks is too small to build a robust model. Concerning the hyperparameters, we highlight the importance of the learning rate. While we used a small value of the learning rate in the fine-tuning approach to avoid changing the architectures' original weights dramatically, the training-from-scratch approach could require a higher value of the learning rate. A complete study on the hyperparameters' effect will be considered in future work, aiming to fully understand the best approach to be used when dealing with fracture images. Focusing on this study's results, it is possible to state that transfer learning is recommended in the context of fracture images. In our future work, we plan to introduce a novel CNN to classify musculoskeletal images, aiming at outperforming fine-tuned CNNs. This would be the first step towards the design of a CNN-based system, which classifies the image and provides the probable position of the fracture if the fracture is present in the image.

Chapter 5. A Novel Architecture to Classify Histopathology Images Using Convolutional Neural Networks[3]

Abstract: Histopathology is the study of tissue structure under the microscope to determine if the cells are normal or abnormal. Histopathology is a very important exam that is used to determine the patients' treatment plan. The classification of histopathology images is challenging to even an experienced pathologist, and a second opinion is often needed. Convolutional neural network (CNN), a particular type of deep learning architecture, obtained outstanding results in computer vision tasks like image classification. In this paper, we propose a novel CNN architecture to classify histopathology images. The proposed model consists of 15 convolution layers and two fully connected layers. A comparison between different activation functions was performed to detect the most efficient one, taking into account two different optimizers. To train and evaluate the proposed model, the publicly available PatchCamelyon dataset was used. The dataset consists of 220,000 annotated images for training and 57,000 unannotated images for testing. The proposed model achieved higher performance compared to the state-of-the-art architectures with an AUC of 95.46%.

5.1. Introduction

Cancer, also called malignancy and neoplasms, is an abnormal growth of cells in a multistage process that generally progresses from a pre-cancerous lesion to a malignant tumor, which can then invade adjoining parts of the body and spread to other organs. It is considered the second leading cause of death globally, and it is responsible for an estimated 9.6 million deaths in 2018 ("Cancer," 2018; Siegel et al., 2020). There are more than 100 types of cancer, including breast cancer, skin cancer, lung cancer, colon cancer, prostate cancer, and lymphoma (Siegel et al., 2020). Through the blood and the lymph systems of the body, cancer can spread across the entire body affecting different organs. It is perceived to be the result of the interaction between a person's genetic factors and exposure to specific environmental external agents, known as carcinogens. Usually, the definitive cancer diagnosis is through histopathology, the study of the tissue structure. Histopathology is done by studying the tissues under a microscope to detect any cell alterations. It is one of the essential steps in the treatment plan because it can help in the early detection of cancer (He et al., 2012). Histopathology study is a very time-consuming process and requires a very experienced pathologist. Due to its difficulty, a second opinion from another pathologist is often needed, especially for certain kinds of tumors. Even very experienced pathologists can disagree with each other in the classification of the histopathology images (Robbins et al., 1995). Moreover, the number of active pathologists decreased dramatically in the last decade. For example, in the US, the number of pathologists decreased by 17.5%, which led to an increase in the workload by more than 40% (Metter et al., 2019). Currently, the used microscopes are digital, meaning that they can produce a digital image that can be viewed and stored on computers. These digital images can be used to make an automatic classifier.

[3] This chapter has been published in MDPI Journal as Kandel, I.; Castelli, M. A Novel Architecture to Classify Histopathology Images Using Convolutional Neural Networks. *Appl. Sci.* **2020**, *10*, 2929. https://doi.org/10.3390/app10082929

The automatic classification of histopathology will save a lot of time and can give a second opinion to the pathologists.

Deep learning architectures have demonstrated their suitability to successfully address optimization problems over different domains. A convolutional neural network (CNN) corresponds to a particular type of deep learning model, and it was proposed by LeCun et al. (1989) to address problems in the computer vision domain. The real advancement happened in 2012 when Krizhevsky et al. (2012) won the ILSVRC challenge (Russakovsky et al., 2015) with an accuracy of 84.6%. Since then, CNNs are considered state-of-the-art models for image classification. CNNs have been successfully applied in different fields like traffic sign classification (Luo et al., 2018; Sermanet & LeCun, 2011), text classification (Kim, 2014; Zhang et al., 2015), speech recognition (Abdel-Hamid et al., 2014; Abdel-Hamid et al., 2012), and machine translation (Gehring et al., 2017; Gehring et al., 2017).

In this work, we present a novel CNN architecture to classify lymph node stained histopathology images. The publicly available PatchCamelyon dataset (Bejnordi et al., 2017; Veeling et al., 2018) was used to train and test the proposed architecture. The main contributions of this paper are as follows:

- We propose a novel CNN architecture that can classify histopathology images with high accuracy.
- We investigate the impact of dropout layers and the impact of the location of the normalization layer.
- We test six activation functions to study their impact on the proposed architecture, rather than choosing the de-facto ReLU activation function.
- We study the impact of two different optimizers on CNN performance.
- We consider four popular state-of-the-art CNNs to compare the performance of our model. These CNNs were trained on the PatchCamelyon dataset. These results can be used by researchers instead of training these models again from scratch, which can take hours (if not days), especially in the lack of computational power.

The rest of this paper is organized as follows: In Section 2, we review the previous studies that introduced novel CNN architectures. The proposed methodology is stated in Section 3. The obtained results are shown in Section 4. The discussion and conclusion are presented in Section 5 and Section 6, respectively.

5.2. Literature Review

In recent years, many algorithms were introduced to help in classifying histopathology images. Nowadays, CNNs are considered as the state-of-the-art algorithm for classifying images. In 1989, Lecun et al. (1998) presented the first CNN with 5 convolution layers. Before Lecun's work, CNN

was not a popular choice for image classification because of the computational cost and the (small) size of the available datasets. In 2012, Krizhevsky et al. (2012) re-introduced CNNs by winning the ImageNet challenge with their AlexNet CNN that obtained a 16% classification error rate compared to 25% of the second-place model. Since then, CNN became the de-facto algorithm for image classification, and many CNNs were subsequently defined.

To address an image classification task by using CNNs, it is possible to rely on two different approaches: use a pre-existing architecture that was developed to classify natural images or, as done in this paper, develop a novel architecture. This section focuses on novel architectures that were introduced mainly to classify histopathology images. We summarize the recent contributions in the area of histopathology images in Table 18, while Table 19 reports the important information associated with the architectures presented in the papers of Table 18. For a complete description of the architectures considered in Tables 7 and 8, the reader is referred to their respective paper.

Table 18. Recent studies in the area of histopathological image classification.

Authors	Main Contribution	Number of Classes	Metric	Test Set Performance
Nguyen et al. (2019)	• A CNN is used to classify breast cancer images belonging to 8 classes (four benign subclasses and four malignant subclasses). • This was the first attempt to classify breast cancer images in eight subclasses.	8	Accuracy	73.68%
Bayramoglu et al. (2016)	• A single task CNN is used to predict malignancy from breast cancer images. • A multi-task CNN is used to predict both malignancy and image magnification levels simultaneously (for a total of eight classes). • Efficient method that allows to use new data with the same or different magnification levels than previous data.	2 for the malignant/ not malignant task 8 for the magnification task	Accuracy	83.25% 80.10%
Arjmand et al. (2019)	• Fully automated diagnostic tool for non-alcoholic fatty liver disease classification, based on an optimized CNN architecture.	2	Accuracy	95%
Sirinukunwattana et al. (2016)	• Deep learning model for nucleus detection and classification from histology images of colorectal adenocarcinomas. • Novel Neighbouring Ensemble Predictor (NEP) coupled with CNN to more accurately predict the class label of detected cell nuclei.	4	F1 Score	0.692 (This is the combined performance on nucleus detection and classification)
Lai et al. (2018)	• Deep learning model that integrates Coding Network with Multilayer Perceptron (CNMP).	2	Accuracy	90.1% and 90.2% on two benchmark

	• Combination of high-level features that are extracted from a deep convolutional neural network with traditional features of an image that can be extracted using simple image analysis concepts.			medical image datasets
Basha et al. (2018)	• Definition of a CNN architecture for the classification of histological routine colon cancer nuclei. • Significant reduction of the number of learnable parameters compared to the popular CNN models such as AlexNet, and GoogLeNet.	4	F1 Score	0.7887

Table 19. Information related to the architectures defined in the papers identified in Table 18.

	Conv Layers	FC layers	Dropout Layers	Normalization Layers	Activation Function	Pooling Layers
Nguyen et al. (2019)	5	1	3	6	LeakyReLU	3
Bayramoglu et al. (2016)	3	2	2	2	ReLU	3
Arjmand et al. (2019)	3	1	2	3	ReLU	2
Sirinukunwattana et al. (2016)	2	2	0	0	ReLU	2
Lai et al. (2018)	6	0	0	0	ReLU	2
Basha et al. (2018)	4	2	2	6	ReLU	2

To assess the performance of the model proposed in this paper, we perform a comparison against the performance of three state-of-the-art models, namely, VGG, InceptionV3, and ResNet architectures. The following paragraphs provide a short description of these models. For a complete description, the reader is referred to the papers where these models were originally presented.

5.2.1. VGG Architectures

Simonyan et al. (2014) proposed a novel CNN called VGG, which achieved an 8.1% error rate, a great achievement compared to the AlexNet network. In particular, two main architectures were introduced: VGG16 and VGG19. The main difference among them is in the number of convolution layers. VGG16 consisted of 13 convolution layers and 3 fully connected layers, while VGG19 considered 16 convolution layers and 3 fully connected layers. All the convolution layers have a 3×3 kernel size, with the number of kernels ranging from 64 till 512. VGG16 can be divided into 5 convolution blocks, where each block contains three convolution layers, followed by a max-pooling

layer. VGG16 with fully connected layers has 138 million parameters, whereas VGG19 with fully connected layers has 144 million parameters.

5.2.2. InceptionV3 Architecture

InceptionV3 is a recent CNN architecture with 22 layers that was introduced by Szegedy et al. (2016). The main difference between this architecture to others is the fact that it connects the convolutions in parallel instead of connecting them sequentially. The authors named this module the inception module. The point of the inception module is to process the images at different scales. The InceptionV3 architecture consists of 9 inception modules and one fully connected layer. In total, InceptionV3 has 23.8 million parameters.

5.2.3. ResNet Architecture

He et al. (2016) noticed that the CNN accuracy would get saturated as soon as the model gets 30 layers deep. The main reason for that saturation is the vanishing gradients problem, where the bottom layers of the CNN will stop being updated. The authors introduced ResNet architecture that could overcome the problem of vanishing gradients. The main difference between ResNet architecture and other networks is the residual connection, where this connection will skip a few convolution layers at a time. ResNet architecture won the ImageNet challenge in 2015 with an error rate of 3.57% that surpassed the human error rate for the first time. In total, ResNet has 25 million parameters.

5.3. Methodology

This section describes the proposed CNN and the dataset used in this study.

5.3.1. Proposed Architecture

The proposed CNN was chosen by analyzing the extant literature in the area, thus exploiting the contributions of different works. The main inspiration of the proposed model is the VGG16 architecture (2014), with the main differences being (1) the use of normalization layer in every convolution block, (2) the use of three convolution layers in every block instead of four, and (3) the use of fully connected layers with 512 neurons instead of 4096. The proposed architecture is shown in Figure 11. In the proposed architecture, the input image size is 96×96 pixels. The architecture has 5 convolution blocks that act as feature extractors and one fully connected block that acts as a classifier. In the first convolution block, three convolution layers followed by a batch normalization layer and a max-pooling layer are used. The padding is kept the same for the three convolution layers to make use of every pixel, especially in the first convolution block and, for the same reason, the stride is set to 1. For the three convolution layers, 32 kernels were used with a size of 3×3. To reduce the dimensions by two to keep the most relevant features obtained from the first convolution block, a max-pooling layer is added. The second, third, and fourth convolution blocks use the same hyperparameters used in the first convolution block, except for the number of kernels used, which

was set to 64, 128, and 256, respectively. In the fifth convolution block, the second and the third convolution layers have no padding to decrease the spatial information with 512 kernels. The output of the convolution blocks is flattened and, subsequently, it becomes the input of the classifier block. The classifier block has two fully connected layers, each with 512 neurons. Overall, the inputs to the convolution blocks are $96 \times 96 \times 3$, $48 \times 48 \times 32$, $24 \times 24 \times 64$, $12 \times 12 \times 128$, and $6 \times 6 \times 256$, respectively. The final layer is a sigmoid function that is used to classify into two classes. In total, the proposed architecture consists of 15 convolution layers, 2 fully connected layers, 5 pooling layers, and 5 normalization layers.

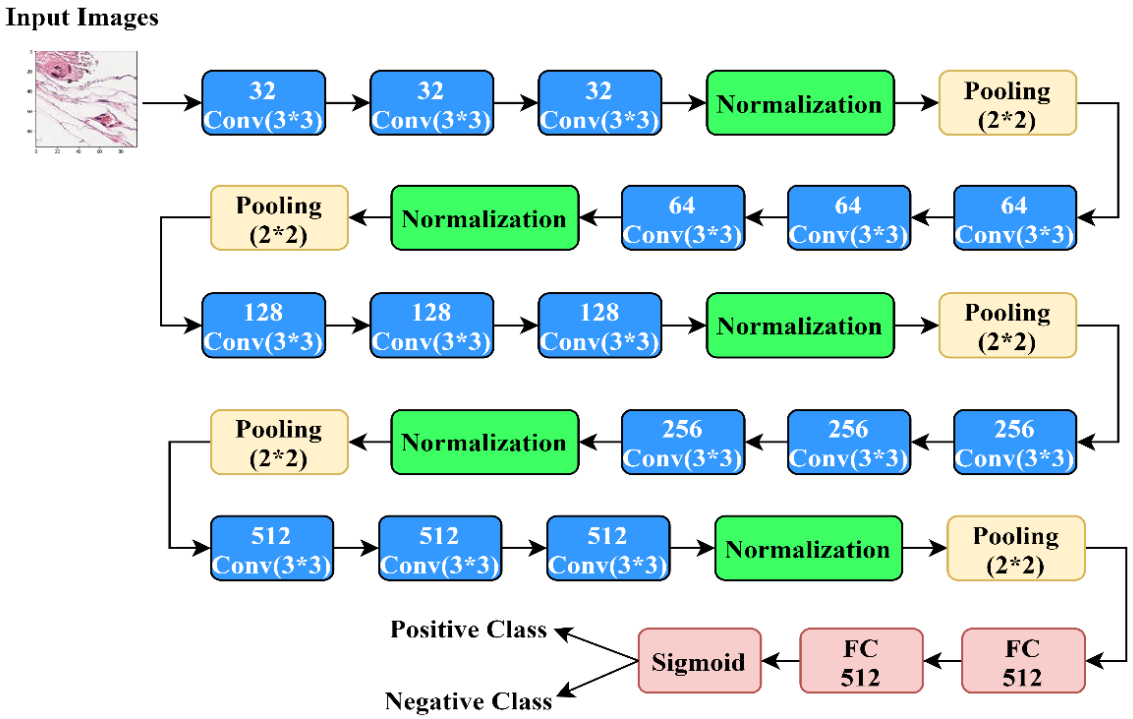


Figure 11. A schematic diagram of the proposed network architecture.

Activation functions have a huge impact on the speed and the accuracy of the networks, and that is why, instead of using the ReLU activation function, which is considered as the de-facto activation function, five additional activation functions were tested to determine the best for the proposed architecture.

Four designs were tested to determine the optimal architecture. The first design is to test the effect of dropout layers in the classifier block, where a dropout layer with a dropout ratio of 50% will be inserted after each fully connected layer. The second design is to test the performance of the architecture without the dropout layers. The third design is to test the performance of the CNN after placing the normalization after the activation function. A dropout layer with a 50% dropout ratio will be added after each fully connected layer. The fourth design is the same as the third design but without dropout layers. Different designs are shown in Figure 12.

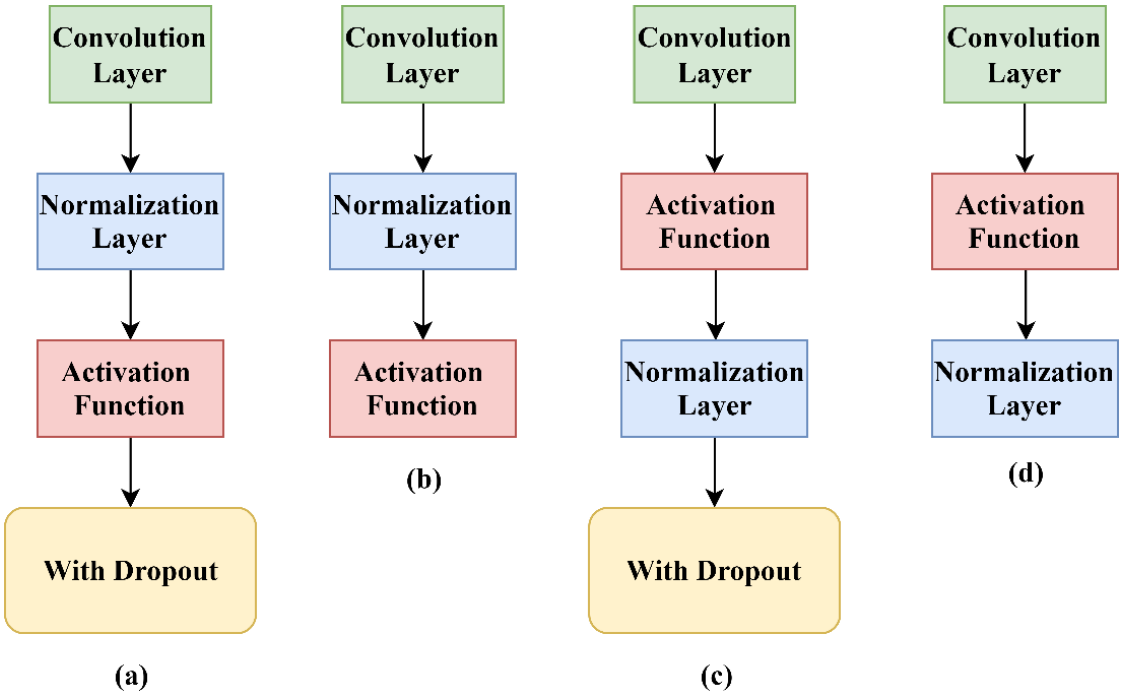


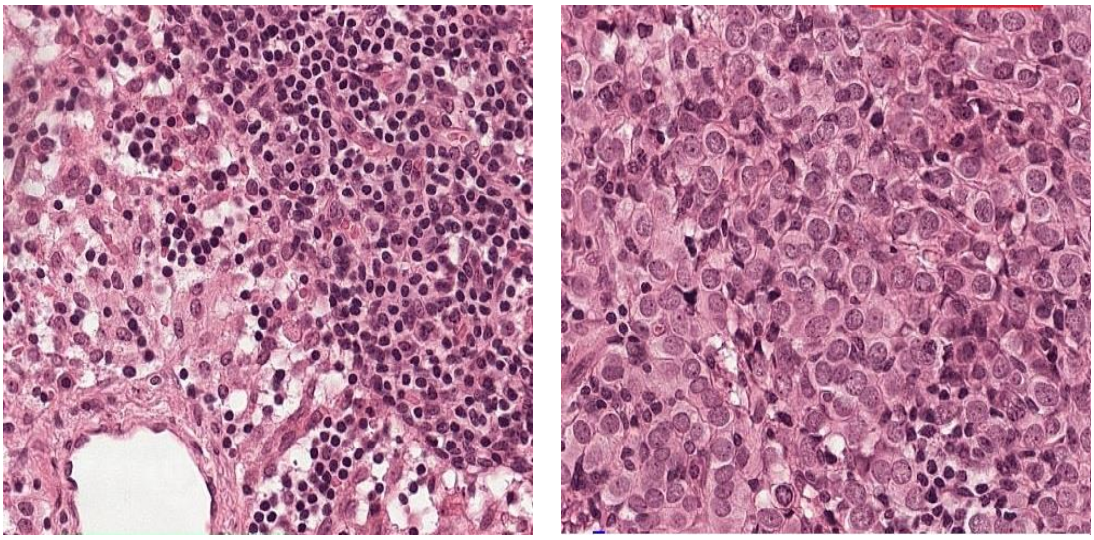
Figure 12. Different designs tested: (a) shows the first design, the proposed architecture with the normalization layer placed before the activation function, and two dropout layers are placed in the classifier block; (b) shows the second design, which is similar.

5.3.2. Dataset

As described in Bejnordi et al. (2017), the Camelyon dataset was sampled from 399 patients from two hospitals in the Netherlands. The dataset was annotated with the help of experienced pathologists from the Netherlands. In particular, the dataset labels were manually annotated by two students and, subsequently, were checked in deep detail by two experienced pathologists. To check the performance of the pathologists on this dataset, two sets of experiments were performed. The first experiment was performed without any time constrain, and 11 experienced pathologists were asked to annotate a first subset of the images. In the second experiment, a two-hour time limit was given to the experts for annotating a second subset of images. The challenge organizers chose the AUC of the ROC curve as an evaluation criterion for this competition; thus, every score is presented in terms of the AUC of the ROC curve. The AUC score achieved by the first experiment was 96.6%, and the score of the second experiment was 81%.

The PatchCamelyon dataset (Bejnordi et al., 2017; Veeling et al., 2018) is an extension of the Camelyon dataset, which contains 277,000 histopathology images with an image size of 96×96 pixels at 10x magnification. A total of 220,000 images are annotated with 60% positive cases and 40% negative cases. A total of 57,000 images are un-annotated images to test the classifier, and there are no duplicates in the PatchCamelyon dataset. The resulting model is subsequently used to predict the labels of the test set and finally uploaded to the Kaggle platform to obtain the model AUC. This process was necessary since the test labels are available only to the owner of the data. To increase the

size of the dataset and to make the model more robust to overfitting, the following augmentation techniques were applied: horizontal flip, vertical flip, rotation range, zoom range, width shift range, height shift range, shear range, and channel shift range. Figure 13 shows two images of the considered dataset.



(**a**) **Normal** (**b**) **Cancerous**

Figure 13. Samples of the PatchCamelyon dataset. (a) A normal sample. (b) A cancerous sample.

5.4. Results

This section presents the hyperparameters used in the experiments, the results obtained using different activation functions, the results of using different designs, and the results obtained using different state-of-the-art CNN architectures.

5.4.1. Experimental Setup

Six activation functions were used to determine the optimum for the proposed architecture. The tested activation functions are Tanh, Sigmoid, ReLU (Nair & Hinton, 2010), LeakyReLU (Maas et al., 2013), ELU (Clevert et al., 2016), and SELU (Klambauer et al., 2017). Two optimizers were used as well, namely, Adam (Kingma & Ba, 2014) and RMSProp (Ruder, 2016). The PatchCamelyon dataset was divided into 80%/20% for training and validation. For all the experiments, the batch size used was 64. Because of the size of the training dataset and the computational cost of CNN training, early stopping of 10 epochs was used for all the experiments. According to Shorten and Khoshgoftaar (Shorten & Khoshgoftaar, 2019), image augmentation techniques can be used to increase the size of the dataset. This results in improved model performance and in the reduction of the overfitting that may occur when using small datasets. Thus, we applied image augmentation in all the experiments conducted in this paper, using the transformation specified in Section 3.2. The positive to negative ratio was kept the same after the application of the augmentation techniques. All the experiments

were implemented using Keras API (Chollet et al., 2015) with TensorFlow API (Abadi et al., 2015) in the backend.

5.4.2. Results

This section presents the results obtained and is divided into four parts. In the first part, we present the results obtained from training the proposed architecture using two optimizers and six different activation functions. In the second part, we present the results of the four different designs to enhance the performance of the network. In the third part, we present the results of the comparison of our architecture against state-of-the-art models. In the fourth part, we present the results of the comparison between our architecture and different architectures that were developed for histopathology image classification.

5.4.2.1. The Results of Different Activation Functions

Six different activation functions (and two different optimizers) were used to test the model performance. The sigmoid activation function achieved the lowest accuracy among all the activation functions taken into account over both the optimizers, and so it was removed from other tests. The ReLU activation function achieved a low accuracy compared to the other competitors, with comparable results obtained with the Adam and RMSProp optimizers. The LeakyReLU activation function scored the same as ReLU for the Adam optimizer but achieved a lower accuracy with the RMSprop optimizer. This could indicate that this activation function makes the network slightly unstable. The ELU activation function scored the best accuracy compared to all the functions, with comparable results for Adam and RMSProp. The SELU activation function had a high accuracy with the Adam optimizer, while its performance degraded when using the RMSProp optimizer. The saturated function Tanh scored very high in both the optimizers, where the result of the RMSProp optimizer was higher than the Adam optimizer. Overall, the best achieving activation function was the ELU, followed by SELU and Tanh activation functions. The results are shown in Table 20.

Table 20. The AUC results of both optimizers for the different activation functions.

	ReLU	**LeakyReLU**	**ELU**	**SELU**	**Sigmoid**	**Tanh**
Adam	87,68%	87,37%	**93,66%**	92,73%	84,03%	91,70%
RMSprop	85,01%	83,02%	**92,99%**	88,43%	84,49%	92,00%

5.4.2.2. The Results of Different Designs

Four different designs were tested to check the performance of the network. The first design considers the network with two dropout layers in the classifier block, and the normalization layer before the activation layer. The second design is similar to the first design but without any dropout layers. The third design places the normalization layer after the activation layer and adds dropout layers in the classifier block. The fourth design is similar to the third design but without adding any

dropout layers in the classifier block. Tables 21 and 22 show the results for both the Adam and RMSProp optimizers.

Table 21. The AUC results of the Adam optimizer.

	Tanh	**ReLU**	**LeakyReLU**	**ELU**	**SELU**
*H*1	91.70%	87.68%	87.37%	93.66%	92.73%
*H*2	92.96%	85.45%	87.17%	91.76%	**92.82%**
*H*3	94.39%	**91.78%**	**89.96%**	**94.40%**	90.16%
*H*4	**95.46%**	89.33%	89.11%	93.85%	92.71%

Table 22. The AUC results of the RMSProp optimizer.

	Tanh	**ReLU**	**LeakyReLU**	**ELU**	**SELU**
*H*1	92.00%	85.01%	83.02%	92.99%	88.43%
*H*2	**94.09%**	**89.77%**	90.37%	87.62%	**94.25%**
*H*3	93.35%	88.40%	**90.38%**	92.26%	93.86%
*H*4	93.43%	87.70%	90.11%	**94.86%**	91.58%

The ReLU activation function has the lowest score compared to other activation functions across the four designs. The best performance of ReLU (with Adam optimizer) was obtained in the third design, followed by the fourth design. The worst score was by using the second design. The results with the RMSprop optimizer were poorer than the Adam optimizer. The best result was achieved with the second design, followed by the third design. The lowest result was obtained with the first design. Overall, the best result achieved by ReLU was 91.78%. The LeakyReLU activation function had slightly lower accuracy than the ReLU function. With Adam optimizer, the third design was the best performer, followed by the fourth design. The worse result was obtained with the second design. With the RMSprop optimizer, the best performance was obtained in the third design, followed by the second design. The worst performance was achieved by using the first design. The results of Adam and RMSprop optimizers were significantly different, especially for the first design. Overall, the best result achieved by LeakyReLU was 90.38%.

The ELU activation function had higher results compared to both ReLU and LeakyReLU. With Adam optimizer, the best performance was achieved by using the third design, followed by the fourth design. The lowest performance was achieved with the second design. With the RMSProp optimizer, the best result was achieved by using the fourth design, followed by the first design. The worst result was achieved with the second design. The results of both Adam and RMSprop optimizers were comparable, except for the second design. Overall, the best result achieved by the ELU activation function was 94.86%. The SELU activation function had a slightly lower performance than the ELU activation function. With Adam optimizer, the best performance was achieved by using the second

design, followed by the first design. The lowest performance was obtained with the third design. With the RMSprop optimizer, the best performance was achieved by using the second design, followed by the third design. The lowest performance was noticed with the first design. The results of Adam and RMSprop optimizers were significantly different, especially for the first design. Overall, the best result achieved by the SELU activation function was 94.25%.

The Tanh activation function achieved the highest results compared to all the other tested activation functions. With Adam optimizer, the best performance was obtained with the fourth design, followed by the third design. The lowest performance was achieved with the first design. With the RMSProp optimizer, the best performance was achieved by using the second design, followed by the fourth design. The lowest performance was obtained by using the first design. The results of both optimizers (Adam and RMSProp) were comparable. Overall, the best result achieved by the Tanh optimizer was 95.46%.

Using Adam optimizer, the first design was, overall, similar to the second design, meaning that the presence of the dropout layer did not increase the performance of the model. However, the performance of the third and fourth designs was higher, which indicates that the location of the normalization layer has an impact on the performance of the architecture. There are no significant differences between the third and the fourth designs, which indicates that the presence of the dropout layer does not increase the network performance.

Using the RMSProp optimizer, the performance of the first design was the lowest compared to the other designs. The second design achieved greater accuracy than the first design, which can indicate that the dropout layer can limit overfitting. The second, third, and fourth designs achieved a different performance, thus corroborating the hypothesis that the location of the normalization layer has a significant impact on the performance of the model. The third and the fourth designs were similar as well, indicating that the presence of the dropout layer has no effect on the model accuracy. All in all, based on the aforementioned results, we recommend practitioners to rely on the fourth design.

5.4.2.3. The Results over Benchmark CNN Architectures

In this section, the results of four benchmark CNN architectures are presented. Two sets of experiments were performed to compare our proposed architecture with four popular CNN benchmark architectures, namely, VGG16, VGG19, InceptionV3, and ResNet. The first set of experiments aims at comparing the architectures' performance under the first design (dropout layer in the classifier block). All the original classifier blocks of the CNN were removed and replaced by two fully connected layers with a dropout layer after each fully connected layer, with a dropout probability of 0.5. The second set of experiments compared the performance of the architectures under the fourth design (without a dropout layer in the classifier block). Just like the first set of experiments,

all the original fully connected layers were removed and replaced with two fully connected layers without any dropout layers. All the architectures were trained from scratch (i.e., no transfer learning was used).

In the first sets of experiments, using the first design, two optimizers were used as well. By using the RMSprop optimizer, the best performing architecture was our proposed architecture, followed by the VGG19 network. The worse performance was achieved by the InceptionV3 architecture. By using Adam optimizer, the highest performance was obtained by our proposed architecture, followed by VGG19. The poorest performance resulted from the ResNet architecture. Overall, our proposed architecture outperformed the other architectures taken into account. The results of the first set of experiments are shown in Table 23. In the second set of experiments, using the fourth design, two optimizers were used as well. By using RMSprop, the highest performance was obtained by using our architecture, followed by VGG16. The lowest performance was achieved by using ResNet. By using Adam optimizer, the highest performance was achieved with our architecture, followed by VGG16. The lowest performance was obtained with the ResNet architecture. Overall, our proposed architecture achieved higher performance than the other tested CNNs. The results of the second set of experiments are shown in Table 24.

Table 23. The AUC results obtained with benchmark architectures under the first design.

	RMSProp	**Adam**
Our Model	**92.99%**	**93.66%**
VGG16	84.22%	89.53%
VGG19	89.08%	90.64%
InceptionV3	82.66%	82.47%
ResNet	85.24%	81.21%

Table 24. The AUC results obtained with benchmark architectures under the fourth design.

	RMSProp	**Adam**
Our Model	**94.86%**	**95.46%**
VGG16	89.33%	91.00%
VGG19	89.24%	89.20%
InceptionV3	87.15%	85.88%
ResNet	78.01%	83.52%

5.4.2.4. The Results of State-of-the-art CNN Architectures

It is impossible to compare the results of our architecture against other architectures unless both the architectures were trained and tested on the same dataset and using the same hyperparameters like batch size, image augmentation, optimizer, and learning rate. That is why we trained different state-of-the-art CNN architectures on the PatchCamelyon dataset to easily compare our proposed CNN architecture with others. The image size was the only hyperparameter that was different between

different architectures: The images were rescaled to follow the requirements of each architecture. Using the RMSProp optimizer, the best performance was achieved by using our proposed architecture followed by the architecture of Lai et al. (2018). The lowest performance was obtained with the Sirinukunwattana et al. (2016) architecture. Using Adam optimizer, the best performance was achieved by our architecture, followed by Lai et al. (2018) architecture. Similar results were obtained when the RMSProp optimizer was considered. Overall, our proposed architecture outperformed all the other architectures tested, as summarized in Table 25.

Table 25. The AUC results of the State-of-the-art architectures.

	RMSProp	**Adam**
Our Model	**94.86%**	**95.46%**
Bayramoglu et al. (2016)	77.22%	86.68%
Arjmand et al. (2019)	85.69%	88.23%
Lai et al. (2018)	86.06%	92.11%
Sirinukunwattana et al. (2016)	72.28%	68.85%
Basha et al. (2018)	80.69%	75.04%
Nguyen et al. (2019)	81.16%	86.68%

5.5. Discussion

In this work, a novel CNN architecture was proposed to classify histopathology images. This section discusses the results obtained.

5.5.1. Histopathology Images Importance And Challenges

Histopathology images classification is considered a very difficult task and very subjective as well, where two experienced pathologists can have very different opinions, and that is where an automatic classifier can be very important by providing a second opinion.

5.5.2. The Presented Architecture Choice

The architecture presented in this work was chosen after an extensive design phase, where different architectures were tested. As pointed out by Sirinukunwattana et al. (2016), giving theoretical justification for the network architecture is very challenging and is still a matter of ongoing research. Based on our results, three aspects affected the performance of the CNN: the position of the normalization layers in regards to the activation function, the presence of the dropout layers in the classifier block, and the activation function used.

5.5.3. The Effect of Different Activation Functions

Six activation functions were tested using our proposed architecture, two of them are saturated, and four are non-saturated. The sigmoid function did not achieve a satisfying result, and so it was

removed from further testing. The Tanh function achieved outstanding results compared to other state-of-the-art non-saturated activation functions, which was quite surprising.

5.5.4. The Effect of the Location of the Normalization Layer and the Dropout Layer

Four different designs were tested to detect the optimal location of the normalization layer and the effect of the dropout layer on it. From the obtained results, we can conclude that the location of the normalization layer is very important, for which we recommend placing the normalization layer before the activation function. The presence of the dropout layer is not always guaranteed to increase network performance. Moreover, the optimizers played a very important role in the network performance, and from our results, Adam achieved better performance than RMSProp.

5.5.5. Comparison between Different Benchmark CNN

From our experiments, we noticed that adding a dropout layer after each fully connected layer decreases the performance of the VGG16 network, especially for the RMSProp optimizer. For the VGG19 network, the performance did not change significantly when considering the addition of dropout layers. The performance of the InceptionV3 network was affected by the inclusion of the dropout layer, for both the Adam and RMSprop optimizers. The ResNet network was the only network that benefited from adding the dropout layer, and the performance of RMSProp with dropout was higher with respect to the Adam optimizer. The VGG architectures were the best performers among the different competitors taken into account, and they were outperformed only by our proposed architecture. InceptionV3 and ResNet were the poorest performers. For the InceptionV3 model, we can speculate that the inception module did not increase the performance, while adding too many layers is not beneficial for the ResNet model. All in all, our proposed architecture achieved higher results for both Adam and RMSprop optimizers.

5.5.6. Comparison between Different State-of-the-Art CNN

Arjmand et al. (2019) presented a network with three convolution layers, where the number of kernels is 64, 32, and 16, respectively. The kernels size used was 5×5, 3×3, and 3×3, respectively. Every convolution layer was followed by a batch normalization layer and max-pooling layer, except for the third convolution layer, which was followed by a batch normalization layer only. There are two dropout layers, one after the second convolution layer and the second before the fully connected layer. Compared to our network, Arjmand architecture was the second-best architecture, among the set of competitors, for both RMSprop and Adam optimizers. The authors placed the normalization layer before the activation layer and the dropout layer in the middle of the convolution blocks. The main difference between our proposed architecture and the architecture proposed by Arjmand et al. (2019) relies on the number of convolution layers used. From the results obtained, it seems that the choice of the number of convolution layers plays a fundamental role in the performance

of the model. In particular, a network with just three convolution layers seems to be not sufficient for dealing with the complexity of the application at hand. The architecture of Arjmand et al. (2019) is shown in Figure 14.

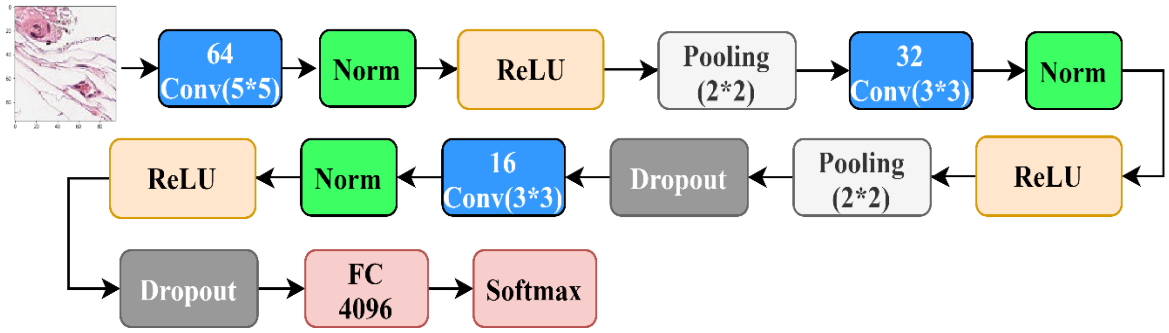


Figure 14. The architecture of Arjmand et al. (2019).

Lai et al. (2018) introduced a network with six convolution layers and no fully connected layers. This network achieved the best results, among the competitors of our proposed architecture, for both the optimizers. The kernel size used in both the convolution layers and the pooling layers is the highest compared to all the tested CNNs. No dropout layers or batch normalization layers were used. The results of Adam and RMSProp optimizers are comparable. The differences between this network and our network are the size of the kernels used (which is higher than our proposed network), the absence of any regularization layers, the absence of any fully connected layers, and the number of convolutional layers, (which is higher in our model). The main difference in terms of design choices between the architecture of Lai et al. (2018) and our architecture is the absence of any regularization layers from their architecture. According to the experiments we performed, this seems to be the main cause for the lower performance of the network proposed by Lai and coauthors. Figure 15 shows the architecture of Lai et al. (2018).

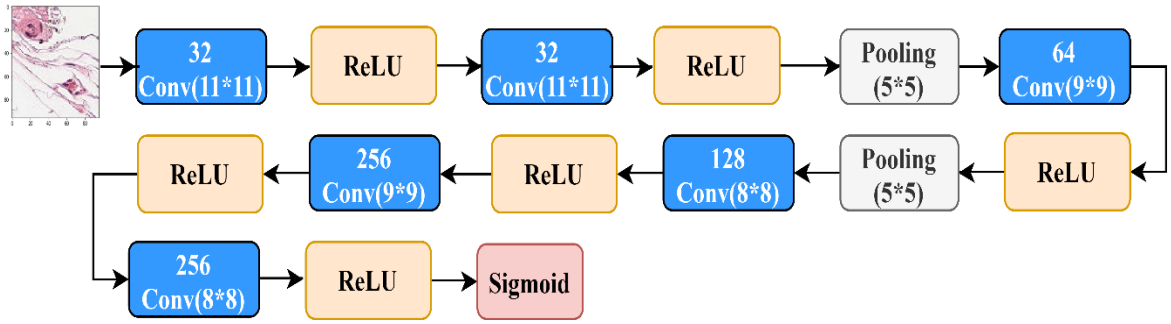


Figure 15. The architecture of Lai et al. (2018).

Nguyen et al. (2019) presented a network with five convolution layers, one fully connected layer, three dropout layers, and six normalization layers. The authors placed the normalization layer after the activation layer, and the dropout layer in the middle of the convolution blocks. This is the only

network with an activation function other than the ReLU function. The results of Adam and RMSProp optimizers are different, with Adam being the best optimizer. Compared to our proposed network, the main differences are the number of convolution layers, the presence of the normalization layer after the activation function, and the usage of only one fully connected layer. The difference between the performance of our proposed architecture and the architecture presented by Nguyen et al. (2019) can be explained by the number of convolution layers (that is lower with respect to our architecture) and by the position of the normalization layer after the activation layer. In particular, placing the normalization layer after the activation function was not suggested in the original paper that introduced the normalization layer (Ioffe & Szegedy, 2015). Figure 16 shows the architecture of Nguyen et al. (2019).

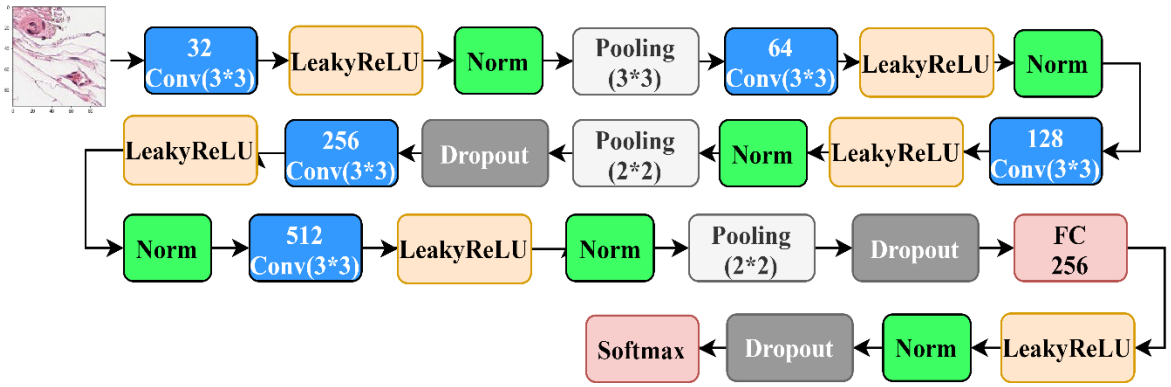


Figure 16. The architecture of Nguyen et al. (2019).

Basha et al. (2018) designed an architecture with 4 convolution layers and two fully connected layers. The authors also placed the normalization layer after the activation layer. Moreover, the authors placed a normalization layer in the classifier block. The results of Adam and RMSProp optimizers are significantly different, with RMSProp being the best optimizer. Compared to our architecture, the main differences are the number of convolution layers used (we used 15 instead of 4), the presence of the normalization layer after the activation function, and the usage of the normalization layer in the classifier block. The difference between the performance of our proposed architecture and the architecture proposed by Basha et al. (2018) can be due to the number of the convolution layers, placing the normalization layer after the activation layer, and the usage of dropout layers in the classifier block. Figure 17 shows the architecture of Basha et al. (2018).

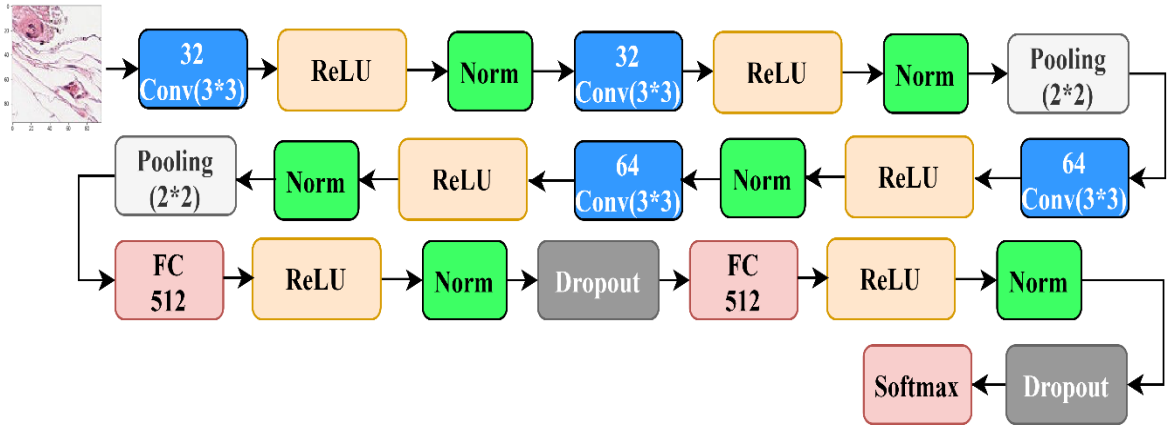


Figure 17. The architecture of Basha et al. (2018).

The architecture of Bayramoglu et al. (2016) has three convolution layers and two fully connected layers. The authors placed the normalization layer after the activation layer and after the pooling layer as well. The results of Adam and RMSProp optimizers are significantly different, with Adam being the best optimizer. Compared to our design, this architecture has low capacity, since it has only three convolution layers. The difference between the performance of our proposed architecture and the architecture proposed by Bayramoglu et al. (2016) can be explained (beyond the number of convolution layers) considering the usage of a larger kernel size in the first convolution layer. This is something that could be detrimental to the performance of the network. Using smaller kernel sizes allows the network to learn complex, more non-linear features. The architecture of Bayramoglu's (2016) is shown in Figure 18.

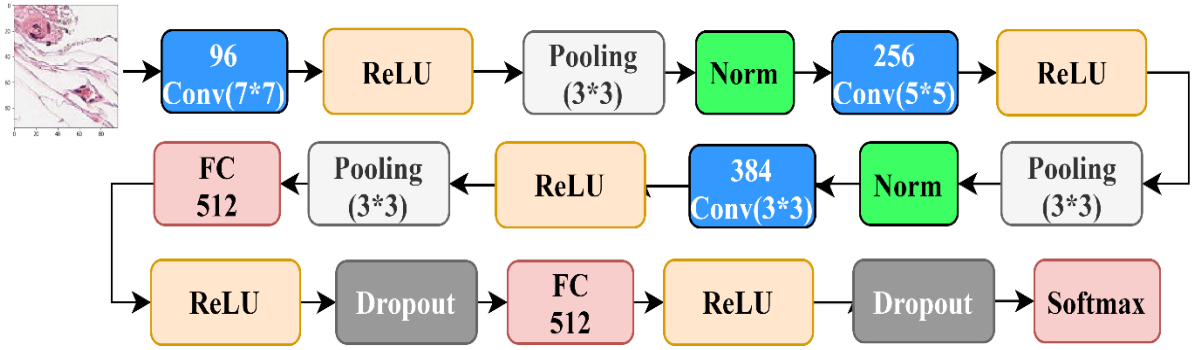


Figure 18. The architecture of Bayramoglu et al. (2016).

Sirinukunwattana et al. (2016) proposed a network with only two convolution layers and two fully connected layers. The network is the worst performer in our comparison, probably because it has a very low capacity compared to the other networks taken into account. The results of Adam and RMSProp optimizers are significantly different, with RMSProp being the best optimizer. In this network, the ReLU activation function was used, and two pooling layers were placed after each convolution layer. Figure 19 shows the architecture of Sirinukunwattana's (2016).

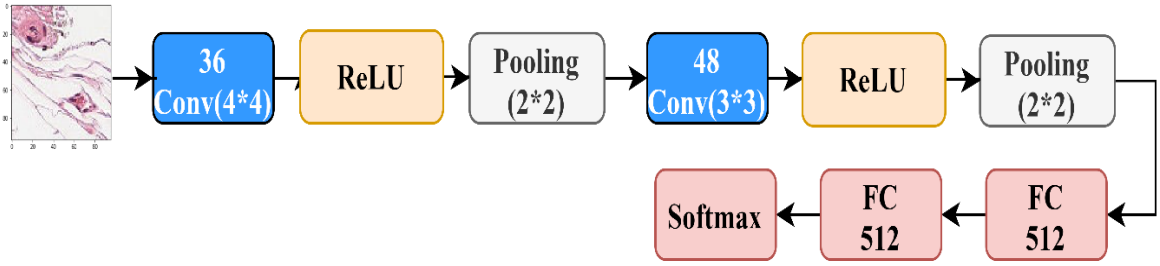


Figure 19. The architecture of Sirinukunwattana et al. (2016).

5.6. Conclusions

In this paper, we introduced a novel CNN architecture that is designed to classify histopathology images. The training and evaluation of the architecture were conducted on the publicly available PatchCamelyon dataset. The proposed architecture has fifteen convolution layers and two fully connected layers. The highest AUC obtained using our architecture was 95.46%. In this work, we have also studied the effect of different activation functions on the CNN performance and the effect of the location of the activation function on the performance of the network. Based on the obtained results, two main points must be highlighted: We recommend authors to try different activation functions and to fully analyze their impact other than choosing the ReLU activation function as a default. Based on our results, we recommend placing the normalization layer before the activation function. We do encourage researchers to examine our proposed CNN on different datasets and report the performance achieved.

We acknowledge several limitations in this work that can be addressed in future work. First, the performance of the proposed model slightly increases with respect to the models that were presented in the literature. Second, the proposed model was inspired by the VGG16 architecture, and the differences are the positioning of the regularization layers, the activation functions used, and the number of neurons in the fully connected layers. We believe that the two limitations could be overcome using neuroevolution algorithms that can provide a different way of exploring the search space of deep learning architectures. Finally, the human performance was reported on the Camelyon dataset, which is a reduced dataset with respect to the PatchCamelyon dataset used in this study. Thus, it would be interesting to assess the human experts' performance on this bigger dataset.

Chapter 6. The Effect of Batch Size on the Generalizability of the Convolutional Neural Networks on a Histopathology Dataset[4]

Abstract: Many hyperparameters must be tuned to have a robust convolutional neural network that will be able to classify images accurately. One of the most important hyperparameters is the batch size, which is the number of images used to train a single forward and backward pass. In this study, the effect of batch size on the performance of convolutional neural networks and the impact of learning rates will be studied for image classification, specifically for medical images. To train the network faster, a VGG16 network with ImageNet weights was used in this experiment. Our results concluded that a higher batch size doesn't usually achieve high accuracy, and the learning rate and the optimizer used will have a significant impact as well. Lowering the learning rate and decreasing the batch size will allow the network to train better, especially in the case of fine-tuning.

6.1. Introduction

Since its introduction nearly two decades ago, convolutional neural networks (CNNs) (Lecun et al., 1998) have been used as primary image classification algorithms. The true power of the CNN has been rediscovered by the ImageNet competition (Russakovsky et al., 2015), where AlexNet architecture (Krizhevsky et al., 2012) succeeded in classifying millions of images with thousands of labels with an accuracy of 85% compared to 74% of the traditional algorithms, and that's when the CNN again became one of the most important algorithms for image classification. One of the main benefits of using a CNN is that it doesn't need any manual feature extraction to work, which makes it robust against new datasets. CNNs not only succeed in the image classification domain but are also successfully applied in text classification (Hughes et al., 2017), climate change detection (Liu et al., 2016), and speech recognition (Abdel-Hamid et al., 2014), among others.

Medical images can be considered very complicated datasets because of the complexity and seriousness, and they require an experienced physician with years of experience to be able to classify the images. Examples of medical images that CNN can be applied to are histopathology images, which are images assessed by pathologists to evaluate whether tissue is cancerous. Histopathology images are very challenging to classify, even for an experienced pathologist, and that's where the CNN can be applied, either in giving a second opinion or giving assistance to the pathologist in classifying these images.

To correctly train the CNN to be able to classify images, many hyperparameters need to be adjusted; these hyperparameters will affect the performance of the network along its time to

[4] This chapter has been published in ICT express journal as I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," ICT Express, 2020. https://doi.org/10.1016/j.icte.2020.04.010

convergence. One of the main hyperparameters that need to be tuned is the batch size (Ioffe & Szegedy, 2015), which is the number of images used in every epoch to train the network. Setting this hyperparameter too high can make the network take too long to achieve convergence (no more gain in accuracy); however, if it's too low, it will make the network bounce back and forth without achieving acceptable performance. Also, the nature of the dataset can have an impact on the batch size, especially the medical dataset because of its complexity.

In this study, we investigated the effect of batch size on the performance of CNNs and the impact of learning rates for image classification. Two different optimizers were used to assess the impact of batch size. The CNN architecture used in this experiment was the VGG16 (Simonyan & Zisserman, 2014); the network was fine-tuned to suit this dataset and to avoid training the network from scratch. This experimental study aims at providing a better understanding of the batch size value to be considered before addressing a given problem through a CNN. In fact, despite the importance of the batch size value for the learning process of a CNN, scientific literature only provides a few studies on this topic. Additionally, as discussed in Section 2, the results reported in the literature do not report unanimous conclusions, with some authors indicating a preference for large batch size values and other works suggesting the usage of small batch size values. The rest of the paper is organized as follows. In section 2, previous research done on batch size is presented. In section 3, our methodology is presented. In sections 4 and 5, we present our results and then the conclusion.

6.2. Literature Review

Many hyperparameters need to be adjusted before training the CNN to classify images. One of the main hyperparameters that need to be adjusted before beginning the training process is the batch size, where the batch size is the number of images that will be used in the gradient estimation process. Many researchers have studied the effect of batch size on the network performance—either the accuracy of the network or the time that was taken till convergence—to determine which was better: small batches or large batches. On one hand, a small batch size can converge faster than a large batch, but a large batch can reach the optimum minima that a small batch size cannot reach. Also, a small batch size can have a significant regularization effect because of its high variance (Wilson & Martinez, 2003), but it will require a small learning rate to prevent it from overshooting the minima (Goodfellow et al., 2016). Below are some researches that were done to investigate the pros and cons of using small and large batch sizes.

In 2017, Radiuk (2017) investigated the effect of batch size on CNN performance for image classification, the author used two datasets in the experiment, namely, MNIST and CIFAR-10 datasets. Radiuk tested batch sizes with the power of 2, starting from 16 until 1024 and 50, 100, 150, 200, and 250 as well. Radiuk opted for a LeNet architecture for the MNIST dataset and a custom-

made network with five convolutional layers for the CIFAR-10 dataset. The optimizer used for both networks was the stochastic gradient descent optimizer with a learning rate of 0.001 for the MNIST and 0.0001 for the CIFAR-10 dataset. For both the datasets, the best accuracy was achieved by the 1024 batch size, and the worst result was with the 16 batch size. The author stated that based on their results, the higher the batch size the higher the network accuracy, meaning that the batch size has a huge impact on the CNN performance.

Bengio (2012) stated that a batch size of 32 is a good default value, also he stated that the larger batch size will quicken the computation of the network but will decrease the updates required for the network to reach convergence. The author stated that the batch size likely impacts the convergence time and not network performance. Meanwhile, Masters and Luschi (Masters & Luschi, 2018) tested the effect of batch sizes between 2^1 and 2^{11}, on AlexNet (Krizhevsky et al., 2012) and ResNet (He et al., 2015) architectures with SGD as an optimizer without momentum to exclude the effect of momentum on the training. The authors studied the effect of batch size on three datasets: CIFAR10, CIFAR100, and ImageNet. The authors stated that the best results were obtained with batch sizes between 2 and 32, and the authors noted the small batch sizes are more robust than the large batch sizes. In general, the main question regarding the batch size is which is the optimal batch size for training CNNs that will help the network achieve the highest accuracy in the shortest time, especially for complex datasets like a medical image dataset.

6.3. Methodology

The training of a CNN to classify images can be defined as minimizing a non-convex loss function $L(\theta)$ by using an optimizer like a stochastic gradient descent or Adam optimizer, where $L(\theta)$ is the average cost of training image $L_i(\theta)$ over the dataset, and M is the size of the image dataset.

$$\arg\min_{\theta\in\mathbb{R}} L(\theta); L(\theta) = \frac{1}{M}\sum_{i=1}^{M} L_i(\theta)$$

The gradient update has three options to be calculated: using the entire image dataset M, using a single image, or using a number between $1\ and\ M$. The previous methods are named batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, respectively. Batch size hyperparameter B is the number of images used to update the gradients per time. By using the SGD optimizer, the network weights will be updated using the following equation:

$$w_{t+1} = w_t - \eta\frac{\partial L}{\partial w_t}; \frac{\partial L}{\partial w_t} = \nabla_W C(w_t; x^{(B)}; y^{(B)})$$

Where η is the learning rate, x are the sample images used, y are the image labels, and w are the weights being updated. For the Adam optimizer, the weights will be updated using the following:

$$w_t^i = w_{t-1}^i - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} . \hat{m}_t$$

Where $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$, $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$, $m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$,$v_t = \beta_2 v_{t-1} + (1-\beta_2)[\frac{\partial L}{\partial w_t}]^2$ and $\frac{\partial L}{\partial w_t} = \nabla_W C(w_t; x^{(B)}; y^{(B)})$

Where $\beta_i \in [0,1]$ is used to determine how much information is needed from the previous update, m_t is the first momentum where it is the gradients' running average, and v_t is the second momentum where it is the squared gradients' running average. The bias-corrected first and second momentums are $\hat{m}_t$ and $\hat{v}_t$. As is shown from the previous equations, batch size and learning rate have an impact on each other, and they can have a huge impact on the network performance.

To speed up the network training and to increase its robustness, fine-tuning of the VGG16 network was applied. Fine-tuning a network is considered a method of transfer learning, where the knowledge transfer between networks that have been trained on different datasets. Because training CNN weights from scratch requires millions of images and training for days and this amount of images is not available for medical images, usage of transfer learning can be very useful in the medical field (Tajbakhsh et al., 2016).

The VGG16 (Simonyan & Zisserman, 2014) network is considered one of the most important CNNs for image classification because of its deep yet simple architecture, which gives it robustness against overfitting while providing good performance; VGG16 is presented in fig.19.

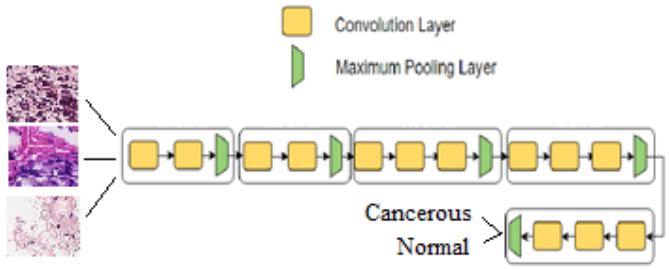


Figure 20. VGG16 network architecture.

The dataset used in this experiment was the PatchCamelyon (Bejnordi et al., 2017; Veeling et al., 2018) a public dataset that contains 220,000 binary labeled images to train the CNN. The dataset was balanced, meaning it contained 60% positive to 40% negative images. Another 57,458 images were provided on the Kaggle platform to test the algorithm. All the images were 96×96 pixels. A sample of the dataset is presented in Figure 21.

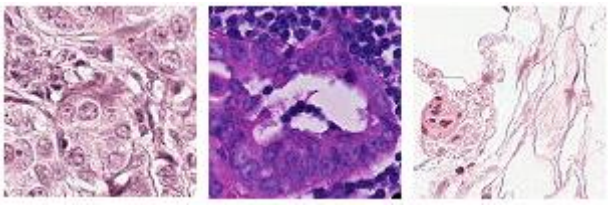


Figure 21. A sample of the PatchCamelyon dataset.

Image augmentation is usually used to increase the image dataset and also to make the network more robust against translation invariance. Image augmentation is defined as creating duplicates of the original image datasets by flipping, rotating, zooming, and adjusting brightness. In this work, the images were horizontally and vertically flipped, with an image rotation of 180 degrees; some images were zoomed in; and some images were shifted.

To evaluate the CNN classifier performance (i.e., to determine the classifier ability to classify positive images as positive and negative images as negative), the area under the ROC curve was used (AUC), which can be formally defined as (Idrees et al., 2017):

$$AUC = \frac{1}{2}\left(\frac{TP}{TN + FN} + \frac{TN}{TN + FP}\right)$$

Where TP is the true positive metric, which is the positive images classified as positive; TN is the true negative metric, which is the negative images classified as negative; FP is the false positive metric, which is the negative images classified as positive; FN is the false negative metric, which is the positive images classified as negative. The minimum value of the AUC metric was 0.5, which represents that the model had no predictive power, and the maximum was 1, which represents that the model had perfect power in classifying images.

6.4. Results

The last two blocks of the VGG16 network were fine-tuned using 80% of the dataset and were validated on the remaining 20% of the dataset, after which the best model was saved and used to classify the Kaggle online test set. The batch sizes used in this experiment were $B = [16,32,64,128,256]$; two optimizers were used, namely SGD and Adam optimizers, and two learning rates were used for each optimizer of 0.001 and 0.0001. For consistency of results and due to the size of the dataset, the number of epochs was fixed to 50 epochs. To overcome overfitting, only the best model was saved, meaning that during the training phase, if the validation accuracy of the epoch was higher than the highest accuracy, then the model was saved. The results of the Kaggle online test set are shown in tables 15 and 16.

Table 26 shows the results of the Adam optimizer with a learning rate of 0.001 and a learning rate of 0.0001. For a learning rate of 0.001, the lowest batch size (16) achieved the lowest AUC. The highest performance was from using the largest batch size (256); it can be shown that the larger the batch size, the higher the performance. For a learning rate of 0.0001, the difference was mild; however, the highest AUC was achieved by the smallest batch size (16), while the lowest AUC was achieved by the largest batch size (256).

Table 26. The results of the test AUC of the Adam optimizer.

Test AUC		
Batch Size	Adam LR = 0.0001	Adam LR = 0.001
16	**0.9677**	0.9144
32	0.9636	0.9332
64	0.9616	0.9381
128	0.9567	0.9432
256	0.9585	**0.9652**

Table 27 shows the result of the SGD optimizer with a learning rate of 0.001 and a learning rate of 0.0001. For a learning rate of 0.001, we can see that the large batch size achieved the highest AUC, while the lowest was by using the smallest batch size (16). For a learning rate of 0.0001, it was the opposite; the largest batch size (256) achieved the lowest AUC, while the 32 batch size achieved the highest, followed by the lowest batch size.

Table 27. The results of the test AUC of the SGD optimizer.

Test AUC		
Batch Size	SGD LR = 0.0001	SGD LR = 0.001
16	0.9555	0.9461
32	**0.9570**	0.9521
64	0.9512	0.9545
128	0.9302	0.9567
256	0.9077	**0.9579**

The highest overall AUC achieved during the experiments was by the Adam with a learning rate of 0.0001 and batch size of 16.

Our results agree with the ones obtained by Masters and Luschi (Masters & Luschi, 2018), where the authors stated that smaller batch sizes should be used. According to Radiuk (Radiuk, 2017), when a large learning rate is used, the higher the batch size, the better the performance of a CNN. While the use of large batch size values is not recommended in our study, the results of Radiuk match our findings on the relation between the batch size and the learning rate. In particular, we highlighted that higher learning rates require larger batch sizes. Finally, Bengio (2012) suggested that 32 is a good default value for the batch size. While this is corroborated by our experiments (in which a batch size of 32 provided good results), the best performance was achieved with a batch size of 16.

6.5. Conclusion

Convolutional neural networks have shown superior accuracy in image classification, but to accurately train a CNN many hyperparameters need to be tuned depending on the dataset being used. The medical field can benefit greatly by using CNN in image classification to increase accuracy. In this paper, we compared the performance of CNN using different batch sizes and different learning rates. According to our results, we can conclude that the learning rate and the batch size have a significant impact on the performance of the network. There is a high correlation between the learning rate and the batch size, when the learning rates are high, the large batch size performs better than with small learning rates. We recommend choosing a small batch size with a low learning rate. In practical terms, to determine the optimum batch size, we recommend trying smaller batch sizes first(usually 32 or 64), also keeping in mind that small batch sizes require small learning rates. The number of batch sizes should be a power of 2 to take full advantage of the GPUs processing. Subsequently, it is possible to increase the batch size value till satisfactory results are obtained.

Chapter 7. How Deeply to Fine-Tune a Convolutional Neural Network: A Case Study Using a Histopathology Dataset [5]

Abstract: Accurate classification of medical images is of great importance for correct disease diagnosis. The automation of medical image classification is of great necessity because it can provide a second opinion or even a better classification in case of a shortage of experienced medical staff. Convolutional neural networks (CNN) were introduced to improve the image classification domain by eliminating the need to manually select which features to use to classify images. Training CNN from scratch requires very large annotated datasets that are scarce in the medical field. Transfer learning of CNN weights from another large non-medical dataset can help overcome the problem of medical image scarcity. Transfer learning consists of fine-tuning CNN layers to suit the new dataset. The main questions when using transfer learning are how deeply to fine-tune the network and what difference in the generalization that will make. In this paper, all of the experiments were done on two histopathology datasets using three state-of-the-art architectures to systematically study the effect of block-wise fine-tuning of CNN. Results show that fine-tuning the entire network is not always the best option; especially for shallow networks, alternatively fine-tuning the top blocks can save both time and computational power and produce more robust classifiers.

7.1. Introduction

Medical images play a very crucial role in patient treatment; however, usually, the shortage of manpower, the time required to reach a decision, and the need for a second opinion are factors that greatly impact the process. Correctly and more quickly classifying images is an absolute need for certain medical images fields, like pathology. Histopathology images are very important for detecting certain kinds of diseases like cancer or even determining the kind of cancer itself to see if it is benign or malignant and its degree. Histopathology is defined as examining a tissue sample taken by a biopsy to diagnose certain diseases microscopically (Gurcan et al., 2009). It plays a very important role in the detection of diseases, enabling doctors to carefully create a treatment plan. The physician who is responsible for classifying histopathology images is called a pathologist. The image-examining process is extremely difficult and requires an experienced pathologist with years of experience. In the United States, the number of active pathologists dropped 17.5% in the last decade while the workload increased by 41% (Metter et al., 2019), which indicates a real need for assisting the pathologists in their work by providing them with an autonomous classifier that is able to classify histopathology images with a high level of accuracy.

A recent breakthrough in the artificial intelligence field is machine learning, in which an algorithm can be developed that will be able to extract image features automatically. When the algorithm used is a neural network with more than one hidden layer, it is called deep learning. Deep learning can be implemented in the image classification domain in which a feed-forward convolutional neural network (CNN) (LeCun et al., 1989) can be used to classify images

[5] This chapter has been published in MDPI journal as Kandel, I.; Castelli, M. How Deeply to Fine-Tune a Convolutional Neural Network: A Case Study Using a Histopathology Dataset. *Appl. Sci.* **2020**, *10*, 3359. **https://doi.org/10.3390/app10103359**

automatically. Making the CNN able to classify images is called training; in training, the CNN's weights will be adjusted to suit the image dataset under study. The main point of CNN is that it is able to map important features of images that can be used to classify those images without the CNN's being explicitly programmed to do so. CNN has been proven to work with great accuracy in the classification of many medical domains like diabetic retinopathy detection and classification (Mohammadian et al., 2017; Prentašić & Lončarić, 2016), Alzheimer's disease detection (Farooq et al., 2017; Khan et al., 2019), and skin lesion detection (Harangi, 2018; Hosny et al., 2019), among others.

Image classification, which is defined as grouping images into successive predefined labels, plays a very important role in many areas, like the medical field. Deep learning algorithms can detect important features of images without any manual feature engineering, which can be thought of as using autonomous algorithms that can learn by themselves how to differentiate between distinct image classes. The earliest attempt to construct an automatic classifier that could learn how to differentiate between classes was introduced by LeCun (1989), who was inspired by the work of Fukushima et al. (1980) and Hubel and Wiesel (1977), and was named convolutional neural networks (CNN), but this attempt was limited because of the size of datasets and the computational power available then. In 2012, Krizhevsky et al. (2012) introduced their CNN architecture that was named AlexNet and won first place in the ILSVRC competition, with an error rate of 16% compared to the second-place winner's 25%, and since then, CNN has become a state-of-the-art image classifier. CNN is considered a feed-forward artificial neural network that has at least one convolution layer. A diagram of CNN is shown in Figure 22.

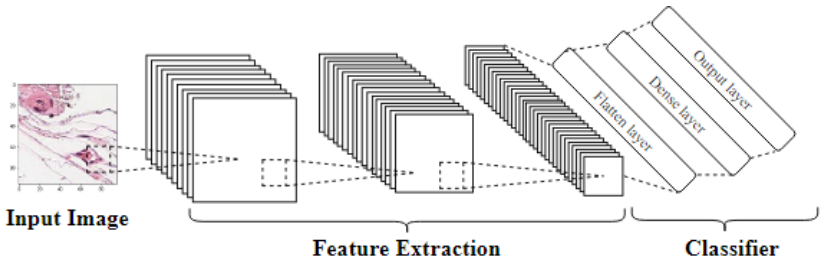


Figure 22. Convolutional neural network (CNN) diagram.

To accurately train CNN and to overcome the overfitting issues associated with feed-forward neural networks, usually, hundreds of thousands or even millions of images are needed (Tajbakhsh et al., 2016), which can limit the usage of CNN to certain domains, like nature images, for classification. Transfer learning is a novel domain that has been introduced to help overcome the before-mentioned issues in CNN training, where instead of initializing the network weights from scratch, the weights of a previous network that was trained on a different large dataset can be used. The medical field could benefit a lot from the usage of CNN for image classification, but the main drawback is the number of images available for training, and that is where transfer learning can be used. Using transfer

learning for medical images has been successfully applied in References (Hosny et al., 2019; Khan et al., 2019; Mohammadian et al., 2017). As pointed out by Chollet (Francois Chollet, 2017a) , transfer learning uses two main techniques: fine-tuning, wherein the original weights will be re-tuned to suit the new dataset, or feature extraction, in which the original weights will be fixed and the original layers will serve for feature extraction only. Both techniques can help a lot based on the size of the dataset, as pointed out by Yosinski et al. (2014). If the dataset is large enough, the original layers can be fine-tuned, while if the dataset is small, the original layers can serve in feature extraction. Also, the similarity between the original dataset and the target dataset can play a very important role. For this study, we opted for fine-tuning instead of feature extraction because of the following reasons: (1) the large number of images contained in the dataset used, and (2) the huge difference (with respect to the domain of the images) between the ImageNet dataset and the histopathology dataset taken into account.

Fine-tuning an entire CNN architecture can take hours and requires certain hardware, and the effect of each block can play a very important role. Fine-tuning the entire network does not always guarantee to yield the best performance. The main purpose of this study is to determine the effect of fine-tuning a CNN block-wise to assess the performance gained by training each block. Three state-of-the-art CNN architectures with three learning rates are used in this study. The performance measure used is the AUC of the ROC curve. A separate unlabeled test set is used to evaluate the performance of the CNN architectures. The rest of this paper is organized as follows: In Section 2, a brief literature review about using transfer learning in histopathology is given. In Section 3, the proposed methodology is discussed. In Section 4, the results obtained are stated. In Section 5, a brief overview of the findings is discussed. In Section 6, the conclusion is stated.

7.2. Literature Review

The usage of deep learning techniques for medical image classification is a sore subject right now because of the need to assist pathologists and give them a second opinion.

Sharma and Mehra (2018) investigated the effect of CNN transfer learning on the performance of three CNN architectures, namely VGG16, VGG19, and ResNet50. The authors used the BreakHis dataset (Spanhol et al., 2016), which consists of 7909 breast cancer histopathology images. The authors opted for binary classification, and the original last classification layer of all the architectures was removed and replaced by a logistic regression classifier. The authors used the AUC of the ROC curve, accuracy, precision, recall, F1 score, and APS as evaluation criteria. The authors tried three different splitting techniques, namely 90% and 10% for training and testing respectively, 80% and 20%, and 70% and 30%. The best result reported by the authors was found using the VGG16 architecture, which achieved an AUC of 95.65% for the first splitting technique, followed by the VGG19 architecture, which achieved an AUC of 91.85% for the first splitting technique as well. The ResNet architecture’s performance did not improve by using transfer learning.

Kassani et al. (2019) proposed a novel model to classify histopathology images. The authors used four datasets, namely PatchCamelyon (Ehteshami Bejnordi et al., 2017; Veeling et al., 2018), BreakHis (Spanhol et al., 2016), Bach (Aresta et al., 2019), and BioImaging (2015), to train and validate their model. The authors proposed a novel binary-classification ensemble model composed of three CNN architectures, namely, VGG19 (Simonyan & Zisserman, 2014), MobileNet (Howard et al., 2017), and DenseNet (Huang et al., 2017). The accuracy results reported by the authors were 98.13%, 95%, 94.64%, and 83.10% for datasets BreakHis (Spanhol et al., 2016), Bach (Aresta et al., 2019), PatchCamelyon, and BioImaging, respectively. Different image augmentation techniques were used to increase the size of the training dataset to make the CNN models more robust against overfitting. Some of the augmentation techniques applied were flipping the images horizontally and vertically, increasing the zoom range, and rotating the images. The authors opted for an Adam optimizer with a learning rate of 0.0001, and the batch size used was 32. All of the images were resized to $224 \times 224\, pixels$, and all of the models were trained for 1000 epochs. The authors claimed that by using the three-model ensemble, the accuracy of the BreakHis dataset increased from 97.42% for the best single classifier to 98.13%, for the PatchCamelyon dataset, it increased from 90.84% to 94.64% for the best single classifier; for the Bach dataset, it increased from 92% to 95%, and for the BioImaging dataset, it increased from 81.69% to 83.10%.

Vesal et al. (2018) investigated the effect of transfer learning on the Bach (Aresta et al., 2019) dataset. The authors compared two CNN architectures, namely InceptionV3 (Szegedy et al., 2016) and ResNet50 (He et al., 2016), and opted for multi-class classification into four categories. The batch size used was 32, and the optimizer used was stochastic gradient descent with Nesterov momentum with a learning rate of 0.0001, with the dataset trained for 100 epochs for both architectures. The authors reported that the fine-tuned ResNet50 architecture outperformed the InceptionV3 architecture with an accuracy of 97.50% and 91.25%, respectively.

Deniz et al. (2018) used the BreakHis (Spanhol et al., 2016) dataset to investigate the effect of transfer learning. The authors opted for the AlexNet and VGG16 architectures. The authors conducted three experiments, two of them using the AlexNet and VGG16 for feature extraction, concatenating their results then adding the SVM classifier, and the third fine-tuning the AlexNet network. The optimizer authors used was SGD with momentum, and the learning rate chosen was 0.0001. The authors set the batch size at 10. The authors reported that the fine-tuned AlexNet outperformed the feature extraction of both the AlexNet and VGG16 networks.

Ahmad et al. (2019) investigated the effect of transfer learning on a multiclass histopathology dataset, using three CNN architectures, namely the AlexNet, GoogleNet, and ResNet architectures. The dataset used was the BioImaging dataset, and the authors used image augmentation to increase the size of the dataset from 260 images to 72,800 images. The authors reported that the ResNet network achieved the best accuracy, at 85%. Table 28 shows a summary of the studies mentioned.

The above-mentioned studies did not investigate the fine-tuning block-wise effect on the CNN's performance, and that is where this study will come into use: where the effect of each block in three CNN architectures will be investigated to detect how deeply the CNN should be fine-tuned given that fine-tuning a CNN is very computationally expensive.

Table 28. Summary of the studies mentioned.

Paper	Dataset Name	Dataset Size	Architectures Used	Classes	Best Accuracy
Sharma et al. (2018)	BreakHis (Spanhol et al., 2016)	7909	VGG16 VGG19 ResNet50	Binary	92.6%
Ahmad et al. (2019)	BioImaging (2015)	260	AlexNet GoogleNet ResNet50	Multiclass	85%
Deniz et al. (2018)	BreakHis (Spanhol et al., 2016)	7909	AlexNet VGG16	Binary	91.37%
Vesal et al. (2018)	Bach (Aresta et al., 2019)	400	InceptionV3 ResNet50	Multiclass	97.50%
Kassani et al. (2019)	PatchCamelyon (Ehteshami Bejnordi et al., 2017) BreakHis (Spanhol et al., 2016) Bach (Aresta et al., 2019) BioImaging (2015)	327,680 7909 400 249	Ensemble of: VGG19 DenseNet ImageNet	Binary	94.64% 98.13% 95% 83.10%

7.3. Methodology

This paper focuses on fine-tuning three CNN architectures' weights from the ImageNet dataset (non-medical) to classify histopathological images into normal or not. Below is a description of the methods used in this research.

7.3.1. Convolutional Neural Networks

Formally, in supervised machine learning given a training dataset of $\Phi = \{(x_i, y_i)\}_{i=1}^{N}$, where N is the training dataset size and (x_i, y_i) is a single training example, $x_i \in \Phi$ is the training images and $y_i \in \Phi$ is the respective label of each image x_i, θ is the model parameters and $\hat{y}_i$ is the predicted label using the function $f(x_i; \theta)$. The purpose of training a machine learning classifier can be described as minimizing the loss function (1):

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i; \theta), y_i) \tag{1}$$

where, in classification problems, the loss function used is the cross-entropy loss function, which can be formally defined as in Equation (2):

$$L(\hat{y}_i, y_i) = -\sum_{i=1}^{N} y_i(N) \log \hat{y}_i(N) \tag{2}$$

given that $\hat{y}_i = f(x_i; \theta)$.

For binary classification, the cross-entropy loss function can be defined as in Equation (3):

$$L_{BCE}(\hat{y}_i, y_i) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \tag{3}$$

The convolution operation that composes the convolution layer is a mathematical operation that combines two signals, which can be formally defined as in Equation (4):

$$o[u, v] = f[m, n] * g[u, v] = \sum_m \sum_n f[m, n] \odot g[u + m, v + n] \tag{4}$$

where $f[m, n]$ is the convolution filter, $g[u, v]$ is the input image, and $o[u, v]$ is the output feature map. The convolution operation is shown in Figure 23.

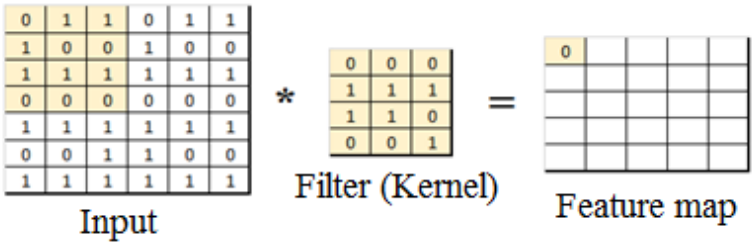


Figure 23. Convolutional operation.

The convolution layers convolve the input images with a small grid shape called the kernel, or convolution filter, starting from the top left corner of the image, as shown in Figure 24. The convolution filter is used to extract important features from the input images that will help in classifying the images. The weights of the convolution filter are the most important in the CNN, which will be learned from the iterative nature of the backpropagation algorithm. Many filters will be used to extract as many features from the images as possible to be able to correctly classify the images

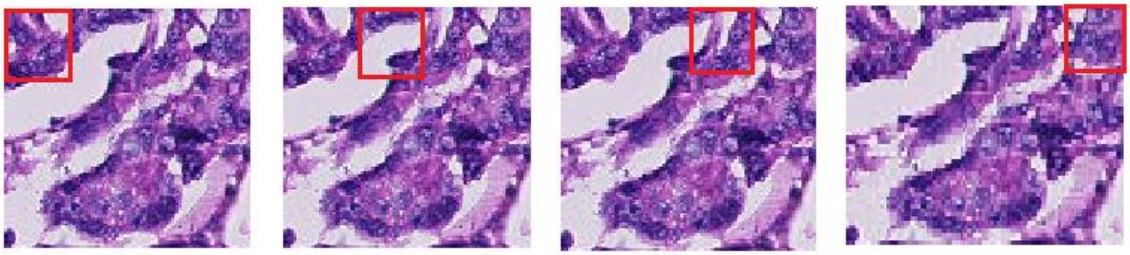


Figure 24. Convolution filter to extract features.

The result of convolving the filter over the input image will produce a matrix that is called a feature map, where it can be considered as a reduced image that will be used instead of the original image in the subsequent layers. Many feature maps will be outputted from the filters used, every one of these maps will capture different features from each image, and these features will be used to correctly classify the images (Kassani et al., 2019). The backpropagation algorithm is used to tune the weights of the CNN by updating the weights from the last layers to the first. To protect the CNN from vanishing or exploding gradients, the weights of the CNN are initialized with a certain distribution and not from zero. The weights of the network can be transferred from one to another to avoid initializing the weights from scratch.

7.3.2. Transfer Learning

The source dataset is the dataset being used to train the CNN weights to be used for another target dataset; usually, the source dataset contains millions of images with thousands of classes, like the ImageNet dataset (Deng et al., 2009). The following four techniques for transfer learning were introduced in the literature:

- The first technique is to freeze the source CNN's weights (like ImageNet's weights) and then remove the original fully connected layers and add another classifier, either a new fully-connected layer or any machine learning classifier, like support vector machine (SVM), that is, to use the original weights for feature extraction.
- The second technique is to fine-tune the top layers of the source CNN with a very small learning rate and freeze the bottom layers, under the assumption that the bottom layers are very generic and can be used for any kind of image dataset (Yosinski et al., 2014).
- The third technique is to fine-tune the entire network's weights using a very small learning rate to avoid losing the source weights, then remove the last fully connected layers, and add another layer to suit the target dataset.
- The fourth technique is to use the CNN's original architecture without importing any weights, that is, to initialize the weights from scratch. The point of this technique is using a well-known architecture that has been experimented with challenging datasets and proven to be good. Different transfer learning techniques are shown in Figure 25.

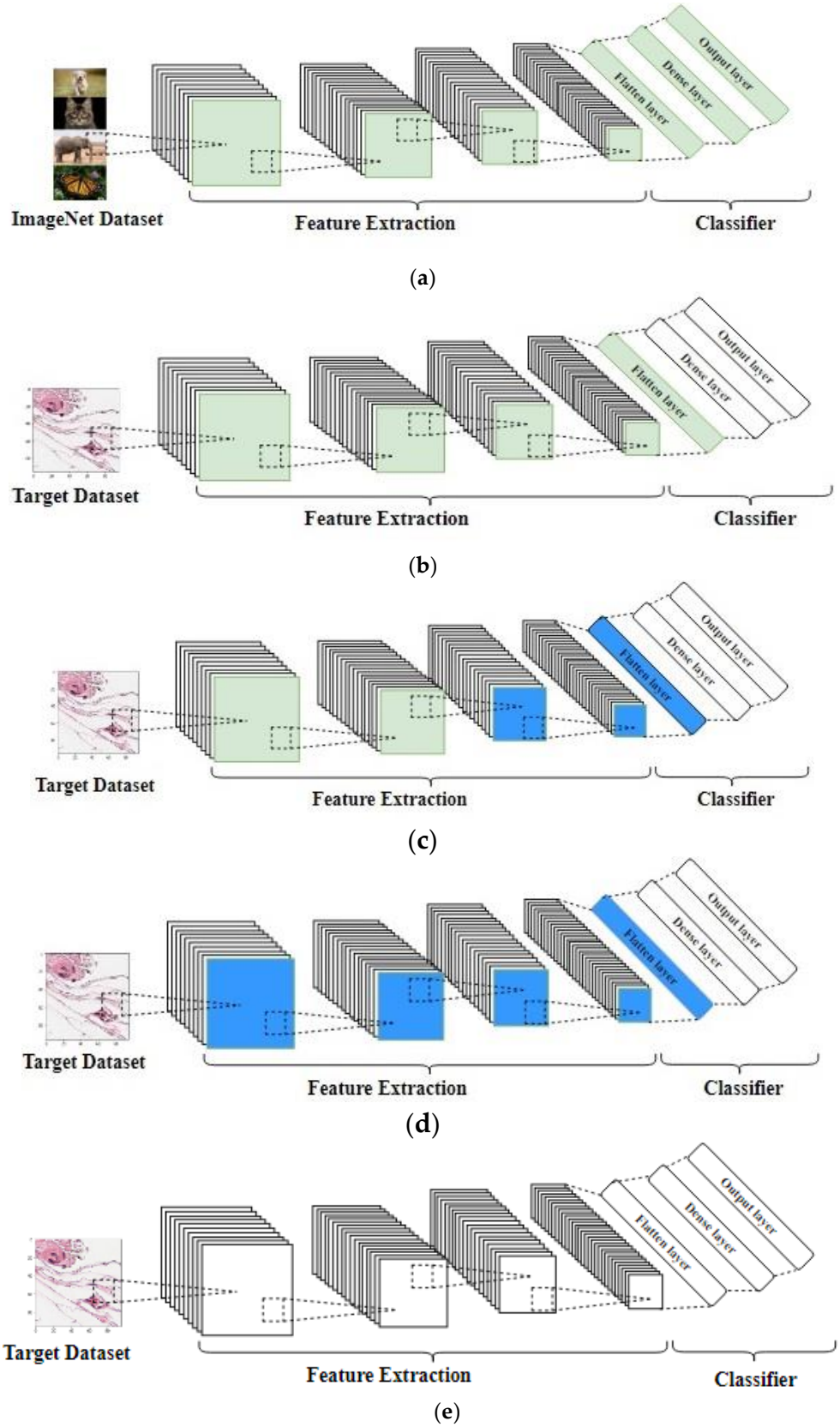


Figure 25. This figure shows different transfer learning techniques. (a) a generic CNN trained on ImageNet dataset, (b) the first technique, in which the source weights are fixed and the original classifier layers will be replaced by new layers to suit the target dataset, (c) the second technique, in which the top layers will be fine-

tuned, the weights of the bottom layer will be fixed, and the last fully connected layers will be replaced, (d) the third technique, in which the original classifier layers will be replaced and the entire network will be fine-tuned, and (e) the fourth technique, in which the original architecture will be used without any weights. The green color represents the weights learned from the ImageNet dataset, the blue color represents the fine-tuning of the ImageNet weights using the target dataset, and the white color means that the weights will be initialized from scratch.

According to Pan and Yang (2010), transfer learning in the image classification domain can be defined given the following parameters: $S: Source$, T: $Target$, $n: Source\ dataset\ size$, $m: Target\ dataset\ size$, $\phi(.)$: objective function, $\mathcal{D}: Domain$, $\mathcal{F}: feature\ space$, $P(X): Marginal\ probability\ distribution$, and $learning\ samples: X = \{x_1, x_2, \dots, x_n\} \in \mathcal{F}$, $\Upsilon: Label\ space$, $task: \mathrm{K} = \{\Upsilon, \phi(.)\}$, $n \gg m$.

An image domain, $\mathcal{D}$, is defined as having two components: a feature space, $\mathcal{F}$, and a probability distribution, $P(X)$:

$$\mathcal{D} = \{\mathcal{F}, P(X)\} \tag{4}$$

For a source domain, S, the dataset is $X_S = \{x_{S_1}, x_{S_2}, \dots, x_{S_n}\} \in \mathcal{F}_S$.

The source domain data can be denoted by:

$$\mathcal{D}_S = \{(x_{S_1}, y_{S_1}), (x_{S_2}, y_{S_2}), \dots, (x_{S_n}, y_{S_n})\} \tag{5}$$

where $x_{S_i} \in \mathcal{F}_S$ is the data instance, and $y_{S_i} \in \Upsilon_S$ is the corresponding class label.

For the target domain, T, the target domain can be denoted by:

$$\mathcal{D}_\mathrm{T} = \{(x_{\mathrm{T}_m}, y_{\mathrm{T}_m}), (x_{\mathrm{T}_m}, y_{\mathrm{T}_m}), \dots, (x_{\mathrm{T}_m}, y_{\mathrm{T}_m})\} \tag{6}$$

where $x_{\mathrm{T}_i} \in \mathcal{F}_\mathrm{T}$ is the data instance, and $y_{\mathrm{T}_i} \in \Upsilon_\mathrm{T}$ is the corresponding class label.

A task, K, consists of two components: a label space, Υ, and an objective function, $\phi(.)$:

$$\mathrm{K} = \{\Upsilon, \phi(.)\} \tag{7}$$

given a source domain, $\mathcal{D}_S$, and learning task, $\mathrm{K_S}$, a target domain, $\mathcal{D}_\mathrm{T}$, and its learning task, $\mathrm{K_T}$. Transfer learning aims to help improve the learning of the target predictive function $\phi_\mathrm{T}(.)$ in $\mathrm{K_T}$ using the knowledge in $\mathcal{D}_S$ and $\mathrm{K_S}$, where $\mathcal{D}_\mathrm{S} \neq \mathcal{D}_\mathrm{T}$ or $\mathrm{K_S} \neq \mathrm{K_T}$. $\mathcal{D}_S$ represents the ImageNet dataset in our research and its learning task, $\mathrm{K_S}$, where $\mathrm{K_S} = \{\Upsilon_\mathrm{S}, \phi_S(.)\}$, given that Υ_S is the source label space, and $\phi_S(.)$ is the source predictive function that is used to map a new image x_i to its label y_i. $\mathcal{D}_\mathrm{T}$ represents the histopathology dataset in our research and its learning task, $\mathrm{K_T}$, where $\mathrm{K_T} =$

$\{\mathcal{Y}_{\mathrm{T}}, \phi_{\mathrm{T}}(.)\}$. As stated before, we want to enhance $\phi_{\mathrm{T}}(.)$ of the histopathology dataset, $\mathcal{D}_{\mathrm{T}}$, using the ImageNet dataset, $\mathcal{D}_S$, and its objective function, $\phi_S(.)$ of $\mathrm{K_S}$.

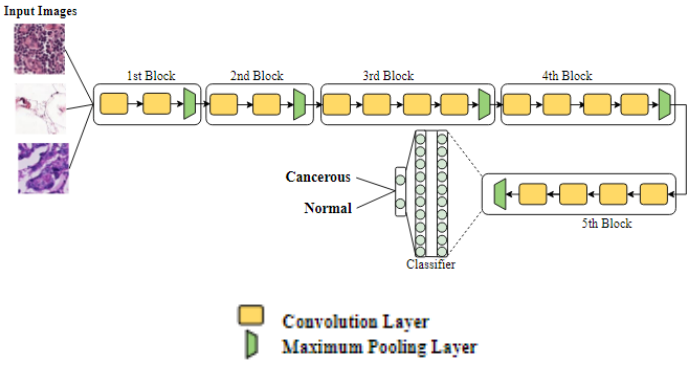
7.3.3. CNN Architectures

Since AlexNet architecture achieved first place in the ImageNet challenge in 2012 with an error rate of 16%, many architectures were introduced using CNN to classify images. In 2014, VGG (Simonyan & Zisserman, 2014) architectures, which are considerably deeper than AlexNet architecture (Krizhevsky et al., 2012), were introduced, and in the same year, GoogLeNet architecture (Szegedy et al., 2016) was introduced as well, which is considered deeper and wider than AlexNet. Then, in 2015, ResNet architecture (He et al., 2016) was introduced and was deeper and contained the residual connection, and it was followed by DenseNet (Huang et al., 2017) in 2017. All of these state-of-the-art CNNs were trained on the ImageNet dataset, and their weights are publicly available. In this study, we decided to take into account three CNNs, namely, VGG16, VGG19, and InceptionV3 architectures. This choice is related to the fact that these three networks are the ones commonly used in the Kaggle competition associated with this dataset, and, even more important, they produced better performance with respect to the other competitors. Additionally, the choice of the architectures to be considered is not fundamental for developing our study that focuses on understanding whether fine-tuning is a suitable approach for analyzing the histopathology dataset at hand. Below, we briefly describe the CNNs used in this paper.

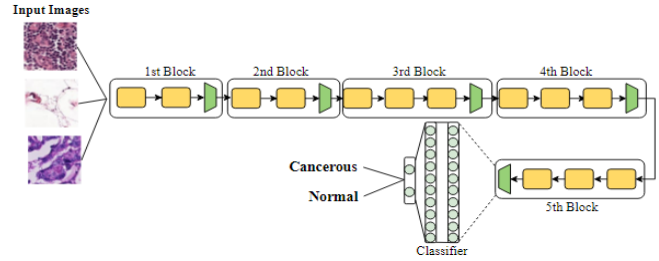
7.3.3.1. VGG Architectures

VGG architectures (Simonyan & Zisserman, 2014) were introduced by Oxford's Visual Geometry Group in 2014 to participate in the ILSVRC competition, where it achieved a top-5 error rate of 7.3%. Two networks, namely VGG16 and VGG19, were introduced, and the only difference between the two networks is the number of convolution layers used. VGG16 consists of 13 convolution layers, and VGG19 consists of 16 convolution layers and so is considered deeper than VGG16. Instead of using a convolution layer with a large filter size, the authors concatenated two layers with a smaller filter size, which reduced the number of parameters by 28%. VGG networks consist of five convolution blocks, where the first two blocks consist of two convolution layers each with a filter size of 3×3, the convolution layers in the first block have 64 filters each, while the convolution layers in the second block have 128 filters each. The third block in VGG16 consists of three convolution layers, and in VGG19, it has four convolution layers, all of the layers have 256 filters with size 3×3. The fourth and fifth convolution blocks consist of three convolution layers in VGG16 and four convolution layers in VGG19, and all of the layers have 512 filters with size 3×3. The five blocks are separated by a maximum pooling layer. Two fully connected layers are used as a classifier for the network with 4096 neurons. VGG16 has 138 million parameters with 23 layers'

depth, and VGG19 has 143 million parameters with 26 layers' depth. VGG architectures are shown in Figure 26.



(**a**)



(**b**)

Figure 26. This figure shows VGG network architectures: (a) VGG19 architecture and (b) VGG16 architecture.

7.3.3.2. InceptionV3 Architecture

Inception architectures (Szegedy et al., 2015) were first introduced by the authors of Reference (Szegedy et al., 2015) in 2015 to participate in the ImageNet competition, winning first place with a top-5 error rate of 6.65%. They were designed under the hypothesis that different scales of the same object require different filter sizes to be observed correctly. The inception module starts with the same input, and then it will be split into different convolutional layers with different kernel sizes and one max pooling layer—these filters will be parallel to each other, and then the output will concatenate to a single layer. Having these layers parallel to each other and not subsequent like in VGG models will save a lot of memory and will increase the model's capacity without increasing its depth. The inception module is shown in Figure 27. The version used in this paper is the third. Inception architecture consists of nine inception modules that are sequentially arranged. InceptionV3 architecture has 23.8 million parameters with 159 layers' depth. Three filter sizes, namely 1×1, 3×3, and 5×5, are used in a single inception module; in the updated version, the authors replaced the 5×5 with two 3×3 convolution layers, influenced by Reference (Simonyan & Zisserman, 2014). Inception architecture is shown in Figure 28.

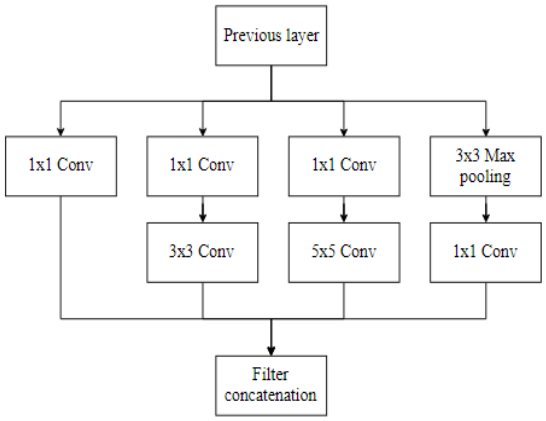


Figure 27. The inception module.

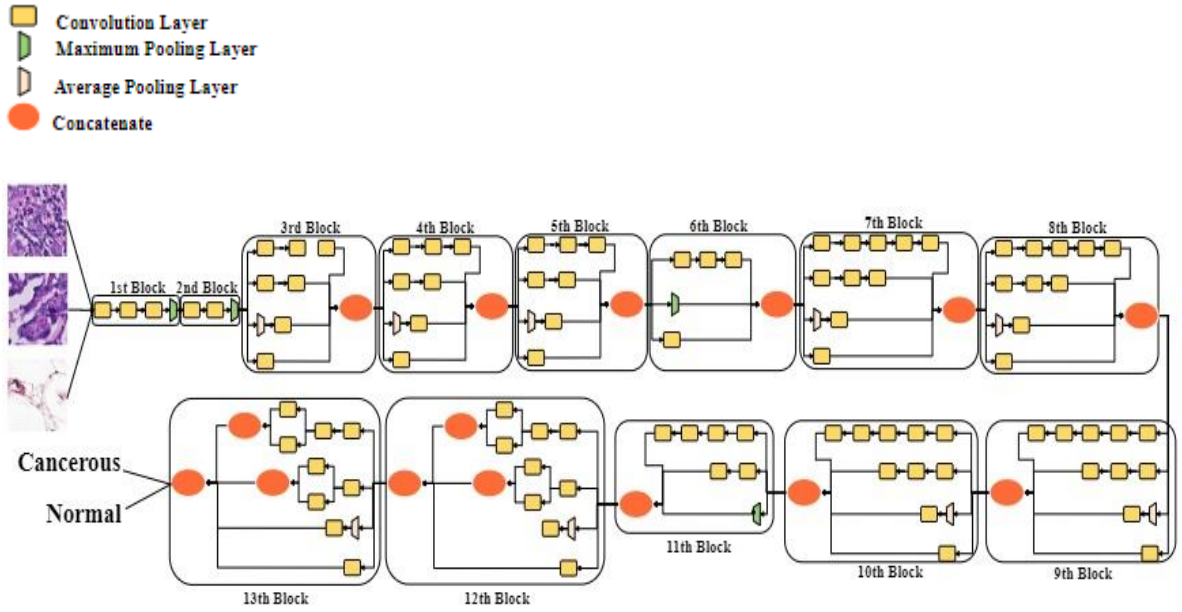


Figure 28. The Inception V3 architecture.

7.3.4. Datasets Used

CNN algorithms, especially for the state-of-the-art architectures that are very deep, are very data-hungry and need hundreds of thousands of images to be accurately trained.

Two datasets were used in this study. One very large dataset the ImageNet dataset, also called the original dataset—was used to train the CNN. The second dataset, the histopathology dataset, was the target dataset that we wanted to classify and that we used to fine-tune the weights that the ImageNet dataset learned. A brief description of both datasets follows.

7.3.4.1. ImageNet Dataset

In 2010, the ImageNet challenge, also known as the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), was introduced to push forward advances in the field of computer vision.

ImageNet has millions of hand-labeled images using thousands of labels. The images are mainly everyday images, including animals, bridges, cars, and furniture, among others. A sample of the ImageNet dataset is shown in Figure 29.

Figure 29. A sample of the ImageNet dataset.

7.3.4.2. PatchCamelyon Histopathology Dataset

The PatchCamelyon histopathology dataset (Bejnordi et al., 2017; Veeling et al., 2018) is a publicly available dataset that consists of 220,000 labeled images and 57,000 unlabeled images and contains a 60% positive class and a 40% negative class. The positive class means the middle region of the image (32×32) contains tumor tissue. The dataset has no duplicates. The test set can be evaluated on the Kaggle website, which produces the AUC of the ROC curve. A sample of the PatchCamelyon dataset is shown in Figure 30.

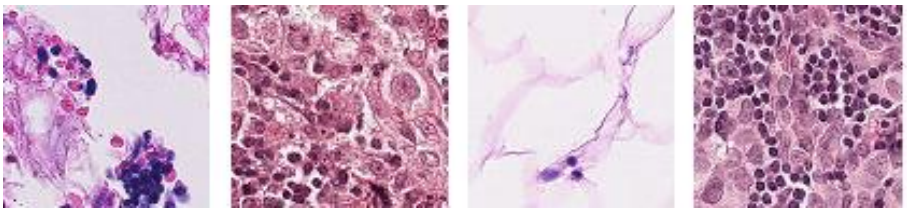


Figure 30. A sample of the PatchCamelyon dataset.

7.3.5. Performance Measures

Several performance measures have been introduced in the literature to assess the quality of the classifier. One of the most popular measures is the accuracy metric, wherein the classification domain can be defined as in Equation (9):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

where TP is the true positive, in which the positive class is predicted as positive, TN is the true negative, in which the negative class was predicted as negative, FP is the false positive, in which the negative class is incorrectly classified as positive, and FN is the false negative, in which the positive class is incorrectly classified as negative. The main drawback of the accuracy metric is that it is only robust for balanced datasets; otherwise, it can be misleading. Another metric introduced was the sensitivity metric (also known as the true positive rate, or TPR), which is defined as in Equation (10):

$$Sensitivity = \frac{TP}{TP + FN} \tag{9}$$

which focuses on the positive classes and how accurately they were classified. Another metric, called the $Specificity$ (also known as the true negative rate, or TNR), was introduced to focus on the negative classes, which is defined as in Equation (11):

$$Specificity = \frac{TN}{TN + FP} \tag{10}$$

To combine the performance of both $Sensitivity$ and $Specificity$, a new metric called the receiver operating characteristic (ROC) curve was introduced. The ROC curve was developed during the Second World War (Goncalves et al., 2014). It is a metric that is widely used to measure and visualize the classifier's ability to distinguish between two classes (Fawcett, 2006). The curve plots the $sensitivity$ against the $(1 - specificity)$. To represent the area under the ROC curve, a metric called the area under the curve (AUC) is used. The AUC of the ROC curve ranges from 0.5 to 1, $0.5 \leq AUC \leq 1$, where an AUC equal to 1 means that the classifier is able to distinguish between the positive and negative classes perfectly, and an AUC equal to 0.5 means that the classifier is just a random guess.

7.3.6. Measures to Avoid Overfitting

Due to the importance of any medical-images classifier, all of the measures necessary should be taken to ensure that the image classifier is robust against overfitting. Overfitting is defined as the model's bad performance on the test dataset and perfect performance on the training dataset (Srivastava et al., 2014), which emphasizes the model's generalizability. Many regularization techniques were introduced in the literature to overcome the problem of classifier overfitting by increasing the training error in exchange for decreasing the testing error, meaning to decrease the model variance by increasing the model bias (Goodfellow et al., 2016). The following measures aim to decrease errors in the test dataset irrespective of errors in the training dataset.

7.3.6.1. Early Stopping

A technique used to control the number of training epochs is early stopping. The model training will stop if the accuracy of the validation dataset does not change for a predefined number of epochs, meaning that the training will stop if the model starts to overfit the training dataset. Early stopping saves computational power and helps the CNN from overfitting the training dataset by doing useless epochs that will take a long time and decrease the network's accuracy (Goodfellow et al., 2016). Early stopping can help optimize the epoch's size hyperparameter by setting the epoch size too high and then letting early stopping automatically halt the training if no increase in accuracy occurs (Bengio, 2012).

7.3.6.2. Best Model Saved

The model will be saved only if the accuracy in that epoch is larger than the largest accuracy achieved so far. The final model is the last saved one.

7.3.6.3. Dropout

Srivastava et al. (2014) introduced dropout, which is considered a very important regularization technique that is usually used as an ensemble technique and is similar to a bagging algorithm (Breiman, 1996). Every single neuron that each layer is applied to has a probability, p, to be dropped temporarily during training, so it can construct many networks from the same single network. Dropout can make a CNN very robust against overfitting (Goodfellow et al., 2016).

7.3.6.4. Image Augmentation

One of the main methods to reduce overfitting is to have a huge training dataset to train the model on every single possible image, but practically, that is impossible, which is how augmentation was introduced. To increase the training dataset, image augmentation can be applied. Image augmentation is defined as an algorithm that can be used to create artificial images by modifying the original images through a series of transformations, rotations, and altered brightness, among other possible modifications.

7.4. Results

In this study, three CNN architectures were fine-tuned block-wise to determine the effect of fine-tuning each block on the generalizability of the network. The CNN architectures used were VGG16, VGG19, and InceptionV3. The original weights used were the ImageNet dataset weights. The dataset used was the publicly available PatchCamelyon dataset, and the dataset size was 220,000 labeled images for training, which consists of 60% of the positive class and 40% of the negative class. A separate 57,458 unlabeled images were provided to assess the classifier's performance, and the results of classifying the test dataset were uploaded to the Kaggle website to determine the AUC of the ROC curve. In all of the experiments, the Keras library was used with Python. A schematic diagram of the proposed model is shown in Figure 31.

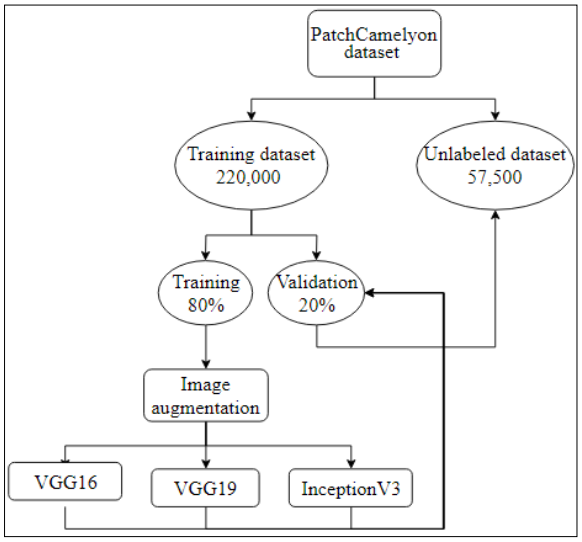


Figure 31. A schematic diagram of the proposed model.

7.4.1. Experiment Parameters

Three CNN architectures were fine-tuned block-wise. The batch size used was 64, and three learning rates, namely 10^{-3}, 10^{-4}, and 10^{-5}, were applied. The Adam optimizer (Kingma & Ba, 2014) was used in all of the experiments. All of the images were kept at the original dimensions of $96 \times 96\ pixels$. The original fully connected layers of the architectures were removed and replaced by a dropout layer with a probability of 50% to increase the network accuracy, and a new fully-connected layer was used as a classifier. The training dataset was divided into two partitions: 80% and 20% for training and validation, respectively. To increase the size of the training dataset and to overcome the translation of the images, image augmentation was applied with the following parameters: horizontal and vertical flipping, $180°$ rotation, width and height shifting, and shearing and zooming. It is worth noting that image augmentation was applied to the training dataset only and not the validation dataset. The best model will be saved in every epoch, and the early stopping criteria of 10 epochs were applied.

Each CNN architecture was divided into blocks based on their design. For VGG networks, the blocks were divided based on the max-pooling layers, and for the InceptionV3 network, the blocks were defined based on the inception module, meaning that every inception module represents a block. VGG architectures have a total of 5 blocks, and the InceptionV3 network has 13 blocks. Each network was fine-tuned backward, meaning that the first block to be fine-tuned in the VGG networks was the 5th, and the first block to be fine-tuned in the InceptionV3 network was the 13th.

7.4.2. Experiment Results

The results of fine-tuning VGG16 with three different learning rates are shown in Table 29. These results are the AUC of the ROC curve from the Kaggle website using the unlabeled dataset. The first block that was trained was the 5th block, and the results of the three learning rates are

approximately the same, although the highest was with the 10^{-3} learning rate. The second block was the 4th, which was fine-tuned with the 5th layer, and the highest AUC obtained was by the 10^{-4} learning rate followed by the 10^{-3} and 10^{-5}. The AUC results did increase by fine-tuning the 4th as well. The third experiment fine-tuned the blocks from the 5th to the 3rd, and the results obtained were higher than those of the previous two experiments. The highest was obtained by using the learning rate of 10^{-4}. The fourth experiment fine-tuned the blocks from the 5th to the 2nd, and the results show that the accuracy started to decrease compared to the previous experiments. The fifth experiment fine-tuned the entire network, and the results were higher than the previous experiment but overall lower than the third experiment. Overall, the highest AUC was 96%, which was achieved by fine-tuning the last 3 blocks and using the 10^{-4} learning rate.

Table 29. Results of the VGG16 architecture. Where LR is the learning rate.

VGG16 Test AUC Results			
Blocks	**LR = 10^{-3}**	**LR = 10^{-4}**	**LR = 10^{-5}**
Fine-Tuning 5th Block	0.9303	0.9260	0.9212
Fine-Tuning 4th Block	0.9398	0.9480	0.9382
Fine-Tuning 3rd Block	0.9364	0.9603	0.9475
Fine-Tuning 2nd Block	0.8893	0.9350	0.9384
Fine-Tuning ALL	0.9383	0.9310	0.9404

The results of fine-tuning the VGG19 network are shown in Table 30. The same procedure followed to fine-tune the VGG16 architecture was employed. According to the results reported in Table 30, it is possible to see that the highest AUC was obtained by fine-tuning the last three blocks, by freezing the other blocks, and with a learning rate of 10^{-4}. A comparable performance is obtained with a learning rate of 10^{-5}, while when a learning rate of $10^{-3}$was used, the best performance was obtained by fine-tuning only the last two layers.

Table 30. Results of the VGG19 architecture.

VGG19 Test AUC Results			
Blocks	**LR = 10^{-3}**	**LR = 10^{-4}**	**LR = 10^{-5}**
Fine-Tuning 5th Block	0.9082	0.9028	0.9058
Fine-Tuning 4th Block	0.9268	0.9266	0.9235
Fine-Tuning 3rd Block	0.9087	0.9514	0.9440
Fine-Tuning 2nd Block	0.8377	0.9480	0.9254
Fine-Tuning ALL	0.8669	0.9427	0.9342

The results of fine-tuning the InceptionV3 architecture are shown in Table 31. The first experiment conducted using the InceptionV3 architecture was to freeze the entire network and fine-tune the last block (13th block) three times using three learning rates. The results were similar across the three learning rates, with the highest found by using the highest learning rate (10^{-3}) and the lowest by using the medium learning rate (10^{-4}). This procedure (that consists of fine-tuning the last *n* blocks and freezing the remaining blocks) was subsequently iterated for *n* in the range [2-13] and by considering at each iteration the three different learning rates. According to the results reported in Table 31, when considering a learning rate of 10^{-4} and 10^{-5}, the best performance was obtained by fine-tuning the whole architecture (i.e., *n* = 13). On the other hand, with a learning rate of 10^{-3}, the best performance was achieved by fine-tuning the last eight blocks (from block 6 to block 13), thus freezing the first five blocks.

Table 31. Results of the InceptionV3 architecture.

InceptionV3 Test AUC Results			
Blocks	**LR = 10^{-3}**	**LR = 10^{-4}**	**LR = 10^{-5}**
Fine-Tuning 13th Block	0.8250	0.8220	0.8221
Fine-Tuning 12th Block	0.8514	0.8466	0.8446
Fine-Tuning 11th Block	0.8702	0.8446	0.8274
Fine-Tuning 10th Block	0.8648	0.8675	0.8429
Fine-Tuning 9th Block	0.8526	0.8816	0.8538
Fine-Tuning 8th Block	0.8574	0.8637	0.8469
Fine-Tuning 7th Block	0.8673	0.8429	0.8907
Fine-Tuning 6th Block	0.8923	0.8468	0.8950
Fine-Tuning 5th Block	0.8680	0.8730	0.8883
Fine-Tuning 4th Block	0.8483	0.8335	0.8686
Fine-Tuning 3rd Block	0.7715	0.8575	0.8641
Fine-Tuning 2nd Block	0.7175	0.8636	0.8785
Fine-Tuning ALL	0.8071	0.9058	0.9280

To demonstrate the potential of the fine-tuning approach, we decided to compare its performance against the one obtained by using CNNs trained from scratch, specifically for the PatchCamelyon dataset.

The results of training the architectures from scratch are summarized in Table 32. The first experiment was to train the three CNNs from scratch using three different learning rates. For VGG16 architecture, using the highest learning rate, the model did not converge at all for the stopping criteria we imposed. By using the medium learning rate, the model converged to an acceptable performance,

which is the highest in this set of experiments but was lower than the result obtained by fine-tuning the network. The lowest learning rate did not produce satisfactory performance, and performs poorer with respect to the network trained by considering the medium learning rate value. Concerning the VGG19 architecture, it did behave the same as the VGG16 for the highest learning rate. The performance achieved using the medium learning rate value was lower than the one achieved by using the lowest learning rate. InceptionV3 architecture did converge for the smaller learning rate, and the best performance was obtained by using the medium learning rate value. Overall, training the network from scratch did not achieve better results than fine-tuning the network.

Table 32. Results of different architectures trained from scratch.

Training from Scratch AUC Results			
Networks	**LR = 10^{-3}**	**LR = 10^{-4}**	**LR = 10^{-5}**
VGG16	50%	90.55%	85.91%
VGG19	50%	84.77%	85.81%
InceptionV3	84.83%	87.93%	81.97%

7.4.3. Experiment Results on a Different Histopathology Dataset

To corroborate the results obtained with the PatchCamelyon dataset and to strengthen our findings, we performed a second set of experiments using a different histopathology dataset, namely the BreakHis (Spanhol et al., 2016) dataset. The BreakHis dataset consists of 7909 images split into 2480 benign images and 5429 malignant images. To conduct our experiments, we split the dataset into 80% to train the model, 10% to validate the model during training, and 10% to test the model on unseen data. The early stopping was increased to 50 epochs because of the dataset size. The results of fine-tuning the BreakHis dataset using the VGG16 architecture are shown in Table 33, the results of using VGG19 are summarized in Table 34, and lastly, the results of using InceptionV3 are shown in Table 35.

As one can see from the analysis of Table 33, the results obtained by fine-tuning VGG16 using the BreakHis dataset match the results obtained on the PatchCamelyon dataset. In particular, fine-tuning the top layers and using the smallest learning rates of 10^{-4} and 10^{-5} did achieve the highest results. More in detail, with a learning rate of 10^{-3}, the best performance was achieved by fine-tuning only the 5th block; with a learning rate of 10^{-4}, the best performance was obtained through the fine-tuning of blocks from the 5th to the 3rd. Finally, the overall best performance was obtained with a learning rate of 10^{-5} and with the fine-tuning of blocks from the 5th to the 3rd.

Table 33. Results of the VGG16 architecture using the BreakHis dataset.

VGG16 Test AUC Results			
	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
Fine-Tuning 5th Block	89.51%	91.76%	89.73%
Fine-Tuning 4th Block	88.50%	93.58%	94.03%
Fine-Tuning 3rd Block	86.21%	95.39%	95.76%
Fine-Tuning 2nd Block	86.79%	94.90%	93.91%
Fine-Tuning ALL	85.04%	92.47%	93.04%

Also, for the results obtained by fine-tuning VGG19 using the BreakHis dataset (Table 34), one can notice a similar behavior with respect to the analysis performed with the PatchCamelyon dataset: fine-tuning the top layers was sufficient to produce the best results, and the smaller learning rates yielded the best performance. More in detail, the best performance when considering a learning rate of 10^{-3} was obtained by only fine-tuning the 5th block. On the other hand, for both the remaining learning rates of 10^{-4} and 10^{-5}, the best performance was achieved by fine-tuning blocks from the 5th to the 3rd.

Table 34. Results of the VGG19 architecture using the BreakHis dataset.

VGG19 Test AUC Results			
	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
Fine-Tuning 5th Block	89.14%	90.11%	88.68%
Fine-Tuning 4th Block	87.41%	91.67%	93.80%
Fine-Tuning 3rd Block	88.72%	96.79%	94.46%
Fine-Tuning 2nd Block	50.00%	95.46%	94.39%
Fine-Tuning ALL	87.29%	95.26%	94.30%

Moving to the results obtained by fine-tuning InceptionV3 using the BreakHis dataset (Table 35), it is possible to observe the same pattern already discussed when we considered the PatchCamelyon dataset. In particular, with this architecture, fine-tuning the top layers did not achieve any satisfactory results. As for the PatchCamelyon dataset, the best results were achieved by fine-tuning the entire network.

Table 35. Results of the InceptionV3 architecture using the BreakHis dataset.

InceptionV3 Test AUC Results			
Blocks	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
Fine-Tuning 13th Block	55.09%	55.80%	55.26%

Fine-Tuning 12th Block	61.61%	57.78%	53.45%
Fine-Tuning 11th Block	56.92%	56.74%	62.26%
Fine-Tuning 10th Block	61.58%	54.85%	53.54%
Fine-Tuning 9th Block	59.33%	55.35%	51.61%
Fine-Tuning 8th Block	51.26%	50.79%	50.63%
Fine-Tuning 7th Block	58.38%	51.62%	50.41%
Fine-Tuning 6th Block	56.63%	50.90%	53.38%
Fine-Tuning 5th Block	56.87%	50.52%	50.64%
Fine-Tuning 4th Block	57.46%	54.39%	50.57%
Fine-Tuning 3rd Block	52.00%	50.00%	50.00%
Fine-Tuning 2nd Block	50.00%	50.00%	52.00%
Fine-Tuning ALL	85.69%	94.72%	94.41%

Finally, Table 36 summarizes the results obtained by training the architectures from scratch on the BreakHis dataset. According to the results reported in Table 36, when considering the VGG16 architecture and the highest learning rate, the model did not converge at all for the stopping criteria we imposed. By using the medium learning rate, the model converged to an acceptable performance, which is the highest in this set of experiments but was lower than the result obtained by fine-tuning the network, it showed the lowest learning rate and performs poorer with respect to the network trained by considering the medium learning rate value. Focusing on the VGG19 architecture, it did behave the same as the VGG16 for all the considered learning rates. More in detail, the best performance was obtained with a learning rate of 10^{-4}. Considering the InceptionV3 architecture, the results follow the same trend observed with the PatchCamelyon dataset. In particular, the training process did converge for the smaller learning rate, and the best performance was obtained by using the medium learning rate value. Similar to the results achieved on the PatchCamelyon dataset, Table 36 shows that training the network from scratch did not achieve better results than fine-tuning the network.

Table 36. Results of different architectures trained from scratch using the BreakHis dataset.

Training from Scratch AUC Results			
Networks	**LR = 10^{-3}**	**LR = 10^{-4}**	**LR = 10^{-5}**
VGG16	50%	90.45%	86.51%
VGG19	50%	92.02%	85.65%
InceptionV3	88.32%	92.65%	81.64%

7.5. Discussion

Training a CNN for a medical dataset is a very difficult process because of the scarcity of medical images due to many reasons, and that is where transfer learning, in which the weights of another CNN that was trained on a different very large dataset can be used, can be very important for the medical field. This process of re-training the CNN on a target dataset is called fine-tuning. Fine-tuning the entire CNN is very time-consuming and does not guarantee the best performance. As pointed out by Yosinski et al. (2014), for natural image datasets, the lower layers will learn more generic features, like circles and edges, that are common to mainly all image datasets, while the top layers can learn the very specific features of the original dataset.

In this study, we compared the block-wise effect of fine-tuning three state-of-the-art CNNs to classify the images of two histopathology datasets using three learning rates. The results of this study showed that fine-tuning the entire network did not give the best performance for the VGG architectures; instead, fine-tuning only the top blocks yielded the best performance. For the InceptionV3 architecture, fine-tuning the entire network did increase its performance. The main argument is in regard to the generalizability of the bottom layers, which can be used for any kind of dataset, meaning the weights of the bottom layer can be frozen.

The results show that the learning rate should be low in order to not to mess up the original weights of the network; also, making the learning rate very low will not increase the performance and will slow down the learning process. From the experiments, we found out that a 0.0001 learning rate achieved the perfect combination between training time and performance. Also, training the architectures from scratch was compared to fine-tuning techniques, and our results showed that training the network from scratch did not achieve higher results than fine-tuning the network. Also, using higher learning rates made shallow architectures like VGGs very unstable and prevented them from convergence.

Comparing the results that we obtained with those obtained by other authors was difficult because the other methods used either different evaluation criteria or different datasets. For example, Kassani et al. (2019) used the PatchCamelyon dataset as well and achieved accuracy of 94.64%, but the authors did not explain how they used transfer learning or whether they fine-tuned the entire networks or not, did not state the size of their test dataset, and did not state whether or not they used the unlabeled dataset. The highest result we achieved was done by fine-tuning the VGG16 top blocks to the 3rd block, and it was an AUC of 96%. As stated earlier, the results were obtained by predicting the unlabeled test dataset and submitting that to the Kaggle platform; therefore, all of our results are in the AUC of the ROC curve because that is the criteria used by Kaggle to assess the performance of the classifier.

As for other authors who used different histopathology datasets, Ahmad et al. (2019) reported an accuracy of 85% for the BioImaging dataset using a fine-tuned ResNet architecture, but the authors did not report the transfer learning technique they used to achieve this result. Sharma et al. (2018) achieved an accuracy of 92.6% and an AUC of 95.65% by using the VGG16 network with logistic regression as a classifier, but the authors did not investigate the role each block plays in the network performance. Further, the authors split the dataset into two partitions to assess its accuracy using three heuristics: the first split the dataset into 90% and 10%, the second split the dataset into 80% and 20%, and the last split the dataset into 70% and 30%. The authors did not use any validation algorithms like k-fold validation. The results of the three heuristics are approximately similar to each other. Deniz et al. (2018) compared fine-tuning with feature extraction for the BreakHis dataset, and the authors reported that the fine-tuned AlexNet outperformed the VGG16 with SVM as a feature extractor. We can conclude from these studies that fine-tuning a CNN outperformed other techniques and that encouraged us to investigate the role of each block to further increase the CNN's performance.

7.6. Conclusions

This paper concentrated on the impact of CNN architecture depth while fine-tuning the network. The architectures used, namely VGG16, VGG19, and InceptionV3, were pre-trained on the ImageNet dataset. All of the networks were fine-tuned on two different histopathology datasets to determine the effect of each block on the generalizability of the network. Three learning rates were used with an Adam optimizer to determine the effect of learning on the performance as well. Our results suggest that for shallow networks like VGG, fine-tuning the top layers can be sufficient to obtain decent results, while for deep networks like InceptionV3, fine-tuning the entire network can yield better results. In all cases, the low learning rate is highly recommended when fine-tuning the network in order to not mess up the original weights. Our recommendation in the case of a scarcity of images is that fine-tuning a pre-trained CNN can be a feasible option. The conclusions drawn from this study, concerning fine-tuning, refer to the particular application at hand. To further strengthen our findings and to draw more general conclusions that could be applied independently from the applicative domain, it would have been necessary to perform a study that involved dozens and dozens of datasets over different domains.

In future work, we will use different optimizers to detect the effect of the optimizers as well and will use different datasets, including non-medical ones, to detect the block-wise effect of fine-tuning CNNs.

Chapter 8. Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images[6]

Abstract: The classification of histopathology images requires an experienced physician with years of experience to classify the histopathology images accurately. In this study, an algorithm was developed to assist physicians in classifying histopathology images; the algorithm receives the histopathology image as an input and produces the percentage of cancer presence. The primary classifier used in this algorithm is the convolutional neural network, which is a state-of-the-art classifier used in image classification as it can classify images without relying on the manual selection of features from each image. The main aim of this research is to improve the robustness of the classifier used by comparing six different first-order stochastic gradient-based optimizers to select the best for this particular dataset. The dataset used to train the classifier is the PatchCamelyon public dataset, which consists of 220,025 images to train the classifier; the dataset is composed of 60% positive images and 40% negative images, and 57,458 images to test its performance. The classifier was trained on 80% of the images and validated on the rest of 20% of the images; then, it was tested on the test set. The optimizers were evaluated based on their AUC of the ROC curve. The results show that the adaptive based optimizers achieved the highest results except for AdaGrad that achieved the lowest results.

8.1. Introduction

To evaluate whether tissue is cancerous, a sample is taken from the suspicious area and then evaluated, under an optical microscope, by the pathologist. This procedure is very time-consuming and extremely complicated (Jukić et al., 2011), and therefore, it requires an expert pathologist with years of experience. Depending on the particular task, even an expert pathologist could make errors. This complicated procedure often demands a second opinion or even assistance, which is where artificial intelligence assumes a role. Artificial intelligence (AI) can provide significant help, whether through a lot it is automation or to furnish the pathologist with a second opinion. AI can be defined as using a computer to generate a prediction of each image by training a deep neural network model. The training process consists of feeding the system labeled pathology images, after which the algorithm seeks; first, to map a function between the input label and the prediction and second, measures the error and tries to minimize it. The state-of-the-art algorithm used in the image classification is the convolutional neural network.

In the context of image classification, deep learning may be defined as a computer program is said to learn from experience E, like pathology images, concerning some task T, like image classification that differentiates between cancerous and non-cancerous images, and is capable of recognizing the relevant image without being explicitly programmed to do so, and using a performance measure like the AUC of the ROC curve. The algorithm's performance on the image classifier, as measured by the AUC, improves by adding more images. Practically speaking, machine

[6] This chapter has been published in MDPI journal: Kandel, I.; Castelli, M.; Popovič, A. Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *J. Imaging* 2020, *6*, 92. https://doi.org/10.3390/jimaging6090092

learning is the task of recognizing patterns from training images and applying these patterns to identify an image with an unknown label.

The convolutional neural network (CNN) has been used as an image classification algorithm for nearly two decades (Fukushima, 1980). The real power of CNN was rediscovered in the context of the ImageNet competition, where millions of images, with thousands of labels, were classified with 85% accuracy; at that time, CNN resumed its former role as one of the most important algorithms for image classification (Tajbakhsh et al., 2016). CNN has been applied in different image classification domains, such as agriculture [4–6] and traffic detection [7,8]. With the rapid improvements in GPU cards and the increasing size of datasets, many influential and robust architectures, like AlexNet (Krizhevsky et al., 2012), VGG16 (Simonyan & Zisserman, 2014), VGG19 (Simonyan & Zisserman, 2014), ResNet50 (He et al., 2016), and InceptionV3 (Szegedy et al., 2015), were introduced. Transfer learning is a deep learning technique, which allows the knowledge acquired during training on previous models to be applied to new tasks. Transfer learning has many advantages. It saves time by starting from the end point of the most recent training, instead of training the new model from scratch; it extends the knowledge it acquired from previous models; transfer learning is particularly useful when the size of the new training dataset is small. Transfer learning has made significant contributions to the fields of computer vision, audio classification, and natural language processing.

The difference between the predicted label and the correct label is called the cost function; the whole point of the algorithm is to minimize this cost function. As the algorithm most commonly used to minimize the cost function, backpropagation is an iterative algorithm, where each of its iterations consists of two passes: A forward pass throughout the entire network, where the inputs are propagated from the input layer to the output layer. At this point, the cost function is be calculated to measure the performance of the network; then there is the backward pass, where the weights are backpropagated from the output to the input of the network. The optimizers are used to minimize this cost function.

This work evaluates different first-degree optimizers used to classify pathology images as cancerous or non-cancerous. Each optimizer is evaluated based on its performance and convergence time. Four CNN architectures will be used to compare the performance of each optimizer to those of the others.

8.2. Related Works

Many works compared the performance of different optimizers in the context of different neural network architectures; the reported approaches differ in relation to the network architecture, datasets, and the optimizers under study.

In a study by Dogo et al. (2018), the authors evaluated the performance of seven optimizers on three image datasets: Natural Images dataset, Cats and Dogs dataset, and Fashion MNIST dataset. The authors evaluated the performance of each optimizer based on accuracy achieved and the convergence time, where convergence consists of reaching the minimum of the function. To determine the performance quality of each optimizer, the authors proposed a simple CNN architecture, with three convolutional layers, and one dense layer with 64 neurons. For the Cats and Dogs dataset, the Nadam optimizer achieved the best performance, and the Adadelta optimizer produced the most mediocre performance; the RMSProp represents the shortest convergence time, and the Nadam optimizer achieved the longest convergence. For the Fashion dataset, the Adam optimizer achieved the highest degree of accuracy, and the Adadelta optimizer displayed the lowest accuracy; the Adamax optimizer achieved the shortest convergence time, and the Adadelta optimizer had the longest convergence time was the Adadelta optimizer. For the Natural dataset, the Nadam optimizer was the best performer, and the Adagrad optimizer exhibited the most inferior accuracy; the SGD algorithm achieved the shortest convergence time, and the Adadelta algorithm had the longest convergence time. The authors concluded that the Nadam optimizer was the best of all tested optimizer, due to its combined mastery of the momentum and the adaptive gradient estimation.

The authors Prilianti et al. (2019) compared the performance of seven optimizers on the digital plant dataset. To evaluate each optimizer, the authors used three CNN architectures; the first was a shallow network with only one convolutional layer and without any dense layers; the second CNN architecture used was the LeNet architecture, which was introduced by Lecun et al., (1998); and the third CNN architecture was the AlexNet (Krizhevsky et al., 2012). The authors evaluated the performance of each optimizer on each CNN architecture, based on the mean square error (MSE). The Adam optimizer achieved the lowest MSE for the shallow net architecture, as well as the LeNet architecture, while the Adadelta achieved the lowest MSE on the AlexNet architecture. The authors concluded that Adam optimizer achieved the best performance.

Jangid and Srivastava (2019) assessed the performance of three optimizers on handwritten Devanagari characters. The optimizers tested were Adam, Adamax, and RMSProp. To evaluate each optimizer, the authors introduced a CNN architecture with three convolutional layers and one dense layer with 1000 neurons. For this architecture, RMSProp achieved the best accuracy. Swastika et al. (2019) evaluated three optimizers to classify vehicle types: Adam, Adadelta, and SGD. The authors used three CNN architectures to evaluate each optimizer: a shallow network, LeNet, and MiniVGGNet. The optimizers were evaluated based on their accuracy, which meant that the Adadelta optimizer was the best for the Mini VGGNet architecture.

This study uses four CNN architectures to perform a comparative evaluation of six first-degree stochastic gradient descent optimizers: the optimizers tested are Nesterov gradient descent, Adagrad, Adam, Adamax, Nadam, and RMSProp; and the CNN architectures tested are VGG16, InceptionV3, DenseNet, and ResNet50. The optimizers are evaluated based on their AUC of the ROC curve and their convergence time. All the optimizers' default hyperparameters were kept constant throughout the experiment, except the learning rate, which was set to three values 0.001, 0.0001, and 0.0001. Fine-tuning was applied to each network to adjust its weight to the new dataset.

8.3. Methodology

8.3.1. Dataset

The public available PatchCamelyon dataset (Bejnordi et al., 2017; Veeling et al., 2018) was used in this study. The images represent sentinel axillary lymph nodes to investigate the spread of breast cancer. The dataset was sampled from two hospitals in the Netherlands, experienced pathologists from the Netherlands annotated the dataset labels. The dataset was acquired from the Kaggle platform. The dataset consists of 220,025 images to train the classifier; the dataset is composed of 60% positive images and 40% negative images, and it includes 57,458 unlabeled images to test the classifier performance. All images have dimensions of 96 × 96 pixels. Eighty percent of the dataset is used to train the classifier, which is subsequently evaluated with the other 20% of the dataset images; the classifier is also tested on the online set of 57,458 images, and the results are uploaded to the Kaggle platform to detect the model performance. A sample of images is presented in Figure 32.

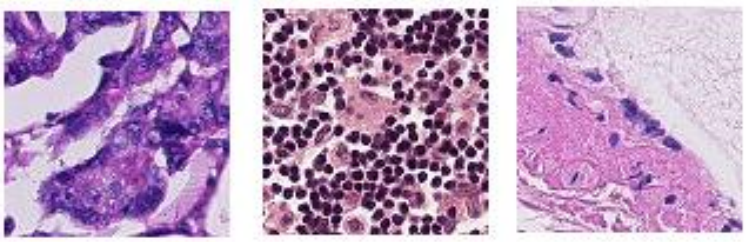


Figure 32. Example of images available in the PatchCamelyon dataset.

8.3.2. Convolutional Neural Networks

CNN is the most used algorithm in image classification, where it is understood to be a deep learning algorithm that serves as a feed-forward neural network with more than one hidden layer. The CNN for image classification was introduced by Fukushima (1980) to mimic the biological visual cortex of the brain. CNN combines sophisticated features obtained from the higher layers of the network with the generic features obtained from the lower layers of the network. The most critical layer of CNN is the convolution layer, which is responsible for capturing the temporal and spatial information of each image; the convolutional layer must conduct the convolution operation, which is

a mathematical operation performed between the input and the filter to produce the feature map. Equation (1) shows the convolution operation,

$$O[u,v] = F[m,n] * I[u,v] = \sum_m \sum_n F[m,n] \cdot I[u+m, v+n] \quad (1)$$

where $F[m,n]$ is the convolution filter, $I[u,v]$ is the input image and $O[u,v]$ is the output feature map.

A filter is convolved over the input image to produce a feature map. Another CNN layer is the activation function, which is used to present non-linearity because usually, the image classification task is highly non-linear. To reduce overfitting and to reduce the spatial footprint of each filter, two main techniques can be used to extract the essential pixels and removing the noise. The first involves using a stride value larger than 1, which reduces the output of each filter. The second technique is called pooling, where a pooling layer usually follows the activation layer. Pooling layers can strengthen network spatial invariance (Scherer et al., 2010). The two main types of pooling layers are the maximum pooling layer and the average pooling layer.

Then, fully connected layers follow, usually defined at the end of the network, which takes the output of the feature extraction layers. The primary purpose of the dense layer is to consider all the features extracted from the previous layers and employ these features to classify the output. The dense layers are followed by an activation function, which usually consists of a rectified linear unit (ReLU) layer; finally, at the end of the network, a softmax or sigmoid function is used to output the target probability.

8.3.3. Optimizers

The model learns (trains) on a given dataset by comparing the actual label of the input (available in the training set) to the predicted label, thereby, minimizing the cost function. Hypothetically, if the cost function is zero, the model has learned the dataset correctly. However, an optimization algorithm is needed to achieve the minimum of a cost function. The next section discusses different optimization algorithms, introduced in the literature, to minimize the cost function.

8.3.3.1. Vanilla Gradient Descent Optimizers

Gradient descent is the primary class of optimizers capable of finding the minimum value of the cost function. The literature has introduced three versions of gradient descent.

Batch Gradient Descent

The first optimization algorithm was the batch gradient descent optimization algorithm (BGD), which updates the network weights after scanning the whole training dataset; in the case of images,

convergence takes much time, as there may be millions of weights to optimize and the whole dataset needs to be reevaluated at every step (i.e., epoch). For the convex loss function, it is guaranteed that the BGD will converge to the global minimum, while it converges to a local minimum for non-convex functions. The weights are updated based on Equation (2):

$$w_{t+1} = w_t - \eta \frac{\partial C}{\partial w_t} \tag{2}$$

$$\frac{\partial C}{\partial w_t} = \nabla_w C(w_t) \tag{3}$$

where Equation (3) is the gradients update equation, and η is the learning rate hyperparameter. w_t are the weights at step t, $C(.)$ is the cost function and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t.

Stochastic Gradient Descent

To overcome the shortcomings of BGD, stochastic gradient descent (SGD) was introduced. SGD allows to update the network weights per each training image, that is why SGD is sometimes called online training. However, such updates engender massive fluctuation in the loss function, due to the high variance between different images, which can create much noise in the training phase:

$$w_{t+1} = w_t - \eta \frac{\partial C}{\partial w_t} \tag{4}$$

$$\frac{\partial C}{\partial w_t} = \nabla_w C(w_t; x^{(i)}; y^{(i)}) \tag{5}$$

The weights are updated based on Equation (4), where Equation (5) is the gradient update equation, and η is the learning rate hyperparameter. w_t are the weights at step t, $C(.)$ is the cost function, and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y.

Mini-Batch Gradient Descent

Mini-batch gradient descent was introduced to overcome the shortcomings of the previous two algorithms, because it allows for the weights to be updated per batch, and not per image. As such, mini-batch gradient descent may be regarded as a particular case of SGD, where the number of samples is more than one. In the literature and it follows, in this paper, the mini-batch is referred to as stochastic gradient descent (SGD):

$$w_{t+1} = w_t - \eta \frac{\partial C}{\partial w_t} \tag{6}$$

$$\frac{\partial C}{\partial w_t} = \nabla_w C\left(w_t; x^{(i:i+n)}; y^{(i:i+n)}\right) \quad (7)$$

The weights are updated based on Equation (6), and Equation (7) is the gradient update equation. η is the learning rate hyperparameter, w_t are the weights at step t, n is the number of data points, $C(.)$ is the cost function and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y.

8.3.3.2. Momentum-Based Gradient Descent Optimizers

The main drawback of using mini-batch SGD is the presence of oscillations during the updating of the weights. These oscillations usually result in a long time to reach convergence. Momentum, also known as moving average gradients, was introduced, in order to overcome this issue and to fix the gradients' direction.

Momentum Gradient Descent

Understanding the right direction for the gradient avoids oscillations in the wrong directions, and knowing the right direction relies on using the previous position for guidance. Considering the previous position, the updating rule adds a fraction of the previous update, which gives the optimizer the momentum needed to continue moving in the right direction. The weights are updated based on Equation (10),

$$V_t = \lambda V_{t-1} + \eta \frac{\partial C}{\partial w_t} \quad (8)$$

$$\frac{\partial C}{\partial w_t} = \nabla_w C\left(w_t; x^{(i:i+n)}; y^{(i:i+n)}\right) \quad (9)$$

$$w_{t+1} = w_t - V_t \quad (10)$$

where V is the velocity, and it is initialized to 0. λ is used to select the amount of information needed from the previous update. η is the learning rate hyperparameter, w_t are the weights at step t, n is the number of data points, $C(.)$ is the cost function, and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y.

Nesterov Momentum Gradient Descent

If the momentum is sufficiently high, close to the minimum, the optimizer may overshoot the minimum. The previous optimization algorithms take the current and the previous gradients into account for updating the weights. However, to make the optimization algorithm more robust, we must take the future gradients into account as well, to approximate the gradients' direction. The weights are updated based on Equation (13),

$$V_t = \lambda V_{t-1} + \eta \frac{\partial C}{\partial w_t} \tag{11}$$

$$\frac{\partial C}{\partial w_t} = \nabla_w C\left(w_t - \lambda V_{t-1}; x^{(i:i+n)}; y^{(i:i+n)}\right) \tag{12}$$

$$w_{t+1} = w_t - V_t \tag{13}$$

where V is the velocity, and it is initialized to 0, λ is used to select the amount of information needed from the previous update. While, η is the learning rate hyperparameter, w_t are the weights at step t, n is the number of data points, $C(.)$ is the cost function, and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y. $(w_t - \lambda V_{t-1})$ is the look-ahead position that is capable of approximating the next gradient position, thereby allowing it to slow down if it threatens to overshoot the minimum.

8.3.3.3. Adaptive Gradient Descent Optimizers

All the optimization mentioned above has a fixed learning rate, while, in practice, deep learning algorithms are non-convex problems. That may be a problem, as we may face a sparse weight matrix, where we require different updates for different weights, especially for infrequent weights, where significant updates are needed to reach to avoid oscillating.

AdaGrad Optimizer

To scale the learning rate for each weight, the AdaGrad optimization algorithm (C. Duchi, Hazan, & Singer, 2011) was introduced to establish different updates for different weights. The learning rate is tuned automatically, by dividing the learning rate by the sum of squares of all previous gradients. The weights are updated based on Equation (14),

$$w_t^i = w_{t-1}^i - \frac{\eta}{\sqrt{\sum_{\mathcal{T}=1}^{t}\left(\nabla_w C\left(w_{\mathcal{T}}^i\right)\right)^2 + \epsilon}} \cdot \nabla_w C\left(w_t^i\right) \tag{14}$$

where η is the learning rate hyperparameter, w_t are the weights at step t, $C(.)$ is the cost function, and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y. The sum of squares $\sqrt{\sum_{\mathcal{T}=1}^{t}(\nabla_w C(w_{\mathcal{T}}^i))^2}$ is used to scale the learning rate; it gives a high learning rate for the least frequent gradients and a low learning rate for the more frequent gradients.

RMSProp Optimizer

The main drawback of AdaGrad is that the learning rate decreases monotonically because every added term is positive. After many epochs, the learning rate is so small that it stops updating the

weights. RMSProp was introduced to address the problem of the monotonically decreasing learning rate (Hinton, 2012). The weights are updated based on Equation (17),

$$G = \nabla_w C(w_t) \tag{15}$$

$$E[G^2]_t = \lambda\, E[G^2]_{t-1} + (1-\lambda) G_t^2 \tag{16}$$

$$w_t^i = w_{t-1}^i - \frac{\eta}{\sqrt{E[G^2]_t + \epsilon}} \cdot \nabla_w C\left(w_t^i\right) \tag{17}$$

where η is the learning rate hyperparameter, w_t are the weights at step t, $C(.)$ is the cost function, and $\nabla_w\, C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y. λ is used to select the amount of information needed from the previous update. $E[G^2]_t$ is the running average of the squared gradients, which has been used to avoid the monotonically decreasing gradients of the AdaGrad optimizer.

Adam Optimizer

The Adam optimization (Kingma & Ba, 2014) algorithm was introduced to combine the benefits of Nesterov momentum, AdaGrad, and RMSProp algorithms. The weights are updated based on Equation (18):

$$w_t^i = w_{t-1}^i - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t \tag{18}$$

where:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{19}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{20}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) G \tag{21}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)[G]^2 \tag{22}$$

$$G = \nabla_w C(w_t) \tag{23}$$

where η is the learning rate hyperparameter, w_t are the weights at step t, $C(.)$ is the cost function, and $\nabla_w\, C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y, β_i is used to select the amount of information needed from the previous update, where $\beta_i \in [0,1]$, m_t is the running average of the gradients, also known as the first moment, v_t is the running average of the squared gradients, and known as the second moment. If the first and second moments get initialized at zero, they are biased toward it, to solve this zero-biased problem, these moments are bias-corrected by dividing them by their respective β.

Adamax Optimizer

Adamax (Kingma & Ba, 2014) is the update of the Adam algorithm, where the uncentered variance tends to ∞. The weights are updated based on Equation (24):

$$w_t^i = w_{t-1}^i - \frac{\eta}{v_t + \epsilon} \cdot \hat{m}_t \tag{24}$$

where:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{25}$$

$$v_t = \max(\beta_2 \cdot v_{t-1}, |G_t|) \tag{26}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) G \tag{27}$$

$$G = \nabla_w C(w_t) \tag{28}$$

where η is the learning rate hyperparameter, w_t are the weights at step t, $C(.)$ is the cost function, and $\nabla_w C(w_t)$ is the gradient of weight parameters w_t for image x and its corresponding label y. β_i is used to select the amount of information needed from the previous update, where $\beta_i \in [0,1]$. m_t is the first moment, v_t is the second moment.

Nadam Optimizer

Nadam (Dozat, 2016) is an extension of the Adam algorithm by combining it with Nesterov momentum gradient descent. The weights are updated based on Equation (29):

$$w_t^i = w_{t-1}^i - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot \bar{m}_t \tag{29}$$

where:

$$\bar{m}_t = \beta_1^{t+1} \hat{m}_t + (1 - \beta_1^t) \hat{g}_t \tag{30}$$

$$\hat{m}_t = \frac{m_t}{1 - \prod_{i=1}^{t} \beta_1^i} \tag{31}$$

$$\hat{g}_t = \frac{g_t}{1 - \prod_{i=1}^{t} \beta_1^i} \tag{32}$$

where η is the learning rate hyperparameter, and w_t are the weights at step t. While, β_i is used to select the amount of information needed from the previous update, where $\beta_i \in [0,1]$, m_t is the first moment.

Fine-Tuning

According to (Pan and Yang 2009; Xie et al. 2016) transfer learning can be formalized as follows: by having a domain D, where $D = \{X, P(X)\}$, in which X is the feature space and $P(X)$ is the marginal probability distribution and task $\mathcal{T} = \{Y, f(.)\}$, where Y is the label space and $f(.)$ is the predictive function which models $P(y|x)$ for $y \in Y$ and $x \in X$. By having a source domain D_s and learning task $\mathcal{T}_S$ and a target domain D_T and learning task $\mathcal{T}_T$, by using weights from D_S and $\mathcal{T}_S$, learning the target predictive function $f(.)$ in $\mathcal{T}_T$ can be improved a lot, where $D_S \neq D_T$, $\mathcal{T}_S \neq \mathcal{T}_T$, or both. Fine-tuning is very important in the classification of medical images, because the neural network usually needs many images to be trained. However, in the medical field, for many reasons, labeled medical images are scarce. Instead of initializing the weights from scratch, ImageNet weights can be used. In this paper, the networks were trained on ImageNet dataset and all the networks blocks were fine-tuned using the PatchCamelyon dataset.

8.3.3.4. VGG16 Network

VGG16 (Simonyan & Zisserman, 2014) was introduced in 2014 by the researchers at Oxford's Visual Geometry Group. It was one of the top algorithms involved in the ImageNet classification challenge, and it had an 8.1% error rate. VGG16 consists of five convolution blocks, where the first block contains two convolution layers, stacked together with 64 filters. The second block consists of two stacked convolution layers with 128 filters, where the second convolution block is separated from the first block by a max pool layer. The third block consists of three convolution layers, stacked together with 256 filters and separated from the second block by another max pool layer. The fourth and fifth layers have the same architecture, but instead, have 512 filters. The convolution filter used throughout this network is of size 3 × 3 and stride of 1. Then, a flatten layer is added between the convolution blocks and the dense layers, converting the 3D vector into a 1D vector. The last block consists of three dense layers, each of which has 4096 neurons, to classify each image. The last layer is a softmax layer, which is used to ensure that the probability summation of the output is one. ReLU was used as an activation layer throughout the network. This network was trained on the ImageNet dataset for three weeks on four GPUs to detect the ImageNet classification task. A summary of the VGG16 network is presented in Table 37.

8.3.3.5. InceptionV3 Network

The authors Szegedy et al. (2015) introduced a novel architecture, called Inception, to participate in the ImageNet competition in 2015; Inception had an accuracy rate of 92.2%. The architecture consists of 48 layers and total parameters of 22,000,000. This architecture has a concatenated layer of convolutions, stacked in parallel to decrease the size of the architecture while maintaining its complexity. InceptionV3 network architecture is shown in Figure 33. A summary of the InceptionV3 network is presented in Table 37.

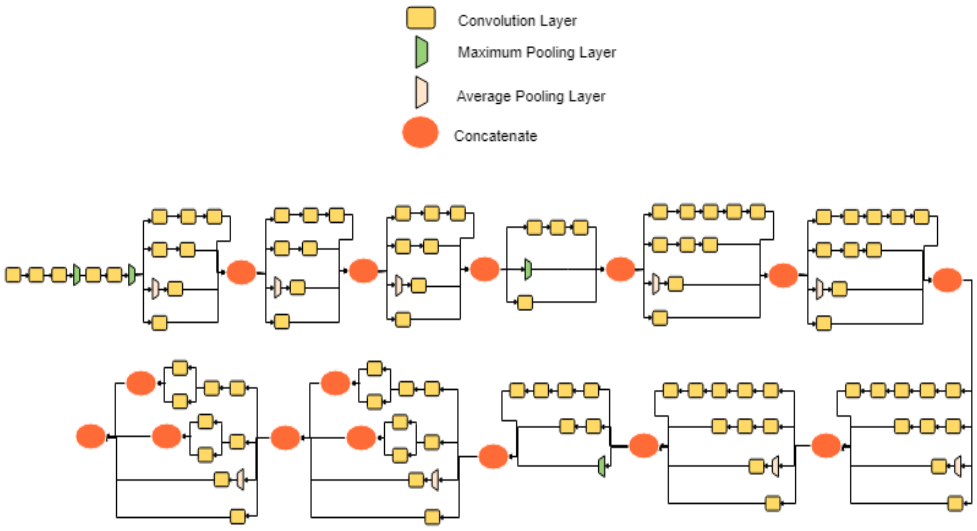


Figure 33. InceptionV3 network architecture.

8.3.3.6. ResNet Network

The authors, He et al. (2016), investigated the effect of increasing the depth of the convolutional neural network and its impact on network performance. The authors noticed that increasing the depth of the network decreases the generalizability of the network, which means that the test error of the network is higher than a shallow network. This may be due to the vanishing gradients, where the weights are not updated in deep layers. Therefore, He et al. (2016) introduced a novel architecture called ResNet, where Res signifies the application of a residual connection between the convolutional layers, which is then passed to the ReLU activation layer. One of the main benefits of adding the residual connection is that the weights learned from the previous layers can be carried to the next layers during the backpropagation step. ResNet won the ImageNet competition in 2015 with Top-5 accuracy of 94.29%. It has a total of 23,587,712 parameters, and its ImageNet weights are available in the Keras package. A summary of ResNet network is presented in Table 37.

8.3.3.7. DenseNet Network

DenseNet network (Huang et al., 2017) was inspired by the residual connection of the ResNet architecture. All the layers are connected to all their subsequent layers, meaning that a residual connection is established between all the layers. Merging will be used instead of adding to combine the layers. DenseNet has many variants depends on the number of layers; some of the variants are DenseNet 121, DenseNet169, and DenseNet201. In this paper, we opted for the DenseNet121 network. A summary of the DenseNet network is presented in Table 37.

Table 37. Number of layers and parameters of the CNNs used in this study.

Networks	Number of Layers	Number of Parameters

VGG16	16	14,714,688
InceptionV3	48	21,802,784
ResNet50	50	23,587,712
DenseNet121	121	7,037,504

8.3.4. Overcoming Overfitting

Overfitting generally consists of memorization of the training dataset and usually leads to poor performance on the test dataset. This means that the performance on the training set can be excellent, but the performance on the test set is quite poor. The loss of the generalizability of the network may be due to many issues, such as the capacity of the network or the nature of the training dataset itself. Many measures have been introduced in the literature to overcome overfitting. Below are some techniques that were used in this research to overcome overfitting.

8.3.4.1. Dropout

A regularization layer introduced by (Srivastava et al., 2014) can be applied to any layer in the network. During network training, some neurons are disabled with a pre-defined dropout-rate probability *P*. This can be understood as a sort of bagging for neural networks.

8.3.4.2. Image Augmentation

Increasing the size of the training set improves the performance of the network. For image datasets, many duplicates can be created with simple changes to the original dataset, including rotation, flipping, zooming, and cropping. These transformations make the network more robust in defending against overfitting, and it enhances network performance as well. In our case, the original images are flipped, rotated, zoomed, and shifted. The rotation range used was 180°; and the images were randomly flipped horizontally and vertically; the shifting range used was 25%; and the zoom range used was 40%.

8.3.4.3. Early Stopping

Early stopping is a precautionary measure used to prevent the network from overfitting, which may be defined as stopping the training phase of the network when the performance on the validation set stops improving for a pre-defined number of epochs. This pre-defined number usually ranges from 10–50 epochs. In our case, the number of epochs is 10.

8.3.5. Evaluation Metrics

To assess the quality of the trained CNN, many measures have been developed. For classification tasks, a confusion matrix is constructed to assess the model quality; it categorizes the model predictions, according to whether they match the correct label of the image. It has four central values:

TP: True positive (A positive example, classified as a positive example)
TN: True negative (A negative example, classified as a negative example)
FP: False positive (A negative example, but classified as a positive example)
FN: False-negative (A positive example, but classified as a negative example)

To visualize the model performance, the ROC curve was introduced to examine the trade-off between sensitivity and specificity visually. The main idea of the ROC curve is to plot the specificity of the algorithms, which is the percentage of the correctly classified negatives against the sensitivity, which is the percentage of the correctly classified positives of the algorithm (Fawcett, 2003). The ROC curve has a diagonal line, which represents a random guess model. It means that the model cannot differentiate between true positives and false positives; this diagonal line can be considered as the baseline where models can be judged. The best model has a curve that passes through the top left corner for "100% Sensitivity" and has a 0% false-positive rate. To measure the quality of the model using the ROC curve, a statistic known as AUC or "Area under the ROC curve," is used; this treats the ROC diagram as a two-dimensional square and measures the area under the curve. AUC has a minimum value of 0.5 and a maximum value of 1, where 0.5 represents a model with no predictive power, and 1 represents a model with 100% predictive power. According to Vuk (2006), the AUC is calculated by Equation (33):

$$\mathrm{AUC} = \int_0^1 \frac{\mathrm{TP}}{\mathrm{P}}\, \mathrm{d}\frac{\mathrm{FP}}{\mathrm{N}} = \frac{1}{\mathrm{PN}} \int_0^{\mathrm{N}} \mathrm{TP}\ \mathrm{dFP} \tag{33}$$

where $TP + FN = P$ and $TN + FP = N$.

8.4. Results

The following section details the results obtained from training the four network architectures using the six selected optimizers with three learning rates, namely, 1×10^{-3},1×10^{-4}, and 1×10^{-5}. Many experiments have been conducted on this dataset, to determine the behavior of each optimizer with each network architecture to determine the best combination. The performance of each optimizer with the VGG16 architecture is presented in Table 38, with the InceptionV3 architecture in Table 39, with the ResNet architecture in Table 40, and with the DenseNet architecture in Table 41. To test the performance of each configuration, two types of evaluation were used; the first consists of splitting the training dataset into 80%/20% to train and validate the dataset. After training, the model is used to predict the class of the images in the test set, and the result is submitted to the Kaggle platform to assess the performance of each model. The AUC of the ROC curve measured the performance. The optimizers are ranked based on their test AUC that was acquired from the Kaggle platform.

In all the experiments, the default settings of each optimizer were chosen, except the learning rate, and image augmentation used for rotating, flipping, and cropping all the images. The size of the images was kept constant at 96 × 96; a batch size of 64 images was used, and early stopping was applied with the number of epochs equal to 10.

8.4.1. VGG16 Architecture Result

Table 38 shows the results for the VGG16 architecture, which shows that the highest AUC was achieved by Adam optimizer, which also took the shortest time to achieve convergence. At the same time, the lowest test AUC was achieved by RMSProp and Adamax optimizers that did not converge at all. For the highest learning rate (1×10^{-3}) the highest AUC achieved was by the Adam optimizer, and the lowest AUC achieved was by the RMSProp and Adamax optimizers. For the medium learning rate (1×10^{-4}) the highest AUC achieved was by NAG optimizer, and the lowest AUC achieved was by both AdaGrad and Adam optimizers. For the lowest learning rate (1×10^{-5} The AdaGrad optimizer achieved), the highest AUC achieved by the Adam optimizer and the lowest AUC. Overall, the medium learning rate achieved the best results, followed by the lowest learning rate. Adam optimizer was the most stable optimizer with high results and low variance between different learning rates.

Table 38. Results obtained with the VGG16 architecture. Where LR stands for learning rate; NAG represents Nesterov momentum; AdaGrad represents the adaptive gradient optimizer; RMSProp represents the root mean square propagation optimizer; Adam represents adaptive moment estimation optimizer; AdaMax represents maximum adaptive moment estimation optimizer; and Nadam represents Nesterov and Adam optimizer.

Optimizers	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
NAG	89.45%	94.64%	94.25%
AdaGrad	88.50%	87.40%	88.07%
RMSProp	50.00%	94.33%	93.45%
Adam	90.88%	90.39%	95.01%
Adamax	50.00%	94.02%	94.20%
Nadam	85.00%	91.14%	94.33%

8.4.2. InceptionV3 Architecture Result

Table 39 shows the results for the InceptionV3 architecture, which shows that the highest AUC was achieved by the RMSProp optimizer, which also took the shortest time to achieve convergence. At the same time, the lowest test AUC was achieved by AdaGrad optimizer, which also took the longest time to convergence. For the highest learning rate (1×10^{-3}), the AdaGrad optimizer achieved the highest AUC, while, the Adam optimizer achieved the lowest AUC. For the medium learning rate (1×10^{-4}) the highest AUC achieved was by the RMSProp optimizer, and the lowest AUC achieved was by the AdaGrad optimizer. For the lowest learning rate (1×10^{-5} The AdaGrad optimizer achieved), the highest AUC achieved by the AdaMax optimizer and the lowest AUC. Overall, the medium learning rate achieved the best results, followed by the lowest learning rate. Adamax optimizer was the most stable optimizer with high results and low variance between different learning rates.

Table 39. Results obtained with the InceptionV3 architecture. Where LR stands for learning rate; NAG represents Nesterov momentum; AdaGrad represents the adaptive gradient optimizer; RMSProp represents the root mean square propagation optimizer; Adam represents adaptive moment estimation optimizer; AdaMax represents maximum adaptive moment estimation optimizer; and Nadam represents Nesterov and Adam optimizer.

Optimizer	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
NAG	93.18%	93.25%	90.81%
AdaGrad	93.64%	90.46%	86.32%
RMSProp	91.41%	94.91%	92.65%
Adam	90.44%	92.53%	93.22%
Adamax	93.44%	93.11%	93.95%
Nadam	91.97%	91.33%	92.46%

8.4.3. ResNet Architecture Result

Table 40 shows the results for the ResNet architecture, which shows that the best AUC was achieved by the Nadam optimizer, while the AdaGrad optimizer achieved the lowest AUC. For the highest learning rate (1×10^{-3}) the highest AUC achieved was by the AdaGrad optimizer, and the lowest AUC achieved was by the RMSProp optimizer. For the medium learning rate (1×10^{-4}) the highest AUC achieved was by the NAG optimizer, and the lowest AUC achieved was by the AdaGrad optimizer. For the lowest learning rate (1×10^{-5} The AdaGrad optimizer achieved), the highest AUC achieved by the Nadam optimizer and the lowest AUC. Overall, the medium learning rate achieved the best results, followed by the lowest learning rate. Adamax optimizer was the most stable optimizer with high results and low variance between different learning rates.

Table 40. Results obtained with the ResNet architecture. Where LR stands for learning rate; NAG represents Nesterov momentum; AdaGrad represents the adaptive gradient optimizer; RMSProp represents the root mean square propagation optimizer; Adam represents adaptive moment estimation optimizer; AdaMax represents maximum adaptive moment estimation optimizer; and Nadam represents Nesterov and Adam optimizer.

Optimizer	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
NAG	90.07%	93.84%	89.00%
AdaGrad	93.04%	89.11%	83.46%
RMSProp	89.56%	89.62%	93.04%
Adam	90.24%	90.24%	93.84%
Adamax	90.24%	92.24%	93.70%
Nadam	91.91%	89.36%	93.85%

8.4.4. DenseNet Architecture Result

Table 41 shows the results for the DenseNet architecture, where the best AUC was achieved by the Adamax optimizer, while the Adam optimizer achieved the lowest AUC. For the highest learning rate (1×10^{-3}) the highest AUC achieved was by the AdaGrad optimizer, and the lowest AUC achieved was by the Adam optimizer. For the medium learning rate (1×10^{-4}) the highest AUC achieved was by the Adamax optimizer, and the lowest AUC achieved was by the Adam optimizer.

For the lowest learning rate (1×10^{-5} The AdaGrad optimizer achieved), the highest AUC was achieved by the RMSProp optimizer and the lowest AUC. Overall, the medium learning rate achieved the best results, followed by the lowest learning rate. Adamax optimizer was the most stable optimizer with high results and low variance between different learning rates.

Table 41. Results obtained with the DenseNet architecture. Where LR represents learning rate; NAG represents Nesterov momentum; AdaGrad represents the adaptive gradient optimizer; RMSProp represents the root mean square propagation optimizer; Adam represents adaptive moment estimation optimizer; AdaMax represents maximum adaptive moment estimation optimizer; and Nadam represents Nesterov and Adam optimizer.

Optimizer	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
NAG	93.08%	94.31%	91.64%
AdaGrad	93.89%	93.57%	87.70%
RMSProp	88.19%	93.98%	94.61%
Adam	84.18%	89.69%	94.43%
Adamax	90.62%	95.12%	93.91%
Nadam	86.77%	94.21%	93.77%

Overall, in terms of performance across all four networks, the highest results were achieved by the adaptive learning optimizers, like Adam, Adamax, Nadam, and RMSProp. However, these optimizers needed a lower learning rate to be able to converge, while the high learning rate did not achieve good results. One exception was the AdaGrad optimizer that did not achieve high results with a low learning rate; on the contrary, it needed a high learning rate to be able to converge to an acceptable result. From our results and the results obtained by Prilianti et al. (2019), it is apparent that every combination of network and optimizer will produce a unique combination. However, the general behavior of each optimizer can be noted, which can be concluded from the results. Overall, the NAG optimizer did achieve high results overall the four architectures and overall, the three learning rates used with the medium learning rate (1×10^{-4}) achieved the best results. The AdaGrad optimizer did not achieve high results compared to other optimizers used, especially when trained using low learning rates. RMSProp optimizer did achieve high results with a low learning rate but was unstable with high learning rates. Adam optimizer needed a low learning rate to be able to converge to high results. While, the AdaMax optimizer behaved similarly to the Adam optimizer, except for a high learning rate with the VGG16, where it did not converge at all, one reason may be the shallow depth of the VGG16 network. Nadam optimizer did achieve high results with both the medium learning rate and the low learning rate.

8.5. Discussion

Taking into account the results achieved from the experimental campaign, it is possible to draw some interesting observations of the behavior of the CNNs and optimizers considered in this paper. By focusing on the choice of the optimizer and its relation with the learning rate, the experimental

results confirm that the choice of the learning rate may result in an unstable behavior of the training process. This is particularly evident, for some of the considered networks and optimizers, when considering the smallest learning rate used in the experiments. As one can see, when LR $= 10^{-3}$, the training process of VGG16 with both RMSProp and Adamax optimizer result in a poor performance of the model. As explained in the previous sections, this can be motivated by the fact that the weights of the network change abruptly from one epoch to the next. Moving to lower LR values allows the convergence of the training process in all the configurations that were investigated. Overall, the results match the theoretical expectation: A lower LR value allows for a smoother convergence, but it requires more time with respect to a greater LR value.

Another interesting observation relates to the importance of the hyperparameters. While this is a topic of fundamental importance in the area of deep learning, it is particularly evident from the results of the experimental phase. In particular, all the considered architecture produced a comparable performance when the best configuration of the learning rate and optimizer (that is different for each type of architecture) was considered. In other words, it seems that the choice of the hyperparameters not only plays an essential role in determining the performance of the model, but the CNNs under exam are indistinguishable in terms of performance. We believe that this is an interesting observation that should further stress the importance of the tuning of the hyperparameters.

Focusing on the optimizers, AdaGrad produces the best performance with LR $= 10^{-3}$ and, under this aspect, it behaves differently with respect to the other optimizers under analysis. Conversely, Adam, Adamax, and Nadam obtained the best performance on the considered CNNs when LR $= 10^{-5}$ (except Adamax on the DenseNet architecture, where the best performance is obtained with LR $= 10^{-4}$).

Globally, the best result on the considered dataset was achieved by Adamax optimizer and DenseNet network. Anyway, the differences in terms of performance among the best configurations of each network, are not statistically significant. Overall, every optimizer behaved differently according to the particular architecture. For instance, for the deep architectures like ResNet, AdaGrad outperformed Adam and Adamax. For the shallow architectures like VGG16, AdaMax and NAG had the same performance. Given a specific network, each optimizer requires a different amount of time for converging (i.e., concluding ten epochs). In particular, RMSProp was the fastest optimizer, whereas training a CNN with AdaGrad resulted in the slowest training process. This result is coherent with respect to the one discussed proposed by Dogo et al. (Dogo et al., 2018), in which the authors investigated the effect of different optimizers in terms of required time to reach. More in detail, training the VGG16 architecture requires a minimum of 90 min (RMSProp optimizer and learning rate of 10−3) and a maximum of nine hours (AdaGrad optimizer and learning rate of 10−5). InceptionV3 requires approximately one hour more than VGG16; in this case, the use of RMSProp (with a learning rate of 10−3) resulted in the fastest training process (approximately 150 min), while

the use of AdaGrad (with a learning rate of 10−5) required approximately 10 h to finish. An identical pattern was observed for ResNet and DenseNet, that are requiring approximately two hours more than VGG16 for concluding the training process.

Finally, it is important to compare the results achieved with transfer learning against the ones obtained with CNNs that were specifically built for classifying the images of the PatchCamelyon dataset. The winner of the Kaggle competition obtained an AUC of 1, while the second-best performing network obtained an AUC of 0.98. On the other hand, the best performing network obtained with transfer learning (DenseNet architecture, with Adamax optimizer, and a learning rate of 10−4) was able to obtain an AUC of 0.95. This result confirms the suitability of transfer learning for the task at hand. More in detail, we believe that, by considering deeper architectures and more epochs, it could be possible to improve the results of this study, thus equaling the performance achieved by the winner of the Kaggle competition. On the other hand, we highlight a fundamental difference between the best performance reported in the present study and the best performance of the Kaggle competition: The former was obtained using an existing network (used for addressing different computer vision tasks) and by fine-tuning it, while the latter was achieved by designing an ad hoc CNN, a time-consuming task that requires some expertise.

8.6. Conclusions

CNN represents an analysis of images created using current computation techniques. This is mostly due to their ability to obtain a performance that is similar to, or better than, the one achieved by human beings. Nevertheless, similarly to other deep learning models, training a CNN is a task that usually requires a vast amount of images. This is an essential limitation in all the domains, like the medical one, in which data are scarce and difficult to obtain. In such a situation, transfer learning may provide a viable option. The idea of transfer learning is to use a model trained over thousands of observations (i.e., images in this study) to provide an initial architecture and set of weights for addressing a similar problem over a different domain. Motivated by the success of transfer learning in the analysis of medical images, and for further studying this promising research area, this paper compared the performance optimizers used in popular CNNs for the classification of histopathology images. In particular, four network architectures were used in the evaluation process. These networks were trained on the ImageNet dataset, which consists of millions of images, and their weights were fine-tuned to suit the considered histopathology images dataset. The results obtained from the experimental phase, in which different combinations of network, optimizer, and learning rate were considered, corroborated the initial hypothesis on the importance of the optimizer and the learning rate. While the choice of CNN is essential, it is clear that by fixing the value of the learning rate, the results obtained using different optimizers could be significantly different. On the other hand, once a particular optimizer is selected, the choice of the learning rate plays an essential role in determining the final performance of CNNs.

Interestingly, for each of the different CNNs under exam, it is possible to notice that the best performing configuration of optimizer and learning rate produces an AUC that is approximately 94%. This result strengthens the importance of selecting the hyperparameters of the network, and, in a future investigation, we will extend this work to include additional hyperparameters and datasets aiming at providing formal guidelines for medical experts that want to use CNN models to support their daily work.

Chapter 9. Conclusion

The computer-assisted analysis of medical images is a recently emerging application of artificial intelligence that can save time, money, and the workforce. The main challenge of using CNNs in medical image classification is the size of the training dataset, which is typically limited since an experienced doctor is required to annotate each image and, sometimes, even resort to a second opinion to classify some difficult images. Two main techniques have been developed to help in the classification of medical images: (1) transfer learning and (2) designing a new CNN and training it from scratch. The first technique, transfer learning, can be a viable option considering its suitability when a limited number of training observations are available to address the image classification task. Thus, transfer learning can play an essential role in the medical field. The second technique is to design a new CNN, where complex and deep architectures are developed to solve tasks related to computer vision. These architectures can be successfully applied to solve the challenges of the field of medical images. In this chapter, we present our main findings and future works.

In the third chapter, we reviewed the most recent literature about the classification of DR using transfer learning of CNN. Many open questions were presented as well to help future researchers. In the fourth chapter, we investigated the usage of transfer learning, comparing it to training from scratch. We concluded that transfer learning is highly beneficial in training CNN, especially in the case of medical images. In the fifth chapter, we introduced a novel CNN trained from scratch to help classify histopathology images. In this work, we have also studied the effect of different activation functions on the CNN performance and the effect of the location of the activation function on the performance of the network. Based on the obtained results, we recommend authors try different activation functions and comprehensively analyze their impact other than choosing the ReLU activation function as a default. Moreover, we recommend placing the normalization layer before the activation function.

In the sixth chapter, we studied the effect of the batch size on the classification power of CNN. According to our results, we can conclude that the learning rate and batch size significantly impact the network's performance. There is a high correlation between the learning rate and the batch size. When the learning rates are high, a large batch size performs better than with small learning rates. In the seventh chapter, we investigated the effect of depth on the accuracy of CNN. We studied the effect of three CNN architectures with three different learning rates on two different histopathology datasets. Our results suggest that fine-tuning the top layers for shallow networks like VGG can be sufficient to obtain decent results, while for deep networks like InceptionV3, fine-tuning the entire network can yield better results. In all cases, a low learning rate is highly recommended when fine-tuning the network not to mess up the original weights. In the eighth chapter, we compared the performance of six different optimizers using three learning rates. We concluded that the optimizer's choice could significantly impact both the performance and the convergence of the CNN. For our future works, we

intend to investigate the use of neuroevolution algorithms that can provide a different way of exploring the search space of deep learning architectures, especially for CNN.

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrada, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv 2015, arXiv:1603.04467.

Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014). Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *22*(10), 1533–1545. https://doi.org/10.1109/TASLP.2014.2339736

Abdel-Hamid, O., Mohamed, A., Jiang, H., & Penn, G. (2012). Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 4277–4280). https://doi.org/10.1109/ICASSP.2012.6288864

Abramoff, M.; Folk, J.; Han, D.; Walker, J.; Williams, D.; Russell, S.; Massin, P.; Cochener, B.; Gain, P.; Tang, L.; et al. Automated analysis of retinal images for detection of referable diabetic retinopathy. JAMA Ophthalmol. 2013, 131, 351–357. doi:10.1001/jamaophthalmol.2013.1743

Abramoff, M.; Niemeijer, M.; Russell, S. Automated Detection of Diabetic Retinopathy: Barriers to Translation into Clinical Practice. Expert Rev. Med. Devices 2010, 7, 287–296. https://doi.org/10.1586/erd.09.76

Ahmad, M., Ghaffar, S., & Khurshid, K. (2019). Classification of Breast Cancer Histology Images Using Transfer Learning. In *IEEE International Bhurban Conference on Applied Sciences and Technology (IBCAST)* (Vol. 16th). Pakistan. https://doi.org/10.1109/IBCAST.2019.8667221

Almubarak, H., Bazi, Y., & Alajlan, N. (2020). Two-Stage Mask-RCNN Approach for Detecting and Segmenting the Optic Nerve Head, Optic Disc, and Optic Cup in Fundus Images. *Applied Sciences*, *10*. https://doi.org/10.3390/app10113833

Amin, J., Sharif, M., & Yasmin, M. (2016). A Review on Recent Developments for Detection of Diabetic Retinopathy. *Scientifica*, *2016*, 1–20. https://doi.org/10.1155/2016/6838976

Aresta, G.; Araújo, T.; Kwok, S.; Chennamsetty, S.S.; Safwan, M.; Alex, V.; Marami, B.; Prastawa, M.; Chan, M.; Donovan, M.; et al. BACH: Grand challenge on breast cancer histology images. Med. Image Anal. 2019, 56, 122–139. https://doi.org/10.1016/j.media.2019.05.010

Arjmand, A.; Angelis, C.T.; Tzallas, A.T.; Tsipouras, M.G.; Glavas, E.; Forlano, R.; Manousou, P.; Giannakeas, N. Deep Learning in Liver Biopsies using Convolutional Neural Networks. In Proceedings of the 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), Budapest, Hungary, 1–3 July 2019; pp. 496–499. https://doi.org/10.1109/TSP.2019.8768837

Basha, S.H.S.; Ghosh, S.; Babu, K.; Dubey, S.; Pulabaigari, V.; Mukherjee, S. RCCNet: An Efficient Convolutional Neural Network for Histological Routine Colon Cancer Nuclei Classification. In Proceedings of the 2018 15th International Conference on Control,

Automation, Robotics and Vision (ICARCV), Singapore, 18–21 November 2018. doi: 10.1109/ICARCV.2018.8581147.

Bayramoglu, N.; Kannala, J.; Heikkila, J. Deep learning for magnification independent breast cancer histopathology image classification. In Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 4–8 December 2016. https://doi.org/10.1109/ICPR.2016.7900002

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In Neural Networks: Tricks of the Trade; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7700, pp. 437–478. https://doi.org/10.1007/978-3-642-35289-8_26.

BioImaging Dataset. http://www.bioimaging2015.ineb.up.pt/dataset.html. (2015). Retrieved from http://www.bioimaging2015.ineb.up.pt/dataset.html

Bourne, R.; Stevens, G.A.; White, R.; Smith, J.L.; Flaxman, S.R.; Price, H.; Jonas, J.B.; Keeffe, J.; Leasher, J.; Naidoo, K.; et al. Causes of vision loss worldwide, 1990–2010: A systematic analysis. Lancet Glob. Health 2013, 1, e339–e349.

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, *24*(2), 123–140. https://doi.org/10.1023/A:1018054314350

Cancer. (2018). Retrieved from https://www.who.int/news-room/fact-sheets/detail/cancer

CDC. (2017). *National Hospital Ambulatory Medical Care Survey: 2017 Emergency Department Summary Tables*. Retrieved from https://www.cdc.gov/nchs/data/nhamcs/web_tables/2017_ed_web_tables-508.pdf

Chada, G. (2019). Machine Learning Models for Abnormality Detection in Musculoskeletal Radiographs. *Reports*, *2*, 26. https://doi.org/10.3390/reports2040026

Chen, H., Zeng, X., Luo, Y., & Ye, W. (2019). Detection of Diabetic Retinopathy using Deep Neural Network. In *International Conference on Digital Signal Processing, DSP* (Vol. 2018-Novem). https://doi.org/10.1109/ICDSP.2018.8631882

Chen, L., Magliano, D. J., & Zimmet, P. Z. (2012). The worldwide epidemiology of type 2 diabetes mellitus—present and future perspectives. *Nature Reviews Endocrinology*, *8*(4), 228–236. https://doi.org/10.1038/nrendo.2011.183

Cho, N. H., Shaw, J., Karuranga, S., Huang, Y., da Rocha Fernandes, J. D., Ohlrogge, A., & Malanda, B. (2018). IDF Diabetes Atlas: Global estimates of diabetes prevalence for 2017 and projections for 2045. *Diabetes Research and Clinical Practice*, *138*. https://doi.org/10.1016/j.diabres.2018.02.023

Choi, J. Y., Yoo, T. K., Seo, J. G., Kwak, J., Um, T. T., & Rim, T. H. (2017). Multi-categorical deep learning neural network to classify retinal images: A pilot study employing small database. *PLOS ONE*, *12*(11), e0187336. Retrieved from https://doi.org/10.1371/journal.pone.0187336

Chollet, Francois. (2017a). *Deep Learning with Python* (1st ed.). Greenwich, CT, USA: Manning Publications Co.

Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017. https://doi.org/10.1109/CVPR.2017.195

Chollet, François, & others. (2015). Keras. GitHub. https://github.com/fchollet.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2016). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. ArXiv Prepr. 2016. ArXiv:1511.07289

Cohen, J. A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement 1960, 20, 37–46. https://doi.org/10.1177/001316446002000104

Collobert, R.; Weston, J. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In Proceedings of the 25th International Conference on Machine Learning; ACM: New York, NY, USA, 2008. https://doi.org/10.1145/1390156.1390177

Cuadros, J., & Bresnick, G. (2009). EyePACS: An Adaptable Telemedicine System for Diabetic Retinopathy Screening. *Journal of Diabetes Science and Technology*, *3*, 509–516. https://doi.org/10.1177/193229680900300315

Decencière, E.; Cazuguel, G.; Zhang, X.; Thibault, G.; Klein, J.-C.; Meyer, F.; Marcotegui, B.; Quellec, G.; Lamard, M.; Danno, R.; et al. TeleOphta: Machine learning and image processing methods for teleophthalmology. IRBM 2013, 34, 196–203. https://doi.org/10.1016/j.irbm.2013.01.010

Decencière, E.; Zhang, X.; Cazuguel, G.; Lay, B.; Cochener, B.; Trone, C.; Gain, P.; Ordonez, R.; Massin, P.; Erginay, A.; et al. Feedback on a publicly distributed image database: The Messidor database. Image Anal. Stereol. 2014, 33, 231–234. https://doi.org/10.5566/ias.1155

Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). https://doi.org/10.1109/CVPR.2009.5206848

Deniz, E., Sengur, A., Kadiroğlu, Z., Guo, Y., Bajaj, V., & Budak, Ü. (2018). Transfer learning based histopathologic image classification for breast cancer detection. *Health Information Science and Systems*, *6*. https://doi.org/10.1007/s13755-018-0057-x

Dogo, E.M.; Afolabi, O.J.; Nwulu, N.I.; Twala, B.; Aigbavboa, C.O. Aigbavboa, A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks. In Proceedings of the 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), Belgaum, India, 21–22 December 2018. doi: 10.1109/CTEMS.2018.8769211.

Dos Santos, C.; Gatti de Bayser, M. Deep Convolutional Neural Networks for Sentiment Analysis

of Short Texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics; Dublin City University and Association for Computational Linguistics: Dublin, Ireland, 2014.

Dozat, T. (2016). Incorporating Nesterov Momentum into Adam. In Proceedings of the 4th International Conference on Learning Representations, Workshop Track, San Juan, Puerto Rico, 2–4 May 2016.

Duchi, J.C.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research. 2011, 12, 2121–2159. http://jmlr.org/papers/v12/duchi11a.html

Dumoulin, V., & Visin, F. (2016). *A guide to convolution arithmetic for deep learning*. *ArXiv 2016, arXiv:abs/1603.07285.*

Ehteshami Bejnordi, B.; Veta, M.; Johannes van Diest, P.; van Ginneken, B.; Karssemeijer, N.; Litjens, G.; van der Laak, J.A.W.M.; the CAMELYON16 Consortium, Hermsen, M. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. JAMA 2017, 318, 2199–2210. doi:10.1001/jama.2017.14585

Farooq, A., Anwar, S., Awais, M., & Rehman, S. (2017). A deep CNN based multi-class classification of Alzheimer's disease using MRI. In *2017 IEEE International Conference on Imaging Systems and Techniques (IST)* (pp. 1–6). https://doi.org/10.1109/IST.2017.8261460

Fawcett, T. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Available online: https://www.hpl.hp.com/techreports/2003/HPL-2003-4.pdf (accessed on 27 August 2020)

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Fuentes, A., Yoon, S., Kim, C. S., & Park, S. D. (2017). A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. *Sensors* . https://doi.org/10.3390/s17092022

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/BF00344251

Gao, Z., Li, J., Guo, J., Chen, Y., Yi, Z., & Zhong, J. (2019). Diagnosis of Diabetic Retinopathy Using Deep Neural Networks. *IEEE Access*, *7*, 3360–3370. https://doi.org/10.1109/ACCESS.2018.2888639

Gehring, J., Auli, M., Grangier, D., & Dauphin, Y. (2017). *A Convolutional Encoder Model for Neural Machine Translation*. https://doi.org/10.18653/v1/P17-1012

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; Dauphin, Y. Convolutional Sequence to Sequence

Learning. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017. http://proceedings.mlr.press/v70/gehring17a.html

Goldbaum, M., Katz, N., Nelson, M., & Haff, L. (1990). The discrimination of similarly colored objects in computer images of the ocular fundus. *Investigative Ophthalmology & Visual Science*, *31*, 617–623. https://iovs.arvojournals.org/article.aspx?articleid=2199653

Goncalves, L., AnaSubtil, Oliveira, M., & Bermudez, P. (2014). ROC curve estimation: An overview. *Revstat - Statistical Journal*, *12*, 1–20. https://www.ine.pt/revstat/pdf/rs140101.pdf

Goodfellow, I.; Bengio, Y.; Courville, A. Deep Learning; The MIT Press: Cambridge, MA, USA, 2016.

Gulshan, V.; Peng, L.; Coram, M.; Stumpe, M.; Wu, D.; Narayanaswamy, A.; Venugopalan, S.; Widner, K.; Madams, T.; Cuadros, J.; et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA 2016, 316, 2402–2410. https://doi.org/10.1001/jama.2016.17216

Gupta, A., & Chhikara, R. (2018). Diabetic Retinopathy: Present and Past. *Procedia Computer Science*, *132*, 1432–1440. https://doi.org/https://doi.org/10.1016/j.procs.2018.05.074

Gurcan, M. N., Boucheron, L. E., Can, A., Madabhushi, A., Rajpoot, N. M., & Yener, B. (2009). Histopathological Image Analysis: A Review. *IEEE Reviews in Biomedical Engineering*, *2*, 147–171. https://doi.org/10.1109/RBME.2009.2034865

Hallas, P., & Ellingsen, T. (2006). Errors in fracture diagnoses in the emergency deparment - Characteristics of patients and diurnal variation. *BMC Emergency Medicine*, *6*, 4. https://doi.org/10.1186/1471-227X-6-4

Harangi, B. (2018). Skin lesion classification with ensembles of deep convolutional neural networks. *Journal of Biomedical Informatics*, *86*, 25–32. https://doi.org/https://doi.org/10.1016/j.jbi.2018.08.006

Hazim Johari, M.; Abu Hassan, H.; Ihsan Mohd Yassin, A.; Tahir, N.; Zabidi, A.; Ismael Rizman, Z.; Baharom, R.; Wahab, N. Early Detection of Diabetic Retinopathy by Using Deep Learning Neural Network. Int. J. Eng. Tech. 2018, 7, 198–201. https://doi.org/10.14419/ijet.v7i4.11.20804

He, K, Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90

He, L., Long, L., Antani, S., & Thoma, G. (2012). Histology image analysis for carcinoma detection and grading. *Computer Methods and Programs in Biomedicine*, *107*, 538–556. https://doi.org/10.1016/j.cmpb.2011.12.007

Hinton, G. (2012). Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch

gradient descent. *COURSERA: Neural Networks for Machine Learning*.

Hosny, K., Kassem, M., & Fouad, M. (2019). Classification of skin lesions using transfer learning and augmentation with Alex-net. *PLoS ONE*, *14*, e0217293. https://doi.org/10.1371/journal.pone.0217293

Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv 2017, arXiv:1704.04861.

Huang, G., Liu, Z., Maaten, L. v. d., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2261–2269). https://doi.org/10.1109/CVPR.2017.243

Hubel, D. H., & Wiesel, T. N. (1977). Ferrier lecture. Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, *198 1130*, 1–59.

Hughes, M., Li, I., Kotoulas, S., & Suzumura, T. (2017). Medical Text Classification Using Convolutional Neural Networks. *Studies in Health Technology and Informatics*, *235*. https://doi.org/10.3233/978-1-61499-753-5-246

Idrees, F., Rajarajan, M., Conti, M., Chen, T. M., & Rahulamathavan, Y. (2017). PIndroid: A novel Android malware detection system using ensemble learning methods. *Computers & Security*, *68*, 36–46. https://doi.org/https://doi.org/10.1016/j.cose.2017.03.011

Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv Prepr. 2015, arXiv:1502.03167

Ishtiaq, U.; Kareem, S.; Rahayu, E.; Mujtaba, G.; Jahangir, R.; Ghafoor, H. Diabetic retinopathy detection through artificial intelligent techniques: A review and open issues. Multimed. Tools Appl. 2019, 78, 1–44. https://doi.org/10.1007/s11042-018-7044-8

James, G.; Witten, D.; Hastie, T.; Tibshirani, R.; Trevor Hastie, G.J.; Robert Tibshirani, D.W. An Introduction to Statistical Learning: With Applications in R.; Springer Publishing Company: Berlin/Heidelberg, Germany, 2014; ISBN 9781461471370.

Jangid, M., & Srivastava, S. (2019). Deep ConvNet with Different Stochastic Optimizations for Handwritten Devanagari Character: Proceedings of IC4S 2017, Volume 1 (pp. 51–60). https://doi.org/10.1007/978-981-13-0341-8_5

John, V., Yoneda, K., Qi, B., Liu, Z., & Mita, S. (2014). Traffic light recognition in varying illumination using deep learning and saliency map. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (pp. 2286–2291). https://doi.org/10.1109/ITSC.2014.6958056

Jukić, D.M.; Drogowski, L.M.; Martina, J.; Parwani, A.V. Clinical examination and validation of primary diagnosis in anatomic pathology using whole slide digital images. Arch. Pathol. Lab.

Med. 2011, 135, 372–378

Kaggle. PatchCamelyon dataset. Retrieved from https://www.kaggle.com/c/histopathologic-cancer-detection/data

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, *1*. https://doi.org/10.3115/v1/P14-1062

Kandel, I., & Castelli, M. (2020). A novel architecture to classify histopathology images using convolutional neural networks. *Applied Sciences (Switzerland)*, *10*(8). https://doi.org/10.3390/APP10082929

Kandel, I, & Castelli, M. (2020). Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review. *Applied Sciences* , *10*(6). https://doi.org/10.3390/app10062021

Kassani, S. H., Kassani, P. H., Wesolowski, M. J., Schneider, K. A., & Deters, R. (2019). Classification of Histopathological Biopsy Images Using Ensemble of Deep Learning Networks.
ArXiv:1909.11870.

Khan, N.; Abraham, N.; Hon, M. Transfer learning with intelligent training data selection for prediction of Alzheimer's Disease. IEEE Access 2019, 7, 72726–72735.

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing ({EMNLP})* (pp. 1746–1751). Doha, Qatar: Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1181

Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. arXiv 2014, arXiv:1412.6980.

Kitamura, G., Chung, C. Y., & Moore, B. E. (2019). Ankle Fracture Detection Utilizing a Convolutional Neural Network Ensemble Implemented with a Small Sample, De Novo Training, and Multiview Incorporation. *Journal of Digital Imaging*, *32*(4), 672–677. https://doi.org/10.1007/s10278-018-0167-7

Klambauer, G.; Unterthiner, T.; Mayr, A.; Hochreiter, S. Self-Normalizing Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 971–980.

Kornblith, S.; Shlens, J.; Le, Q. Do Better ImageNet Models Transfer Better. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–21 June 2019; pp. 2656–2666.

Krizhevsky, A., Sutskever, I., & E. Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, *25*. https://doi.org/10.1145/3065386

Kruskal, W. H., & Wallis, W. A. (1952). Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, *47*(260), 583–621. https://doi.org/10.2307/2280779

Lai, Z., & Deng, H. (2018). Medical Image Classification Based on Deep Features Extracted by Deep Model and Statistic Feature Fusion with Multilayer Perceptron. *Computational Intelligence and Neuroscience*, *2018*, 1–13. https://doi.org/10.1155/2018/2061516

Lam, C., Yi, D., Guo, M., & Lindsey, T. (2018). *Automated Detection of Diabetic Retinopathy using Deep Learning*. *AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science* (Vol. 2017).

Lam, C., Yu, C., Huang, L., & Rubin, D. (2018). Retinal Lesion Detection With Deep Learning Using Image Patches. *Investigative Ophthalmology & Visual Science*, *59*(1), 590–596. https://doi.org/10.1167/iovs.17-22721

Längkvist, M., Karlsson, L., & Loutfi, A. (2014). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Pattern Recognition Letters*, *42*(1), 11–24. https://doi.org/10.1016/j.patrec.2014.01.008

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, X., Pang, T., Xiong, B., Liu, W., Liang, P., & Wang, T. (2017). Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)* (pp. 1–11). https://doi.org/10.1109/CISP-BMEI.2017.8301998

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., & Lederberg, J. (1993). DENDRAL: A case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, *61*(2), 209–261. https://doi.org/https://doi.org/10.1016/0004-3702(93)90068-M

Lindsey, R.; Daluiski, A.; Chopra, S.; Lachapelle, A.; Mozer, M.; Sicular, S.; Hanel, D.; Gardner, M.; Gupta, A.; Hotchkiss, R.; et al. Deep neural network improves fracture detection by clinicians. Proc. Natl. Acad. Sci. USA 2018, 115, 11591–11596. https://doi.org/10.1073/pnas.1806905115

Liu, Y., Racah, E., Prabhat, M., Correa, J., Khosrowshahi, A., Lavers, D., & Collins, W. (2016). Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets. ArXiv:1605.01156.

Lotfy, M., Adeghate, J., Kalasz, H., Singh, J., & Adeghate, E. (2015). Chronic Complications of Diabetes Mellitus: A Mini Review. *Current Diabetes Reviews*, *13*. https://doi.org/10.2174/1573399812666151016101622

Luo, H., Yang, Y., Tong, B., Wu, F., & Fan, B. (2018). Traffic Sign Recognition Using a Multi-Task Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems*, *19*(4), 1100–1111. https://doi.org/10.1109/TITS.2017.2714691

Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of ICML 2013, Atlanta, GA, USA, 16–21 June 2013.

Masood, S., Luthra, T., Sundriyal, H., & Ahmed, M. (2017). *Identification of diabetic retinopathy in eye images using transfer learning*. https://doi.org/10.1109/CCAA.2017.8229977

Masters, D., & Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks. ArXiv:1804.07612.

Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, Vijayan K. Asari. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approache*s. ArXiv:1803.01164.

Metter, D., Colgan, T., Leung, S., & Timmons, C. (2019). Trends in the US and Canadian Pathologist Workforces From 2007 to 2017. *JAMA Network Open, 2*, e194337. https://doi.org/10.1001/jamanetworkopen.2019.4337

Mitchell, T.M. Machine Learning, 1st ed.; McGraw-Hill Inc.: New York, NY, USA, 1997; ISBN 0070428077.

Mohammadian, S., Karsaz, A., & Roshan, Y. M. (2017). Comparative Study of Fine-Tuning of Pre-Trained Convolutional Neural Networks for Diabetic Retinopathy Screening. In *2017 24th National and 2nd International Iranian Conference on Biomedical Engineering (ICBME)* (pp. 1–6). https://doi.org/10.1109/ICBME.2017.8430269

Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, *7*, 1419. https://doi.org/10.3389/fpls.2016.01419

Moonen, P.-J., Mercelina, L., Boer, W., & Fret, T. (2017). Diagnostic error in the Emergency Department: follow up of patients with minor trauma in the outpatient clinic. *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, *25*(1), 13. https://doi.org/10.1186/s13049-017-0361-5

Nair, V.; Hinton, G. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. In Proceedings of ICML; ACM: New York, NY, USA, 2010; Volume 27, pp. 807–814.

Nguyen, P. T., Nguyen, T. T., Nguyen, N. C., & Le, T. T. (2019). Multiclass Breast Cancer Classification Using Convolutional Neural Network. In *2019 International Symposium on Electrical and Electronics Engineering (ISEE)* (pp. 130–134). https://doi.org/10.1109/ISEE2.2019.8920916

Nørgaard, M. F., & Grauslund, J. (2018). Automated Screening for Diabetic Retinopathy – A

Systematic Review. *Ophthalmic Research*, *60*. https://doi.org/10.1159/000486284

Okur, M., Karantas, I., & Siafaka, P. (2017). Diabetes Mellitus: A Review on Pathophysiology, Current Status of Oral Medications and Future Perspectives. *Acta Pharmaceutica Sciencia*, *55*, 61–82. https://doi.org/10.23893/1307-2080.APS.0555

Ouahabi, A. (2013). A review of wavelet denoising in medical imaging. In *2013 8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA)* (pp. 19–26). https://doi.org/10.1109/WoSSPA.2013.6602330

Padhy, S. K., Takkar, B., Chawla, R., & Kumar, A. (2019). Artificial intelligence in diabetic retinopathy: A natural step to the future. *Indian Journal of Ophthalmology*, *67*, 1004–1009. https://doi.org/10.4103/ijo.IJO_1989_18

Pan, S., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineeringions on Knowledge and Data Engineering*, *22*, 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Paranjpe, M. (2014). REVIEW OF METHODS FOR DIABETIC RETINOPATHY DETECTION AND SEVERITY CLASSIFICATION. *International Journal of Research in Engineering and Technology*, *03*, 619–624. https://doi.org/10.15623/ijret.2014.0303115

Pires, R., Jelinek, H. F., Wainer, J., Valle, E., & Rocha, A. (2014). Advancing Bag-of-Visual-Words Representations for Lesion Classification in Retinal Images. *PLOS ONE*, *9*(6), e96814. Retrieved from https://doi.org/10.1371/journal.pone.0096814

Prentašić, P., & Lončarić, S. (2016). Detection of exudates in fundus photographs using deep neural networks and anatomical landmark detection fusion. *Computer Methods and Programs in Biomedicine*, *137*, 281–292. https://doi.org/https://doi.org/10.1016/j.cmpb.2016.09.018

Prilianti, K., Brotosudarmo, T., Anam, S., & Suryanto, A. (2019). *Performance comparison of the convolutional neural network optimizer for photosynthetic pigments prediction on plant digital image*. https://doi.org/10.1063/1.5094284

Radiuk, P. (2017). Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. *Information Technology and Management Science*, *20*. https://doi.org/10.1515/itms-2017-0003

Raghu, M.; Zhang, C.; Kleinberg, J.; Bengio, S. Transfusion: Understanding Transfer Learning with Applications to Medical Imaging. arXiv 2019, arXiv:1902.07208.

Rajpurkar, P.; Irvin, J.; Bagul, A.; Ding, D.Y.; Duan, T.; Mehta, H.; Yang, B.J.; Zhu, K.; Laird, D.; Ball, R.L.; et al. MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs. arXiv 2017, arXiv:1712.06957.

Robbins, P.; Pinder, S.; de Klerk, N.; Dawkins, H.; Harvey, J.; Sterrett, G.; Ellis, I.; Elston, C. Histological grading of breast carcinomas: A study of interobserver agreement. Hum. Pathol. 1995, 26, 873–879. doi.org/10.1016/0046-8177(95)90010-1

Ruder, S. An overview of gradient descent optimization algorithms. arXiv 2016, arXiv:1609.04747.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vis. 2015, 115, 211–252. https://doi.org/10.1007/s11263-015-0816-y

Santurkar, S.; Tsipras, D.; Ilyas, A.; Madry, A. How Does Batch Normalization Help Optimization? In Advances in Neural Information Processing Systems 31; Curran Associates, Inc.: Red Hook, NY, USA, 2018; pp. 2483–2493.

Scherer, D.; Müller, A.; Behnke, S. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition; Springer: Berlin/Heidelberg, Germany, 2010. https://doi.org/10.1007/978-3-642-15825-4_10

Schmidt, R.; Schneider, F.; Hennig, P. Descending through a Crowded Valley Benchmarking Deep Learning Optimizers. arXiv 2020, arXiv:2007.01547.

Sermanet, P., & LeCun, Y. (2011). Traffic sign recognition with multi-scale Convolutional Networks. In *The 2011 International Joint Conference on Neural Networks* (pp. 2809–2813). https://doi.org/10.1109/IJCNN.2011.6033589

Sharma, S.; Mehra, D.R. Breast cancer histology images classification: Training from scratch or transfer learning? ICT Express 2018, 4, 247–254. https://doi.org/10.1016/j.icte.2018.10.007

Shin, H.; Roth, H.R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R.M. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. IEEE Trans. Med. Imaging 2016, 35, 1285–1298. https://doi.org/10.1109/TMI.2016.2528162

Shorten, C.; Khoshgoftaar, T.M. A survey on image data augmentation for deep learning. Journal of Big Data 2019, 6, 60. https://doi.org/10.1186/s40537-019-0197-0

Siegel, R. L., Miller, K. D., & Jemal, A. (2020). Cancer statistics, 2020. *CA: A Cancer Journal for Clinicians*, *70*(1), 7–30. https://doi.org/10.3322/caac.21590

Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 2014, arXiv:1409.1556.

Singh, A. K., Ganapathysubramanian, B., Sarkar, S., & Singh, A. (2018). Deep Learning for Plant Stress Phenotyping: Trends and Future Perspectives. *Trends in Plant Science*, *23*(10), 883–898. https://doi.org/10.1016/j.tplants.2018.07.004

Sirinukunwattana, K., Raza, S. E. A., Tsang, Y., Snead, D. R. J., Cree, I. A., & Rajpoot, N. M. (2016). Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images. *IEEE Transactions on Medical Imaging*, *35*(5), 1196–1206. https://doi.org/10.1109/TMI.2016.2525803

Spanhol, F. A., Oliveira, L. S., Petitjean, C., & Heutte, L. (2016). A Dataset for Breast Cancer Histopathological Image Classification. *IEEE Transactions on Biomedical Engineering*, *63*(7),

1455–1462. https://doi.org/10.1109/TBME.2015.2496264

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 2014, 15, 1929–1958.

Swastika, W., Febriant Ariyanto, M., Setiawan, H., & Lucky Tirma Irawan, P. (2019). *Appropriate CNN Architecture and Optimizer for Vehicle Type Classification System on the Toll Road*. *Journal of Physics: Conference Series* (Vol. 1196). https://doi.org/10.1088/1742-6596/1196/1/012044

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Szegedy, Christian, Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. B. (2016). Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2818–2826). Las Vegas. https://doi.org/10.1109/CVPR.2016.308

Tae-Hyun, H.; In-Hak, J.; Seong-Ik, C. Detection of Traffic Lights for Vision-Based Car Navigation System BT-Advances in Image and Video Technology; Springer: Berlin/Heidelberg, Germany, 2006; pp. 682–691. DOI:10.1007/11949534_68

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., & Liang, J. (2016). Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE Transactions on Medical Imaging*, *35*(5), 1299–1312. https://doi.org/10.1109/TMI.2016.2535302

Takahashi, H.; Tampo, H.; Arai, Y.; Inoue, Y.; Kawashima, H. Applying artificial intelligence to disease staging: Deep learning for improved staging of diabetic retinopathy. PLoS ONE 2017, 12, e0179790. https://doi.org/10.1371/journal.pone.0179790

Tanzi, L., Vezzetti, E., Moreno, R., & Moos, S. (2020). X-Ray Bone Fracture Classification Using Deep Learning: A Baseline for Designing a Reliable Approach. *Applied Sciences*, *10*, 1507. https://doi.org/10.3390/app10041507

Tsighe Hagos, M.; Kant, S. Transfer Learning based Detection of Diabetic Retinopathy from Small Dataset. arXiv 2019, arXiv:1905.07203.

Vashist, P., Singh, S., Gupta, N., & Saxena, R. (2011). Role of Early Screening for Diabetic Retinopathy in Patients with Diabetes Mellitus: An Overview. *Indian Journal of Community Medicine : Official Publication of Indian Association of Preventive & Social Medicine*, *36*, 247–252. https://doi.org/10.4103/0970-0218.91324

Veeling, B.S.; Linmans, J.; Winkens, J.; Cohen, T.; Welling, M. Rotation Equivariant CNNs for Digital Pathology BT–Medical Image Computing and Computer Assisted Intervention–

MICCAI 2018; Springer Cham: Granada, Spain, 2018; pp. 210–218.

Vesal, S., Ravikumar, N., Davari, A., Ellmann, S., & Maier, A. (2018). *Classification of Breast Cancer Histology Images Using Transfer Learning*. (A. Campilho, F. Karray, & B. ter Haar Romeny, Eds.), *Image Analysis and Recognition. ICIAR*. Póvoa de Varzim, Portugal: Springer International Publishing. https://doi.org/10.1007/978-3-319-93000-8_92

Vuk, M. ROC Curve, Lift Chart and Calibration Plot. Comput. Sci. 2006, 3, 89–108.

Wan, S., Liang, Y., & Zhang, Y. (2018). Deep convolutional neural networks for diabetic retinopathy detection by image classification. *Computers & Electrical Engineering*, *72*, 274–282. https://doi.org/https://doi.org/10.1016/j.compeleceng.2018.07.042

Wang, X., Lu, Y., Wang, Y., & Chen, W. (2018). Diabetic Retinopathy Stage Classification Using Convolutional Neural Networks. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)* (pp. 465–471). https://doi.org/10.1109/IRI.2018.00074

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1578–1585). https://doi.org/10.1109/IJCNN.2017.7966039

Wilkinson, C.P.; Ferris, F.L.; Klein, R.E.; Lee, P.P.; Agardh, C.D.; Davis, M.; Dills, D.; Kampik, A.; Pararajasegaram, R.; Verdaguer, J.T. Proposed international clinical diabetic retinopathy and diabetic macular edema disease severity scales. Ophthalmology 2003, 110, 1677–1682. doi: 10.1016/S0161-6420(03)00475-5

William J. Clancey and Edward H. Shortliffe. 1984. Readings in medical artificial intelligence: the first decade. Addison-Wesley Longman Publishing Co., Inc., USA.

Wilson, D. R., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, *16*(10), 1429–1451. https://doi.org/https://doi.org/10.1016/S0893-6080(03)00138-2

Xie, M.; Jean, N.; Burke, M.; Lobell, D.; Ermon, S. Transfer learning from deep features for remote sensing and poverty mapping. In Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016, Phoenix, AZ, USA, 12–17 February 2016; pp. 3929–3935.

Xu, B.; Wang, N.; Chen, T.; Li, M. Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv 2015, arXiv:1505.00853.

Xu, X., Lin, J., Tao, Y., & Wang, X. (2018). An Improved DenseNet Method Based on Transfer Learning for Fundus Medical Images. In *2018 7th International Conference on Digital Home (ICDH)* (pp. 137–140). https://doi.org/10.1109/ICDH.2018.00033

Yip, M.Y.T.; Lim, Z.W.; Lim, G.; Quang, N.D.; Hamzah, H.; Ho, J.; Bellemo, V.; Xie, Y.; Lee, X.Q.; Lee, M.L.; et al. Enhanced Detection of Referable Diabetic Retinopathy via DCNNs and Transfer Learning BT-Computer Vision–ACCV 2018 Workshops; Carneiro, G., You, S., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 282–288.

https://doi.org/10.1007/978-3-030-21074-8_23

Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. How Transferable are Features in Deep Neural Networks; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2014; pp. 3320–3328.

Zeng, X., Chen, H., Luo, Y., & Ye, W. (2019). Automated diabetic retinopathy detection based on binocular siamese-like convolutional neural network. *IEEE Access*, *7*, 30744–30753. https://doi.org/10.1109/ACCESS.2019.2903171

Zhang, W., Zhong, J., Yang, S., Gao, Z., Hu, J., Chen, Y., & Yi, Z. (2019). Automated identification and grading system of diabetic retinopathy using deep neural networks. *Knowledge-Based Systems*, *175*, 12–25. https://doi.org/10.1016/j.knosys.2019.03.016

Zhang, X.; Zhao, J.; LeCun, Y. Character-Level Convolutional Networks for Text Classification. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2015; Volume 1, pp. 649–657.

Zhao, B., Lu, H., Chen, S., Liu, J., & Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, *28*, 162–169. https://doi.org/10.21629/JSEE.2017.01.18

Zheng, Y.; He, M.; Congdon, N. The Worldwide Epidemic of Diabetic Retinopathy. Indian journal of ophthalmology. 2012, 60, 428. https://doi.org/10.4103/0301-4738.100542