FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Detection of COVID-19 in X-Ray images using Neural Networks
Student:	Dominik Chodounský
Supervisor:	Ing. Jakub Žitný
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Research current state-of-the-art techniques that are used for detection and segmentation tasks in the medical imaging domain, and focus on X-Ray images. Implement your own prototype model that will work on open COVID-19 datasets available online. Compare the performance of your architecture with reference results from literature and existing models. Discuss their pros and cons. Publish your prototype code and make sure your results are reproducible.

Electronically approved by Ing. Karel Klouda, Ph.D. on 10 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Detection of COVID-19 in X-ray images using Neural Networks

Dominik Chodounský

Department of Applied Mathematics
Supervisor: Ing. Jakub Žitný

May 13, 2021

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor, Ing. Jakub Žitný, for his guidance, advice and positive attitude throughout the process of writing this thesis. I would also like to extend my special thanks to my family and friends, who supported me and patiently sat through my lengthy lectures and remarks on the subject of this research — it is greatly appreciated.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 13, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Chodounský, Dominik. *Detection of COVID-19 in X-ray images using Neural Networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstract

The COVID-19 pandemic is a very pressing issue that continues to affect the lives of people around the globe. To combat and overcome the disease, it is necessary for infected patients to be quickly identified and isolated to prevent the virus from spreading. The traditional detection techniques based on molecular diagnosis, such as RT-PCR, are expensive, time-consuming, and their reliability has been shown to fluctuate. In this thesis, we research the detection of COVID-19 in chest X-ray images using convolutional neural networks. We use our findings to implement a prototype that performs binary detection of the disease, evaluate its performance on a collection of open data repositories available online, and compare its results to existing models. Our proposed light-weight architecture called the *BaseNet* achieves an accuracy of 95.50 % on the chosen test set, with a COVID-19 sensitivity of 93.00 %. We further assemble an ensemble of the *BaseNet* along with several other fine-tuned architectures, whose combined classification accuracy is 99.25 % with a measured sensitivity of 98.50 %.

Keywords computer-aided diagnosis, chest radiography, COVID-19 detection, prototype implementation, convolutional neural networks, CNN architecture comparison, ensemble learning

Abstrakt

Pandemie způsobena nemocí COVID-19 je velmi naléhavým problémem, který nadále ovlivňuje životy lidí po celém světě. K překonání této nemoci je nutné včas identifikovat a izolovat infikované pacienty, aby se zabránilo šíření viru. Tradiční detekční techniky založené na molekulární diagnostice, jako například RT-PCR, jsou nákladné, časově náročné a studie ukazují, že jejich spolehlivost značně kolísá. V této práci jsme zkoumali detekci nemoci COVID-19 v rentgenových snímcích hrudníku pomocí konvolučních neuronových sítí. Poznatky z provedené rešerše dále využíváme k implementaci prototypu pro provádění binární detekce a jeho následnému vyhodnocení na souboru otevřených datových repozitářů dostupných online. Tyto výsledky poté porovnáváme se stávajícími řešeními a modely. Naše navrhovaná jednoduchá architektura s názvem *BaseNet* dosahuje na zvolené testovací sadě dat přesnosti 95.50 % a senzitivity 93.00 %. Zmíněný *BaseNet* jsme dále spolu s několika dalšími vyladěnými architekturami spojili do souboru modelů, jejichž kombinovaná klasifikační přesnost je 99.50 % s naměřenou senzitivitou 98.50 %.

Klíčová slova počítačem podporovaná diagnóza, rentgenové snímání hrudníku, detekce COVID-19, implementace prototypu, konvoluční neuronové sítě, porovnání architektur CNN, ensemble learning

Contents

List of Figures

List of Tables

Introduction

As the global pandemic caused by the SARS-CoV-2 virus continues to affect the everyday lives of people around the world, we are presented with an opportunity to utilize the latest advances in the field of computer vision and deep learning to ensure early detection and containment of the disease that it causes. Not unlike other pulmonary diseases, COVID-19 may be detectable in X-ray images of the lungs, where it forms specific lung markings, especially in the more severe cases. Creating models that extract these specific features and training them to classify whether the X-ray shows symptoms of COVID-19 would provide an alternative diagnosis technique to the standard molecular tests, which tend to be more expensive, time-consuming, and less accessible in certain parts of the world. If found to be reliable, the trained models could potentially be directly incorporated into the clinical diagnostic workflow in hospitals, where they would assist medical professionals in identifying infected patients in a timely manner.

Motivation

Although the subject of computer-aided diagnosis of COVID-19 in X-ray images is still relatively new, there is already a number of works that have attempted to approach it with varying degrees of success. Among the most common models used to perform this task are convolutional neural networks, which have been rapidly evolving over the past decade. The individual solutions usually focus on a specific part of the model building process, such as the architecture of the model itself, the preprocessing methods applied to the X-ray images or finding solutions to problems that are commonly found with medical datasets such as class imbalance and lack of variation. In this thesis, we will take a comprehensive approach to thoroughly exploring all aspects and possibilities of utilizing deep neural networks in search of a reliable method of diagnosing COVID-19. The collection of observations will hopefully provide a solid basis for prototyping models to be used in detecting this

disease and possibly several others that affect the lungs. Such models could potentially end up becoming a standard method of acquiring a second opinion when performing patients' diagnoses.

Objectives

The primary goals of this thesis are the following:

- Describe the COVID-19 disease and the virus that causes it, as well as the methods currently used to perform its diagnosis,
- Provide the theoretical background of machine learning and neural networks needed to comprehend the research and experimentation related to this subject,
- Research common approaches to dealing with medical datasets, preprocessing of X-ray images, and building convolutional neural network architectures for detection tasks,
- Analyze existing research specifically oriented towards the detection of COVID-19 in X-ray images,
- Utilize the findings to implement a prototype that is able to perform the detection of COVID-19 on open datasets available online, experiment with different configurations of its parameters and training, and compare its performance to existing models.

Our main focus will be on the automated binary detection of COVID-19 in chest X-ray images using current state-of-the-art machine learning and image preprocessing methods. To achieve this, we will employ the use of convolutional neural networks, which will be trained and evaluated on a collection of publicly available datasets with a wide selection of patients from around the world. We will utilize optimization, regularization, and preprocessing techniques in an attempt to improve the accuracy of these models.

Chapter 1

Detection of COVID-19

This chapter introduces the COVID-19 disease and the standard methods used to diagnose it, as well as their advantages and disadvantages. Furthermore, we discuss how the disease manifests itself in chest X-ray images, which is relevant to its automated detection, which is discussed in the latter part of this thesis.

1.1 COVID-19

Coronaviruses are a family of viruses that cause illnesses such as the common cold or, among the more serious, Severe Acute Respiratory Syndrome (SARS). The coronavirus disease of 2019, also known as COVID-19, is an infectious disease caused by the SARS-CoV-2 coronavirus, whose origin traces back to an outbreak in Wuhan, China, in December 2019. Common symptoms include fever, cough, shortness of breath, and in more severe cases, the infection can cause pneumonia, SARS, kidney failure, and even death. In March of 2020, The World Health Organization (WHO) declared the ongoing outbreak a global pandemic. [1]

Since the distribution of the COVID-19 vaccine is still limited, early detection of the disease and isolation of the patients is essential to containing its spread. The primary method of detecting COVID-19 on a molecular level is the reverse transcription-polymerase chain reaction (RT-PCR) [2]. However, studies have shown that this method alone may not suffice. While RT-PCR produces very few false positives and its specificity is therefore high (98-100 % [3, 4]), its sensitivity has been recorded to vary a lot from study to study (77-95 % [3, 4, 5, 6]). Furthermore, obtaining results through RT-PCR is costly, and it may take several hours or even a few days, depending on the laboratory's capacity. This suggests that using a secondary diagnostic such as radiography may be of use. Examples of radiography imaging techniques used to detect COVID-19 are computed tomography (CT) scans and chest X-rays, which will be the main focus of the experimental part of this thesis.

1.2 Chest X-rays

The chest X-ray (CXR) is the most commonly performed diagnostic X-ray examination [7]. It is a non-invasive test, which helps medical professionals diagnose and treat a wide range of medical conditions. The imaging process involves exposing a part of the body, in this case, the chest, to a small dose of ionizing radiation to produce images of the inside of the body. The result is acquired by taking the negative of this image, and the structures that blocked the radiation appear bright, while the less dense structures let a portion of the radiation pass through and therefore appear darker.

There is a distinction between several different types of projections based on the different views of the chest that are obtained by changing the relative orientation of the body and the direction of the X-ray beam. The projections commonly used in diagnosing COVID-19 are the posterior-anterior (PA) and anterior-posterior (AP) views. In PA view, the patient is positioned in a manner where the beam first enters through their back and exits the chest through the front, while AP is the other way around. It is recommended that PA radiography be prioritized as it produces better images than its AP counterpart. It does, however, increase the risk of cross-infection among the patients. [8]

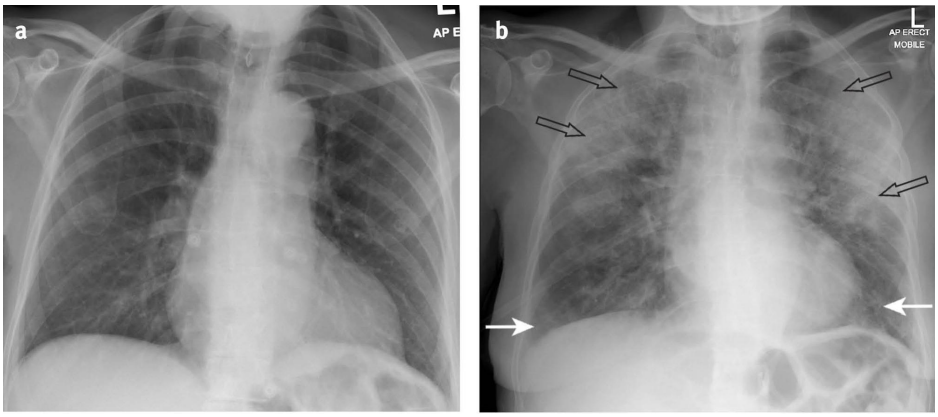


Figure 1.1: Serial radiological progression seen with COVID-19 pneumonia [8].

There are several indicators of COVID-19 pneumonia in the CXR images. One example is the loss of black appearance in the lungs. The increased whiteness gives the part of the lungs a ground-glass appearance, and it marks an area of increased density. The disease can also take the form of coarse horizontal linear opacities on the lungs, and in more severe cases, the opacities can become denser and progress into consolidation with a complete loss of lung markings. Figure 1.1 shows the serial progression of CXR images containing the COVID-19 pneumonia. Image (a) is an AP chest radiograph of a woman in her 70s on the first day of admission to the hospital with a confirmed case

of COVID-19. Image (b) shows the same patient's radiograph after eight days. The white arrows indicate ground-glass opacities present in both lungs, while the black outlined arrows show areas where the opacities are progressing into consolidations. [8]

In most cases, it is challenging to distinguish COVID-19 pneumonia from other types of respiratory infections. However, it does have certain atypical aspects like the fact that the radiographic appearances of ground-glass opacity tend to be multifocal, meaning they affect multiple regions of the lungs [9]. Unfortunately, these findings on chest images are not COVID-19 specific; they have substantial overlap with other more severe infections, such as influenza, H1N1, SARS, or MERS. Another significant drawback of this diagnosis technique is that since most patients with COVID-19 only have a mild form of the illness, they do not develop pneumonia, and the condition may therefore not be detectable in the CXR images at all [10, 11].

Chapter 2

Machine Learning

Machine learning (ML) is a subset of artificial intelligence, which constructs mathematical models and uses experience in order to improve their performance. The experience generally refers to the past information available to the learner, which is used to extract rules and features or discover underlying relationships in the data. We usually call this past information the **training data**. Learning takes the form of data-driven methods, which combine fundamental concepts from computer science, statistics, probability, and optimization in a manner that best satisfies the given criteria. [12, 13]

With the growing availability of online data and low-cost computational resources, the variety of fields utilizing machine learning algorithms keeps expanding. The application domains range from science and technology to manufacturing, marketing, commerce, and even health care.

This chapter introduces the fundamental machine learning concepts needed to establish theoretical background knowledge about the problems and solutions discussed in the latter part of the thesis.

2.1 Supervised vs. Unsupervised Learning

There are many different machine learning scenarios that differ in the nature of the training data, the training itself, and the evaluation criteria. Each approach is fit for a particular class of problems and has its unique advantages and disadvantages. We will discuss the most common distinction of learning approaches: supervised and unsupervised learning.

2.1.1 Supervised Learning

Supervised learning is a process where the training data is labelled with a corresponding class label or value. If we refer to X as the input variable and Y as the output variable, our training data can be expressed as a set of n pairs: $(X_{train}, Y_{train}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. The learner receives this labelled

data from the supervisor and attempts to approximate a mapping function $f : X \to Y$ that predicts the correct labels for a given sample [13]. Such an approximation should be not only able to predict the values given in the training data but also any other data that belongs to the same problem domain.

The learning process is usually iterative. The learner is first initialized (e.g., with random parameters) and subsequently tested on the training data. The supervisor provides the learner with outputs determined not by the process of inference but rather by direct empirical evidence, otherwise known as the ground truth. Knowing the ground truth, the learner proceeds to alter its internal parameters to try and approximate the actual values in the training data. The learning stops when the algorithm reaches an acceptable level of performance, measured by a formerly established metric.

Supervised learning problems can be further grouped into regression and classification problems. In **regression**, we train the model to make continuous predictions. Commonly used models for regression tasks are, for example, linear regression or regression trees. On the other hand, **classification** aims to categorize data into discrete classes. It attempts to understand and be able to distinguish between the classes. An example of a classification problem is the MNIST dataset[1], where the supervisor provides images of handwritten digits, and the learner attempts to classify these images by the digits they represent. The classification itself typically happens on the basis of estimating the probability of the examined item being in each class, and the class with the highest probability is chosen as the prediction. Representatives of the models used in classification are, for example, classification trees, support vector machines, or logistic regression.

2.1.2 Unsupervised Learning

Unsupervised learning has a very different goal to supervised learning. In this case, the input data $X_{train} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ has no corresponding output variable, and the task of the learner is to find solutions on its own. The solution is defined very loosely; it can take the form of patterns, structures, or some other kind of underlying relationships in the unlabelled data. Since there are generally no labelled examples available in the training data, quantifying the learner's performance may be difficult. The most common unsupervised task in machine learning is **clustering**. This is the problem of partitioning a set of points in the domain space into homogeneous subsets, called clusters, which contain points more similar (in some sense) to each other than to those in other clusters [12]. A common use case for clustering is market segmentation in the e-commerce sector. Algorithms used to find these patterns and perform clustering include hierarchical clustering or k-means.

[1] `http://yann.lecun.com/exdb/mnist/`

2.2 Evaluation Metrics

Model evaluation is a core part of the machine learning process. There are many different metrics we can use to rate the model's performance, depending on the task at hand. Since this thesis mainly deals with supervised classification tasks within the medical imaging domain, we will not delve into regression metrics such as mean squared error, mean absolute error, etc.

2.2.1 Accuracy

The most basic classification metric is the **classification accuracy**. It merely describes the portion of the predictions classified correctly by the evaluated model as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} .$$

The ratio is commonly expressed as a percentage, and it should come as no surprise that we look for high accuracy in our models. In a case where there are only two possible classes for the model to pick from, we refer to the metric as **binary accuracy** and usually establish a positive and a negative class.

Under these circumstances, we can witness four different types of classification results. The first case is when a sample from the positive class is classified as positive; we call this a True Positive (TP). If this sample were to be classified wrongly as negative, then it would be a False Negative (FN). On the other hand, a negative sample classified as negative is a True Negative (TN), and if it were to be misclassified, it would be a False Positive (FP). ML researchers will often visualize these metrics in the form of a **confusion matrix** [14]. Having defined these terms, an alternative formula for calculating binary accuracy may be

$$\text{Binary Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} .$$

In the context of medical diagnosis, top-level accuracies are crucial, and any mistakes may result in serious health implications for the patients. Medical professionals should never solely rely on automated diagnosis techniques, but having the models evaluated with a high accuracy score provides the doctors with the possibility of taking these results into consideration as a second opinion.

2.2.2 Predictive Values

Having established the types of outcomes that classification can result in, we further specify the following metrics:

- **Positive Predictive Value**

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Negative Predictive Value**

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}}.$$

The **PPV** informs us about the probability that a positive result in the hypothesis test indicates a real effect. We often use this metric to express the probability of the tested disease's presence in a patient that tested positive for it. Contrastingly, the **NPV** describes the probability of not having this disease considering that the patient tested negative. Both PPV and NPV are highly influenced by the prevalence of the disease in the tested population. Specifically, PPV increases with a higher prevalence of the disease, while NPV decreases. [15]

2.2.3 Area Under the ROC Curve

Another extensively used metric in medical binary classification is the **Area Under the ROC Curve (AUC)**. To be able to define AUC, we must first introduce the following terms:

- **True Positive Rate (Sensitivity)**

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **True Negative Rate (Specificity)**

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

as described by [14]. **Sensitivity** is a metric that helps us evaluate the potential of the diagnostic technique to recognize patients with the disease. It is essentially the probability of getting a positive test result in those that truly have the disease. Highly sensitive tests are useful for ruling out certain diseases if their tests come back as negative [15]. On the other hand, **specificity** tells

us the proportion of patients without the disease that have a negative result. A highly specific test is most informative when it comes out as positive since we can be quite certain that the patient does indeed have the disease [15]. The so-called false positive rate is closely related to specificity, as it can be expressed as $1 - \text{specificity}$.

The **receiver operating characteristic curve (ROC curve)** is a graph where true positive rates (sensitivity) and false positive rates are plotted against each other at different classification thresholds in the interval $[0; 1]$; see Figure 2.1 for an example. By decreasing the classification threshold, we classify more samples as positive, thus producing more true positives and false positives. AUC measures the entire area underneath this ROC curve from the point $(0, 0)$ to $(1, 1)$. A possible interpretation of the AUC score is the probability that the model classifies a random positive sample as actually positive with higher confidence than it would a random negative sample.

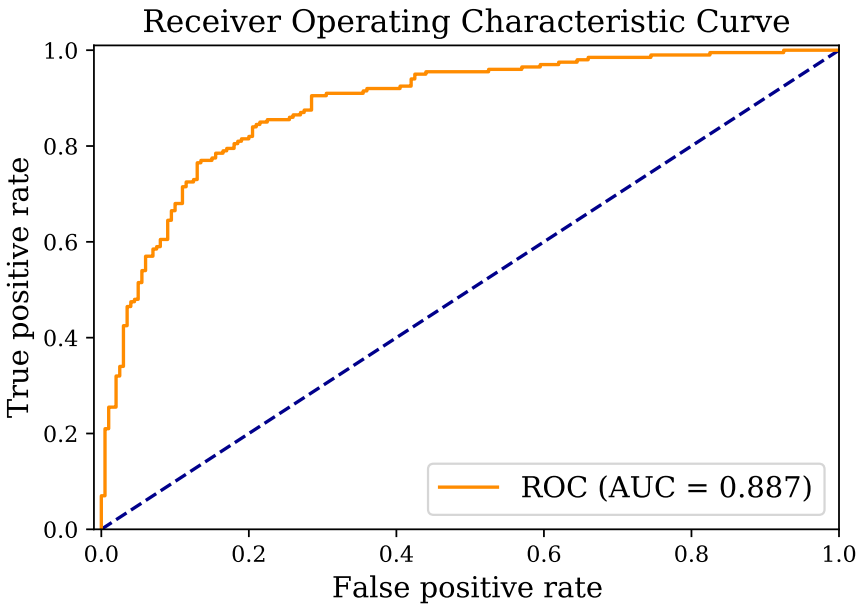


Figure 2.1: An example of an ROC curve with an AUC of 0.887 [14].

We generally look for a model with a high AUC, as an AUC of 1.0 means that the model predicted 100 % of the items correctly, while an AUC of 0.0 means it got all the predictions wrong [16]. An AUC of 0.5 indicates that the model cannot discriminate between the positive and the negative class.

Radiologists often use AUC to compare the overall performance of diagnostic tests and evaluate their discriminatory capacity [17]. It is important to note that two tests with the same AUC may not have the same ROC. One could have a better performance in the higher sensitivity range while the other outperforms it in the lower sensitivity range.

2.2.4 Cross-entropy Loss

Most machine learning algorithms approach the problem of training a model as an optimization task. In mathematical optimization, a **loss function** L is a function that maps a particular state onto a real number, which expresses some sort of a penalty for failing to achieve the desired goal. In machine learning, this loss function takes on the role of representing an error of the predictions made by an examined model. We may write this as $L : (Y, \hat{Y}) \to \mathbb{R}$, where Y is the actual data label and $\hat{Y}$ is the model's prediction of this label. We commonly refer to the loss function's average across the whole dataset as the cost function.

Given a discrete classification task with c possible output classes, the frequently used loss function is the **categorical cross-entropy**. If we mark the estimated conditional probability of a specific $\boldsymbol{x}$ from the data belonging to class i as $\hat{p}_i = \hat{\mathrm{P}}(Y = i|\boldsymbol{X} = \boldsymbol{x})$ and a vector of these probability estimates for each possible class as $\hat{\boldsymbol{p}} = (\hat{p}_1, \ldots, \hat{p}_c)^T$, then the calculation of categorical cross-entropy for this point is

$$L(Y, \hat{\boldsymbol{p}}) = -\sum_{j=1}^{c} \mathbb{1}_{Y=j} \log \hat{p}_j = -\log \hat{p}_Y \ ,$$

where

$$\mathbb{1}_{Y=j} = \begin{cases} 1 & \text{for } Y = j, \\ 0 & \text{otherwise.} \end{cases}$$

If we simplify the task to a binary classification, we can use **binary cross-entropy**. In this case, we denote the estimated conditional probability of a specific $\boldsymbol{x}$ belonging to the positive class as $\hat{p} = \hat{\mathrm{P}}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$ and get the following expression for the loss function

$$L(Y, \hat{p}) = -Y \log \hat{p} - (1 - Y) \log(1 - \hat{p}) \ .$$

The above-mentioned equations were derived from [18].

2.2.5 Bias-varince Tradeoff

We are able to split the error expressed by the loss function into three separate parts. The first part is the inevitable randomness of the problem domain. It is an irreducible error that cannot be explained by our model. Next, we have the bias, which is the difference between the average predictions of the model and the true values, which the model is trying to predict. Bias is caused by the simplifying assumptions made by the model in order to make the target mapping function easier to approximate. The last part of the error is the

variance of the predictions themselves. Variance expresses how much the predictions vary for different sets of training data. [19, 20]

Mathematically, we can mark the irreducible error as σ^2, the bias of the model's predictions $\hat{Y}$ of the output variable Y as $bias\,\hat{Y}$, and their variance as $var\,\hat{Y}$. This gives us the final decomposition of the model's expected error

$$\mathrm{E}\,L(Y,\hat{Y}) = \sigma^2 + (bias\,\hat{Y})^2 + var\,\hat{Y}\ ,$$

as shown by [21].

The phenomenon called the **bias-variance tradeoff** describes the adversarial relationship between bias and variance. Simple models with few internal parameters are characterized by having high bias and low variance. In contrast, more complex models tend to form more complicated approximations leading to high variance and low bias [19]. The goal of any supervised machine learning algorithm is to find a balance between bias and variance, which minimizes the total error. An example of finding such an optimal model complexity can be seen in Figure 2.2.

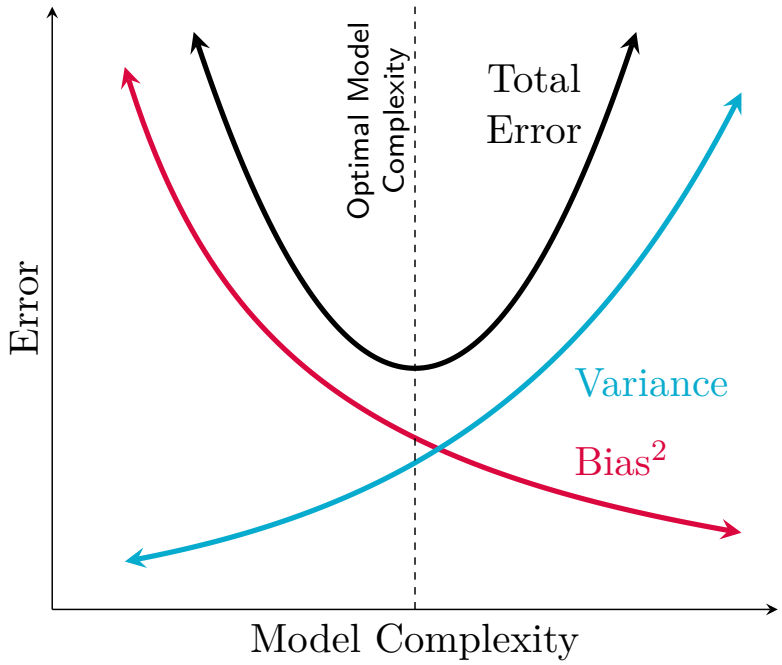


Figure 2.2: Comparison of bias and variance for different model complexities and how they affect the total prediction error [19].

2.3 Training, Validation and Test Set

We have mentioned that machine learning models should be not only able to predict the training data but also any other data that may come in the future. Since we do not have access to future data at the time of training, and not all domains can be characterized by a regular flow of new data, we perform a so-called train-test split. We keep one random portion of the data aside and train the model on the remaining portion. Thus, we are left with a small subset of data which we call the **test set**. The model has never seen this data before, and its training process was therefore not affected by it. We can reliably evaluate the model by taking its predictions on the test set and comparing them to the real values. In doing so, we get an idea of the model's ability to generalize.

Arguably, all machine learning models are parametrized in one way or another. We call these the **hyperparameters (HPs)** of the model, and they are set before the training process begins. Hyperparameters are not determined by the learning algorithm but rather specified as its inputs. Determining these hyperparameters is not straightforward, as different problems require different approaches. If we were to adjust the hyperparameters according to the results they lead to on the test set, then we can no longer claim that the model was trained independently of this data. For this reason, we introduce the **validation set**. The validation set is created by a random split of the training set, and it contains a sample used to tune the hyperparameters of the learning algorithm; see Figure 2.3. [12]

In practice, a so-called 80:20 split is often used for setting up the training and test set. The training set is made up of a random selection of 80 % of the data, while 20 % is used for the overall evaluation. To determine which hyperparameters to use in training, each combination is evaluated on the validation set, which could be randomly selected 15-20 % of the data from the training set. The hyperparameters that led to the best scoring model on the validation data are then used to train the final model. This model is then evaluated on the test set, which has not been involved until this point.

2.3.1 Cross-validation

The aforementioned approach works well in most cases, but it dismisses one problem. There are many possible combinations of train-test sets, and this method experiments with only one. The data could have been randomly split in a way, which is not representative of the whole sample. To bypass this problem, we introduce the concept of cross-validation, which is not dependant on the way the data was split. Specifically, we will focus on **k-fold cross-validation**.

We select a k of at least 2 and at most equal to the number of samples in the training set. We subsequently split the training set into k almost equally large

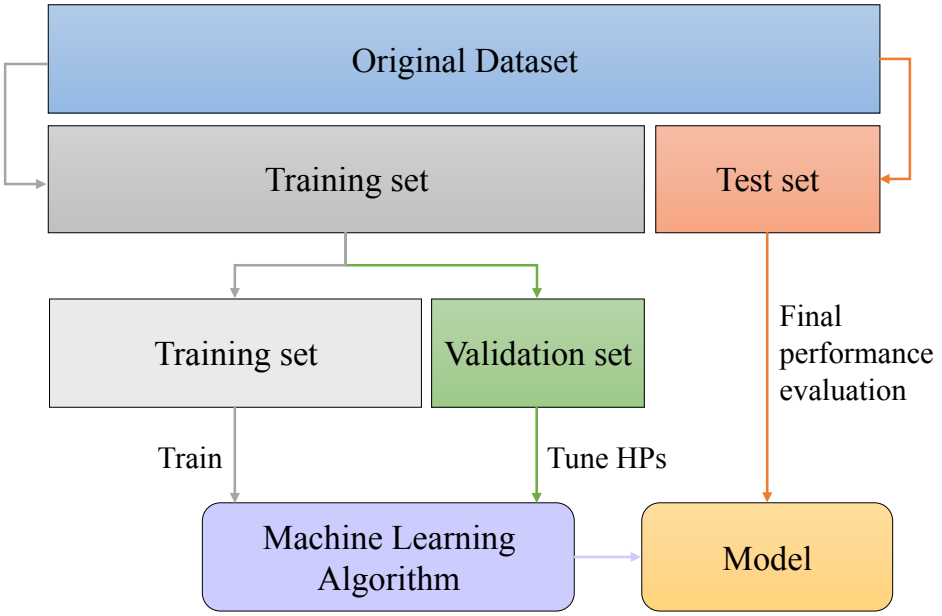


Figure 2.3: Illustration of the composition of subsets used in training and testing a model.

subsets, or **folds**, and label them as $F_1, F_2, \ldots, F_k$. For each $j = 1, 2, \ldots, k$, we proceed to train the model with given hyperparameters on data from the set defined as

$$(\bigcup_{i=1}^{k} F_i) \setminus F_j$$

and test its performance on F_j. We save the result for each j and calculate the model's average performance on all k folds. This process is repeated for every combination of hyperparameters. In the end, those that led to the best average cross-validation performance are chosen to train the final model, which is evaluated on the test set. Splitting the data into k folds is demonstrated in Figure 2.4. [22]

In traditional k-fold cross-validation, we perform random sampling to split the training data into k folds. This could lead to severe issues, especially if the data is very imbalanced. To resolve this problem, we can use the so-called stratified sampling, which captures the distribution of the data within each fold. In **stratified k-fold cross-validation**, each fold is stratified so that it contains approximately the same proportion of class labels as the original dataset. This reduces variance among the estimates and stabilizes the algorithm. [23, 24]

Leave-one-out cross-validation is an extreme case of cross-validation, where k is equal to the number of instances in the dataset. The algorithm is applied once for each instance, using all other instances as the training set and the selected instance as a single-item test set. [25, 22]

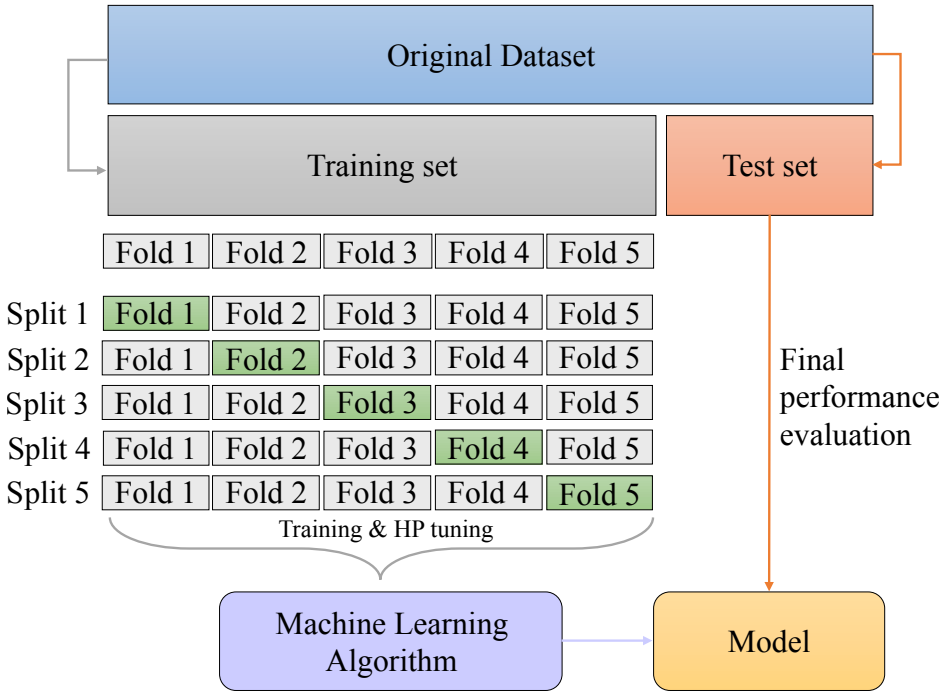


Figure 2.4: Illustration of k-fold cross-validation for $k = 5$.

2.3.2 Overfitting and Underfitting

Common issues surrounding the training process of a model are overfitting and underfitting. **Overfitting** happens when the mapping function is too closely fit to accommodate for a limited set of data points in the training set. It usually takes the form of creating an overly complex model, which does not capture the real underlying relationships within the data. In reality, data will often have some degree of noise caused either by the randomness of the problem space, measuring errors, or anomalies (the previously mentioned irreducible error). Attempting to conform too closely to imperfect data can increase the variance and lead to substantial errors and reduced predictive power.

An overfitted model will fail to generalize well; in other words, its performance on the training set will be significantly better than its performance on the test set. We call this disparity the generalization gap, and it tells us that the model failed to capture the information hidden within the training data [26]. Instead, it took advantage of its high capacity to remember specific

details about the training data, which help it decrease the loss function but do not actually capture the sought-after patterns. A model's **capacity** roughly corresponds to the number of trainable parameters it has [20].

If we look to the opposite side of the problem, we can also encounter models with high bias and low ability to fit the training data. This issue is referred to as **underfitting**, and it can be caused by a shortage of training data, low-quality data, having classes with few distinct features, or simply a model unable to extract those features [27]. We find that an underfitted model usually oversimplifies the problem and performs poorly both on the training set and the test set. See Figure 2.5 for examples of underfitting and overfitting. We will discuss possible ways of dealing with the issues of overfitting and underfitting in the context of artificial neural networks in Section 2.6.8.

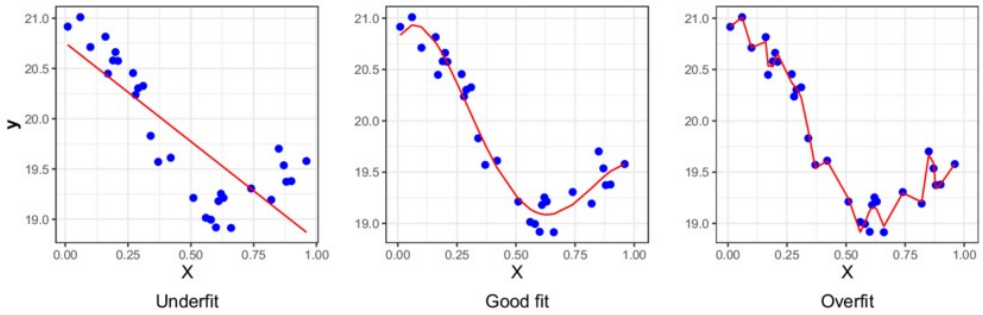


Figure 2.5: Example of an underfitted high bias model on the left, a high variance overfitted model on the right, and a good-fitting model that minimizes the total prediction error in the middle [27].

2.4 Hyperparameter Optimization

In the previous section, we have described using the validation set to evaluate the performance of a model with a given set of hyperparameters. The process of searching for the optimal model configuration is called **hyperparameter optimization (HPO)**. This process may take on a different form depending on the task and the characteristics of the model.

One of the most common and straightforward approaches is **grid search**, also known as full factorial design. This method defines a finite space of hyperparameter configurations, where each hyperparameter is represented by a single dimension. Each dimension is reduced to a finite range of pre-selected discrete values, and grid search evaluates the Cartesian product of all of the sets of values. Such evaluation is often done by calculating the value of the loss function for the model trained with the given set of hyperparameters. The optimum is a point in the hyperparameter space that leads to the lowest validation loss. As pointed out by [28], a significant drawback is that the

required number of evaluations grows exponentially with the dimensionality of the configuration space. This makes the method fit for use with relatively simple models, where the training and evaluation are computationally cheap.

An alternative suggested by [28] is the **random search** method. It views the hyperparameter space in the same manner as grid search, but instead of an exhaustive search of all combinations, it only evaluates a fixed number of random samples from the space. Random search has been shown to be significantly more efficient than grid search, and in many cases, it achieves comparable results [29].

Weighted random search is an attempt at optimizing the random search algorithm by adding a probabilistic greedy heuristic. It assigns probabilities of change to each of the hyperparameters, and values that have been found to perform well will have this probability decreased, as described by [30].

There are many other elaborate optimization methods used in determining hyperparameters of machine learning models, including convolutional neural networks, such as simulated annealing [31], Bayesian optimization [32], evolutionary algorithms [33], etc. These are mainly useful in cases where grid search or random search will not suffice, such as problems with high dimensionality.

2.5 Ensemble Model

It is practically impossible to build a model that can generalize real-world data perfectly. However, if we have several models that come to the same conclusion, we are likely to have higher confidence in that prediction.

Ensemble models are a way to aggregate the predictions from several diverse base models. The collective prediction can be formed by taking the average of the individual predictions in the case of regression, or in our classification case, it is simply the class predicted by the majority of the base models. We may give each base model's vote a different weight; for instance, the votes of the model whose individual prediction accuracy was the highest may be favoured in the final decision. The key aspect that often decreases prediction error and improves the ensemble's ability to generalize is diversity and independence of the base models. This can be achieved by training the models on different sets of data, setting different hyperparameters, or using completely different modelling algorithms altogether.

While there is no guarantee that an ensemble will outperform individual base models, many experimental studies in different areas have shown this to be a widespread occurrence [34, 35, 36].

2.6 Artificial Neural Networks

An **artificial neural network (ANN)** is a computational model inspired by networks of biological neurons found in the human brain. The biological neurons are interconnected by synapses and perform a particular function when activated by electrical impulses from other neighbouring nerve cells. A key concept is that not all synaptic connections are equally weighted, and specific neural pathways may be adjusted and strengthened over time [37]. Similarly, the artificial neuron receives inputs and sends them as an impulse through the network to produce some sort of output based on predefined activation functions. To simulate learning, the weights of individual connections are adjusted by an appropriate optimization algorithm.

The idea of ANNs has been around since 1943 when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper describing how neurons may work, and to support their hypothesis, they used electrical circuits to model a simple neural network. In the following decades, the topic became progressively more popular, with the first simple perceptron model appearing around the 1960s as a result of studies conducted by Frank Rosenblatt and his research group [38]. Soon after, researchers realized the potential of using multi-layer perceptrons, and finally, in 1974, Paul Werbos published a dissertation [39] describing an efficient algorithm for determining the synaptic connection strengths known as the error backpropagation.

Nowadays, ANNs form the basis of countless academic and commercial applications. We use them to make predictions, recognize patterns, and solve all kinds of problems where other approaches fail to deliver satisfactory results.

2.6.1 Single-layer Perceptron

The **single-layer perceptron** is the simplest form of an ANN. As shown in Figure 2.6, it consists only of one layer of connections, where the inputs are directly fed into the output layer, which contains a single artificial neuron. The activation of the output neuron is determined by a non-linear activation function f, which takes the argument ξ, also known as the inner potential. Denoting the inputs of the model as $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and the weights of their corresponding connections to the output neuron as $\boldsymbol{w} = (w_1, \ldots, w_n)^T$, we can calculate the value of the neuron's inner potential as

$$\xi = w_0 + \sum_{i=1}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} + w_0 \ ,$$

where w_0 is a so-called bias that enables the network to fit the data more accurately by allowing for a linear shift of the inner potential [18]. We acquire the perceptron's prediction by applying the activation function to the value of the inner potential as follows

$$\hat{Y} = f(\xi) = f\left(\boldsymbol{w}^T \boldsymbol{x} + w_0\right) .$$

In the case of a single-layer perceptron, the activation function is a step function defined as

$$f(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0, \\ 0 & \text{for } \xi < 0. \end{cases}$$

While the single-layer perceptron is capable of performing binary classification tasks, its limitation comes when dealing with data that is not linearly separable. An example of such a problem is implementing an XOR function. [40]

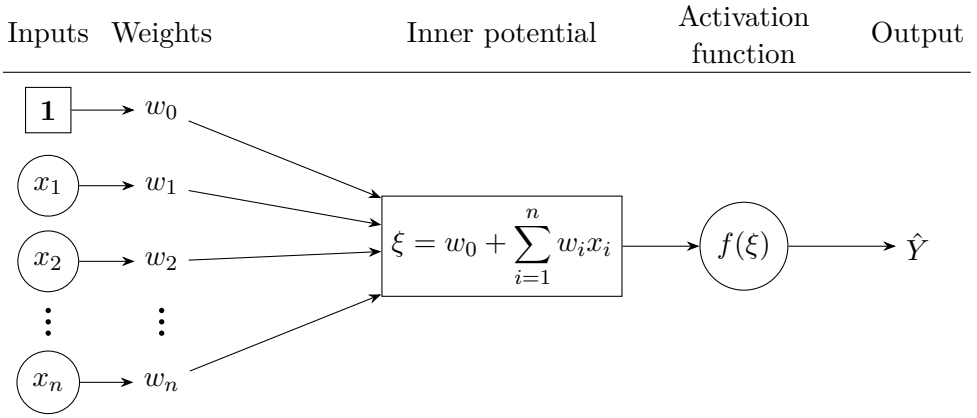


Figure 2.6: Architecture of the single-layer perceptron.

2.6.2 Multi-layer Perceptron

The **multi-layer perceptron (MLP)** is an extension of the single-layer perceptron that introduces an architecture with additional hidden layers between the inputs and the outputs. The layers are interconnected in such a manner that the outputs of one layer, along with a specific bias term, form the inputs to the next layer in line. As was the case with single-layer perceptrons, the output of a neuron is acquired by applying an activation function f to its weighted inputs. To obtain the whole network's output, we must compute the output of each neuron in each of the layers.

For our purposes, let l be the number of layers in the network and let each layer have an arbitrary number of neurons expressed by numbers $n_1, \ldots, n_l$. The number of neurons in a layer is often referred to as its width. We further consider n_0 to be the width of the inputs. With this information, we can model the output of j-th neuron in the i-th layer as function $g_j^{(i)} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}$, where

$$g_j^{(i)} = f\left(w_{0,j}^{(i)} + \sum_{k=1}^{n_{i-1}} w_{k,j}^{(i)}\, g_k^{(i-1)}\right) \quad \text{for } i = 2, 3, \ldots, l \ ,$$

and $w_{k,j}^{(i)}$ is the weight between the j-th neuron in the i-th layer and the k-th neuron in the previous layer that acts as its input. The neuron's bias term is $w_{0,j}^{(i)}$. As the first layer in the network does not receive its inputs from a previous hidden layer but rather directly from the inputs, we consider this a special case where

$$g_j^{(1)} = f\left(w_{0,j}^{(1)} + \sum_{k=1}^{n_0} w_{k,j}^{(1)}\, x_k\right) \ .$$

Since the neurons within one layer cooperate to form inputs for the following layer, we can denote their collective output as a multivariable function, as proposed by [41]. In consideration of the foregoing, the i-th layer of the network can be described by function $\boldsymbol{g}^{(i)} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, where

$$\boldsymbol{g}^{(i)} = \begin{pmatrix} g_1^{(i)} \\ g_2^{(i)} \\ \vdots \\ g_{n_i}^{(i)} \end{pmatrix} = \begin{pmatrix} f\left(w_{0,1}^{(i)} + \sum_{k=1}^{n_{i-1}} w_{k,1}^{(i)}\, g_k^{(i-1)}\right) \\ f\left(w_{0,2}^{(i)} + \sum_{k=1}^{n_{i-1}} w_{k,2}^{(i)}\, g_k^{(i-1)}\right) \\ \vdots \\ f\left(w_{0,n_i}^{(i)} + \sum_{k=1}^{n_{i-1}} w_{k,n_i}^{(i)}\, g_k^{(i-1)}\right) \end{pmatrix} \quad \text{for } i = 2, 3, \ldots, l \ ,$$

and the first layer is modelled analogously with the exception of processing direct inputs $\boldsymbol{x}$ rather than outputs from previous layers.

The most common MLPs are fully-connected feedforward networks, where the information passes through the network in a forward direction and the directed graph created by the connections between the neurons is acyclic [42]. The individual functions of the layers are chained together, creating a composition of functions

$$\boldsymbol{g} = \boldsymbol{g}^{(l)} \circ \boldsymbol{g}^{(l-1)} \circ \cdots \circ \boldsymbol{g}^{(2)} \circ \boldsymbol{g}^{(1)} \ .$$

For $l = 3$, the network's function would be

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{g}^{(3)}(\boldsymbol{g}^{(2)}(\boldsymbol{g}^{(1)}(\boldsymbol{x}))) \ ,$$

where $\boldsymbol{x}$ are the inputs, $\boldsymbol{g}^{(1)}$ and $\boldsymbol{g}^{(2)}$ are the so-called hidden layers and $\boldsymbol{g}^{(3)}$ is the output layer [18]. An example of such an architecture can be seen in Figure 2.7, where both hidden layers have a width of 5 and the output layer has 3 neurons. The length of this chain of layers determines the model's depth, which is where the term **deep learning** originates from.

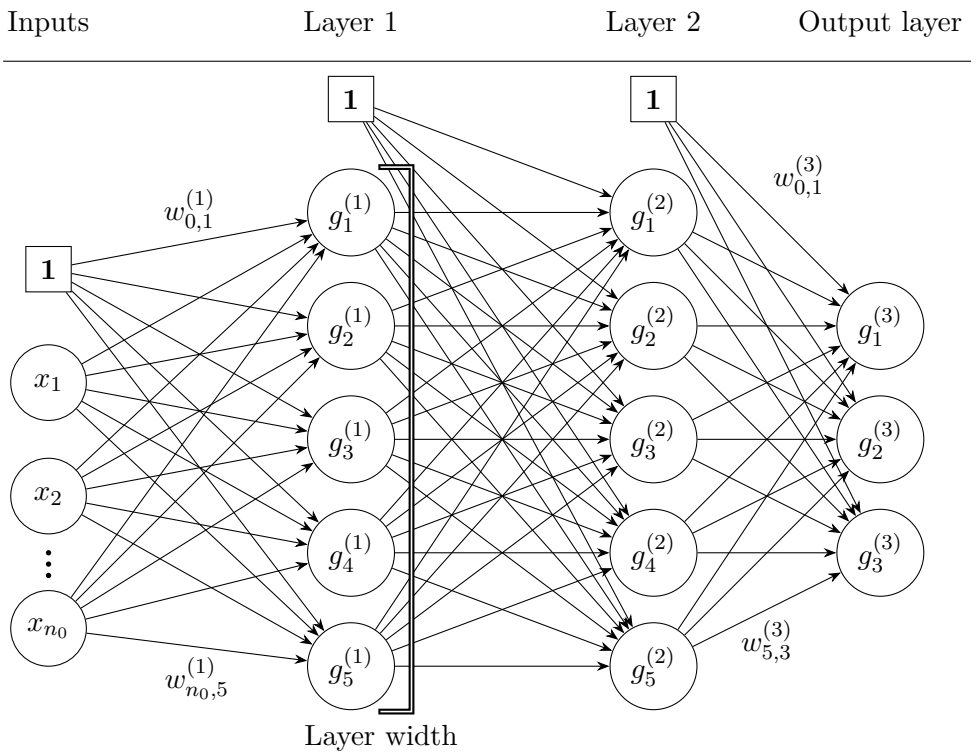


Figure 2.7: Example of a multi-layer perceptron with 3 layers.

2.6.3 Cost Function

As is the case with most machine learning models in supervised learning, the goal of an artificial neural network is to approximate the function that maps inputs in the training data to their labels as accurately as possible. To enable the model to learn, we must first evaluate its performance. For this purpose, we will use the cost function, which calculates the average loss across the training set as

$$C(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L(Y_i, g(\boldsymbol{x_i}, \boldsymbol{w})) \ ,$$

where $\boldsymbol{w}$ are the internal parameters (weights) used in the ANN function g, N is the number of samples in the training set, L is the cross-entropy loss function previously defined in Section 2.2.4, and $\boldsymbol{x}$ are the inputs.

2.6.4 Backpropagation and Gradient Descent

The initial configuration of a neural network usually has randomly initialized weights. Unsurprisingly, such a network will likely not be able to approximate

the desired outputs very accurately. In training an ANN, our goal is to find such a combination of weights that minimizes the cost function. We may simplify this thought by taking a partial derivative $\frac{\partial C}{\partial w_{k,j}^{(i)}}$, which expresses the effect of changing the weight $w_{k,j}^{(i)}$ on the overall cost value. We aim to find this relationship for each weight and then iteratively update the weights throughout the learning process in such a way that decreases the cost.

The vector of these partial derivatives of the cost function with respect to each weight is called the **gradient**, and it is denoted as

$$\nabla_{\boldsymbol{w}} C = \left(\frac{\partial C}{\partial w_{0,1}^{(1)}}, \frac{\partial C}{\partial w_{0,2}^{(1)}}, \ldots, \frac{\partial C}{\partial w_{n_{l-1},n_l-1}^{(l)}}, \frac{\partial C}{\partial w_{n_{l-1},n_l}^{(l)}} \right)^T .$$

A possible interpretation of the gradient vector is the direction and rate of the fastest increase of function C [42]. The method used to calculate the gradient is called **backpropagation**, and it uses the chain rule for finding derivatives of composite functions.

Once we have the gradient, we can use the fact that it points in the direction of the steepest increase and go in the opposite direction to minimize the cost function [43, 18]. The rate by which we step in the opposite direction is called the **learning rate**, and it is a hyperparameter that affects how much the weights get updated during every iteration of the learning process. Choosing a learning rate that is too small may overly prolong the learning process, while too large of a learning rate will cause unnecessarily large weight updates and instability. We may also characterize the learning rate as having some kind of a **decay**, which means its value gradually decreases over time. This is especially useful in cases where we want to start the learning process with larger steps and decrease this step size as we approach the global optimum to make for more precise weight updates.

This method of training ANNs is called the **gradient descent**, and it can be summarized by the following steps:

1. Randomly initialize weights $\boldsymbol{w}$.
2. Evaluate the cost function $C(\boldsymbol{w})$ across the training data $\boldsymbol{x}$.
3. Calculate the gradient $\nabla_{\boldsymbol{w}} C$ using backpropagation.
4. Update the weights as $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} C$, where α is the specified learning rate.
5. Repeat steps 2-4 until cost $C(\boldsymbol{w})$ stops decreasing, or some other termination criteria are met. [18]

When training deep neural networks with gradient descent, we may encounter several problems that negatively impact the model's ability to learn. One such obstacle is the **vanishing gradient** problem, which is characterized by deep near-zero gradients with respect to parameters closer to the input layers. It essentially means that early parameters have a minimal effect on the deeper levels' outputs, which leads to only minor weight updates and a potential inability for the network to learn the parameters correctly. A significant factor causing this problem is the choice of saturating activation functions [44, 45]. These functions map a wide range of input values onto a very narrow range of outputs, meaning they often have small gradients, and the problem gets increasingly worse as we propagate the gradients through multiple saturated layers.

Contrastingly, we may also experience that as we propagate the gradient backwards through the network, it grows exponentially and causes large weight updates. We call this phenomenon the **exploding gradient**, and it may occur when the derivatives of activation functions take on large values and gradually get multiplied. [45]

2.6.5 Optimizers

The gradient descent method for optimizing ANN weights has many different implementations that attempt to resolve some of its possible drawbacks, such as vanishing and exploding gradients or getting trapped in local minima. This section names a few relevant examples commonly used in training neural networks for object detection in images.

2.6.5.1 Stochastic Gradient Descent

The standard gradient descent performs batch training, where the cost is calculated across the whole training set. Since ANNs often require large amounts of data, this may not always be feasible due to computational and storage resource limitations. **Stochastic gradient descent (SGD)** aims to resolve this issue by approximating the expected gradient. Using a smaller representative sample of the training data called a mini-batch, it calculates the cost across the mini-batch and performs gradient descent based on this estimate. As a result, the model's weights get updated more frequently, which may lead to faster convergence. [44]

The hyperparameter determining the number of samples used for calculating the gradient is commonly referred to as the **batch size**, and in some cases, only a single sample (batch size of 1) may be used. One cycle through the complete training set is called an **epoch**.

SGD is often used in combination with the method of **momentum**, which may accelerate convergence in some instances where the gradient has a consistent direction but is otherwise small. The idea of momentum is to incorporate

the previous updates in the current change by keeping an exponentially decaying average of past negative gradients. We represent this as velocity v, which is the direction and speed of the parameters' movement through parameter space. We also introduce the momentum parameter $\beta \in [0; 1)$, which determines the exponential rate of decay for the past gradients. Having stated this notation, we can describe the update to the velocity and the following weight update as

$$v \leftarrow \beta v - \alpha \nabla_{\boldsymbol{w}} C$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + v \ .$$

The larger the momentum parameter is relative to the learning rate, the greater influence the previous gradients have on the following weight update. [46]

2.6.5.2 AdaGrad

AdaGrad is an optimizer that introduces the notion of having a vector of learning rates for each trainable parameter. This allows for adapting the learning rate automatically in different directions. The basic idea is that the weight update is computed by scaling each parameter's learning rate inversely proportional to the square root of the cumulative sum of the squares of the gradient's previous magnitudes [46]. This means that parameters with large partial derivatives will have their learning rate significantly decreased to allow for the other parameters of sparser features to catch up.

Accumulating the squared gradients can often lead to an excessive decrease of the learning rates, which means that the optimizer rapidly slows down and often stops before reaching the global optimum [47].

2.6.5.3 RMSProp

The optimizer **RMSProp** attempts to improve upon the AdaGrad by altering the gradient accumulation. It instead works with an exponentially decaying moving average of the squares of past gradients, which may eliminate the problem of rapidly dropping learning rates [46]. Once again, the parameters that have previously caused large oscillations in the estimate of the cost function will be penalized within the current weight update. Another advantage of this averaging approach is that new data points will not dramatically influence the gradients [48].

2.6.5.4 Adam

Adam is one of the most widely used optimization algorithms in deep learning [46]. Its name is derived from adaptive moment estimation, and it combines the ideas of having both momentum and adaptive learning rates. The

momentum component takes the form of an exponentially decaying moving average of the gradient, or in other words, the estimate of the first-order moment m (the mean of the gradient). On the other hand, the learning rate scaling is accounted for by the exponentially decaying moving average of the squares of the gradient, i.e., the estimate of the second-order moment u (uncentered variance of the gradient). [46, 49]

Considering the aforementioned notation and that β_1 and β_2 mark the exponential decay rates for m and u respectively, the moment estimates are updated in each time step t as

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\boldsymbol{w}_{t-1}} C$$

$$u_t \leftarrow \beta_2 u_{t-1} + (1 - \beta_2)\left(\nabla_{\boldsymbol{w}_{t-1}} C\right)^2 \ .$$

However, the moment estimates are initialized as vectors of zeroes, which means that they are biased towards zero [49] and have to be bias-corrected thusly

$$\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$$

$$\hat{u}_t \leftarrow \frac{u_t}{1 - \beta_2^t} \ .$$

Finally, we update the weight parameters as

$$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \epsilon} \ ,$$

where ϵ is a very small scalar to prevent division by zero.

A popular adaptation of the Adam algorithm is the **Nadam**, which uses Nesterov's accelerated gradient to modify the momentum component and try to achieve faster convergence. It does so by stepping in the direction of the previous accumulated gradients and measuring the gradient at the point where it ends up, where it then makes a correction [50].

2.6.6 Activation Functions

Thus far, we have mentioned that single-layer perceptrons usually use a binary step function as their activation function, but we have yet to address the situation with MLPs. The first aspect to note is that the activation function of an MPL must fulfil the requirement of being differentiable, or at least differentiable in parts. This is due to the fact that the model's learning process involves calculating the gradient using partial derivatives. As was the case with the step function used in single-layer perceptrons, the activation function should be non-linear so as to expand the range of mapping functions that can

be approximated. A wide range of activation functions may be utilized in building a neural network, each with its own advantages and disadvantages. Ultimately, the decision regarding which activation functions to use is a crucial part of the hyperparameter optimization process.

The most frequently used activation function in the output layers is the **logistic sigmoid** [51]. It transforms the input values to range from 0 to 1, which makes it a good fit for binary classification where the result can be represented as a probability. The logistic sigmoid may be defined as

$$f(\xi) = \frac{1}{1 + \mathrm{e}^{-\xi}} = \frac{\mathrm{e}^{\xi}}{1 + \mathrm{e}^{\xi}} \ .$$

We interpret the result as the estimated probability of the given input belonging to the positive class, i.e., $\hat{\mathrm{P}}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$. Aside from ANNs, the logistic sigmoid is also a common tool in logistic regression. Due to it being a saturating function, it can potentially cause the problem of vanishing gradients.

When dealing with multiclass classification, the outputs are generally transformed by the **softmax** function. It is essentially a combination of multiple sigmoids and it provides us with the probability of a given input belonging to a specific class. The function's definition for c classes can be expressed as

$$f_i(\boldsymbol{\xi}) = \frac{\mathrm{e}^{\xi_i}}{\sum_{k=1}^{c} \mathrm{e}^{\xi_k}} \quad \text{for } i = 1, \ldots, c \ ,$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_c)^T$ is a vector of inner potentials of all c neurons in the output layer and the value of the activation function $f_i(\boldsymbol{\xi})$ is interpreted as the estimated probability of input $\boldsymbol{x}$ belonging to class i, i.e., $\hat{\mathrm{P}}(Y = i|\boldsymbol{X} = \boldsymbol{x})$. The chosen prediction will be the class with the highest estimated probability assigned to it. [18]

The sigmoid and softmax are very commonly found in the output layer, however, the hidden layers tend to use other activation functions. One such function is the **hyperbolic tangent (tanh)**. It resembles the sigmoid, although as shown in Figure 2.8, it has a steeper gradient and is symmetric around the origin, which allows for variability in the signs of outputs from previous layers. The transformed range is from -1 to 1, meaning that layers with this activation function are susceptible to saturation. The function is defined as

$$f(\xi) = \tanh(\xi) = \frac{\mathrm{e}^{\xi} - \mathrm{e}^{-\xi}}{\mathrm{e}^{\xi} + \mathrm{e}^{-\xi}} \ .$$

The last noteworthy activation function is the so-called **rectified linear unit (ReLU)**. It is currently the most widely-used activation function due to its efficiency and observed success [52]. It is simple to compute, as its standard definition is

$$f(\xi) = \max(0, \xi) = \begin{cases} \xi & \text{for } \xi \geq 0, \\ 0 & \text{for } \xi < 0. \end{cases}$$

Furthermore, this definition implies that the function is not saturated in its positive direction, and the model is more resistant to the vanishing gradients problem. However, the ReLU does allow for exploding gradients, which could potentially lead to severe instability.

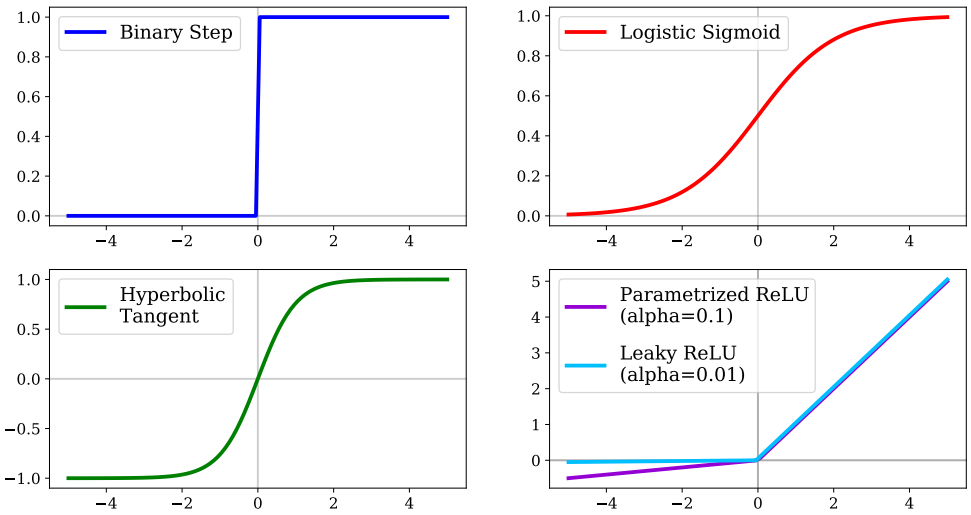


Figure 2.8: Transformations of inner potentials into neuron activations by frequently used activation functions.

Another issue is the so-called dying ReLU problem, which relates to having the same output of 0 for all negative values of the inner potential. This means that the gradient will be zero and the neuron will no longer be able to update its weights during backpropagation [51]. To resolve this negative occurrence, we may use an improvised version of the function called the **Parametrized ReLU**, which introduces the parameter α for the slope of the negative part of the function. The function definition therefore changes accordingly to

$$f(\xi) = \max(0, \xi) = \begin{cases} \xi & \text{for } \xi \geq 0, \\ \alpha\xi & \text{for } \xi < 0. \end{cases}$$

The slope α may be set to be trainable, in which case the optimization algorithm considers it one of the learning parameters [51]. The alternative is to

set the value of α as a hyperparameter before training begins. A common predetermined value for α is 0.01. We call this particular case the **Leaky ReLU**.

2.6.7 Convolutional Neural Networks

One of the most prominent deep multi-layer perceptron architectures of the last decade is the **convolutional neural network (CNN)**. It is a network specifically designed to process data that comes in the form of multiple arrays, for example, colour images composed of 2D arrays of pixels for each of the three colour channels. The CNNs have found many use cases in the fields of image classification, pattern recognition, segmentation, or natural language processing [53].

The architecture is designed to automatically learn spatial hierarchies of features within the grids of data, starting from low-level generic patterns in the first layers to the complex patterns and details in the deeper layers [54]. An example of this may be finding lines and edges and gradually combining them to detect faces. The key aspect is that this detection and pattern recognition must be shift-invariant, meaning that the outputs should not be dependent on the exact location of the objects within the image. The advantage of extracting features using the CNN architecture as opposed to classic ANNs is the reduction of parameters and network complexity. An image contains massive amounts of information, and without its proper preprocessing and spatial analysis, it is difficult to train the network efficiently and effectively.

CNNs generally consist of several building blocks, which are the convolutional layers, pooling layers, and fully connected layers. The typical approach is to stack several convolutional and pooling layers, which perform feature extraction and dimensionality reduction, and then the feature maps get fed into the fully connected layers, which map them to the final outputs of the network.

2.6.7.1 Convolution

The critical component of each convolutional layer is the mathematical operation of linear **convolution**. Within the context of CNNs, convolution is performed on the input data (the image) with the use of a smaller array of weights called the **kernel**. A collection of kernels stacked in multiple dimensions for inputs with more than one channel is called a **filter**. The kernel moves across the width and height of the image, and the dot product of the kernel and the part of the image within the area covered by the kernel is calculated. As we convolve the image with the kernel, we produce an activation map that gives the responses of that kernel at each spatial position within the image [55]. We call this the **feature map** and the network gradually learns filters that activate when they come across specific visual features such as edges or corners, and eventually whole objects. Sharing the kernel weights across

all image positions has the advantage of letting the local feature extraction be shift-invariant as well as reducing the overall number of trainable parameters and therefore increasing the model efficiency [56]. The calculation of the feature map is demonstrated in Figure 2.9.

A neuron in the convolutional layer has local connectivity to only a specific part of the input image. The size of this region, often called the receptive field, is given by the size of the kernel. This is one of the hyperparameters that we set for each convolutional layer. Other hyperparameters include the number of filters in the layer, or the stride, which represents the distance the kernels move across the image in each step of convolution. The bigger the stride, the smaller the produced feature maps.

Since convolution shrinks the original image, we may consider padding the image in places where the kernel reaches out of bounds. Some of the common strategies used are zero-padding, where the outer border of the image is filled with zeroes, or simply using the value of the closest pixel with a defined value. This prevents the image size from shrinking, but it may cause problems since we are artificially adding information that was not present before. [56]

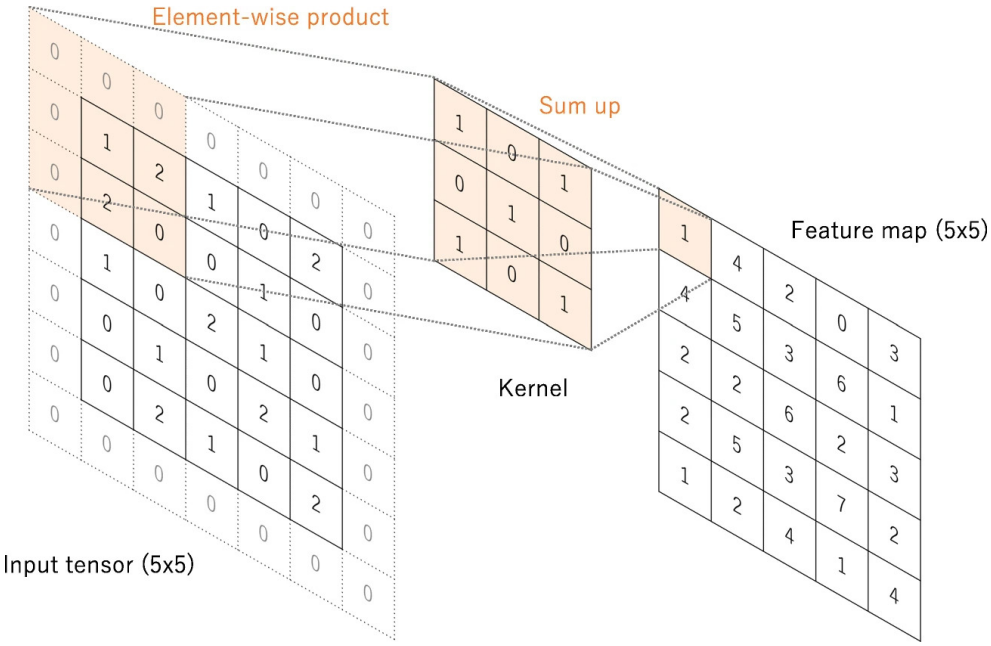


Figure 2.9: Illustration of a feautre map created by convolving a 3×3 kernel across a zero-padded 5×5 input image with a stride of 1 [56].

2.6.7.2 Pooling

The idea behind pooling is to down-sample the feature map and reduce complexity for the following layers. The dimensionality reduction is done by di-

viding the feature map into rectangular subregions of predetermined size and summarizing those regions by single values. One of the most common approaches is **max-pooling**, where the return value is the maximum of the inside of the subregion. It is useful for the extraction of dominant features like edges. Alternatively, we may use **average-pooling** where the subregion is characterized by its average. This method has a smoothening effect on the features but maintains their representation in the further layers, while max-pooling preserves only the most extreme features.

Pooling also helps to maintain invariance towards translational shifts by collecting the features across the whole image. Pooling layers have no trainable parameters, and they have similar hyperparameters as convolutional layers in terms of kernel (subregion) size and strides. [54]

A special pooling operation is **global average pooling** which is an extreme kind of down-sampling, where each feature map gets reduced to its average. The output vector of these feature averages is then fed into the dense fully-connected layers, which act as the classifier.

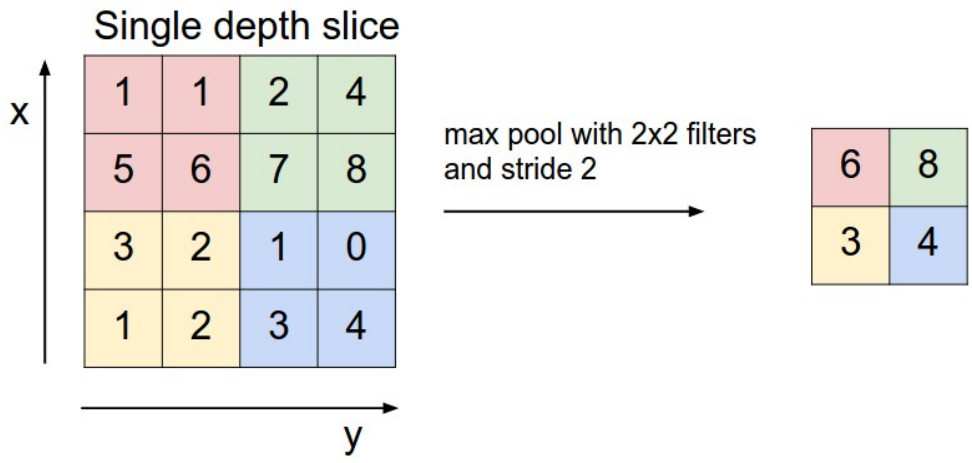


Figure 2.10: Illustration of down-sampling a single dimension of a feature map using the max-pooling method with kernel size of 2×2 and stride of 2 [55].

2.6.8 Regularization

Due to the substantial number of parameters in ANN models, it is often very problematic to prevent overfitting. The most straightforward way to tackle this issue is by reducing the model complexity. We may optimize those hyperparameters that affect the complexity and number of parameters in the model to find a balance between fitting the data well and retaining its ability to generalize. The problem is that reducing the capacity often causes underfitting and lowers the performance on the training set as well. The goal

of **regularization** is to reduce the variance of the model without significantly increasing its bias [57].

2.6.8.1 Data Augmentation

ANNs are among those machine learning models that generally require a lot of training data. Unfortunately, there are domains where gathering new data may not always be an option, or there is a limited flow of new data. This is often the case with datasets for training CNNs. One of the most common techniques used to enhance image datasets for training CNNs is data augmentation. By transforming and adjusting the data we have within certain bounds, we can increase the training set size without too much additional effort. These augmentations usually include affine transformations such as horizontal and vertical shifts, scaling, rotation, shearing, and others such as image mirroring, zooming or random intensity adjustments [58]. It is crucial that the altered images remain realistic to their specific domain. Increasing and diversifying the training set in such a way gives the network a better opportunity to learn more generic patterns rather than focusing on the few original samples and their exact structure.

2.6.8.2 L1 and L2 Regularization

The **L1** and **L2 regularizations** are methods that put constraints on the optimization algorithm when minimizing the model's cost function. The constraint is in penalizing the weights in hopes of simplifying the model. L1 regularization does this by adding a regularization term in the form of the sum of absolute values of the weights to the cost function. This has the effect of feature selection, as features found to be not as important tend to be given zero weight, and the model is therefore simplified [57]. Previously, we have indexed each weight according to which neurons in which layers they connect. In this example, we will simplify the notation by having a vector $\boldsymbol{w} = (w_1, \ldots, w_m)$ of all m weights in the whole network. This enables us to define the L1-regularized cost function C_{L1} as

$$C_{L1}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L(Y_i, g(\boldsymbol{x_i}, \boldsymbol{w})) + \frac{\lambda}{N} \sum_{j=1}^{m} |w_j| \ ,$$

where λ is the regularization parameter directly proportional to the applied penalty [57]. The situation is slightly different with L2 regularization. Here, the regularization term is added to reduce the sum of the squares of the weights. This form of regularization is sometimes referred to as weight decay, as its effect is not driving the weights to zero values, but rather reducing them by a factor proportional to their magnitude at every iteration of the gradient descent [26, 57]. The L2-regularized cost function may be expressed as

$$C_{L2}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} L(Y_i, g(\boldsymbol{x_i}, \boldsymbol{w})) + \frac{\lambda}{2N}\sum_{j=1}^{m} w_j^2 \ .$$

2.6.8.3 Dropout

Dropout is another approach to addressing the problem of overfitting the training data. Its basic principle lies in randomly deactivating neurons along with their connections during each update of the training phase. This reduces the neurons' co-adaptation and makes the network more robust. If we consider a network with n neurons where each of them has a certain probability of being randomly deactivated, then dropout is essentially sampling a thinned network from the collection of all 2^n possible thinned networks. Training a set of thinned networks with extensive weight sharing has been shown to produce optimistic results in terms of improving the generalization of ANNs. [59]

2.6.8.4 Early Stopping

We often encounter a situation where the training loss keeps steadily decreasing over the whole span of the training, but the validation loss first decreases and then starts to return to undesirable values. At that point, the model is likely finding ways to overfit the training data, and it loses its predictive ability on the validation set. **Early stopping** is a technique used to address this issue and determine the point at which the validation loss converges to its minimum. This is achieved by saving the trained weights after each epoch and monitoring a particular evaluation metric, such as the validation loss. If the loss has not been improving for a given number of epochs, early stopping terminates the training and reverts the model to the point where it last achieved its best validation loss. The number of epochs it waits before stopping is often called the **patience**.

Chapter 3

Analysis

In this chapter, we will analyze the current state-of-the-art approaches to acquiring automated diagnoses based on medical images. We will introduce some of the standard methods of preparing data and dealing with medical datasets, as well as the problems that they are burdened with. Finally, we will present some of the existing work in performing COVID-19 detection with CNNs trained on CXR images.

3.1 Medical Imaging

Medical imaging has been an integral part of diagnosing and treating patients for many decades. It is a non-invasive form of getting insight into the structure of the inside of the body. Specifically, CXR imaging can be a very useful diagnostic tool in detecting abnormalities in the chest cavity. It has been successfully applied to diagnosing various conditions such as pneumonia, tuberculosis, cancer, and other pulmonary diseases [7].

As analyzing the chest radiographs may often be tedious work, researchers have been proposing many concepts of computer-aided diagnosis (CAD) systems since the 1960s [60]. The goal was not only to simplify and expedite the process of diagnosis but also expand its reach to areas with a lack of radiological expertise. The CAD systems based their feature extraction on traditional image processing techniques, such as comparing subregions of the images, texture analysis, histograms of gradients, intensity moments, or edge and shape detection [61, 62, 63]. The features would then be handed over as inputs to more traditional machine learning models such as support vector machines or random forest classifiers. These methods generally require a precise image preprocessing pipeline, where several algorithms are stacked on top of each other. Such a design is prone to instability if any of its components fail to deliver decent results. Such a failure is not out of the ordinary, as it is challenging to secure the same conditions in each of the testing facilities, and

the algorithms would often require different settings and threshold values to perform well.

In recent years, new advances made in the field of deep learning and computer vision have allowed us to employ convolutional neural network models to act as automated diagnostic tools that aid medical professionals. One of their main advantages is their lesser need for complex image preprocessing and the fact that they are generally able to localize important features with shift-invariance [56]. Due to extensive weight sharing in the convolutional layers and consequent parameter reduction, CNNs perform these tasks more efficiently than traditional neural networks. A significant drawback is their tendency to overfit and their large training data requirements, which tend to be difficult to fulfil, especially in the medical domain. Nevertheless, their ability to perform unsupervised feature extraction and subsequent classification has been rigorously assessed in tasks such as diabetic retinopathy screening [64], skin lesion classification [65], lymph node metastasis detection [66], and pneumonia detection, where the performance of the proposed model even exceeded practising radiologists [67].

Given that some cases of COVID-19 may manifest themselves as a kind of pneumonia in the infected patients, we are presented with the opportunity of using CNNs for detecting the disease in chest radiographs of the patients' lungs. Performing the diagnosis with this technique could offer the medical professionals a faster and cheaper solution than the molecular RT-PCR alternative [68].

3.2 Preprocessing Methods

As was already mentioned, CNNs generally require a less precise and extensive preprocessing pipeline. Nonetheless, there are certain adjustments that we may experiment with to try and make the process of extracting information from the images more reliable.

3.2.1 Image Resizing

One form of altering the images that must be performed in most practical scenarios is image resizing. The architecture of CNNs does not allow for processing variable sized images; the input shape of the first layer has to be specified and fixated before the model is compiled. The reason for that is that the input shape affects all subsequent layers' width and output size and cannot be changed during training.

Considering that open datasets of medical images rarely come in the same dimensions, it is common practice to resize all images to a fixed width and height, which is specified by the smallest image in the training data. Unfortunately, this results in a loss of precious information and down-sizing the

images too much could end up removing the necessary features needed to discriminate between the classes. A possible solution to this problem could be detecting the main region of interest within the image and cropping the rest of the pixels so that only the useful section remains [69].

A second option for resizing images would be to pad the images to match the dimensions of the largest one. However, we must decide what kind of padding to use and whether artificially adding new information may have a negative impact on the training of the model. Enlarging the images also increases the number of parameters and significantly slows down the training.

3.2.2 Data Transformation

Transforming the pixel values in the images may sometimes lead to faster convergence and better accuracy [70]. One of the commonly used methods for transforming images that form inputs of CNNs is data normalization. This is the process of adjusting the values to a common scale, and one of the most straightforward approaches is **min-max feature normalization** to the interval $[0; 1]$. If we assume that x is a pixel with one dimension per each colour channel, the formula for calculating the normalized value x'_i of its original intensity x_i in the channel i is

$$x'_i = \frac{x_i - \min_{\boldsymbol{x_i}}}{\max_{\boldsymbol{x_i}} - \min_{\boldsymbol{x_i}}} \ ,$$

where $\min_{\boldsymbol{x_i}}$ and $\max_{\boldsymbol{x_i}}$ are the minimal and maximal pixel intensities of the channel i across all pixels $\boldsymbol{x}$ in the training data [22]. We often use the maximum value of 255 and the minimum of 0 on 8-bits-per-channel images, which means that rescaling the images to the desired range can be achieved by dividing each pixel value by 255.

Another form of transforming the data may be **centering**, where we subtract the mean of the specific feature (colour channel) of the training data. This has the effect of centering the brightness around 0 with respect to each dimension. The calculation of the new pixel intensity x'_i in the channel i given that $\bar{\boldsymbol{x}}_{\boldsymbol{i}}$ is the sample mean of the corresponding colour channel across all pixels in the training data is as follows

$$x'_i = x_i - \bar{\boldsymbol{x}}_{\boldsymbol{i}} \ .$$

Standardization extends the image centering further and also divides the values by the standard deviation along each dimension. Standardizing the channels in this fashion ensures a mean of 0 and a variance of 1. Having comparable data with a similar distribution is helpful in cases where we want the learning rate to have proportional influence in all regions of the convolution, where there is extensive weight sharing. This transformation can be denoted as

$$x_i' = \frac{x_i - \bar{\boldsymbol{x}}_{\boldsymbol{i}}}{\boldsymbol{s_i}} \ ,$$

where $\boldsymbol{s_i}$ is a vector of the channel's sample standard deviation across all pixels in the training data. [70, 71]

The modern approach to standardization is incorporating it into the architecture of the network itself. During the training, the distribution of each layer's inputs changes, which slows down the learning process. We refer to this phenomenon as the internal covariate shift, and we counteract it by adding a **batch normalization layer**, which standardizes the outputs of the previous layer with regards to the current mini-batch. Not only does this accelerate the convergence and allow us to use larger learning rates, but it has also been shown to lower the need for using dropout; i.e., it acts as a form of regularization. [72]

Transforming and scaling image values is not always guaranteed to improve the final results, but it is a common step that several preprocessing pipelines implement when working with X-ray images and it has been shown to yield good results [67, 73, 56].

3.2.3 Noise Reduction

Noisy images are a frequent issue throughout all of computer vision, as any type of sensor also detects noise caused by uncontrollable variables. X-ray detectors are no exception; some of the causes of noise, in this case, are the randomness of the quantum characteristics of the X-ray photons, photon collisions, or the false signal caused by re-emission of X-rays absorbed in different parts of the patient's body [74].

Before training CNN models, it is a common practice to denoise the images first. One of the ways this can be accomplished is by convolving the image with a **Gaussian filter**, which has a slight blurring effect. The kernel replaces its central pixel with the weighted average of its surroundings. The weights of the pixels closer to the centre are larger than those farther away. An example of a blurring filter of size 3×3 may be seen in Figure 3.1.

$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$
$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$

Figure 3.1: A 3×3 Gaussian filter used to blurr images when applied by convolution.

The issue with the Gaussian filter is that it also smooths edges and that may be undesirable when detecting abnormalities and disease characteristics in X-ray images. Hence, we can use the so-called **median filter**, which replaces the central pixel with the median of its surroundings. This is especially effective on salt and pepper noise, where random pixels are either saturated to the value of 255 or deactivated on 0. The median filter generally preserves edges, but it may also create new artificial edges, which could add features that resemble image abnormalities, and the model may be trained on incorrect data. [75]

The **bilateral filter** is a method used to smooth the images while also preserving their edges. Similarly to the Gaussian, it is defined as the weighted average of its pixel neighbourhood, but it considers two parameters. These signify the amount of filtering applied to the image; one represents the spatial distance from the central position and the other the difference of the intensity values of the weighted pixel and the central pixel. The rationale of bilateral filtering is that two pixels are close to each other not only if they occupy nearby spatial locations but also if they have similarity in their respective intensities. Pixels that are farther away and have very different intensity values will have a lower weight in the averaging operation. Edges are characterized by having significantly different values than their neighbourhood which means that during bilateral filtering, they are affected much less than by the typical Gaussian blur. [76]

3.2.4 Histogram Equalization

Since features extracted by the convolutional kernels are generally derived from varying pixel intensities within a certain region of the image, having a high contrast can often be beneficial to our cause. **Histogram equalization** is a technique used to enhance the global contrast of an image. A histogram holds the information about the distribution of pixel intensities in a grayscale image. Each intensity value is assigned a number according to the frequency of the occurrence of that value among all the pixels. The algorithm then proceeds to spread out the most frequent values towards the extremes in a way that approximates a uniform distribution, thus enhancing the overall contrast. [78]

X-ray images tend to be acquired through continuous exposure, which requires the exposure to be administered at a low-level, meaning that the resulting contrast is also quite low [79]. It may be difficult to tell apart the denser bone structures from the background, and it ultimately makes the learning process difficult for CNNs. For that reason, many image preprocessing pipelines of classification and segmentation tasks in the X-ray domain use some form of histogram equalization [78, 79, 77].

One significant drawback of this technique is that it also increases the contrast of the noise in the image. As we have mentioned before, noisy images are a common occurrence and highlighting the noise may negatively impact the performance of the model. A popular solution to combat this issue in the med-

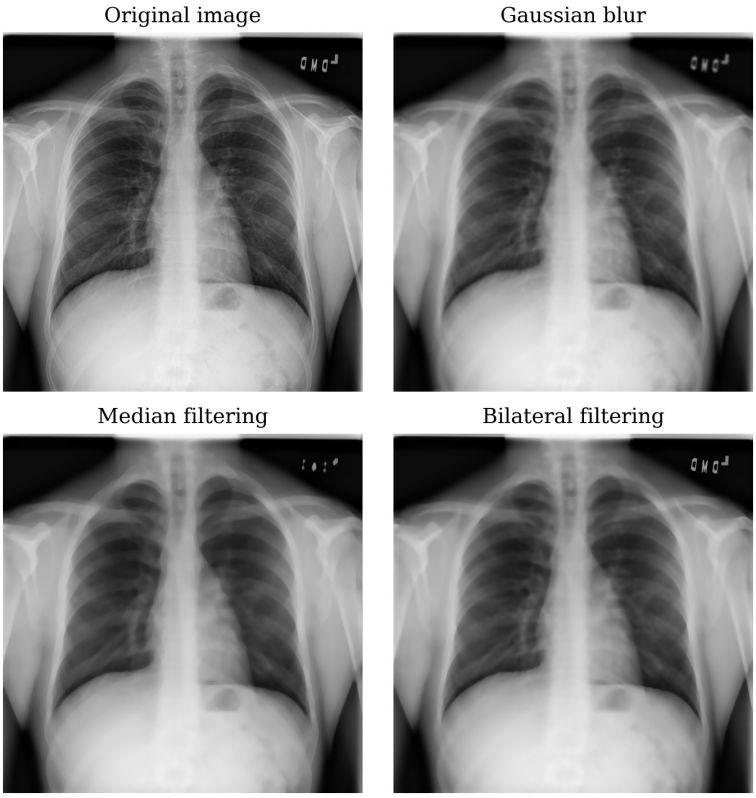


Figure 3.2: Comparison of techniques used to denoise images from the COVIDx dataset [77]. On the upper left is the original CXR image, then to its right follows the image after being blurred with a Gaussian filter, the bottom left is the result after applying the median filter and on the bottom right is the bilateral filtering method. All filters use a 17×17 filer size in order to highlight their effects for the purpose of this demonstration.

ical imaging domain is the **contrast limited adaptive histogram equalization (CLAHE)**. The method is based on dividing the image into non-overlapping regions of almost equal sizes and calculating each individual region's histogram. We also specify a parameter called the clip limit, which limits the number of pixels with the same intensity during their redistribution and thus mitigates the noise amplification. Each region's histogram is then equalized in a way where its height cannot go beyond the clip limit. The equalized value of a pixel in the modified image is finally acquired by linearly combining the results from the mappings of its four nearest regions. The combination of the mappings is based on the distances of the pixel from the centres of those nearest regions. The resulting image frequently contains clearer borders between the objects it depicts, and the sought-after features tend to be more distinguishable. [79]

3.2.5 Image Segmentation

Image segmentation is the process of classifying and partitioning objects captured in images. It works by assigning class labels to each pixel and finding larger areas of pixels that belong to the same class. The goal is typically to help analyze the image or locate certain objects. This can be very useful in medical imaging, where we often want to concentrate on a specific part of the body, such as an organ. When detecting COVID-19, we may find it helpful to locate the lungs within CXR images, as that is the area that will be affected by the disease. Not only will this enable us to crop the image to this region of interest and reduce the dimensionality, but it can also filter out unnecessary parts of the image that would otherwise interfere with and complicate the learning process.

The current state-of-the-art image segmentation pipelines are usually built with CNNs. The architectures for object detection generally include two parts — bounding box proposals and semantic segmentation [80]. The purpose of bounding boxes is to locate the object by enclosing it with a rectangular border, and semantic segmentation defines the specific area and shape of the object by assigning class labels to each pixel. The issue with this deep learning approach is that it requires manually labelling the images by adding bounding boxes and marking the regions of interest, which is a tedious and time-consuming process. That is part of the reason why there is a lack of high-grade datasets for medical segmentation tasks. In spite of that, this area of image preprocessing has given rise to several well-established CNN architectures such as the U-Net [81], or the Mask R-CNN [82].

The alternative approach that was more common in the early days of CAD systems was using traditional rule-based segmentation techniques such as thresholding, edge detection, region growing, and morphological operations [80]. These methods are mostly heuristic and not very robust, which is why they are gradually being replaced in the deep learning era. Nonetheless, they offer interesting insights into image segmentation and are certainly less computationally expensive and data-hungry.

3.2.6 Preprocessing Pipeline for COVID-19 Detection

Since deep learning segmentation is often an expensive task on its own, it is common to use some of the more traditional approaches when preprocessing images for the task of detection with CNNs. In the case of COVID-19 detection, [83] proposes a preprocessing pipeline that smooths and equalizes the images and also uses thresholding to segment the region of interest, i.e. the lungs. The core idea is that the parts of the image with the highest intensity tend to be less informative and could interfere with the regions that hold valuable information. A dominant feature in the CXR images that the method detects and removes is the diaphragm.

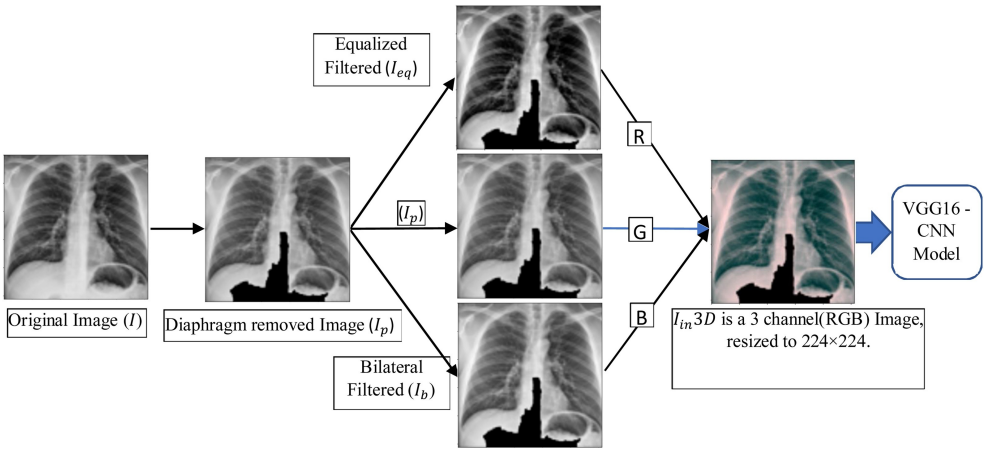


Figure 3.3: A flow diagram that illustrates a suggested image preprocessing pipeline for COVID-19 detection. The original image is segmented in a way that removes the high-intensity diaphragm region, a copy of it is denoised with the bilateral filter, and another copy's contrast is adjusted with histogram equalization. The three images are then combined to form a final pseudocolour image that can be used as an input to CNN models [83].

The algorithm starts by converting the RGB image into a single-channel grayscale version. Then it finds the highest pixel intensity value in the image V_{max} and the lowest value V_{min}, and uses them to calculate a threshold value $T = V_{min} + 0.9 \times (V_{max} - V_{min})$. The threshold is then used to perform binary segmentation, which transforms the data so that any pixel values under the threshold are clipped to value 0, and any values equal to or greater than the threshold are saturated to 255. This marks several high-intensity regions, and we must then select the largest one, which should correspond to the diaphragm, fill in the holes in the binary mask and delete the other smaller regions. The authors also suggest using morphological operators to smooth the boundaries of the marked region. The detected diaphragm is then mapped to the original image, and we remove the overlapping pixels in the corresponding location on the original image. At this point, we are left with the segmented image without its largest region of high intensity, which should hopefully accomplish the removal of the diaphragm. In the following step, the algorithm passes a copy of the segmented image through a bilateral filter, which acts as a denoising mechanism that manages to preserve textural information. The original segmented image is copied once more, and histogram equalization is applied to compensate for the differences in image contrast and brightness caused by the variations in X-ray exposure. The equalization enhances lung tissue patterns that may be associated with the COVID-19 infection.

The final step of the algorithm takes the three modified images (the segmented, denoised, and equalized) and merges them into a three-channel pseudocolour image, which is ready to be fed into the training algorithm for the CNN. The process is demonstrated in Figure 3.3. [83]

3.2.7 Dimensionality Reduction

Many machine learning models are inflicted with a phenomenon called the curse of dimensionality. It refers to a set of problems that often arise when dealing with data in high-dimensional spaces. The volume of such spaces is typically very large, which makes the data points sparsely laid out. In such cases, the training data could be a non-representative sample of the whole space, making it more difficult to understand or capture its distribution. To combat this issue, there are several techniques of reducing the number of features that characterize the input data. We call this process the **dimensionality reduction**, and it aims to simplify the feature space while retaining the key properties of the original data.

A recently proposed approach to dimensionality reduction is the **Uniform Manifold Approximation and Projection (UMAP)**. In its core, UMAP constructs a high-dimensional representation of the data and then optimizes a lower-dimensional representation to be as structurally similar as possible. A key parameter used by the algorithm is the number of neighbours, which is the number of nearby data points used to capture local structure within different parts of the space. Using a lower neighbour count prioritizes local structures and finer details, while larger counts focus more on the global structure of the data. Another important metric is the minimal distance, which limits how closely packed together the projected points in the lower-dimensional representation may be. Lower values result in the formation of clusters, which may be useful in analyzing closely related data points. The measure of distance is also a parameter that may be specified; a common metric is the Euclidean distance. [84]

The UMAP is not only useful for preprocessing data for machine learning algorithms but also for projecting the data to lower-dimensional spaces, which are easier to visualize and subsequently analyze. Another popular dimensionality reduction technique used for visualizations is the t-Distributed Stochastic Neighbour Embedding (t-SNE). However, the UMAP has been found to achieve superior run time performance and preservation of the global structure of the original feature space [84].

3.3 Imbalanced Datasets

Naturally, medical datasets tend to suffer from class imbalance. The publicly available COVID-19 datasets are no exception, as the number of positive images is usually heavily outweighed by those that are negative or show symptoms of other pulmonary diseases [85]. Another form of imbalance is in the nature of the disease manifestation itself. The features that signify the presence of COVID-19 only cover a very small fraction of the overall image, so the semantic interpretation of the pixel labels within a positive image is still underrepresented. These factors can cause serious problems when training CNNs, so it is good practice to try to address the issue.

3.3.1 Cost Sensitive Learning

When training the classifier, we may employ the use of cost sensitive learning to adjust the importance of each class. This method assigns different costs to the misclassification of samples from different classes, which weakens the impact of class imbalance [86]. We can achieve this during training by modifying the loss function with different error penalties for each class, or more commonly, by introducing **class weights** that are inversely proportional to the prevalence of the classes in the training data [86]. The calculation of the weight w_i of class i is commonly defined as

$$w_i = \frac{N}{c \times n_i} \ ,$$

where N is the total number of samples in the training data, c is the number of distinct classes, and n_i is the number of samples belonging to class i [87].

In addition to counteracting an imbalanced dataset, the usage of class weights may also have a deeper meaning, especially in the context of medical imaging and the detection of COVID-19. Higher penalization of the misclassification of COVID-19 positive images is analogous to the concept that it is worse not to detect the disease in a patient that has it, rather than to wrongly classify a healthy patient as ill. Admittedly, if a medical professional tells a healthy patient that they have COVID-19, it may cause them unnecessary stress and other problems, but it is undoubtedly worse to leave the disease undetected and therefore untreated in patients that truly have it.

3.3.2 Undersampling

One data level solution to balancing the occurrence of the class labels in the training data itself is **undersampling**. Samples from the majority class are randomly removed until both classes have the same number of individual images. The major disadvantage of this approach is that it inevitably discards a portion of the available data, which is usually not a recommended step

in data-hungry methods such as deep learning. A somewhat more tolerable alternative is being more deliberate with the removal of the majority class and attempting to remove images that are less valuable in some sense. These can be noisy images or images close to the boundary between the classes, where it is difficult to classify them with certainty. [88]

3.3.3 Oversampling

Another technique that takes the opposite approach to solving class imbalance is **oversampling**. Its most basic form is based on randomly sampling examples from the minority class and re-adding them into the training data. This ensures that the minority class is equally represented and the network is given a higher chance of learning the necessary features. Oversampling has been shown to be effective; however, since there is severe duplication in the data, it is quite susceptible to overfitting. [88, 85]

3.3.4 Data Augmentation and Synthetic Data Generation

In hopes of reducing the likelihood of overfitting, we may perform data augmentation as described in Section 2.6.8.1. One way of doing so is performing oversampling where the re-added images are not exact duplicates but rather augmented versions of images already present in the training data. Not only does this balance the classes, but it also introduces more variation in the training data, meaning that the model has a better chance to learn to generalize. Another approach could be to perform online data augmentation during the training. In this case, the generated mini-batches have a certain chance of being augmented before they are fed into the network, which reduces the risk of the weights being adjusted exactly according to specific inputs. When dealing with medical images, it is recommended that simple data augmentation techniques such as flipping or rotation are applied so as not to damage or remove the information, which is often quite subtle [86]. While data augmentation does provide more diversity, it is not able to introduce new features that are not already present in the training set.

3.3.4.1 Generative Adversarial Networks

In machine learning, we differentiate between two basic types of models: discriminative and generative. The discriminative models' goal is to learn the boundaries between the classes in the data, while the generative models attempt to capture the actual distribution of the data in an unsupervised manner [89]. Knowing the distribution can be very useful, as we can sample from the distribution, thereby creating a synthetic data point. In tasks concerning medical imaging, we could generate synthetic images of the underrepresented class and use them to balance the data and mitigate overfitting without the need for gathering new real-life samples.

In 2014, Ian J. Goodfellow and his colleagues introduced the **generative adversarial network (GAN)** architecture for generating synthetic data [89]. The idea behind it has roots in game theory, where the training corresponds to a two-player min-max game between two networks — the discriminator D and the generator G. The goal of the generator is to approximate the distribution of the training data p_{data} and output artificial data points from that distribution, while the discriminator learns to distinguish between real data from the training set and synthetic data generated by G. The generator never sees the actual training data; its inputs are samples $\boldsymbol{z}$ from a noise distribution $p_{\boldsymbol{z}}$. Considering the notation above and that $D(\boldsymbol{x})$ is interpreted as the discriminator's estimate of the probability that the real sample $\boldsymbol{x}$ is, in fact, real and $D(G(\boldsymbol{z}))$ is its estimate of the probability that a synthetic sample is real, Goodfellow defines the min-max GAN loss function as

$$L(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log(1 - D(G(\boldsymbol{z})))\right] ,$$

where $\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}$ is the expected value over all real data instances $\boldsymbol{x}$ following the training data distribution, and $\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}$ is the expected value over all random inputs $\boldsymbol{z}$ to the generator, which follow the noise distribution. The learning process, or the game, is defined by the generator's effort to minimize the loss function while the discriminator is attempting to maximize it. We can express this as

$$\min_G \max_D L(D,G) ,$$

which represents the generator's tendency to trick the discriminator and increase the probability of its estimates that a generated sample is real, while the discriminator tries to lower this probability and simultaneously maximize the probability of recognizing actual real samples. [89]

The original architecture of GANs uses fully-connected layers, but with the evolution of CNNs, some of the layers were replaced by convolutional layers and such design is referred to as the **deep convolutional generative adversarial network (DCGAN)** [90]. GANs remain a very active area of research, where improvements are still to be made. Learning of the models may sometimes be quite unstable, and the generator and the discriminator have to be well synchronized in order to find balance and converge [90]. Nonetheless, their applications can be found all throughout machine learning, and balancing medical datasets is an increasingly popular example of their use [91, 86].

3.4 Transfer Learning

We may combat the lack of reliably labelled medical data by leveraging the fact that the generic features learned in the first convolutional layers of CNNs can be shared among many seemingly disparate datasets [56]. **Transfer learning** is a common strategy that utilizes this by pre-training a network on an extremely large dataset and then optimizing the weights further on the smaller dataset of the problem at hand.

One way of performing transfer learning is to use the pre-trained model as a fixed feature extractor and build the classifier on top of it. In order to do that, we must freeze the layers of the convolutional base by setting them not to be trainable and using their output as the input to a trainable classifier made of fully-connected dense layers that lead to the output layer. When the term transfer learning is used, it generally refers to this approach. It makes most sense if our dataset belongs to the same domain as the original dataset that the feature extractor was built on, but this is not a very common scenario in medical deep learning [56].

The alternative approach is called **fine-tuning**, and it involves pre-training the model on the source dataset and then further tuning all of the weights on the target dataset. Naturally, optimizing all the weights, including those in the convolutional layers, means that the parameter space is much larger and the training is more computationally expensive [92]. This can be somewhat alleviated by freezing only the first few layers, which usually extract very generic features and continue training from a certain depth, where the features become progressively more domain-specific. In radiology, fine-tuning tends to produce better results than transfer learning [56].

The following sections describe some of the popular source datasets used in transfer learning as well as several specific CNN model architectures that are pre-trained on them. Some of these models will be further experimented with in the latter part of the thesis.

3.4.1 ImageNet

ImageNet [93] is a large-scale image database that has played a key role in advancing modern computer vision and deep learning research. It contains over 14 million hand-annotated images and over one million images with marked bounding boxes. The class categories are based on the WordNet database[2] of English words linked by semantic relationships. Similar words are grouped together into a synonym set called a synset. ImageNet currently covers over 20 000 non-empty synsets, each with 500-1 000 images. [94]

Between 2010 and 2017, an annual computer vision contest was held to test the current state-of-the-art solutions to image classification, object localization, and object detection. The contest was the ImageNet Large Scale

[2] `https://wordnet.princeton.edu/`

Visual Recognition Challenge (ILSVRC), and the training and testing were done with the ImageNet database. The evaluation on the ImageNet test set is usually in the form of top-1 and top-5 error rates, where **top-1** examines whether the highest-confidence output label matches the ground truth class and **top-5** examines whether the ground truth is included in the output labels with the 5 highest confidence levels [94]. The challenge has given rise to many advancements in deep learning with CNNs, and the ImageNet is one of the most common source datasets used in transfer learning today. Many publicly available CNN architectures have the option of being initialized with the pre-trained ImageNet weights, which may help the feature extraction process in cases where the target dataset is too small, as is the common case with medical data.

3.4.2 ChestX-ray

Transferring ImageNet weights to the task of COVID-19 detection may not prove to be very useful, as the two image domains are very distinct. Consequently, some of the existing works in COVID-19 detection have used the **ChestX-ray** dataset [95] as the source where they pre-train their models [96, 97, 98]. The dataset was created specifically to fulfil the purpose of providing a substantial amount of data for training fully automated deep learning CAD systems. The focus is not only on the lungs but also the whole chest cavity that may contain a wide range of thoracic diseases. The latest version of the dataset ChestX-ray14 is made up of over 100 000 X-ray images containing 14 diseases (COVID-19 not included), making it the largest publicly available chest X-ray dataset to date [67].

3.4.3 AlexNet

AlexNet is the first CNN model which has had tremendous success on the ImageNet dataset. Its lead author Alex Krizhevsky entered the network to the ILSVRC-2012 and won first place, outperforming the previous state-of-the-art by a significant amount. On the test data, AlexNet achieved a top-5 error rate of 15.3 %. The architecture comprises of 5 convolutional layers (with kernel size of 11×11), some of which are followed by max-pooling layers, and 3 fully-connected layers that act as the classifier with an output width of 1 000 neurons, which are activated by the softmax function. The design pattern of stacking convolutional and pooling layers was inspired by the LeNet series of convolutional networks that were proposed in the late '90s [99]. Krizhevsky managed to make the network deeper than its predecessors by making use of an efficient GPU implementation of the convolution operation in order to be able to train the 60 million parameters of 650 000 neurons with relative speed. To add non-linearity and prevent vanishing gradients, the network also uses

the ReLU activation function in the hidden layers, and the concept of dropout was applied to act as a form of regularization. [100]

3.4.4 VGG

The VGG is a CNN architecture which improved upon AlexNet by going deeper and using 13 convolutional layers with a much smaller kernel size of 3×3. Along with the 3 fully-connected classifier layers, the depth raised the number of VGG's parameters to around 138 million, giving the network a much larger capacity than AlexNet. This version of the model is called the **VGG16**, and it also has a 19-layer implementation called the **VGG19**. In the ILSVRC-2014, the VGG was the runner-up in the classification task with a top-5 error rate of 7.3 %, only behind GoogLeNet with 6.7 %. [101]

Both of the VGG designs are commonly used to perform automated classification and detection tasks in modern computer vision, and VGG19 has recently found use in COVID-19 detection as well [77]. One of the main disadvantages is that the large number of parameters takes much longer to train and the network itself takes up a lot of memory.

3.4.5 ResNet

Although stacking convolutional layers led to a rapid increase of performance in the evolution of CNNs, there is a point where the deep network starts to converge, and by adding more layers, we saturate the accuracy and the performance starts to degrade. In 2015, the Microsoft Research team came up with a method of building extremely deep models without diminishing their predictive potential. They suggested using residual connections between layers, which essentially accomplish that outputs of one layer skip ahead and are added to outputs of a layer that is deeper down the line. By adding residual connections, features learned in the shallower part of the network are identically mapped to a deeper part which ensures that the deeper model should not produce a higher training error than its shallow counterpart. Not only does this resolve the training accuracy degradation, but it also mitigates the vanishing gradients problem and generally makes the networks easier to optimize and achieve higher accuracies from considerable depths. Aside from popularizing residual connections, the ResNet series was also among the first to use batch normalization. [102]

Some of the more prominent models are the **ResNet-50** or the extremely deep 152-layer **ResNet-152**, which still manages to have only around 60 million parameters, less than half of the VGG architectures [103]. Microsoft Research entered an ensemble model of several ResNet networks into the ILSVRC-2015 and ended up having a 3.57 % top-5 error rate, securing the first place and even outperforming the average human recognition ability with an error rate of 5 % [104].

3.4.6 Inception

We have mentioned that the winner of the classification task of ILSVRC-2014 was the GoogLeNet, which is also referred to as **Inception-v1** and was developed by Google Inc. It is a 27-layer deep architecture (including pooling layers without trainable weights) with just 5 million parameters, which is the result of improved utilization of the computing resources inside the network. The design uses a so-called 'Network-in-Network' principle, where one layer is replaced by a building block defined as a miniature network of several convolutional and pooling layers. These blocks, or Inception modules, use pointwise convolution (1×1 kernel size) to compress the pixel channel depth without altering the feature map dimensions. This is useful for reducing dimensionality before the more costly 3×3 and 5×5 convolutions are computed. Aside from reducing computational bottlenecks, the authors also introduced parallel branches which use differently sized convolution kernels and are then concatenated back together. Parallel branching helps with collecting a wider variety of features that are then compressed by global average pooling. [105]

Over the years, several adaptations of the architecture design were made, with the most notable addition to the Inception series being the **Xception** model proposed in 2016. The authors took the Inception module to the extreme by replacing the traditional convolution operation with a combination of depthwise convolution and pointwise convolution, which has the effect of lowering the number of multiplication operations and making the training process more efficient [106]. The Xception architecture has 36 convolutional layers structured into 14 modules, which all use linear residual connections except for the first and the last [106]. This CNN was never entered into an ILSVRC, but with a top-5 error rate of 5.5 %, it outperforms its predecessor Inception-v3, which was the runner-up in ILSVRC-2015 [104].

3.4.7 DenseNet

DenseNets are a category of CNNs which take maximum advantage of the residual connection design pattern. They are comprised of Dense Block modules, which directly connect all layers with matching feature map dimensions. The inputs of each layer in the Dense Block are made up of all the preceding layers' feature maps, and the output is fed forward into all subsequent layers. The layers between two adjacent Dense Blocks are referred to as transition layers, and their purpose is to alter feature map dimensions via convolution and pooling. This dense connectivity has several compelling advantages, such as the fact that it improves information flow between layers and preserves all features in the last layer of the module. The design also alleviates the vanishing gradient problem, strengthens feature propagation and encourages feature reuse by concatenating feature maps learned by a stack of layers instead of adding them together as the ResNets do. Due to the large input

connectivity, the convolutional layers may be very narrow (e.g., 12 filters per layer), which means that they add only a small set of feature maps to the collective knowledge of the network and require much fewer parameters than other architectures, making their training more efficient. [107]

A popular model from this category is the **DenseNet-121**, which has 4 Dense Blocks interconnected by 3 transitional layers, totalling a depth of 121 layers with roughly 8 million trainable parameters [107, 103]. The authors tested this architecture on the ImageNet validation set and achieved a top-5 error rate of 6.66 % [107].

A significant advantage of this network for our purposes is that aside from ImageNet, it has also been pre-trained on the ChestX-ray dataset. This pre-trained model is dubbed the **CheXNet** and has achieved excellent results, going as far as to outperform practising radiologists in detecting pulmonary diseases from CXR images [67].

3.5 Research in COVID-19 Detection

Considering the ongoing pandemic as of writing this thesis, it is safe to say that the automated detection of COVID-19 is still a rapidly developing area of research. This section will describe some of the work that has already been done to combat this disease and provide alternatives to the typical medical diagnoses. We focus on analyzing the novelty approaches to either data pre-processing or architecture design and training, but there are numerous other works that demonstrate the use of well-established techniques in deep learning and computer vision and their application in the detection of COVID-19 [68, 73, 85, 91, 97, 108].

3.5.1 COVID-Net

One of the most prominent research efforts in this area has been the COVID-Net project [77] conducted by researchers at the University of Waterloo. Since the start of the pandemic in March 2020, they have been working on setting up a comprehensive benchmark dataset of CXR images of COVID-19 and pneumonia and building deep learning classifiers to perform COVID-19 detection on it. The dataset has undergone several updates, and its current version named the **COVIDx8** was published at the end of March 2021. The dataset is a composition of CXR images from several publicly available sources, and it has two versions — one for binary detection of COVID-19 positive and negative images and one for categorical classification of regular CXRs, COVID-19, and non-COVID-19 pneumonia. The data is split into a training set and a test set in order to provide a benchmark for comparison with other architectures. To the best of the authors' knowledge, the dataset has maintained the largest number of publicly available COVID-19 positive samples throughout its version history.

The creators of COVIDx have used the data to train several models over the past year, and they refer to the collection of networks as COVID-Net. Their most recent model is called the **COVID-Net CXR-2**, which was released along with the current dataset version COVIDx8 and performs binary classification. As of writing this, the model lacks extensive documentation. However, based on a published illustration seen in Figure 3.4, we can deduce that the model is 88 layers deep and accepts 480×480 px RGB images as input. The architecture uses some common concepts found in modern CNNs, such as the use of modules as building blocks and residual connections to support feature reuse. According to the authors' report, the COVID-Net CXR-2 contains 8.8 million parameters, and it achieved a binary accuracy of 96.30 % and COVID-19 sensitivity of 95.50 % on the COVIDx8 test set [109].

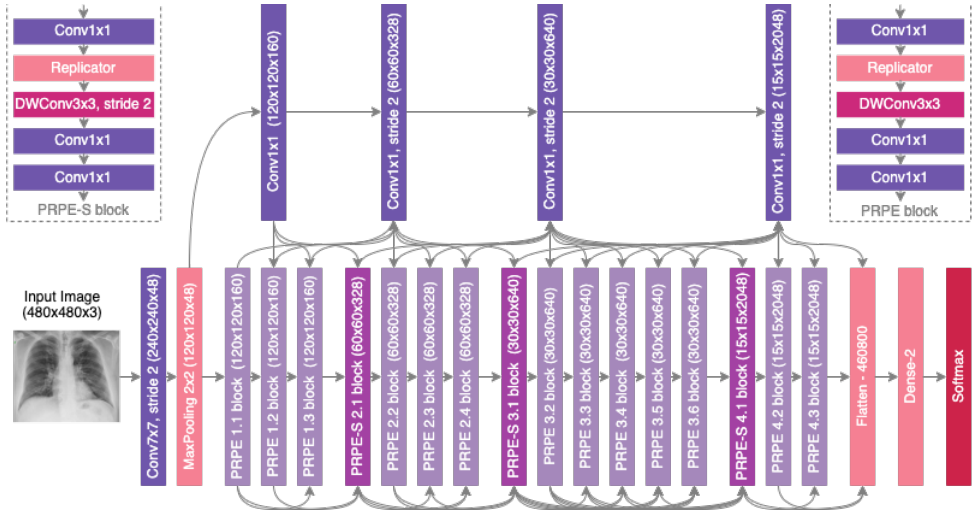


Figure 3.4: Summary of the COVID-Net CXR-2 architecture for binary classification of COVID-19 CXR images [109].

Another significant contribution is the **COVID-Net CXR3-B**. It came from a series of models built on the older COVIDx3 dataset and was primarily built for categorical classification of normal images, COVID-19 images and images of non-COVID-19 pneumonia. The model is not explicitly named in the accompanying publication, but the release timeline and reported evaluation results indicate that the COVID-Net design described by [77] was then implemented under the alias CXR3-B. The architecture was created with the strategy of human-machine collaborative design. The process combines the human-driven principles and best practices in network design with the machine-driven exploration strategy based on adjusting the macro-architecture and micro-architecture of the network in order to fit the target domain [77]. This was accomplished with the use of generative synthesis of deep neural networks, where a generator-inquisitor pair works in tandem to

generate a network prototype to satisfy given criteria. The goal of generative synthesis is to learn a generator that, given a set of seeds, can generate a parametrized deep neural network that maximizes a universal performance function, which is evaluated by the inquistior [110]. Such optimization enables the network to flexibly explore various configurations at different levels of granularity while still maintaining well-established techniques that ensure efficiency and high performance [77]. Given the similarity of the two networks, we assume that COVID-Net CXR-2 was also built by generative synthesis.

The generated COVID-Net CXR3-B can be seen in Figure 3.5; it is 87 layers deep and has 11.7 million parameters. The authors proposed an interpretation of the architecture where the network makes heavy use of lightweight residual projection-expansion-projection-extension (PEPX) design pattern, which consists of the following phases:

- **First-stage Projection:** 1×1 convolutions for projecting input channels into an output tensor with lower dimensionality,
- **Expansion:** 1×1 convolutions for expanding features to a higher dimensionality,
- **Depth-wise Representation:** 3×3 depth-wise convolutions with different filters for each expanded channel, which learns spatial characteristics to minimize computational complexity,
- **Second-stage Projection:** 1×1 convolutions for projecting the features into a lower dimension again, and
- **Extension:** 1×1 convolutions for extending the channel dimensionality and producing final features of the module.

The uniquely generated PEPX design is accompanied by the use of long-range residual connectivity, which improves the representational capacity of the network, but also increases the computational complexity and memory overhead. Consequently, the connectivity has been selectively generated only to certain layers within the network that act as hubs for optimal feature transfer. [77]

The COVID-Net CXR3-B was first pre-trained on the ImageNet dataset and then trained for 22 epochs on COVIDx in batches of 64 images. The Adam optimizer was used with a learning rate of 2×10^{-4}, which decreased when the learning stagnated for 5 epochs. During training, online data augmentation in the form of translation, rotation, horizontal flips, zoom, and intensity shifts was used and batches were re-balanced to maintain distribution of each class type at batch level. The final model was evaluated on the test set and achieved an accuracy of 93.30 % and COVID-19 sensitivity of 91.00 %. [77]

The authors of COVID-Net have published a wide range of other models, some of which detect COVID-19 from CT scans [111], while others assess the severity of the progression of the disease based on its geographic and opacity

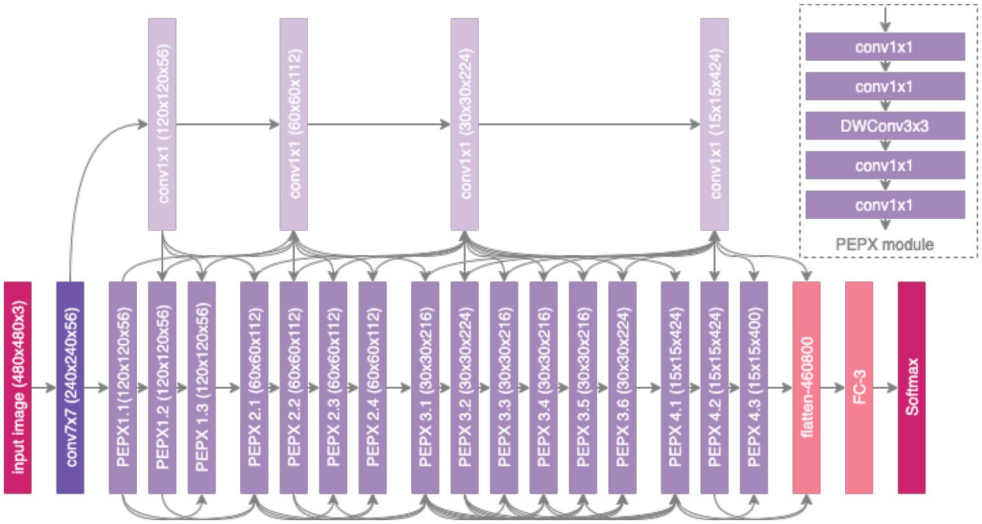


Figure 3.5: Summary of the COVID-Net CXR3-B architecture for categorical classification of pulmonary diseases including COVID-19 in CXR images [77].

extent within the CXR images [112]. All of the aforementioned models, as well as a guide on generating the COVIDx dataset, have been published through a publicly available GitHub repository[3].

3.5.2 Application of VGG16 and Image Preprocessing for COVID-19 Detection

The preprocessing pipeline described in Section 3.2.6 was proposed by [83] and further experimented with on a publicly available dataset of 8 474 posterior-anterior view CXR images. The dataset was created for categorical classification and contains 415 images depicting COVID-19 cases, 6 179 non-COVID-19 pneumonia cases and 2 880 normal healthy radiographs. The architecture used was the VGG16 pre-trained on the ImageNet dataset and then further fine-tuned on the target data. The fine-tuning was done in 200 epochs with a batch size of 4 and the use of Adam optimizer with an initial learning rate of 10^{-5}, which was subjected to decay every 5 epochs by a factor of 0.8. In order to increase the training set, common data augmentation techniques such as shearing factors, intensity adjustments, zooming, translation and rotation were used. Further class balancing was achieved with the use of class weights and cost sensitive learning. The results presented in Table 3.1 indicate that the absence of data augmentation causes the model's ability to generalize on the test set to drop significantly to around 82.00 %. Feeding the network with the original unpreprocessed CXR images from a dataset expanded through data augmentation yields significantly better results; however, by using de-

[3] `https://github.com/lindawangg/COVID-Net`

noising techniques and equalizing the image contrast, the authors improved the accuracy even further by roughly 3 %. Furthermore, using the complete preprocessing pipeline and removing the high-intensity region containing the diaphragm increased the accuracy to 94.50 %, as well as having a very respectable COVID-19 sensitivity and specificity of over 98.00 %. [83]

Table 3.1: Comparison of the results of preprocessing techniques used in COVID-19 detection from CXR images by the VGG16 architecture in [83]. The *Proposed model* is the VGG16 with the complete preprocessing pipeline, *Filter-based model* keeps the bilateral filtering and histogram equalization but not the removal of the diaphragm region, *Simple model* contains no image preprocessing, and the *No-augmentation model* is a model trained on a set of images where no augmentation was applied at all. The listed metrics either came directly from the authors' report or were calculated based on the confusion matrices published in [83].

Model configuration	Accuracy	TPR (Sensitivity)	TNR (Specificity)
Proposed model	**94.50** %	**98.41** %	**98.06** %
Filter-based model	91.20 %	93.65 %	97.39 %
Simple model	88.00 %	84.92 %	96.24 %
No-augmentation model	82.30 %	75.40 %	94.73 %

3.5.3 Twice Transfer Learning for COVID-19 Detection

Another relatively recent publication that makes some interesting advances in this field was released in January 2021 by a team of researchers from the University of Campinas, Brazil [98]. They presented a COVID-19 classifier based on the DenseNet architectures and used a process called the twice transfer learning to combat the small size of their training data. Inspired by the CheXNet (see Section 3.4.7) architecture, which was a DenseNet-121 trained on the ChestX-ray14 dataset (see Section 3.4.2), they created a deeper densely connected CNN called the DenseNet-201. This model was initially pre-trained on the ImageNet dataset and then transferred to the CXR domain of ChestX-ray14, fine-tuned, and then finally transferred again to the target COVID-19 data. The hypothesis was that expanding the sets of training data would reduce the curse of dimensionality tied to the sparsity of features within their 224×224 px input images. ImageNet was supposed to teach the network to extract low-level generic features, and the ChestX-ray14 would focus on adapting that feature extraction to chest radiographs. The COVID-19 data, which was used to fine-tune the final version of the model, was comprised of 439 COVID-19 posterior-anterior view images, 1 255 pneumonia images and

370 healthy images. Samples augmented through rotations, translations and horizontal flipping were used for balancing both the training set and the validation set. The balanced data then contained 8 280 COVID-19 images, 8 640 pneumonia images and 8 640 normal images. [98]

In addition to twice transfer learning, the authors experimented with output neuron keeping. Having two similar datasets like the ChestX-ray14 and their COVID-19 collection, they were presented with an opportunity of retaining a common part of the network trained on more extensive and reliable data. Due to the fact that the datasets share the classes of normal images and pneumonia, they kept their respective original output neurons and combined the rest of the CheXNet outputs into one, which was trainable for assessing the probability of COVID-19 [98].

The authors trained 5 CNNs with different combinations of architecture design, training specifics and output neuron keeping. The progression of the individual training phases can be seen in Figure 3.6. There were 3 top-performing configurations, which all managed to predict the test data with 100 % accuracy. One of the top models was the DenseNet-201 trained with twice transfer learning and output neuron keeping, and the other two models were both CheXNets with and without output neuron keeping. The remaining two networks were the deeper DenseNet-201 models with single transfer learning and with twice transfer learning, both without output neuron keeping. While the latter 2 networks were only behind by around 2 % on the accuracy, the authors concluded that the deeper architecture faced more issues with overfitting than the simpler CheXNet, and output neuron keeping has proved to be a useful technique for utilizing previous learning stages. [98]

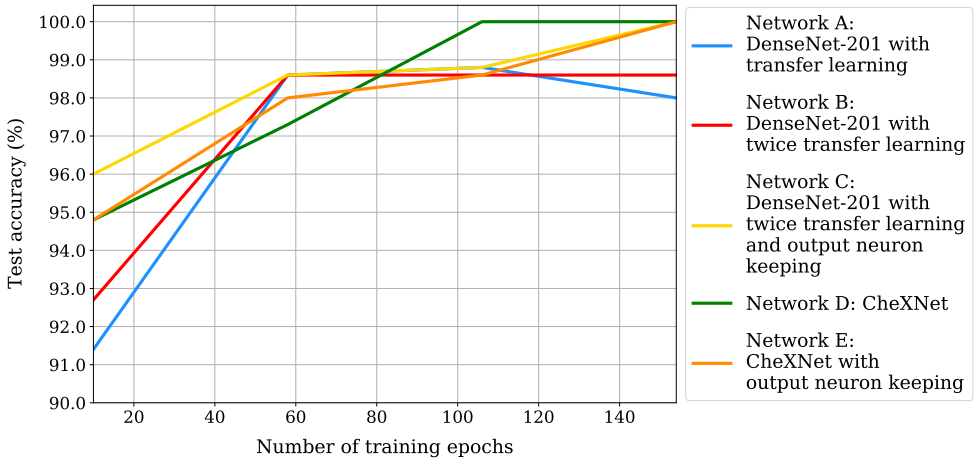


Figure 3.6: Comparison of test accuracy of DenseNet architectures trained with various configurations of transfer learning and output neuron keeping [98].

Chapter 4

Design and Implementation

This chapter describes the fundamental design and implementation specifics that cover the scope of our experimentation.

4.1 Requirements and Technologies

Machine learning, in general, is a very algorithmic field and the pipeline of training CNNs involves many complex steps that cover dataset preparation, image preprocessing, model building, training, and evaluation. Fortunately, many of these steps are implemented in modern-day programming languages and their respective libraries and frameworks, enabling us to build on top of previous work and speed up the development process. This section lists the most relevant technologies that were used throughout our experimentation.

4.1.1 Python

The primary programming language of our choice was Python[4], specifically the version Python 3.7. It is an interpreted, high-level, general-purpose programming language that is one of the most popular languages in machine learning and data science [113]. Due to its large support community, it has many libraries and tools that make the development process more efficient and reliable. It also has good readability and is supported by several interactive computing environments, making the language user-friendly and enabling developers to keep the codebase clean and well-organized.

[4] `https://python.org`

4.1.2 NumPy

A large portion of our numerical calculations and data transformations were done with the use of the open-source library NumPy[5]. With its efficient implementation of multi-dimensional array and matrix operations, it is very useful for applications dealing with image data.

4.1.3 Scikit-learn

Scikit-learn[6] is an open-source Python library that provides implementations of many data analysis tools and machine learning models. We have used some of the metrics offered by the library to evaluate our models and also utilized its infrastructure for preparing datasets for cross-validation.

4.1.4 OpenCV

Our proposed image preprocessing methods were realized with the open-source library OpenCV[7]. It provides a Python interface for optimized computer vision tools and algorithms written in C/C++.

4.1.5 Matplotlib and Seaborn

In order to plot graphs or create visualizations of our data and training and evaluation of individual models, we used the plotting libraries Matplotlib[8] and Seaborn[9]. Analyzing visual representations of the training process may give us more insight into the quality of the models and improve the interpretability of their results.

4.1.6 TensorFlow and Keras

TensorFlow[10] is an open-source framework developed at Google for machine learning applications. Its main focus is on defining the architecture and training of deep neural networks. It is highly optimized for the execution of low-level tensor operations on CPU, GPU, or TPU. Most of our work was done using the current TensorFlow version 2.4, but we downgraded to TensorFlow 1.15 when testing the COVID-Net architecture [77] for compatibility purposes.

Keras[11] is a high-level API that acts as an interface for the TensorFlow framework. It enables faster prototyping of ANNs by providing abstractions

[5] https://numpy.org
[6] https://scikit-learn.org
[7] https://opencv.org
[8] https://matplotlib.org
[9] https://seaborn.pydata.org
[10] https://tensorflow.org
[11] https://keras.io

and building blocks for developing the models. It also provides the implementation of several popular CNN architectures along with their weights which have been pre-trained on the ImageNet dataset, making transfer learning more accessible. The simplest way of defining Keras models is by using the Sequential model API, which is essentially a linear stack of defined layers. The alternative is adopting the Keras functional API, which allows for building arbitrary graphs of layers with multiple inputs and outputs or using residual skipping connections [103]. Most of our models have been implemented with the functional API.

4.1.7 Jupyter Notebook and Google Colab

The Jupyter Notebook[12] is an open-source web application that enables developers to create and share documents with live code, markdown text and visualizations. The document is split up into cells which can be run in any order, making the prototyping and visualizations very interactive and efficient. A small subset of our experiments was conducted in a Jupyter Notebook running inside a Python virtual environment on a *Lenovo ThinkPad X1 Extreme (Gen2)* with an *Intel Core i7-9750H* processor, 16 GB DDR4 RAM and an *NVIDIA GeForce GTX 1650* graphics card with 4 GB of VRAM.

In cases where the model or the training batch size was too large to be handled by the physical machine, we utilized the Google Colab[13] platform. It is a cloud-based interactive computing environment that provides a very similar document format to the Jupyter Notebooks along with access to a virtual machine for executing the code cells. Most of our experiments were performed on the Colab virtual machine with GPU hardware acceleration on an *NVIDIA Tesla T4* graphics card. The allocated resources of the free version vary depending on the usage statistics of the given account, but we were usually assigned roughly 16 GB of VRAM.

4.2 Dataset

The dataset we chose to work with is the COVIDx which originates from the COVID-Net project (see Section 3.5.1). It is the largest publicly available dataset for COVID-19 detection to our knowledge and it aggregates data from 6 different open acess data repositories:

- COVID-19 Image Data Collection [114],
- Figure 1 COVID-19 Chest X-ray Dataset Initiative [115],
- ActualMed COVID-19 Chest X-ray Dataset Initiative [116],

[12] `https://jupyter.org`
[13] `https://colab.research.google.com`

- RSNA Pneumonia Detection Challenge dataset [117],
- COVID-19 radiography database [118],
- RSNA International COVID-19 Open Radiology Database (RICORD) [119].

As the task at hand is currently a very topical subject, the project keeps evolving, and new versions of the dataset are regularly being released. Each version adds new COVID-19-positive cases to the data, which improves the potential of capturing the critical features that discriminate between the classes. During our experimentation, the dataset has undergone several version updates, with the newest one being the COVIDx8. The majority of our work is done with this version, but some tests which will be described at a later point have used some of the older versions as well.

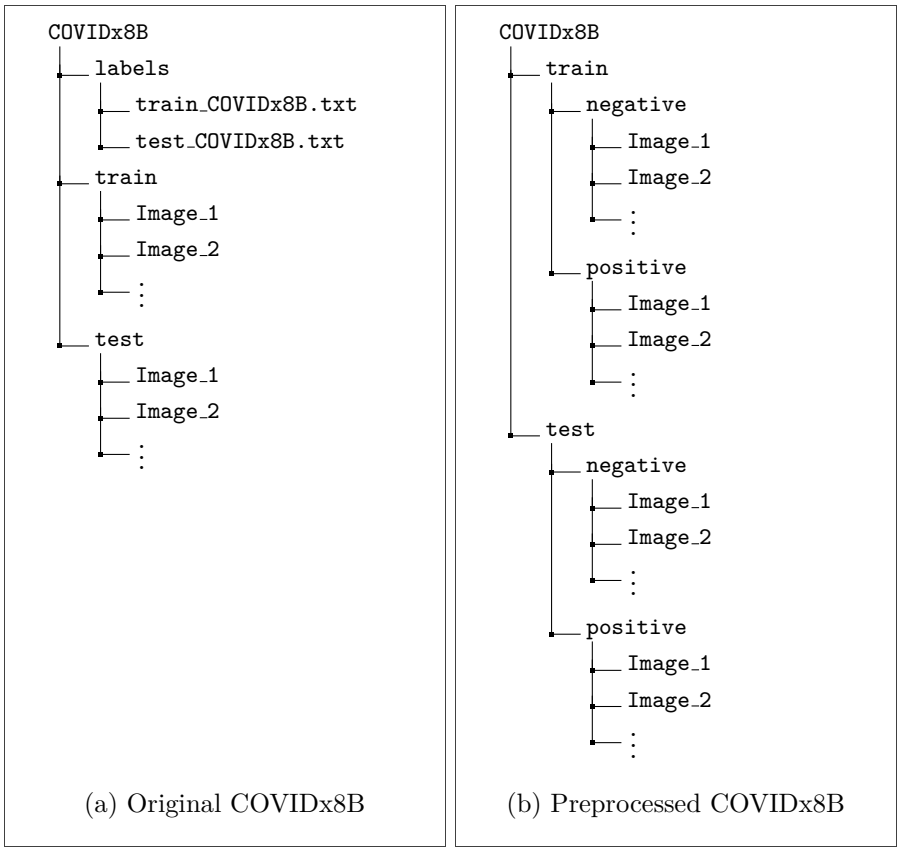
The authors of COVIDx have published a guide on generating the dataset on their public GitHub repository[14]. They also provide two Jupyter notebooks that contain scripts for extracting the image data from the individual datasets and aggregating them into a training set and a test set. One of the notebooks is meant for creating a dataset for binary classification of positive/negative COVID-19 CXRs, while the other prepares the data for categorical classification of no pneumonia/non-COVID-19 pneumonia/COVID-19 pneumonia. The labels of all images and the subset they belong to are also provided as text files.

Our task is focused on the binary classification of COVID-19, so the first of the two provided notebooks was adapted and used to set up the current COVIDx8B version. Due to the amount of image data we were working with, we had to continuously generate data in batches online during training, which can be done using the Keras `ImageDataGenerator` class. The generator requires the dataset to be organized according to its classes, so we prepared a Python script to alter the directory structure so that it is ready to be fed into the generator object. Figure 4.1a shows the data directory structure after generating the COVIDx8B dataset according to instructions by the COVID-Net project, while Figure 4.1b illustrates the changes we made to prepare the dataset for training with the `ImageDataGenerator`.

4.2.1 Data Exploration

The COVIDx8B includes a mixture of posterior-anterior and anterior-posterior views of 16 352 CXR images. The radiographs have been collected from case studies of 15 346 different patients from at least 51 different countries [109]. The training set reflects the scarcity of COVID-19 case data available in the public domain and therefore suffers from being very imbalanced, where the

[14] `https://github.com/lindawangg/COVID-Net/blob/master/docs/COVIDx.md`



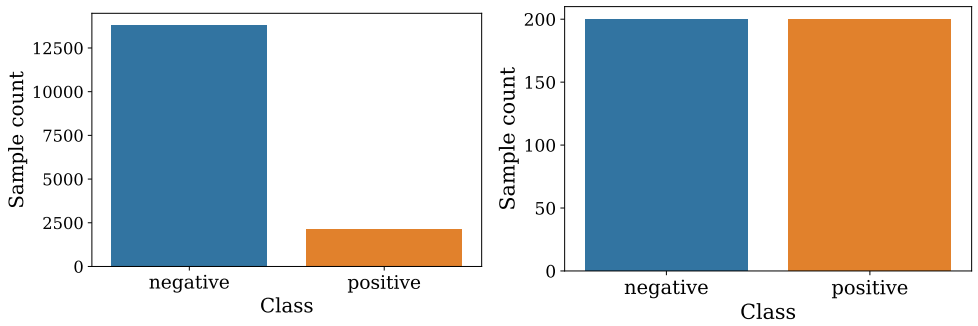
(a) Original COVIDx8B

(b) Preprocessed COVIDx8B

Figure 4.1: Comparison of the directory tree of the original COVIDx8B dataset (a) as used by the COVID-Net project, and the preprocessed version (b) which was used in our experiments for image generation during training.

COVID-19 positive class is heavily outweighed by the negative samples, as seen in Figure 4.2a. It contains 13 794 images belonging to the negative class and 2 158 to the positive class, which is, however, a significant improvement on the previous versions such as COVIDx3 from the first half of the year 2020, which only had around 400 positive samples. Contrastingly to the training set, the test set was designed to be balanced in order to prevent biased evaluations of models that have overfitted to predicting the majority class. The class distribution of the test set is shown in Figure 4.2b and contains 200 samples from each class.

The images vary a lot in size, exposure and captured region of the body, which could possibly have a detrimental effect on the training. The smallest image in our data is 156×157 px while the largest is $3\,480 \times 4\,248$ px, but the side length of the majority is in the range between 256-1 024 px. Since CNN architectures accept a fixed input size, some images will have to be downsampled and some upsampled to unify their dimensions. This inevitably



(a) Distribution of the training set

(b) Distribution of the test set

Figure 4.2: Comaprison of the class distributions in the COVIDx8B training and test set. The training set has a significant class imbalance, while the test is balanced to avoid bias in model evaluation.

results in loss of information and quality, which has to be taken into account. A few examples of the images found in COVIDx8B are shown in Figure 4.3.

The data also includes some abnormal samples which contain medical annotations, such as text describing the type of the radiograph, the type of view, or even arrows that point to specific regions of interest within the image. Another source of anomalies are medical devices captured by the screening apparatus, such as pacemakers or prostheses. Examples of such images are shown in Figure 4.4. Since these abnormalities are only present in a small minority of the samples, we did not feel the need to remove them, and we did not observe any overfitting to specific annotations.

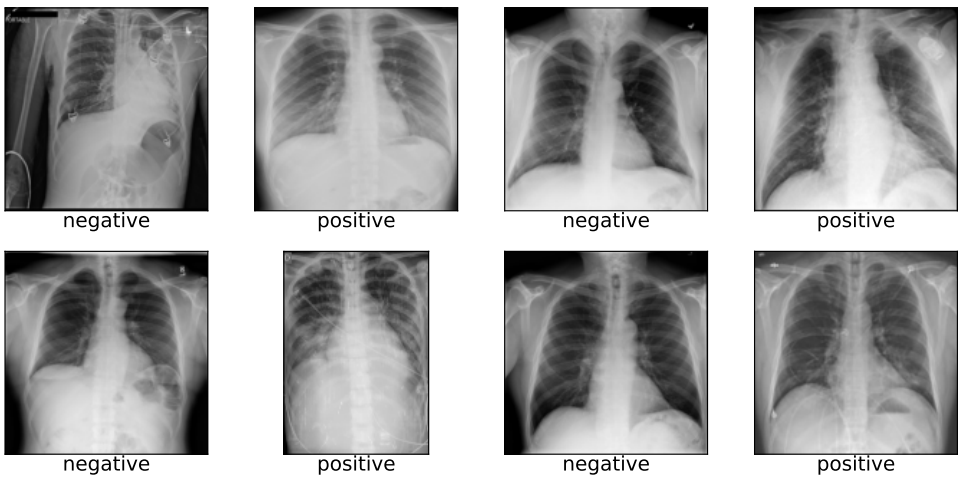
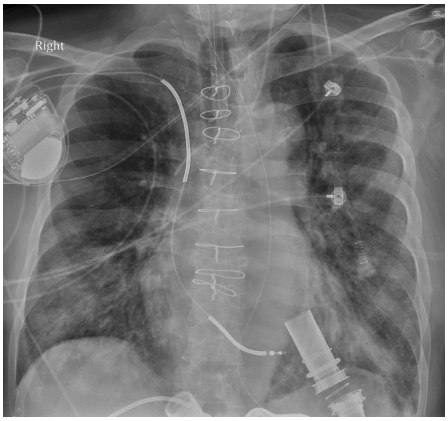
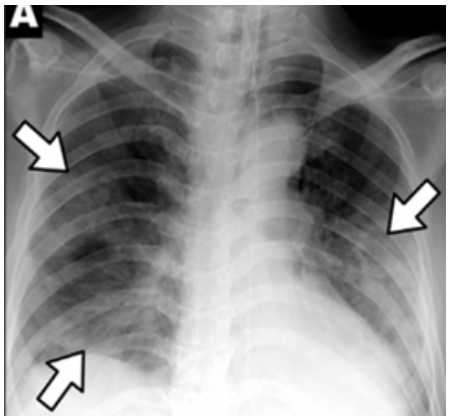


Figure 4.3: Examples of CXR images found in the COVIDx8B dataset and the classes they are labelled with.



(a) Medical equipment captured in a CXR image



(b) Arrows marking the regions of lungs with a suspicion of being infected

Figure 4.4: Examples of abnormalities found among the data samples. Image (a) shows a piece of medical equipment captured in the CXR image and image (b) contains annotations in the form of arrows which mark potentially infected regions of the lungs.

4.2.2 Data Separability in Lower-dimensional Spaces

During data exploration, we applied the UMAP technique for dimensionality reduction, described in Section 3.2.7. The goal was to visualize the dataset in lower dimensions to gain insight into the distribution of the two classes. In order to extract features from the images, we used the CheXNet model, which is a DenseNet-121 architecture pre-trained on the ChestX-ray14 dataset. We decided to use this model as it was previously applied to extracting features from CXR images in pulmonary disease classification tasks; hence its weights are adapted to the structure of the images in our domain as well. We performed feature extraction on 500 positive and 500 negative random samples from the training data of COVIDx8B and passed them on to the UMAP algorithm implemented by its authors and provided as a Python package[15]. We used the Euclidean distance as the metric of the algorithm, with a specified minimum distance of 0.4 and the relatively low number of 50 neighbouring points to capture the local structure of the space. As shown by Figure 4.5, two separate projections were calculated: one into a two-dimensional feature space and the other into a three-dimensional feature space. The COVID-19-positive samples are drawn in red, and the negative samples are blue. While neither of the projections suggests that the encoded features are linearly separable in these dimensions, we may notice a pattern where the negative samples are clustered in the centre and the positive samples surround them. This observation leads us to believe that a CNN trained on CXR images may be able to extract the

[15] `https://umap-learn.readthedocs.io`

necessary features for discriminating between the two classes. The separation will likely be more apparent in a higher dimension and perhaps not linear.

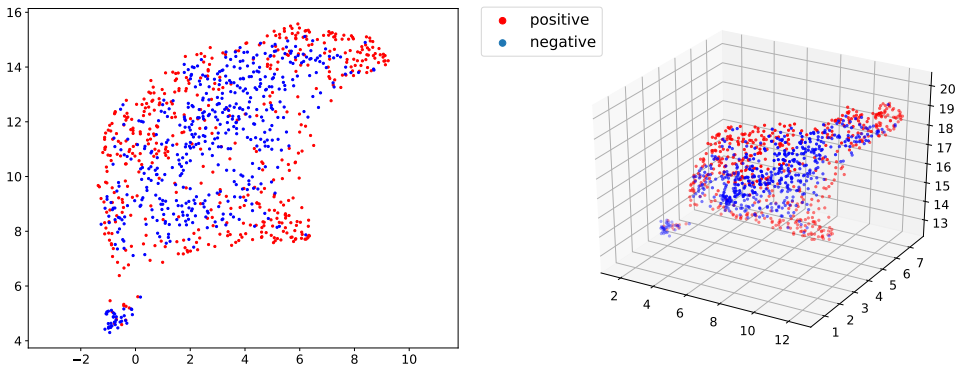


Figure 4.5: Visualization of UMAP projections of extracted features from a sample from COVIDx8B data onto a 2D feature space (on the left) and a 3D feature space (on the right).

4.3 Model Training and Evaluation

As mentioned previously, we used the high-level Keras functional API for defining our models. The specific configurations of each model will be discussed at a later point, along with the experiments in which they were involved. Using the Keras API allowed us to create a training environment where the compiled CNN architecture is continuously fed data from the previously mentioned `ImageDataGenerator`. The generator is given a preprocessing function that is applied to each image and the paths to data sources used to feed the images to the network in batches during training. Most of our experiments were done with a batch size of 32 or 16, depending on the allocated GPU memory that was available. We created 3 separate generators: a training generator, a validation generator, and a test generator. The training generator's source was set to 80 % of the training images, which it fed in batches to the networks during their training. After each batch, the network's weights were updated through gradient descent. Once a whole epoch passed and all of the data took part in the training, the validation generator provided the remaining 20 % of the training images for evaluating the network's performance with the binary accuracy metric. Due to the heavy class imbalance present in the data, we applied cost sensitive learning to prevent the model from degrading to constant prediction of the majority class. The classification error of each sample was weighed inversely proportionally to its class's prevalence in the training data. In the case of COVIDx8B, the weight of COVID-19-positive samples was calculated to be 3.70 and the weight of negatives samples was 0.58.

If not stated otherwise, our experiments were conducted using the Adam optimizer with an initial learning rate of 10^{-4}, exponential decay rate of 0.9 for the first moment estimate, and exponential decay rate of 0.999 for the second moment estimate. Our research found that the Adam optimizer tends to be the favoured optimization algorithm of choice in similar tasks, and we achieved consistently good results with this configuration. Most of the explored architectures were trained for anywhere between 10-20 epochs until their cost functions converged. Calculating the cost function was done by taking the average of binary cross-entropy loss across the training data, as is the conventional approach for binary classification tasks. During training, we used the early stopping and checkpointing method of regularization, where we maintained a saved version of the model configuration with the highest validation accuracy. Once the training stopped, or the validation accuracy did not improve for a certain number of epochs, the training was interrupted, and the model was reverted to its configuration that performed best on the validation set.

Certain experiments involving the training of a single architecture configuration were executed using k-fold cross-validation. More specifically, we used the stratified k-fold cross-validation where the folds are constructed in a way that captures the distribution of the classes in the whole training set. The typical number of folds we used was 4, which means that roughly 75 % of the training data was used to train each fold and 25 % was left for its validation. Upon completion of the cross-validation process, we selected the best model that was found throughout all folds and evaluated it on the test set. This approach was not a viable option for every single experiment, as that would consume too much time and resources. We used it specifically for the testing of individual models, where a reliable evaluation was crucial.

The trained models were primarily evaluated by predicting images provided by the test generator. If the predicted probability of an image containing COVID-19 was greater than 50 %, the image was classified as positive; otherwise, it was classified as negative. After classifying each image supplied by the test generator, we calculated several metrics to rate the model's performance. Since our test set is balanced, our primary metric for evaluating and comparing the models was binary accuracy (labelled as *Test acc.* in results). In order to be able to detect potential overfitting, we also measured the binary accuracy on the whole training set (*Train acc.*). Furthermore, we constructed a confusion matrix and based on its information, we calculated the COVID-19 sensitivity (TPR) and specificity (TNR) on the test set. Finally, we plotted the ROC curve and calculated its AUC metric to evaluate the classifier regardless of the chosen probability classification threshold. Each of the metrics and their interpretation within the medical field is described in Section 2.2 in more detail.

Chapter 5

Experiments and Results

In this chapter, we will describe the experiments we conducted in researching the topic of COVID-19 detection, as well as discuss their results and possible interpretation. We will describe our proposed preprocessing techniques and CNN architectures and compare their performance on the chosen COVIDx dataset.

5.1 Evaluating COVID-Net Performance

5.1.1 COVID-Net CXR-2

To establish a benchmark for the comparison of our created models, we initially set out to perform some tests on the latest state-of-the-art COVID-Net architecture proposed by [77]. It is the COVID-Net CXR-2 model described in Section 3.5.1. We downloaded the model definition along with its fine-tuned weights on the COVIDx8B training set and gathered the model's predictions on both of the subsets. As shown in Table 5.1, the classification accuracy on the training set was 96.07 % and 96.25 % on the test set, which matches the values reported by the original authors. The similarity of these results suggests that the model has not overfitted the training data and is able to generalize well out of sample.

To explore this theory further, we traced back the older version of the dataset called COVIDx3 and performed the same experiments. The dataset had to be adapted to make it fit for our binary detection task by removing non-COVID-19 pneumonia images. Following this preprocessing step, the accuracy on the COVIDx3 training set was 99.51 %, which is very high but does not reflect the model's ability to generalize. We calculated the overlap between the COVIDx8B training set and COVIDx3 training set and found that around 55 % of the COVIDx3 training images have also taken part during the training of the model on the new version of the dataset. Nevertheless, the classification accuracy on the COVIDx3 test set was 93.99 %, which is still very respectable,

and none of this data was included in the training. Therefore, we can conclude that the model predicts previously unseen data with remarkable consistency.

5.1.2 COVID-Net CXR3-B

As we have mentioned before, the COVID-Net CXR-2 lacks proper documentation, and the actual model described by the COVID-Net publication is the COVID-Net CXR3-B. Knowing a little more about its design and the fact that it was created in the very early stages of the pandemic, we set out to evaluate how it handles the data that we have available a year later after its creation. We first evaluated it on the preprocessed COVIDx3, which was the state-of-the-art dataset at the time of the model's release. The observed accuracy on the training set was 96.07 %, and the test set accuracy was 96.17 %. The values slightly deviate from those reported by the original authors, but this is likely due to the fact that their evaluation considered categorical classification while we investigated binary classification. The model evidently had enough capacity to function well on the data available at the time. However, its performance on the new COVIDx8B dataset has significantly degraded, as we measured a training accuracy of 61.66 % and a test accuracy of 71.50 %. These results are not very surprising, considering that the number of COVID-19-positive samples in the training set has increased by a factor of 5 since the older version. The model probably did not have enough examples to properly learn to extract the important features that separate the classes.

Following this discovery, we decided to train the CXR3-B on the new data ourselves to see if the old architecture has enough capacity to fit the current version of the dataset. Since the authors only provided the model as a TensorFlow graph and we struggled to create a training setup in that environment, we decided to re-implement the model using the Keras functional API. To distinguish between the models, we will refer to our adaptation of the architecture as COVID-Net CXR3-B2. We initially followed CXR3-B's description and diagrams in the original publication and then made a few changes to suit our use case better. The input layer was downsized from accepting 480×480 px images to 224×224 px, and the output layer no longer used 3 neurons activated by the softmax function but was instead built for binary classification with a single neuron and the sigmoid activation function. The reduced input size was a measure we took to reduce the training complexity to a manageable task given our resources and also to match the input dimensions of models that we will discuss at a later point. The resulting network has almost 6.3 million trainable parameters and is a composition of 87 layers, most of which are organized into the PEPX blocks described in Section 3.5.1.

We trained the COVID-Net CXR3-B2 on COVIDx8B with stratified 4-fold cross-validation, where each fold ran for 15 epochs. As suggested by the architecture's authors, we used the Adam optimizer with an initial learning rate of 2×10^{-4}. The progression of the training can be seen in Figure 5.1, where

the model of each fold is evaluated after every epoch on the validation set. We saved the best performing model per each fold, and their average validation accuracy was 93.63 %. The best of these was the model trained in the fourth fold, so we evaluated it similarly to the CXR-2 and CXR3-B. As the results reported in Table 5.1 show, our implementation achieved quite consistent results without regard for the dataset version. Its classification accuracy on the COVIDx8B training data was 95.21 % and 97.14 % for COVIDx3 training data, and 89.75 % and 85.79 % on the COVIDx8B and COVIDx3 test sets, respectively. While this performance is relatively good, and our implementation certainly increases the consistency of the model's out of sample prediction accuracy, there is a significantly decreased ability to correctly classify samples in the older test set compared to the original implementation. We suspect that this is either the result of the loss of information caused by downsizing the input images, not pre-training the architecture on ImageNet like the original version or, most likely, the fact that CXR3-B was trained on a more informative dataset. Its purpose was to perform categorical classification, which means that its training involved thousands of additional images containing non-COVID-19 pneumonia. Consequently, the original implementation had access to more images with symptoms that belonged to or were very similar to COVID-19, which gave it a better chance to learn the defining features of pulmonary diseases in general. When examining the model's binary prediction, we simply chose the higher of the probability estimates of COVID-19 pneumonia and non-COVID-19 pneumonia, which likely gave the model a significant advantage over our CXR3-B2 which only saw the binary samples during training. This explanation is also supported by the fact that we measured the CXR3-B's COVID-19 sensitivity to be 97.59 %, which is a lot higher in comparison to the 91.00 % reported in the categorical study by the model's authors.

Table 5.1: Results of evaluating the performance of the COVID-Net CXR-2 and the COVID-Net CXR3-B on two separate versions of the COVIDx dataset, as well as the performance of our own implementation of the proposed architecture, which we call the COVID-Net CXR3-B2.

Model	Dataset version	Train acc.	Test acc.	TPR	TNR	AUC
CXR-2	COVIDx8B	96.07 %	96.25 %	95.50 %	97.00 %	0.994
	COVIDx3	99.51 %	93.99 %	87.95 %	99.00 %	0.991
CXR3-B	COVIDx8B	61.44 %	71.50 %	93.00 %	50.00 %	0.826
	COVIDx3	96.07 %	96.17 %	97.59 %	95.00 %	0.993
CXR3-B2	COVIDx8B	95.21 %	89.75 %	85.50 %	94.00 %	0.957
	COVIDx3	97.14 %	85.79 %	78.31 %	92.00 %	0.931

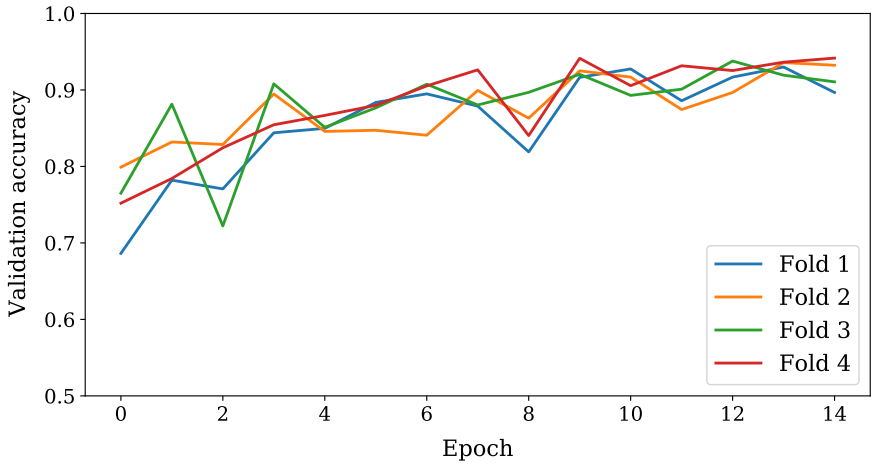


Figure 5.1: Progression of validation set accuracy of the stratified 4-fold cross-validation of our COVID-Net CXR3-B2 implementation.

5.2 BaseNet Architecture and its Hyperparameter Optimization

The following experiment we conducted was creating our own prototype CNN architecture called the BaseNet and testing how it performs on the current COVIDx8B dataset in comparison to the COVID-Net designs produced by generative synthesis. We built a fairly simple feature extraction base that accepts 224×224 px coloured images. It includes 15 convolutional layers, 6 of which have a max-pooling layer, and the final layer is fed into a global average pooling operation which flattens all feature maps by characterizing them with their maximum values. These values are then passed on to a fully-connected dense layer and the final output layer, which contains one neuron activated by the logistic sigmoid function. Each of the max-pooling layers and the dense classifier have a dropout rate to act as a form of regularization and prevent overfitting.

In order to explore different combinations of the network's hyperparameters, we used the Keras Tuner[16], which is a package specifically designed for optimizing the hyperparameters of models defined by the Keras API. It allows for modelling parts of the network as variables and specifying their possible configurations. The parametrized definition is then passed on to one of the provided tuner classes, which attempts to find the optimal hyperparameters by performing short training phases. The hyperparameters we chose to optimize were:

[16] `https://keras-team.github.io/keras-tuner/`

- Dropout rates of neurons in the wide fully-connected dense layer and the max-pooling layers that follow each set of convolutions (values 0.2, 0.3, or 0.5),
- Activation function used in the hidden layers (ReLU, parametrized ReLU with $\alpha = 0.1$, or the hyperbolic tangent),
- Usage of a residual skipping connection that feeds the max-pooled output of the 5^{th} convolutional layer into the 13^{th} convolutional layer where it gets convolved in a parallel branch and added to the standard previous inputs (boolean value to indicate whether the residual features are added or not),
- Width of the fully-connected dense layer in the classification part of the network (128, 256, or 512 neurons).

To perform the search, we used the provided implementation of the Hyperband Tuner. It uses a state-of-the-art hyperparameter optimization algorithm that is partly based on the random search approach that we defined in Section 2.4, but it utilizes adaptive resource allocation and early stopping to make the exploration strategy more efficient. See the original Hyperband publication [120] for further details.

Following the hyperparameter optimization results, our final BaseNet architecture uses a dropout rate of 0.2, the neurons in its hidden layers are activated by the ReLU function, and the fully-connected dense layer that processes the extracted features is 512 neurons wide. We also found that branching the network and using the residual skipping connection improves performance, which could either be due to encouraging feature reuse or reducing the impact of the vanishing gradients problem. This final configuration of our BaseNet design is characterized by 8.5 million trainable parameters and is illustrated in Figure 5.2. See Table C.3 for a more detailed description of the network and its parameters.

To reliably evaluate our prototype, we used the stratified 4-fold cross-validation pipeline once again. The BaseNet was trained with the Adam optimizer with an initial learning rate of 10^{-4} for 15 epochs in each fold. The progress of the model's performance on the validation set throughout each fold is presented in Figure 5.3. We kept track of the best performing models, and the overall highest achieved validation accuracy was during the first fold after all 15 epochs of training. We loaded the BaseNet with the weights from this checkpoint and tested it on the whole training set and the test set. The training accuracy of the classifications was 97.45 %, and the test accuracy was 95.50 %. These results are very impressive considering that the current state-of-the-art COVID-Net CXR-2 only had a slightly higher test accuracy of 96.25 % and its training accuracy was actually lower. Both of the models achieve comparable results, and neither is prone to overfitting the

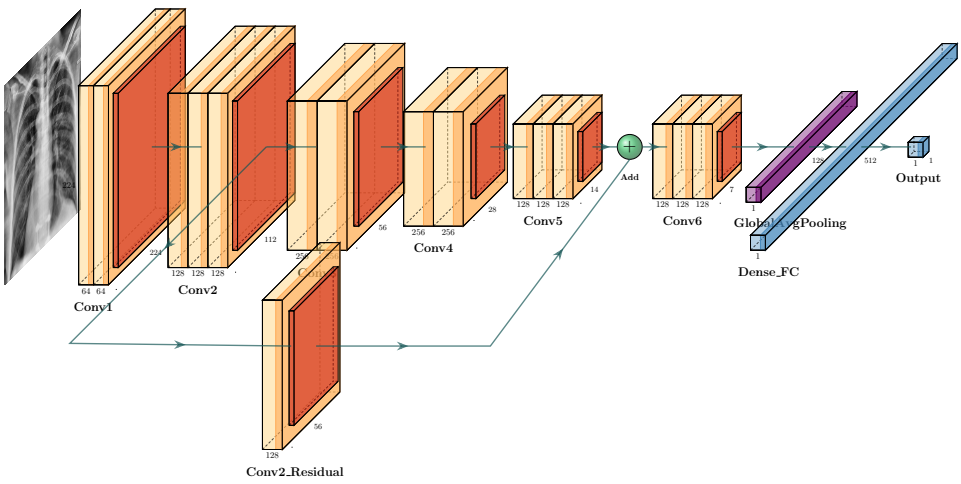


Figure 5.2: Illustration of the optimized BaseNet architecture prototype for the detection of COVID-19 in medical CXR images. The illustration was created with the use of source codes from the PlotNeuralNet tool[17].

training data because the test accuracy does not significantly deviate from the training accuracy. The advantage that our BaseNet architecture has over its COVID-Net counterpart is that it is much simpler to define and understand, it reached similar results within a shorter training duration (faster by about 7 epochs), and it did not require pre-training on the ImageNet dataset. Its weights were randomly initialized, and as the graph in Figure 5.3 shows, the random initialization did not affect the final convergence; each of the models in different folds reached similar results despite their large difference in the initial epochs. Our model is also slightly more light-weight, as it only has 8.5 million trainable parameters in comparison to the CXR-2's 8.8 million, but this is by no means a significant difference.

Aside from the binary accuracy metric, which we used to compare our prototype to the state-of-the-art, we also explored other evaluation criteria to analyze its performance further. We assembled a confusion matrix for the binary classification on the COVIDx8B test set and used it to calculate a COVID-19 sensitivity of 93.00 % and a specificity of 98.00 %. The high sensitivity indicates that our method of diagnosing COVID-19 using the BaseNet predictions has a high potential to recognize patients with the disease. The specificity is even higher, which would mean that a patient whose CXR scan was classified as positive by our BaseNet model is very likely to indeed have a COVID-19-induced pulmonary infection. Furthermore, we plotted an ROC curve and calculated its AUC metric, which equalled 0.987. The collection of these outcomes leaves us fairly confident that our model has substantial capacity to discriminate between the positive and the negative class.

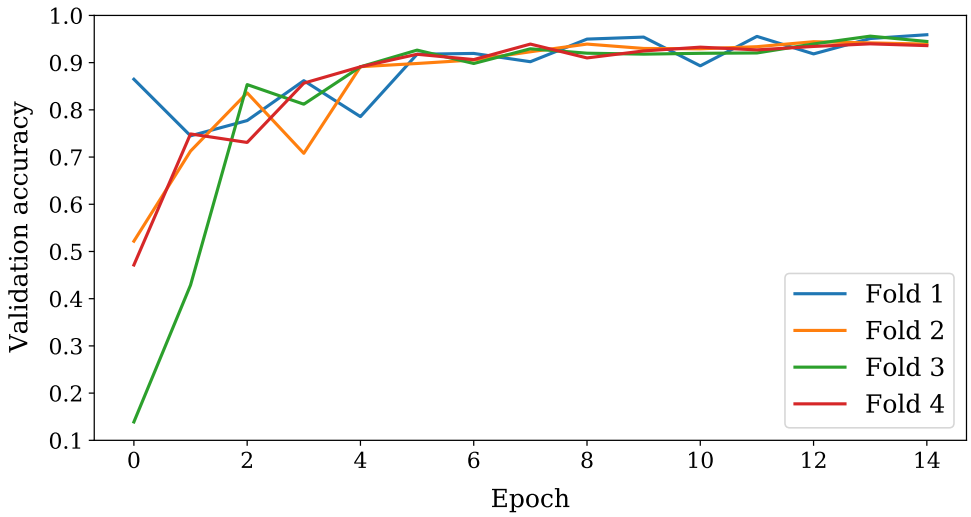
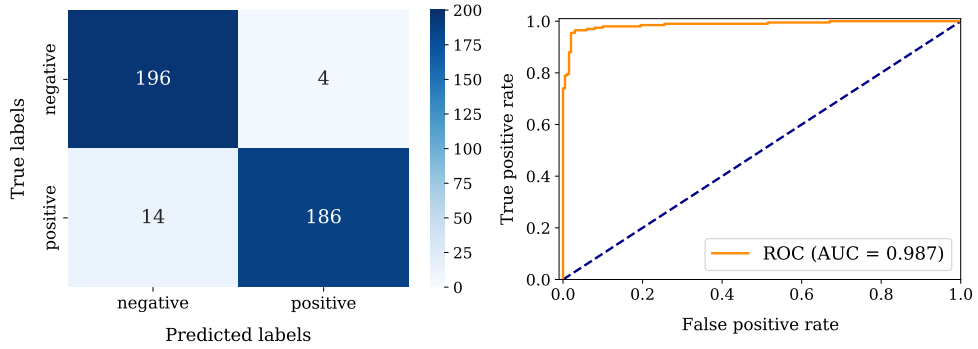


Figure 5.3: Progression of validation set accuracy of the stratified 4-fold cross-validation of our BaseNet prototype.



(a) Confusion matrix

(b) ROC curve

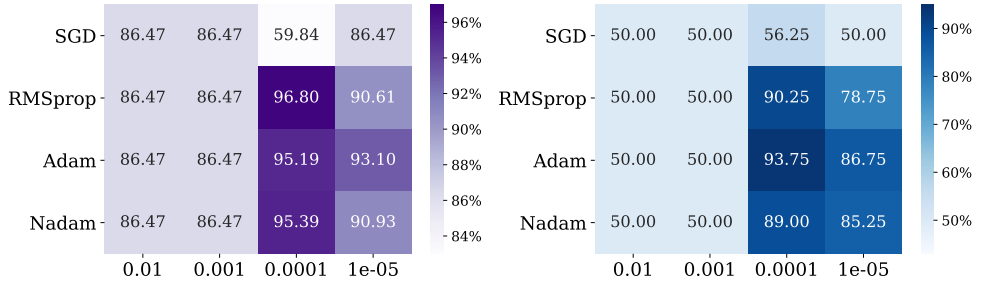
Figure 5.4: The confusion matrix and ROC curve that characterize the BaseNet's binary classification performance on the COVIDx8B test images.

5.3 Optimizer Selection

The early stages of our experimentation were all done with the use of the Adam optimizer with an initial learning rate of 10^{-4}. Although this configuration has performed well in training our implementation of the COVID-Net and in determining the hyperparameters of our BaseNet model, our next step was to compare this optimizer with a selection of other very popular alternatives. We performed a manual hyperparameter grid search over 4 chosen optimizers and 4 learning rate settings that were applied during the training of the BaseNet model on COVIDx8B data. The optimizers were SGD without a momentum

term, RMSprop, Adam and Nadam. The examined learning rates were the values 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. Our aim was to test how well the optimizers can train the weights to minimize the cost function and how quickly they can converge. For this reason, we only trained each model for 10 epochs to make the conditions more difficult and get a better idea about the performance of each optimizer. The resulting accuracies of each configuration on the training set and the test set can be seen in Figure 5.5.

The immediate observation that can be drawn from the results of this experiment is that higher learning rates tend to make rapid weight updates and cause the cost function to converge early to a suboptimal solution. When the values 10^{-2} and 10^{-3} were used, all of the optimizers found that the best solution is to take advantage of the class imbalance in the data and simply predict the majority negative class under all circumstances. This resulted in a training accuracy of 86.47 % and 50.0 % on the balanced test set. This is also a good demonstration of why it is helpful to keep a balanced test set when dealing with medical data, which often suffers from being imbalanced.



(a) Accuracy on the training set

(b) Accuracy on the test set

Figure 5.5: Comparison of the performance of different optimizers and learning rates used to train the BaseNet prototype. The trained model's binary accuracies are shown separately for the COVIDx8B training set (a) and for the COVIDx8B test set (b).

As we moved to the lower learning rates, the results drastically improved. For the value of 10^{-4}, the RMSprop, Adam, and Nadam were able to fit the training data very well and still maintain good generalization on the test data. As for the SGD, we can see that it no longer used the strategy of predicting the majority class, but it was still unable to converge to a reasonable point. Without the use of an accumulating momentum, this optimizer probably converges too slowly and would need more epochs to show improvement. The lowest attempted learning rate of 10^{-5} also gave us relatively good results. Once again, RMSprop, Adam, and Nadam were able to discriminate between the classes both in the training set and the test set, but the accuracy did decrease in comparison to the previous learning rate. We suspect that the weight updates with this learning rate become too small, and the convergence takes

longer, so the model would probably reach comparable results to those with the value of 10^{-4}, but it would need to be trained for more epochs. The same can be said for SGD, which once again stagnated on predicting the majority class.

Based on these results, we conclude that the choice of the Adam optimizer with an initial learning rate of 10^{-4} was indeed a wise choice, as also suggested by our research and analysis of the related works. The RMSprop with this learning rate achieved a higher training accuracy of 96.80 % in comparison to Adam's 95.19 %, but this difference is not very significant, and our primary evaluation criterion is the binary accuracy on the test set, where Adam outperformed the other optimizers by a significant amount with a result of 93.75 %.

5.4 Impact of Image Preprocessing Techniques

While deep learning famously requires very little to no preprocessing of the input data, our research of related studies in COVID-19 detection has shown that preprocessing the images before feeding them into the network may prove to be beneficial in some instances. In order to explore this possibility, we designed several different preprocessing functions that we separately initialized the image generators with. We then trained our BaseNet prototype for 20 epochs with each setting and observed the impact of the various methods on the evaluation metrics.

5.4.1 Min-max Normalization

The benchmark we set for comparing the various preprocessing approaches is the use of min-max normalization. This technique does not remove any information from the image and does not alter its intensity distribution in any way; it simply rescales the values to range from 0 to 1 by dividing each pixel value by 255. Doing so transforms the data to have comparable scales and reduced variation, which frequently leads to better results in many deep learning scenarios. At the very least, this technique should not hinder the model's performance, making it a suitable candidate for being a constant when optimizing other hyperparameters. Hence, we used min-max normalization when performing all the previously described experiments.

Since the BaseNet architecture is built to accept image sizes of 224×224 px, each image was also resized to fit these dimensions. This is true for all the preprocessing techniques that we will discuss.

5.4.2 Histogram Equalization

The first more complex preprocessing technique that we designed uses the histogram equalization described in Section 3.2.4. The images in our dataset

are not limited to grayscale values; they are a composition of three colour channels. Consequently, performing standard histogram equalization in the default RGB colour space would redistribute the channels separately, and the colour components would be incorrect. To resolve this issue, we first converted the image into a colour space that separates the pixel intensity values from the colour components. The standard colour space used in this scenario is the YCbCr, which uses the Y component to represent the pixels' luminance and Cr and Cb are their chromatic components. After converting the image to YCbCr, we equalized the distribution of the whole Y channel and then converted the image back to the RGB colour space.

Due to the fact that histogram equalization has a tendency to highlight noise in the images, we attempted to denoise them first by using bilateral filtering. As described during our analysis in Section 3.2.3, bilateral filtering removes noise while preserving edges, which is crucial to our CXR domain where the COVID-19 may be indicated by linear opacities along the lungs. After denoising, the intensity equalization was performed, and the image was resized.

5.4.3 Contrast Limited Adaptive Histogram Equalization

To further combat noise amplification when increasing image contrast, we designed an alternative preprocessing method that uses CLAHE instead of the global histogram equalization. This may also be helpful in situations where the image does not have a similar contrast level throughout all its regions, so redistributing the intensities locally may improve the overall separation of structures and patterns within the image. As was the case in the previous approach, we first applied bilateral filtering and then converted the colour space to YCbCr and performed CLAHE on the pixels' luminance. The default clip limit of 40 was used initially, but because we found the resulting images to be damaged and overly contrasting, we also tested a clip limit of 3. After increasing the local contrasts, the image was resized and normalized. The effect of applying this technique is shown in Figure 5.6 along with the output of standard histogram equalization for comparison.

5.4.4 Diaphragm Segmentation

The last attempted technique used in training our prototype was the preprocessing pipeline specifically proposed for COVID-19 detection by [83], which aims to remove the diaphragm in the CXR image based on the assumption that it forms a large high-intensity region that interferes with analyzing the actual region of interest — the lungs. We recreated the preprocessing steps as accurately as the level of detail in the original publication allowed; therefore, we will describe the steps specific to our implementation and the rest of the details are described in Section 3.2.6 of our analysis.

Figure 5.6: Results of applying histogram equalization and CLAHE with a clip limit of 3 on a low-contrast CXR image. We can see how the global histogram equalization increases overall contrast, but the localized CLAHE method is able to bring out individual bones and lung markings more clearly.

Each image was converted to a grayscale single-channel version, which was resized to 224×224 px, and a threshold was calculated based on the recommended formula $T = V_{min} + t \times (V_{max} - V_{min})$. The authors used the value 0.9 for the cutoff, which we denote as t; however, we found that this value leads to the removal of only very small areas in our images. We empirically determined that the optimal value for the cutoff t on our data is 0.8, as it removes larger areas of the high-intensity regions. Next, we used the threshold T to perform binary thresholding and traced the contour of the largest remaining object. Considering that the thresholding operation creates very rough and jagged edges, we first had to smoothen the contour to better match the shape of the diaphragm, which we are trying to remove. We attempted this with two different methods. The first involved enclosing the contour with a convex hull, which is the smallest convex set of points that contains the whole contour. The other method approximated the shape of the contour with a polygon to reduce the number of vertices. The outputs of both of these methods were further smoothened using a combination of morphological operators, and the resulting contour of the diaphragm was removed from the original image. In accordance with the proposed pipeline, we then proceeded to take an identity of this segmented image, its filtered version, and its equalized version and combine them into a three-channel pseudocolour image ready to be used in training. The resulting preprocessed images with the use of convex hull enclosing are shown in Figure 5.7, and the ones with polygon approximation in Figure 5.8. It is clear that the convex hull leads to suboptimal results, where it sometimes removes a very large portion of the image, including the lungs. For that reason, we opted for the polygon approximation when finally using the method to train the BaseNet model.

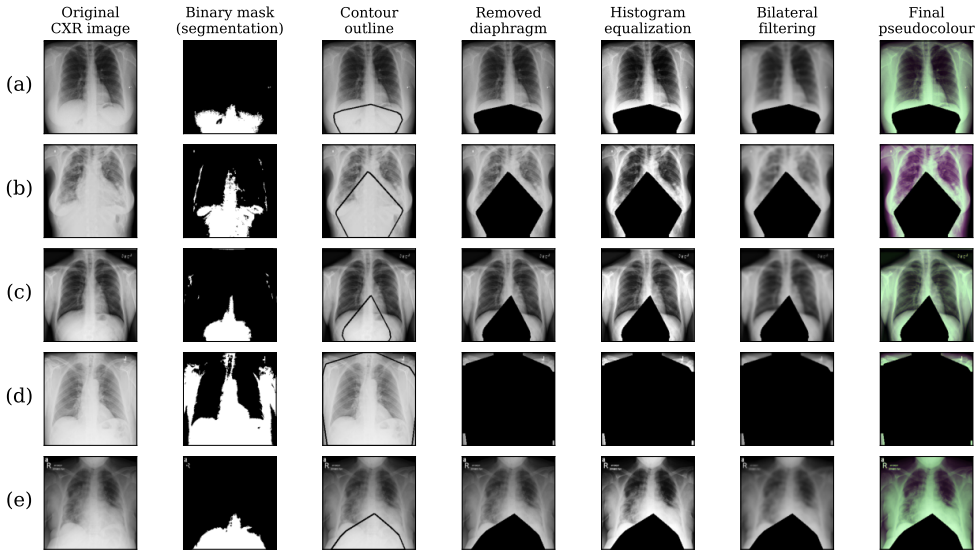


Figure 5.7: Demonstration of the steps of our implementation of the COVID-19 preprocessing pipeline, which uses a convex hull to enclose the contour of the high-intensity region. The convex nature of the final contour causes it to often overlap areas that were not segmented by the calculated threshold. Specifically, in images (b) and (d), this causes significant problems where the majority of the image is removed, leaving very little or no useful information.

5.4.5 Results

To our surprise, we found that the model is able to generalize best when no transformative preprocessing is used — simply using min-max normalization, which is a very standard step in most deep learning pipelines, yields impressive results. The test accuracy of the model trained with this preprocessing technique was 92.75 %, and we know that the potential is even higher, as all of the previous BaseNet models have been trained with the same technique and even reached higher accuracies (cross-validation discovered a configuration that led to 95.50 % test accuracy). We suspect that the reason why the model reached slightly better results in the previous experiments was that they were only run for 10-15 epochs, whilst our preprocessing experiment ran for 20 epochs, which led to overfitting. This hypothesis is supported by the fact that the training accuracy of this preprocessing experiment was 98.33 %, which is higher than any previously encountered results. Its specificity is high, which has been a common occurrence throughout all model configurations, and the sensitivity has slightly dropped to a still respectable 87.00 %.

Analyzing the results in Table 5.2 further, enhancing local contrast with CLAHE proved to be a better approach than using global histogram equalization, where the main effect was to usually highlight the lung region, but not

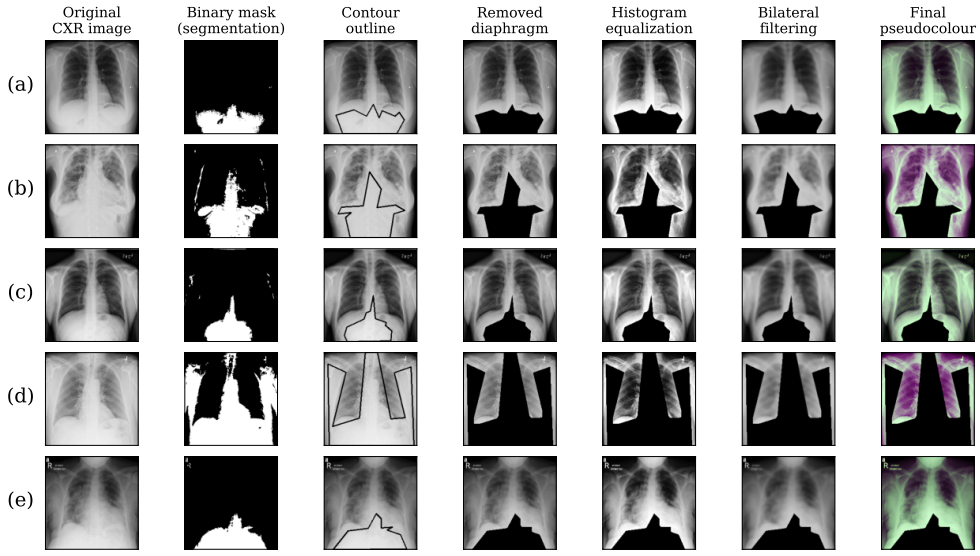


Figure 5.8: Demonstration of the steps of our implementation of the COVID-19 preprocessing pipeline, which uses polygon approximation to outline the contour of the high-intensity region. The computed threshold seems to work well in most images, where it successfully removes the diaphragm region and leaves the lungs undamaged. Image (d) has a relatively low contrast, which leads to a large portion of the image being removed, including a small part of the right lung, which is an undesired outcome.

the individual lung markings. As suspected based on our prior visual assessment of the images, using a clip limit of 3 led to slightly better results than the standard clip limit of 40, which often created several high-intensity artefacts that could have interfered with the natural features. Finally, we discovered that using CLAHE with a clip limit of 3 had very comparable results to the diaphragm segmentation pipeline. This is interesting because the pipeline involves a global histogram equalization step, which leads us to the assumption that removing the high-intensity region from the image improved the ability to focus on the lung area, where the local contrast had a more significant impact as a result.

The root cause behind the more extensive preprocessing methods not being able to outperform the standard min-max normalization is likely the non-homogeneity of the data. The CXR scans all have very different dimensions and levels of exposure and initial contrast, which means that the unified preprocessing steps may only work well with a subset of the data. It is possible that building a pipeline that utilizes more adaptive thresholds and other variables would perhaps lead to better results. We also draw this conclusion based on the fact that during our analysis of other works, the COVID-19 detection pipelines that used complex preprocessing steps were often done on a much

smaller dataset with only one source. Our data comes from 6 different repositories, and the solutions proposed by the COVID-Net team that worked with the same dataset have also used only relatively basic preprocessing steps.

Table 5.2: Comparison of the impact of various preprocessing techniques on the evaluation metrics of COVID-19 detection with the BaseNet prototype.

Preprocessing technique	Train acc.	Test acc.	TPR	TNR	AUC
Normalization	98.33 %	**92.75** %	**87.00** %	98.50 %	**0.982**
Histogram equalization	98.09 %	85.50 %	72.00 %	**99.00** %	0.968
CLAHE (clip = 40)	**98.40** %	88.50 %	78.50 %	98.50 %	0.978
CLAHE (clip = 3)	98.13 %	89.75 %	82.00 %	97.50 %	0.976
Diaphragm segmentation	98.12 %	89.75 %	83.00 %	96.50 %	0.969

5.5 Data Augmentation and Generation

Our primary means of dealing with class imbalance in the training data was using cost sensitive learning and assigning each class with a weight inversely proportional to the class's prevalence in the training data. During this experiment, we explored other methods of either increasing the pool of the minority positive class with oversampling and image generation or augmenting the existing images to increase variation within the training set in an attempt to improve generalization.

5.5.1 Oversampling and Augmentation

We set up two primary ways of increasing the size of our underrepresented class in the training data — randomly sampling images from the COVID-19 positive class and re-inserting them as duplicates into the training set, and doing the same oversampling, but instead of directly duplicating the images, we first augmented them. Both approaches were repeated until the classes in the training data were balanced. The possible augmentations that we defined and randomly applied to each image were up to 10-degree rotations in both directions, up to 5 % horizontal and 3 % vertical shifts, up to 10 % increase or decrease of brightness, and up to 10 % zooming in and out. Vertical flipping certainly makes no sense in the case of CXR images, and we found that using horizontal flipping has also led to worse results. This is likely due to the fact that the chest cavity is not symmetrical, and although the dataset description does not specify this, the majority of images seem to be of the posterior-anterior view, making the horizontally inverted version much less common.

For comparison, we also experimented with using online augmentation of batches of images during the training, where the positive class was not oversampled but merely modified to artificially pose as new data. Our hypothesis was that this would increase variation in the training set, making it harder to fit the data, but also decrease overfitting. Since this alone does not perform class balancing, we paired it with the use of class weights. Given the larger variation in the data, we trained each configuration for 30 epochs. As a control, we also trained the standard class weights balancing solution without any augmentation for the same duration in order to have direct comparability with the other techniques. The evaluation of the models that achieved the highest validation accuracy during those 30 epochs is shown in Table 5.3.

Table 5.3: Comparison of the impact of various class balancing techniques on the evaluation metrics of COVID-19 detection with the BaseNet prototype.

Balancing technique	Train acc.	Test acc.	TPR	TNR	AUC
Class weights	98.31 %	92.75 %	87.00 %	98.50 %	0.977
Oversampling	**99.46** %	91.25 %	83.50 %	99.00 %	**0.986**
Oversampling with augmentation	95.61 %	80.00 %	60.50 %	**99.50** %	0.942
Online augmentation	97.29 %	**94.00** %	**88.50** %	**99.50** %	**0.986**

The first observation we make is that using online data augmentation seems to lead to better results in the long run. Using this technique, we reached a test accuracy of 94.00 %, whereas the class weights balancing with no augmentation achieved 92.75 %. We know that higher accuracy is possible with this configuration because we used this method without augmentation when cross-validating our original BaseNet prototype, which managed to have a test accuracy of 95.50 % in 15 epochs of training. This suggests that using online data augmentation is especially useful in cases where we train the model for more extended periods of time because the increased variation mitigates overfitting. This is consistent with our findings from the analysis of existing research, as the results described in Section 3.5.2 presented by [83] show that not using data augmentation during their 200 epochs of training had a hugely negative impact on their tested models. Our standard oversampling approach had only a slightly lower test accuracy than class weights with no augmentation. This and the fact that its training accuracy was especially high probably means that duplicating the images led to more severe overfitting, and the generalization ability was reduced. The final technique which augmented the duplicates was substantially worse than its alternatives. With a test accuracy of 80.00 % and training accuracy also being the lowest of the techniques, we suspect that there was too much variation in the positive class, so the model learned to properly fit the more stable negative class. This hypothesis is also supported by the very low true positive rate of 60.50 %.

5.5.2 Generating Synthetic CXR Images with DCGAN

One of the latest trends in deep learning is using GANs for generating brand new data based on the distribution of the existing training set. Our research has not uncovered many successful applications of these generative models in balancing the medical datasets for COVID-19 detection, so we attempted to explore this technique and test whether it is a viable option for our data. Inspired by common insights and recommendations in related literature, we built a simple DCGAN architecture where a generator and a discriminator are optimized together in an adversarial manner, as described in Section 3.3.4.1 in more detail. The generator uses transposed convolutional layers to perform trainable upsampling of a random noise vector from the latent space. As the original DCGAN publication [90] suggests, we used a 100-dimensional hypersphere to generate the input vector, where each feature is drawn from the Gaussian distribution with a mean of 0 and a standard deviation of 1. The input of the generator is fed into a wide fully-connected dense layer and then upsampled with 4 transposed convolutional layers with the ReLU activation function, and finally into the convolutional output layer activated by the hyperbolic tangent. This outputs a 256×256 px colour image representing a synthetic sample, which imitates a real CXR image. On the other hand, the discriminator uses 5 convolutional layers with parametrized ReLU ($\alpha = 0.2$) as the activation function, and the flattened features are classified by a single neuron and the logistic sigmoid. The discriminator's output represents the probability that the input image is real and not supplied by the generator.

As per further recommendations by the DCGAN authors, we performed the training of the combined architecture on COVID-19-positive images with the Adam optimizer with an initial learning rate of 2×10^{-4} and a lowered exponential decay rate for the first moment estimate of 0.5 [90]. Contrary to the typical conventions in defining DCGANs, we did not use batch normalization as we found it to produce worse results. The summarized generator and discriminator architectures are shown in Table C.1 and Table C.2 respectively.

Figure 5.9 shows a range of synthetic images generated by our generator after 75 epochs of training. We can see that the model managed to capture the information that the images contain a high-intensity separation in the form of the spine, and we also notice that an attempt to form the two lungs has been made. However, the images are not very clear and certainly not anatomically correct enough to be used in balancing our real data. We observed a significant progression over the course of the training, which leads us to believe that better quality could be achieved with enough computational resources and time. Unfortunately, we were unable to secure these conditions in the Google Colab environment.

Following these findings, we experimented with lowering the computational complexity of the training to explore the potential of the generative models further. We did so by removing one of the upsampling transposed convolu-

tional layers in the generator, which meant that the synthetic images were only 128 × 128 px in size. Our intention was not to use these during the training of our classifiers because we had serious doubts regarding the level of detail signifying the disease that could be captured within these dimensions. The resulting samples generated by our smaller DCGAN are presented in Figure 5.10, and they seem much more hopeful. Firstly, the lungs and the spine are quite clear in each of the images, and there is a much finer level of detail where we can even see ribs and some lung markings. Secondly, the model even managed to capture some of the other organs found in the chest cavity, such as the diaphragm and the heart. Some images are distorted and the anatomy is definitely not perfect, but this experiment indicates that there may be potential in utilizing this method given more computational resources. For our purposes, we decided not to use the images in training, as the likelihood that they contain accurate markings that indicate the presence of a COVID-19 infection is very low.

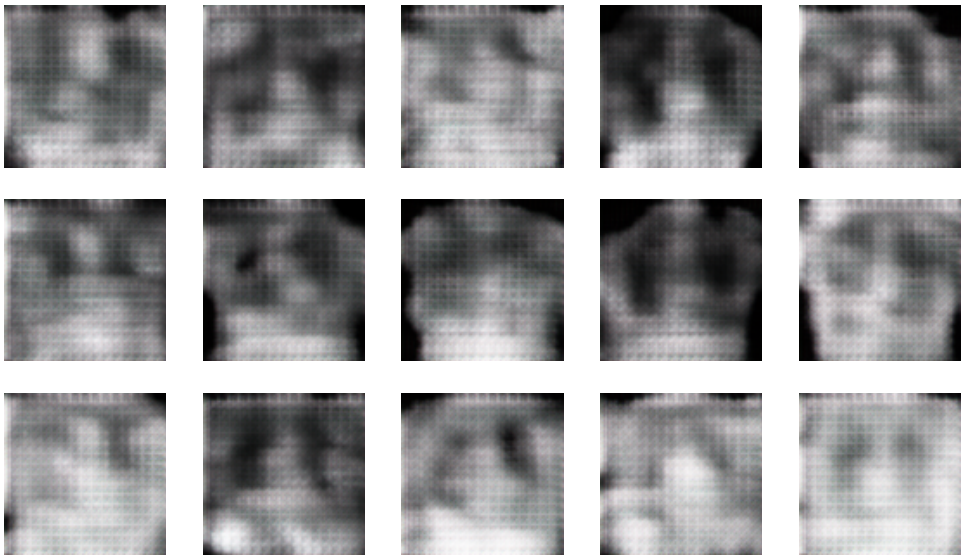


Figure 5.9: Synthetic CXR images generated by our implementation of the DCGAN built for the generation of 256 × 256 px colour images.

5.6 Transfer Learning and Fine-tuning

Our analysis of the related works in COVID-19 detection revealed that the most common approach to this task is utilizing some of the CNN architectures that have performed well in the ILSVRC competitions over the last several years. The Keras framework includes some of these CNNs already built-in and even enables pre-loading them with their weights pre-trained on the ImageNet

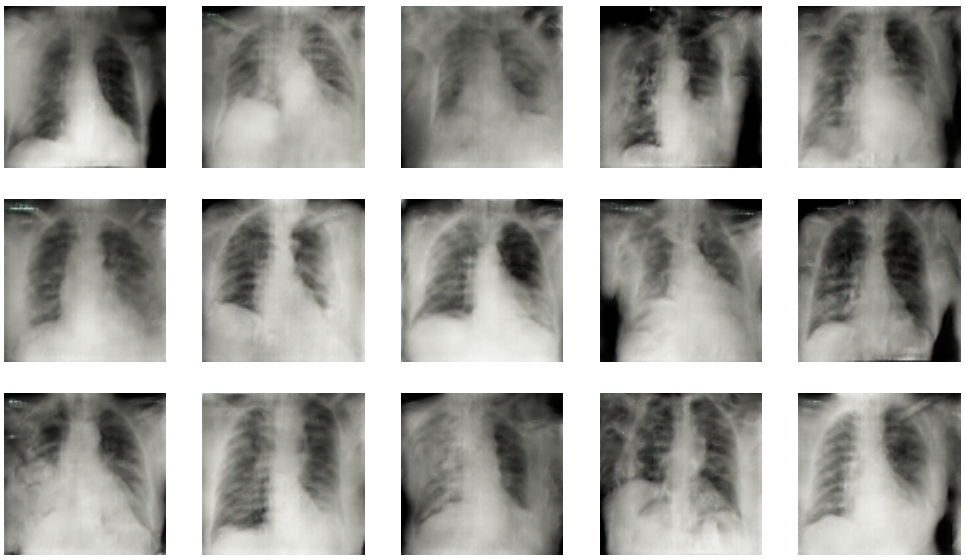


Figure 5.10: Synthetic CXR images generated by our implementation of the DCGAN built for the generation of 128×128 px colour images.

dataset. The specific architectures that we decided to experiment with are the following: VGG16, VGG19, ResNet-50, DenseNet-121, and Xception. We have chosen this selection to include a substantial variety of layer depths, connectivity, and density. The individual specifications and details of the networks are described in Section 3.4.

Each architecture was first loaded with the ImageNet weights and the max-pooling strategy for reducing feature map dimensions, as we suspected that the most significant features within the CXR images would be those with the highest intensities. We also set the input layer size to match the dimensions of our BaseNet models ($224 \times 224 \times 3$). The part of the network that performs classification was then replaced by its adaptation specific to our task. We constructed the classifier by flattening the output of the convolutional base and feeding it into a 256 neuron wide dense layer activated by the ReLU function with a dropout rate of 0.2. This layer was then connected to the final output layer, which contained a single neuron activated by the logistic sigmoid. In the case of the DenseNet-121 architecture, we built two versions. One has the standard ImageNet weights like the other CNNs, and the other is the CheXNet model pre-trained on the ChestX-ray14 dataset. Although these weights are not provided in the Keras framework, they are publicly available for download on Kaggle[18].

In order to make use of the pre-trained weights on the source datasets, we started the training of each architecture with the standard transfer learning

[18] `https://kaggle.com/theewok/chexnet-keras-weights`

approach. The whole convolutional base was locked so that its weights were untrainable, and the classifier was trained independently for 5 epochs. Afterwards, all of the weights in the network were unlocked, and we continued to fine-tune them for 10 more epochs. The reason for this approach was firstly to utilize the highly successful feature extraction from the source data and then gradually try to adapt it to the features present in the COVIDx8B data by delaying the weight updates of the convolutional layers.

The images used in the training of these models were not altered by any of our own preprocessing algorithms but rather by those provided specifically for each individual model. The models come equipped with their own functions named `preprocess_input` which usually apply some form of transformation such as zero-centering or normalizing the pixel intensities with respect to the ImageNet dataset.

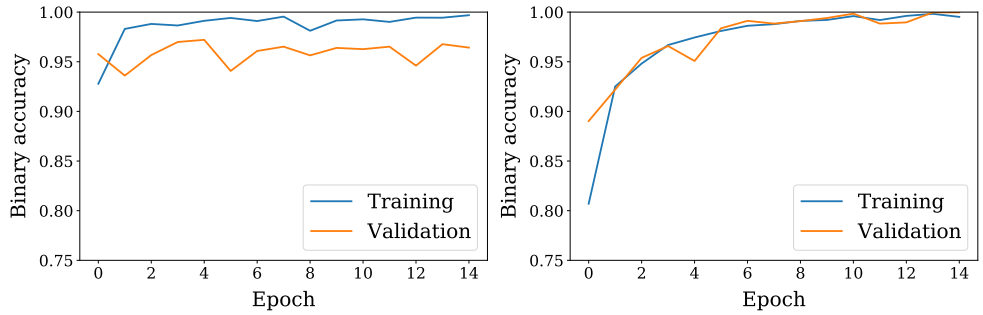
Table 5.4: Comparison of results achieved by various architectures in our transfer learning and fine-tuning experiments.

Model	Params	Source data	Train acc.	Test acc.	TPR	TNR	AUC
VGG16	14.9 M	ImageNet	98.82 %	**97.50** %	**95.00** %	**100.0** %	0.998
VGG19	20.2 M	ImageNet	98.65 %	96.25 %	92.50 %	**100.0** %	0.998
ResNet-50	24.1 M	ImageNet	99.08 %	95.75 %	92.00 %	99.50 %	0.997
DenseNet-121	**7.2** M	ImageNet	99.32 %	95.75 %	91.50 %	**100.0** %	**1.000**
		ChestX-ray14	**99.96** %	96.50 %	93.50 %	99.50 %	0.997
Xception	21.3 M	ImageNet	99.35 %	95.50 %	91.00 %	**100.0** %	**1.000**

Based on the results presented in Table 5.4, we see that almost all of the transferred architectures have been able to outperform our BaseNet prototype, which was to be expected. Only the Xception model has reached an equal test accuracy of 95.50 %, but its COVID-19 sensitivity is lower by 2 %, and the network has almost three times as many trainable parameters as the BaseNet, which leads us to believe that our prototype is superior based on these results. To our surprise, the model that seems to have the highest ability to separate the classes is the VGG16. With a test accuracy of 97.50 %, sensitivity of 95.50 % and specificity of 100 %, it surpassed all of the other models and even the state-of-the-art COVID-Net CXR-2. This is an unexpected result as the VGG architecture is considered to be somewhat outdated by the current standards, though perhaps its relative simplicity gives it the necessary advantage. Being the shallowest CNN in the selection and not utilizing any complex design patterns such as residual connectivity, it seems that it had less overfitting on the training data and was subsequently forced to learn the general patterns and features specific to each class. We also see that the VGG architectures have the lowest training accuracies, which also hints at a lower rate of overfitting. We observed a decrease in accuracy of the VGG19,

which has the same design pattern but has additional layers that add roughly 5 million parameters and complicate the training process.

We are unsure of the reason why the VGG16's generalization outperforms the DenseNet-121 and the COVID-Net CXR-2, because they both have significantly less trainable parameters (DenseNet-121 has 7.2 million and COVID-Net CXR-2 has 8.8 million) and their very large depths are compensated by high residual connectivity which should, in theory, mitigate problems with vanishing gradients. Considering that the difference in their performance is quite minuscule, we suspect it to be simply caused by a random factor in the training process.



(a) DenseNet-121 from ImageNet (b) DenseNet-121 from ChestX-ray14

Figure 5.11: Progression of the training and validation binary accuracies during the training of the DenseNet-121 architecture transferred from the ImageNet (a) and the ChestX-ray14 (b) datasets.

The model with the second-highest test accuracy of 96.50 % is the DenseNet-121 pre-trained on the ChestX-ray14 dataset. Being the only model transferred from source data that belongs to the same image domain as the target COVIDx8B data, we were not surprised that it achieved good results. Its 7.2 million trainable parameters also make it the most light-weight out of any of the models except for our implementation of COVID-Net CXR3-B2, which had 6.3 million parameters but led to significantly worse results. As illustrated in Figure 5.11, pre-training the model on ChestX-ray14 instead of ImageNet provided much better convergence, and the validation accuracy kept steadily rising along with the training accuracy. In the case of the model from ImageNet, we do not see much progression within the span of the 15 epochs, and the generalization ability maintains a relatively constant distance from the ability to fit the training data. Despite this, the ImageNet version was still able to provide very good results, and with its specificity of 100 % (a value shared by several of the models), it remains a useful tool, especially in cases where we search for high confidence in COVID-19-positive diagnoses. Finally, the graphs also show a noticeable drop in validation accuracy around the fifth epoch, which is the point where the fine-tuning portion of the training began.

This is likely caused by the sudden increase in complexity, and we see that the model from ChestX-ray14 was able to recover and fine-tune the weights more successfully.

5.7 Ensemble Model

The last in our series of conducted experiments combines several of the previously described models into an ensemble and evaluates their collective predictive ability. We constructed the ensemble from the best BaseNet model configuration found during its cross-validation described in Section 5.2 and all of the fine-tuned models from Section 5.6 (for DenseNet-121, we used its more successful version pre-trained on ChestX-ray14). To simulate their collective classification, we gathered the predictions of each individual model for every sample in the test data and then calculated the mean of the predicted probabilities. The samples with a mean probability of COVID-19 of over 50 % were classified as positive, the rest as negative.

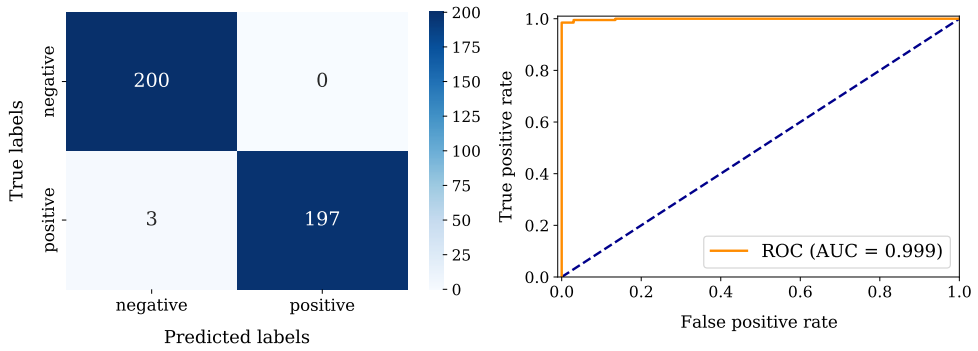
Table 5.5: Evaluation of an ensemble model from several of the best performing base models discovered during previous experimentation. The evaluation is firstly presented for the standard classification threshold of 0.5 and subsequently for a calculated optimal threshold of 0.22.

Dataset	Threshold	Train acc.	Test acc.	TPR	TNR	AUC
COVIDx8B	0.5	99.50 %	97.75 %	95.50 %	100.0 %	0.999
COVIDx3		99.77 %	82.51 %	61.45 %	100.0 %	0.997
COVIDx8B	0.22	98.87 %	99.25 %	98.50 %	100.0 %	0.999
COVIDx3		99.84 %	94.54 %	87.95 %	100.0 %	0.997

By constructing the ensemble this way, we measured a test accuracy of 97.75 % on the COVIDx8B test set, which is the best encountered result yet. To further test its collective ability to generalize, we also performed the experiment on the older COVIDx3 version. While the state-of-the-art COVID-Net CXR-2 retained its good performance on both of the dataset versions, our ensemble's accuracy significantly dropped to 82.51 %. Admittedly, this result does not seem so poor until we calculate the COVID-19 sensitivity, which was only 61.45 %. Worse results were to be expected, but such a low sensitivity was surprising given that the AUC remained high at 0.997. Considering the fact that AUC is a metric independent of the chosen classification threshold and the model had a large number of false negatives, we experimented with lowering the threshold value. We did so not only for the mean probabilities of the whole ensemble but also for probability estimates of every individual

base classifier because their individual performance also decreased on the older dataset (for the chosen BaseNet model, its accuracy on COVIDx3 test set was 84.15 % with a sensitivity of 66.27 %).

Based on the assumption that the optimal threshold value will maximize the true positive rate and minimize the false positive rate, we used the ROC curve to find the value for which the expression $TPR - FPR$ reaches its maximum. The optimal threshold was 0.32 for the COVIDx8B training data and 0.03 for the COVIDx3 training data. We combined the thresholds by calculating their weighted mean to find a value that works well across both datasets. Since the COVIDx3 threshold is extremely low and the newer COVIDx8B data is considerably more reliable, we weighted the average in a way where the COVIDx8B threshold was assigned double the weight of the COVIDx3 threshold. The final combined optimal threshold was 0.22. Using this threshold during the classification of each of the base models and the whole ensemble increased the COVIDx3 test accuracy to 95.54 % with a sensitivity of 87.95 %, and the metrics also improved for COVIDx8B, where the test accuracy even reached 99.25 % as shown in Table 5.5. The evaluation of the ensemble on the COVIDx8B test set is summarized in Figure 5.12 where we find that all of the test images were classified correctly with the exception of 3 false negatives.



(a) Confusion matrix

(b) ROC curve

Figure 5.12: The confusion matrix and ROC curve that characterize our ensemble model's binary classification performance on the COVIDx8B test images.

5.8 Discussion

Our findings from the described experiments and results lead us to the conclusion that our proposed BaseNet prototype is able to achieve comparable performance to both the COVID-Net state-of-the-art model and other popular CNN architectures fine-tuned on our dataset. It is able to do so while having a very simple network structure, whose randomly initialized training stabilizes and begins to converge after only about 6 epochs. This is largely due to the training being paired with the Adam optimizer, which has adaptive learning rates for individual weights and uses a momentum term to speed up the convergence and escape local minima. Based on the evaluations on older COVIDx3 data, the COVID-Net CXR-2 has a distinctive advantage when it comes to its ability to generalize on other datasets. This is likely due to its unique architecture, which was specifically designed by the process of generative synthesis of deep neural networks with a focus on retaining performance on out of sample predictions. This has also proved to be evident when we used the PEPX block design pattern to re-create our own version of the proposed architectures because the CXR3-B2 had relatively consistent test accuracy on both assessed versions of the dataset. Our BaseNet prototype, as well as our fine-tuned models, lack this degree of consistency. However, we find the COVIDx8B evaluation to be more relevant as the authors of the dataset have issued a statement[19] that the current test set was specifically picked from the RICORD data repository, which is curated by the Radiological Society of North America and therefore has more reliable annotations in comparison to the older dataset versions.

The number of diverse CNN models provided by the Keras framework allowed us to compose an ensemble where the BaseNet was grouped with several other common architectures that we pre-trained on larger source datasets. The experiment demonstrated the power of ensemble models, where the majority has the opportunity to override errors made by individual base models. We also found that if the architecture is kept constant, it is beneficial to pre-train its weights on a more similar source dataset such as the ChestX-ray14 rather than the ImageNet, which belongs to a completely different domain.

During the analysis of the ROC curve and its AUC metric of the collective predictions, we discovered that lowering the classification threshold alleviates our models' problem of having inconsistent performance throughout different dataset versions. By lowering the threshold from 0.5 to 0.22, we created a classifier that not only surpasses the state-of-the-art on both COVIDx8B and COVIDx3 data but also delivers competitive results with traditional diagnostic techniques like the RT-PCR, which has been numerously measured to have a lower COVID-19 sensitivity. Lowering the threshold also has some implications in the context of medical diagnosis, where it increases the likelihood

[19] `https://github.com/lindawangg/COVID-Net/issues/159`

of detection. This may inevitably also increase the number of false positive results, but we do not consider those to be as detrimental as false negatives. In other words, telling an infected patient that they are healthy has much more severe consequences than telling a healthy patient that they have the virus, especially in the case of COVID-19, where identification and containment of the infected are crucial to preventing the virus from spreading. It is also important to note that while the ensemble model yields the best results, its memory requirements and inference time are much slower than those of an individual model. If the model was to be incorporated in a CAD system on hospital hardware, the requirements could be for a more light-weight solution. In such a case, our BaseNet prototype or the DenseNet-121 architecture would be the preferred choice.

We found that the COVIDx data is not particularly suitable for more complex image preprocessing techniques, as a simple pixel intensity normalization led to the best results. Other works described in our research have successfully used image preprocessing to enhance their results, but it seems that their datasets were more homogenous. The COVIDx gathers data from several diverse repositories and would require adaptive calculation of the algorithm's thresholds and parameters to work well in general. If the CNN method of automated diagnosis were to be utilized in an actual medical environment, we would suggest that applying only elementary preprocessing steps would ensure the model's consistency; otherwise, the variation in the positioning of patients and X-ray exposure intensities could cause problems. On the other hand, if we can expect a controlled environment, the diaphragm segmentation pipeline along with increased local contrast with CLAHE could prove to be useful in focusing the model on the lung markings that indicate the presence of the virus.

Finally, we discovered that balancing medical datasets is necessary, especially in the case of COVID-19, where the classes suffer from heavy imbalance. The best way to balance the data is by using cost sensitive learning, where each class is assigned a weight inversely proportional to its prevalence in the training data. If the model is trained for more extended periods of time, it has proved beneficial to additionally use online data augmentation in order to reduce the overfitting. A possible alternative could be using the minority class to train a GAN and then generating synthetic images from the minority class to increase its prevalence. We were unable to reach reasonable image quality with our DCGAN implementation, but we suspect that given more computational resources and perhaps a more modern GAN architecture such as StyleGAN [121] or CycleGAN [122] could provide results that truly capture the distinctive features of the various classes.

Conclusion

The main focus of this thesis was to research the possibilities of using convolutional neural networks to detect COVID-19 in X-ray images and to implement a prototype model for performing this task on open datasets available online. We first described the disease and the current methods used to diagnose it as well as their potential drawbacks. In the second chapter, we covered the basics of machine learning theory and defined the necessary terms needed for understanding the proposed solutions in the latter part of the thesis. The following chapter was dedicated to describing the application of neural networks in performing detection tasks in medical imaging. We started by exploring the various methods used for X-ray image preprocessing and resolving issues commonly found within medical datasets. Next, we analyzed existing architectures that are frequently used for similar tasks, and finally, we described existing research on COVID-19 detection that brought relevant insights to our own experimentation. Chapters 4 and 5 describe our own experimentation setup as well as the individual experiments and their results. We designed a comprehensive set of experiments that cover all of the essential steps in solving the task of COVID-19 detection using convolutional neural networks, such as image preprocessing and class balancing techniques, application of different optimization algorithms and architecture designs, and utilization of the collective predictive power of model ensembles. Each experimental configuration was evaluated on a state-of-the-art dataset that combines several open data repositories, and the results were discussed, interpreted, and compared to previous work.

Contribution

By prototyping a basic convolutional neural network and optimizing specific hyperparameters of its structure, we were able to build an architecture that delivers excellent results while remaining relatively simple and compact. We name this light-weight prototype the BaseNet, and it achieves test set accuracies of up to 95.50 % with a COVID-19 sensitivity of 93.00 % and specificity of 98.00 %. These are comparable to results reported by a number of studies surrounding the traditional diagnosis using RT-PCR in a laboratory environment. In addition, our automated method of diagnosis provides an alternative that is more time-efficient, accessible, and less costly.

In order to maximize performance, we proceeded to create an ensemble of the BaseNet prototype grouped together with several fine-tuned CNN models. These are architectures that have previously established their dominance in computer vision contests and other detection tasks, many of which also belonged to the medical domain. Using the collective predictions of such a composition of diverse models yielded results that surpass the state-of-the-art models assessed on the same dataset. More specifically, our proposed ensemble achieves a test accuracy of 99.25 % with a COVID-19 sensitivity of 98.50 % and specificity of 100.0 %, proving the point that deep learning has great potential in assisting medical professionals.

To promote further research on this topic and share our findings, we have published the implementation of our architectures and experiments on a public GitHub repository[20].

Future Improvements

Due to the time constraints and computational limitations, we were unable to explore several aspects of the COVID-19 detection task to a satisfactory level of detail. The following section proposes several ideas for future improvements that could enhance and further the research conducted as part of this thesis.

The CXR images that form the data used in training and testing the models often contain a lot of noise and unnecessary features that may be distracting to the feature classifier. Implementing a localization and segmentation method that would isolate the lung region and crop the image to its bounding box could simplify the task as well as resolve the loss of information that is often caused by resizing the images to the dimensions of the networks' input layers. We have attempted this to a certain degree by using standard image processing techniques to segment the high-intensity diaphragm region, but the algorithm was far from perfect on our data and using a deep learning approach to perform the segmentation would possibly prove to be more successful.

[20] `https://github.com/chododom/COVID-19-Detection`

As shown by [98], using twice transfer learning on previously established architectures can also lead to improved performance. We have used this approach with a DenseNet-121 model, which was first pre-trained on the ImageNet dataset and then the ChestX-ray14 dataset, where the feature extractor learned to focus on features found in CXR images. This improved the model's convergence and final results during its fine-tuning on our COVID-19 data. Applying this process to other architectures is another improvement that could be made in the future.

The next area to focus on could be balancing the minority COVID-19-positive class by generating synthetic images with generative adversarial networks. We observed a constant improvement in the images generated by our DCGAN implementation, but the training process used up too much time and resources to let it converge to an optimal point. We quickly achieved very promising results when we simplified the problem by reducing the image dimensions. This leads us to suspect that giving the training process more time or perhaps using modern GAN implementations such as StyleGAN or CycleGAN would lead to the generation of images with sufficient detail and anatomical accuracy to be used to oversample the positive class and enable the model to learn the discriminatory features using more samples.

As mentioned in Section 1.2, the linear opacities and ground-glass appearances found in the lungs may indicate the presence of a number of pulmonary viruses, no only the SARS-CoV-2. Consequently, it is often challenging to accurately classify the specific disease which could cause problems, as each requires a different treatment. We suggest that expanding the research conducted in this thesis to further explore categorical classification of various pulmonary diseases could be of great use.

Our final proposal for improving the reliability and usability of convolutional neural networks for the automated diagnosis of COVID-19 would be adding an explainability module. Such a module would mark the regions of the image that influenced the model's final decision, thereby allowing the user to validate whether the model is making the right decisions for the right reasons. An example of such a design was presented by the COVID-Net team, who leveraged the GSInquire explainability method for their models [77].

Bibliography

1. The World Health Organization, Regional office for the Eastern Mediterranean. *About COVID-19* [online]. 2020-03 [visited on 2021-02-15]. Available from: http://www.emro.who.int/health-topics/corona-virus/about-covid-19.html.

2. UDUGAMA, Buddhisha; KADHIRESAN, Pranav; KOZLOWSKI, Hannah N.; MALEKJAHANI, Ayden; OSBORNE, Matthew; LI, Vanessa Y.C.; CHEN, Hongmin; MUBAREKA, Samira; GUBBAY, Jonathan B.; CHAN, Warren C.W. *Diagnosing COVID-19: The Disease and Tools for Detection. ACS Nano.* 2020, vol. 14, no. 4. Available from DOI: 10.1021/acsnano.0c02624.

3. PADHYE, Nikhil S. *Reconstructed diagnostic sensitivity and specificity of the RT-PCR test for COVID-19. medRxiv.* 2020. Available from DOI: 10.1101/2020.04.24.20078949.

4. BISOFFI, Zeno; POMARI, Elena; PIUBELLI, Chiara; SILVA, Ronaldo; DEIANA, Michela; RONZONI, Niccolò; BELTRAME, Anna; BERTOLI, Giulia; RICCARDI, Niccolò; PERANDIN, Francesca; GOBBI, Federico; FORMENTI, Fabio; BUONFRATE, Dora. *Sensitivity, Specificity and Predictive Values of Molecular and Serological Tests for COVID-19: A Longitudinal Study in Emergency Room. Diagnostics.* 2020, vol. 10, no. 9. ISSN 2075-4418. Available from DOI: 10.3390/diagnostics10090669.

5. MILLER, Tyler E.; GARCIA BELTRAN, Wilfredo F.; BARD, Adam Z.; GOGAKOS, Tasos; ANAHTAR, Melis N.; ASTUDILLO, Michael Gerino; YANG, Diane; MAHOWALD, Grace K.; THIERAUF, Julia; FISCH, Adam S.; FITZPATRICK, Megan J.; NARDI, Valentina; FELDMAN, Jared; HAUSER, Blake M.; CARADONNA, Timothy M.; MARBLE, Hetal D.; RITTERHOUSE, Lauren L.; TURBETT, Sara E.; BATTEN, Julie; GEORGANTAS, Nicholas Zeke; ALTER, Galit; SCHMIDT, Aaron G.; HARRIS, Jason B.; GELFAND, Jeffrey A.; POZNANSKY, Mark C.; BERNSTEIN, Bradley E.; LOUIS, David N.; DIGHE, Anand;

CHARLES, Richelle C.; RYAN, Edward T.; PIERCE, Virginia M.; BRANDA, John A.; MURALI, Mandakolathur R.; IAFRATE, A. John; ROSENBERG, Eric S.; LENNERZ, Jochen K. *Clinical sensitivity and interpretation of PCR and serological COVID-19 diagnostics for patients presenting to the hospital. The FASEB Journal.* 2020, vol. 34, no. 10, pp. 13877–13884. Available from DOI: https://doi.org/10.1096/fj.202001700RR.

6. DINNES, Jacqueline; DEEKS, Jonathan J; BERHANE, Sarah; TAYLOR, Melissa; ADRIANO, Ada; DAVENPORT, Clare; DITTRICH, Sabine; EMPERADOR, Devy; TAKWOINGI, Yemisi; CUNNINGHAM, Jane; BEESE, Sophia; DRETZKE, Janine; FERRANTE DI RUFFANO, Lavinia; HARRIS, Isobel M; PRICE, Malcolm J; TAYLOR-PHILLIPS, Sian; HOOFT, Lotty; LEEFLANG, Mariska MG; SPIJKER, René; VAN DEN BRUEL, Ann. *Rapid, point-of-care antigen and molecular-based tests for diagnosis of SARS-CoV-2 infection. Cochrane Database of Systematic Reviews.* 2020, no. 8. ISSN 1465-1858. Available from DOI: 10.1002/14651858.CD013705.

7. Radiological Society of North America (RSNA) and American College of Radiology (ACR) [online]. 2020-06 [visited on 2021-02-15]. Available from: https://www.radiologyinfo.org/en/info.cfm?pg=chestrad.

8. CLEVERLEY, Joanne; PIPER, James; JONES, Melvyn M. *The role of chest radiography in confirming covid-19 pneumonia. BMJ.* 2020, vol. 370. Available from DOI: 10.1136/bmj.m2426.

9. HOSSEINY, Melina; KOORAKI, Soheil; GHOLAMREZANEZHAD, Ali; REDDY, Sravanthi; MYERS, Lee. *Radiology Perspective of Coronavirus Disease 2019 (COVID-19): Lessons From Severe Acute Respiratory Syndrome and Middle East Respiratory Syndrome. American Journal of Roentgenology.* 2020, vol. 214, no. 5, pp. 1078–1082. ISSN 0361-803X. Available from DOI: 10.2214/AJR.20.22969.

10. WU, Zunyou; MCGOOGAN, Jennifer M. *Characteristics of and Important Lessons From the Coronavirus Disease 2019 (COVID-19) Outbreak in China: Summary of a Report of 72 314 Cases From the Chinese Center for Disease Control and Prevention. JAMA.* 2020, vol. 323, no. 13, pp. 1239–1242. ISSN 0098-7484. Available from DOI: 10.1001/jama.2020.2648.

11. American College of Radiology. *ACR Recommendations for the use of Chest Radiography and Computed Tomography (CT) for Suspected COVID-19 Infection* [online]. 2020 [visited on 2021-02-15]. Available from: https://www.acr.org/Advocacy-and-Economics/ACR-Position-Statements/Recommendations-for-Chest-Radiography-and-CT-for-Suspected-COVID19-Infection.

12. MOHRI, Mehryar; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. *Foundations of Machine Learning*. The MIT Press, 2018. ISBN 026201825X.

13. ZHANG, Xian-Da. *Machine Learning. In: A Matrix Algebra Approach to Artificial Intelligence*. Springer, Singapore, 2020. Available from DOI: 10.1007/978-981-15-2770-8_6.

14. MISHRA, Aditya. *Metrics to Evaluate your Machine Learning Algorithm* [online]. Towards Data Science, 2020-05 [visited on 2021-02-22]. Available from: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

15. ŠIMUNDIĆ, Ana-Maria. *Measures of Diagnostic Accuracy: Basic Definitions. EJIFCC*. 2009, vol. 19, no. 4, pp. 203–211. ISSN 1650-3414. Available also from: https://pubmed.ncbi.nlm.nih.gov/27683318. PMC4975285[pmcid].

16. Google Developers. *Classification: ROC Curve and AUC; Machine Learning Crash Course* [online]. Google [visited on 2021-02-22]. Available from: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

17. PARK, Seong Ho; GOO, Jin Mo; JO, Chan-Hee. *Receiver operating characteristic (ROC) curve: practical review for radiologists. Korean journal of radiology*. 2004, vol. 5, no. 1, pp. 11–18. ISSN 1229-6929. Available from DOI: 10.3348/kjr.2004.5.1.11. 2004v5n1p11[PII].

18. KLOUDA, Karel; VAŠATA, Daniel. *Vytěžování znalostí z dat: Neuronové sítě* [online]. Czech Technical University in Prague, Faculty of Information Technology, 2021 [visited on 2021-03-05]. Available from: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-handout.pdf.

19. FORTMANN-ROE, Scott. *Understanding the Bias-Variance Tradeoff* [online]. 2012 [visited on 2021-02-23]. Available from: http://courses.washington.edu/me333afe/Bias_Variance_Tradeoff.pdf.

20. GROSSE, Roger. *Generalization* [online]. [N.d.] [visited on 2021-03-02]. Available from: https://www.cs.toronto.edu/~lczhang/321/notes/notes09.pdf.

21. KLOUDA, Karel; VAŠATA, Daniel. *Vytěžování znalostí z dat: Lineární regrese - pokračování, Hřebenová regrese* [online]. Czech Technical University in Prague, Faculty of Information Technology, 2021 [visited on 2021-02-23]. Available from: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-08-cs-handout.pdf.

22. KLOUDA, Karel; VAŠATA, Daniel. *Vytěžování znalostí z dat: Metoda nejbližších sousedů, křížová validace* [online]. Czech Technical University in Prague, Faculty of Information Technology, 2021 [visited on 2021-02-20]. Available from: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-05-cs-handout.pdf.

23. PURUSHOTHAM, Swarnalatha; TRIPATHY, B. K. *Evaluation of Classifier Models Using Stratified Tenfold Cross Validation Techniques*. In: KRISHNA, P. Venkata; BABU, M. Rajasekhara; ARIWA, Ezendu (eds.). *Global Trends in Information Systems and Software Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 680–690. ISBN 978-3-642-29216-3.

24. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

25. SAMMUT, Claude; WEBB, Geoffrey I. (eds.). *Leave-One-Out Cross-Validation*. In: *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, pp. 600–601. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_469.

26. KROGH, Anders; HERTZ, John. *A Simple Weight Decay Can Improve Generalization*. In: MOODY, J.; HANSON, S.; LIPPMANN, R. P. (eds.). *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1992, vol. 4. Available also from: https://proceedings.neurips.cc/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

27. BADILLO, Solveig; BANFAI, Balazs; BIRZELE, Fabian; SIEBOURG-POLSTER, Juliane; DAVYDOV, Iakov; HUTCHINSON, Lucy; KAMTHONG, Tony; STEIERT, Bernhard; ZHANG, Jitao David. *An Introduction to Machine Learning. Clinical Pharmacology & Therapeutics*. 2020, vol. 107. Available from DOI: 10.1002/cpt.1796.

28. FEURER, Matthias; HUTTER, Frank. *Hyperparameter optimization*. In: *Automated Machine Learning*. Springer, Cham, 2019, pp. 3–33.

29. BERGSTRA, James; BENGIO, Yoshua. *Random search for hyper-parameter optimization. Journal of machine learning research*. 2012, vol. 13, no. 2.

30. FLOREA, Adrian-Catalin; ANDONIE, Razvan. *Weighted Random Search for Hyperparameter Optimization. INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL*. 2019, vol. 14,

no. 2, pp. 154–169. ISSN 1841-9844. Available from DOI: 10.15837/ijccc.2019.2.3514.

31. KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. *Optimization by Simulated Annealing. Science*. 1983, vol. 220, no. 4598, pp. 671–680. ISSN 0036-8075. Available from DOI: 10.1126/science.220.4598.671.
32. SNOEK, Jasper; LAROCHELLE, Hugo; ADAMS, Ryan P. *Practical bayesian optimization of machine learning algorithms. arXiv preprint arXiv:1206.2944*. 2012.
33. BOCHINSKI, E.; SENST, T.; SIKORA, T. *Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms*. In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 3924–3928. Available from DOI: 10.1109/ICIP.2017.8297018.
34. BATTITI, Roberto; COLLA, Anna Maria. *Democracy in neural nets: Voting schemes for classification. Neural Networks*. 1994, vol. 7, no. 4, pp. 691–707.
35. GIACINTO, Giorgio; ROLI, Fabio. *Ensembles of neural networks for soft classification of remote sensing images*. In: *European symposium on intelligent techniques*. 1997, pp. 20–21.
36. HUANG, Y. S.; SUEN, C. Y. *A method of combining multiple experts for the recognition of unconstrained handwritten numerals. IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1995, vol. 17, no. 1, pp. 90–94. Available from DOI: 10.1109/34.368145.
37. PARK, Y.-S.; LEK, S. *Chapter 7 - Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling*. In: JØRGENSEN, Sven Erik (ed.). *Ecological Model Types*. Elsevier, 2016, vol. 28, pp. 123–140. Developments in Environmental Modelling. ISSN 0167-8892. Available from DOI: https://doi.org/10.1016/B978-0-444-63623-2.00007-4.
38. MÜLLER, Berndt; REINHARDT, Joachim; STRICKLAND, Michael T. *Neural Networks: An Introduction*. Springer Science & Business Media, 1995.
39. WERBOS, Paul. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Cambridge, MA, 1974. PhD thesis. Harvard University.
40. GRAUPE, Daniel. *Principles of Artificial Neural Networks*. 3rd. WORLD SCIENTIFIC, 2013. Available from DOI: 10.1142/8868.
41. RAMCHOUN, Hassan; AMINE, Mohammed; JANATI IDRISSI, Mohammed Amine; GHANOU, Youssef; ETTAOUIL, Mohamed. *Multilayer Perceptron: Architecture Optimization and Training. International Journal of Interactive Multimedia and Artificial Inteligence*. 2016, vol. 4, pp. 26–30. Available from DOI: 10.9781/ijimai.2016.415.

42. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

43. SATHYANARAYANA, Shashi. *A gentle introduction to backpropagation. Numeric Insight*. 2014, vol. 7, pp. 1–15.

44. RAWAT, Waseem; WANG, Zenghui. *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Computation*. 2017, vol. 29, no. 9, pp. 2352–2449. ISSN 0899-7667. Available from DOI: 10.1162/neco_a_00990.

45. TAN, H. H.; LIM, K. H. *Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization*. In: *2019 7th International Conference on Smart Computing Communications (ICSCC)*. 2019, pp. 1–4. Available from DOI: 10.1109/ICSCC.2019.8843652.

46. SOYDANER, Derya. *A Comparison of Optimization Algorithms for Deep Learning. International Journal of Pattern Recognition and Artificial Intelligence*. 2020, vol. 34, no. 13, p. 2052013. ISSN 1793-6381. Available from DOI: 10.1142/s0218001420520138.

47. GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019. ISBN 9781492032649.

48. PUNTAMBEKAR, Anand. *Strengths and Weaknesses of Optimization Algorithms Used for Machine Learning* [online]. The Startup, 2020-09 [visited on 2021-03-14]. Available from: https://medium.com/swlh/strengths-and-weaknesses-of-optimization-algorithms-used-for-machine-learning-58926b1d69dd.

49. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Available from arXiv: 1412.6980 [cs.LG].

50. DOZAT, Timothy. *Incorporating nesterov momentum into adam*. 2016.

51. SHARMA, Sagar. *Activation functions in neural networks. towards data science*. 2017, vol. 6.

52. RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. *Searching for Activation Functions. CoRR*. 2017, vol. abs/1710.05941. Available from arXiv: 1710.05941.

53. LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. *Deep learning. Nature*. 2015, vol. 521, no. 7553, pp. 436–444. ISSN 1476-4687. Available from DOI: 10.1038/nature14539.

54. ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. *Understanding of a convolutional neural network*. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. Available from DOI: 10.1109/ICEngTechnol.2017.8308186.

55. LI, Fei-Fei; KRISHNA, Ranjay; XU, Danfei. *Convolutional Neural Networks for Visual Recognition* [online]. GitHub, [n.d.] [visited on 2021-03-15]. Available from: `https://cs231n.github.io/convolutional-networks/`.

56. YAMASHITA, Rikiya; NISHIO, Mizuho; DO, Richard Kinh Gian; TOGASHI, Kaori. *Convolutional neural networks: an overview and application in radiology. Insights into Imaging*. 2018, vol. 9, no. 4, pp. 611–629. Available from DOI: `10.1007/s13244-018-0639-9`.

57. MURUGAN, Pushparaja; DURAIRAJ, Shanmugasundaram. *Regularization and Optimization strategies in Deep Convolutional Neural Network. CoRR*. 2017, vol. abs/1712.04711. Available from arXiv: `1712.04711`.

58. ASSIRI, Yahia. *Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods*. 2020. Available from arXiv: `2001.08856 [cs.CV]`.

59. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research*. 2014, vol. 15, no. 56, pp. 1929–1958. Available also from: `http://jmlr.org/papers/v15/srivastava14a.html`.

60. SANTOSH, K. C.; ANTANI, S. *Automated Chest X-Ray Screening: Can Lung Region Symmetry Help Detect Pulmonary Abnormalities? IEEE Transactions on Medical Imaging*. 2018, vol. 37, no. 5, pp. 1168–1177. Available from DOI: `10.1109/TMI.2017.2775636`.

61. VAN GINNEKEN, Bram; KATSURAGAWA, Shigehiko; DOI, Kunio; HAAR ROMENY, Bart M ter; VIERGEVER, Max A. *Automatic detection of abnormalities in chest radiographs using local texture analysis. IEEE transactions on medical imaging*. 2002, vol. 21, no. 2, pp. 139–149.

62. JAEGER, Stefan; KARARGYRIS, Alexandros; CANDEMIR, Sema; FOLIO, Les; SIEGELMAN, Jenifer; CALLAGHAN, Fiona; XUE, Zhiyun; PALANIAPPAN, Kannappan; SINGH, Rahul K; ANTANI, Sameer, et al. *Automatic tuberculosis screening using chest radiographs. IEEE transactions on medical imaging*. 2013, vol. 33, no. 2, pp. 233–245.

63. CHAUHAN, Arun; CHAUHAN, Devesh; ROUT, Chittaranjan. *Role of gist and PHOG features in computer-aided diagnosis of tuberculosis without segmentation. PloS one*. 2014, vol. 9, no. 11, e112980.

64. GULSHAN, Varun; PENG, Lily; CORAM, Marc; STUMPE, Martin C.; WU, Derek; NARAYANASWAMY, Arunachalam; VENUGOPALAN, Subhashini; WIDNER, Kasumi; MADAMS, Tom; CUADROS, Jorge; KIM, Ramasamy; RAMAN, Rajiv; NELSON, Philip C.; MEGA, Jessica L.; WEBSTER, Dale R. *Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA*. 2016, vol. 316, no. 22, pp. 2402–2410. ISSN 0098-7484. Available from DOI: 10.1001/jama.2016.17216.

65. ESTEVA, Andre; KUPREL, Brett; NOVOA, Roberto A.; KO, Justin; SWETTER, Susan M.; BLAU, Helen M.; THRUN, Sebastian. *Dermatologist-level classification of skin cancer with deep neural networks. Nature*. 2017, vol. 542, no. 7639, pp. 115–118. ISSN 1476-4687. Available from DOI: 10.1038/nature21056.

66. BEJNORDI, Babak Ehteshami; VETA, Mitko; VAN DIEST, Paul Johannes; VAN GINNEKEN, Bram; KARSSEMEIJER, Nico; LITJENS, Geert; VAN DER LAAK, Jeroen AWM; HERMSEN, Meyke; MANSON, Quirine F; BALKENHOL, Maschenka, et al. *Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. Jama*. 2017, vol. 318, no. 22, pp. 2199–2210.

67. RAJPURKAR, Pranav; IRVIN, Jeremy; ZHU, Kaylie; YANG, Brandon; MEHTA, Hershel; DUAN, Tony; DING, Daisy Yi; BAGUL, Aarti; LANGLOTZ, Curtis; SHPANSKAYA, Katie S.; LUNGREN, Matthew P.; NG, Andrew Y. *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. CoRR*. 2017, vol. abs/1711.05225. Available from arXiv: 1711.05225.

68. BASSI, Pedro R. A. S.; ATTUX, Romis. *A Deep Convolutional Neural Network for COVID-19 Detection Using Chest X-Rays*. 2021. Available from arXiv: 2005.01578 [eess.IV].

69. KOHLI, Marc; PREVEDELLO, Luciano M.; FILICE, Ross W.; GEIS, J. Raymond. *Implementing Machine Learning in Radiology Practice and Research. American Journal of Roentgenology*. 2017, vol. 208, no. 4, pp. 754–760. ISSN 0361-803X. Available from DOI: 10.2214/AJR.16.17224.

70. PAL, K. K.; SUDEEP, K. S. *Preprocessing for image classification by convolutional neural networks*. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2016, pp. 1778–1781. Available from DOI: 10.1109/RTEICT.2016.7808140.

71. ALBERT, Benji. *Data preprocessing: Should we normalise images pixel-wise?* [Online]. Stack Exchange, Data Science, 2018-01 [visited on 2021-03-24]. Available from: `https://datascience.stackexchange.com/questions/26881/data-preprocessing-should-we-normalise-images-pixel-wise`.

72. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In: BACH, Francis; BLEI, David (eds.). *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 2015, vol. 37, pp. 448–456. Proceedings of Machine Learning Research. Available also from: `http://proceedings.mlr.press/v37/ioffe15.html`.

73. SAIZ, Fatima; BARANDIARAN, Iñigo. *COVID-19 Detection in Chest X-ray Images using a Deep Learning Approach. International Journal of Interactive Multimedia and Artificial Intelligence*. 2020, vol. InPress, p. 1. Available from DOI: `10.9781/ijimai.2020.04.003`.

74. YORKSTON, John. *Understanding and Managing Noise Sources in X-ray Imaging* [online]. Carestream, 2020-06 [visited on 2020-03-24]. Available from: `https://www.carestream.com/blog/2020/04/21/understanding-and-managing-noise-sources-in-x-ray-imaging/`.

75. JIŘINA, Marcel; NOVÁK, Jakub; BRCHL, Lukáš. *Strojové vidění a zpracování obrazu: Filtrace v prostorové a frekvenční oblasti* [online]. Czech Technical University in Prague, Faculty of Information Technology, 2020-05 [visited on 2021-03-24]. Available from: `https://courses.fit.cvut.cz/BI-SVZ/lectures/files/bi-svz-07-filtrace-v-prostorove-a-frekvencni-oblasti.pdf`.

76. PARIS, Sylvain; KORNPROBST, Pierre; TUMBLIN, Jack; DURAND, Frédo. *Bilateral filtering: Theory and applications*. Now Publishers Inc, 2009.

77. WANG, Linda; WONG, Alexander. *COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest X-Ray Images*. 2020. Available from arXiv: `2003.09871 [eess.IV]`.

78. GAÁL, Gusztáv; MAGA, Balázs; LUKÁCS, András. *Attention U-Net Based Adversarial Architectures for Chest X-ray Lung Segmentation*. 2020. Available from arXiv: `2003.10304 [eess.IV]`.

79. REZA, Ali M. *Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement. Journal of VLSI signal processing systems for signal, image and video technology*. 2004, vol. 38, no. 1, pp. 35–44. ISSN 0922-5773. Available from DOI: `10.1023/B:VLSI.0000028532.53893.82`.

80. CHEN, Hsin-Jui; RUAN, Shanq-Jang; HUANG, Sha-Wo; PENG, Yan-Tsung. *Lung X-ray Segmentation using Deep Convolutional Neural Networks on Contrast-Enhanced Binarized Images. Mathematics.* 2020, vol. 8, no. 4. ISSN 2227-7390. Available from DOI: 10.3390/math8040545.

81. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. *U-net: Convolutional networks for biomedical image segmentation.* In: *International Conference on Medical image computing and computer-assisted intervention.* Springer International Publishing, 2015, pp. 234–241. ISBN 978-3-319-24574-4.

82. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross B. *Mask R-CNN. CoRR.* 2017, vol. abs/1703.06870. Available from arXiv: 1703.06870.

83. HEIDARI, Morteza; MIRNIAHARIKANDEHEI, Seyedehnafiseh; QIU, Yuchen; KHUZANI, Abolfazl Zargari; DANALA, Gopichandh; ZHENG, Bin. *Improving the performance of CNN to predict the likelihood of COVID-19 using chest X-ray images with preprocessing algorithms. International Journal of Medical Informatics.* 2020, vol. 144, p. 104284. ISSN 1386-5056. Available from DOI: https://doi.org/10.1016/j.ijmedinf.2020.104284.

84. MCINNES, Leland; HEALY, John; MELVILLE, James. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* 2020. Available from arXiv: 1802.03426 [stat.ML].

85. AHSAN, Mominul; KOWALSKI, Marcin; BASED, Md; HAIDER, Julfikar, et al. *COVID-19 Detection from Chest X-ray Images Using Feature Fusion and Deep Learning. Sensors.* 2021, vol. 21, no. 4, p. 1480. ISSN 1424-8220. Available from DOI: 10.3390/s21041480.

86. BRIA, Alessandro; MARROCCO, Claudio; TORTORELLA, Francesco. *Addressing class imbalance in deep learning for small lesion detection on medical images. Computers in Biology and Medicine.* 2020, vol. 120, p. 103735. ISSN 0010-4825. Available from DOI: https://doi.org/10.1016/j.compbiomed.2020.103735.

87. BUITINCK, Lars; LOUPPE, Gilles; PEDREGOSA, Fabian; BLONDEL, Mathieu; MUELLER, Andreas; GRISEL, Olivier; HOLT, Brian; NICULAE, Vlad; PRETTENHOFER, Peter; GRAMFORT, Alexandre; JOLY, Arnaud; GROBLER, Jaques; LAYTON, Robert; VANDERPLAS, Jake; VAROQUAUX, Gaël. *API design for machine learning software: experiences from the scikit-learn project.* In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning.* 2013, pp. 108–122. Available also from: https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html.

88. BUDA, Mateusz; MAKI, Atsuto; MAZUROWSKI, Maciej A. *A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks.* 2018, vol. 106, pp. 249–259. ISSN 0893-6080. Available from DOI: https://doi.org/10.1016/j.neunet.2018.07.011.

89. GOODFELLOW, Ian J; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. *Generative adversarial networks. arXiv preprint arXiv:1406.2661.* 2014.

90. RADFORD, Alec; METZ, Luke; CHINTALA, Soumith. *Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.* 2015.

91. WAHEED, A.; GOYAL, M.; GUPTA, D.; KHANNA, A.; PINHEIRO, P. R.; AL-TURJMAN, F. *CovidGAN: Data Augmentation Using Auxiliary Classifier GAN for Improved Covid-19 Detection. IEEE Access.* 2020, vol. 8, pp. 91916–91923. Available from DOI: 10.1109/ACCESS.2020.2994762.

92. POOJARY, R.; PAI, A. *Comparative Study of Model Optimization Techniques in Fine-Tuned CNN Models.* In: *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA).* 2019, pp. 1–4. Available from DOI: 10.1109/ICECTA48151.2019.8959681.

93. DENG, J.; DONG, W.; SOCHER, R.; LI, L.; KAI LI; LI FEI-FEI. *ImageNet: A large-scale hierarchical image database.* In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009, pp. 248–255. Available from DOI: 10.1109/CVPR.2009.5206848.

94. RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATHY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael; BERG, Alexander C.; FEI-FEI, Li. *ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision.* 2015, vol. 115, no. 3, pp. 211–252. ISSN 1573-1405. Available from DOI: 10.1007/s11263-015-0816-y.

95. WANG, Xiaosong; PENG, Yifan; LU, Le; LU, Zhiyong; BAGHERI, Mohammadhadi; SUMMERS, Ronald M. *ChestX-ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017.

96. AL-WAISY, Alaa S.; AL-FAHDAWI, Shumoos; MOHAMMED, Mazin Abed; MAASHI, Mashael S.; ABDULKAREEM, Karrar Hameed; ARIF, Muhammad; MOSTAFA, Salama A.; GARCIA-ZAPIRAIN, Begonya. *COVID-CheXNet: hybrid deep learning framework for identifying COVID-19 virus in chest X-rays images. Soft Computing*. 2020. ISSN 1433-7479. Available from DOI: 10.1007/s00500-020-05424-3.

97. MANGAL, Arpan; KALIA, Surya; RAJGOPAL, Harish; RANGARAJAN, Krithika; NAMBOODIRI, Vinay; ARORA, Chetan; BANERJEE, Subhashis. *CovidAID: COVID-19 Detection Using Chest X-Ray*. 2020. Available from arXiv: 2004.09803 [eess.IV].

98. BASSI, Pedro R. A. S.; ATTUX, Romis. *A Deep Convolutional Neural Network for COVID-19 Detection Using Chest X-Rays*. 2021. Available from arXiv: 2005.01578 [eess.IV].

99. ZHANG, Quan. *Convolutional neural networks*. In: *Proceedings of the 3rd International Conference on Electromechanical Control Technology and Transportation*. 2018, pp. 434–439.

100. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks*. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, vol. 25, pp. 1097–1105. Available also from: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

101. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. In: *International Conference on Learning Representations*. 2015.

102. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition. CoRR*. 2015, vol. abs/1512.03385. Available from arXiv: 1512.03385.

103. CHOLLET, François et al. *Keras* [https://keras.io]. 2015.

104. LIU, Quan; FENG, Chen; SONG, Zida; LOUIS, Joseph; ZHOU, Jian. *Deep Learning Model Comparison for Vision-Based Classification of Full/Empty-Load Trucks in Earthmoving Operations. Applied Sciences*. 2019, vol. 9, no. 22, p. 4871. ISSN 2076-3417. Available from DOI: 10.3390/app9224871.

105. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott E.; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. *Going Deeper with Convolutions. CoRR*. 2014, vol. abs/1409.4842. Available from arXiv: 1409.4842.

106. CHOLLET, François. *Xception: Deep Learning with Depthwise Separable Convolutions. CoRR.* 2016, vol. abs/1610.02357. Available from arXiv: 1610.02357.

107. HUANG, Gao; LIU, Zhuang; VAN DER MAATEN, Laurens; WEINBERGER, Kilian Q. *Densely connected convolutional networks.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017, pp. 4700–4708. Available from arXiv: 1608.06993.

108. UMER, Muhammad; ASHRAF, Imran; ULLAH, Saleem; MEHMOOD, Arif; CHOI, Gyu Sang. *COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images. Journal of Ambient Intelligence and Humanized Computing.* 2021. ISSN 1868-5145. Available from DOI: 10.1007/s12652-021-02917-3.

109. WANG, Linda; LIN, Zhong Qiu; WONG, Alexander. *COVID-Net* [https://github.com/lindawangg/COVID-Net]. GitHub, 2021 [visited on 2021-04-05].

110. WONG, Alexander; SHAFIEE, Mohammad Javad; CHWYL, Brendan; LI, Francis. *FermiNets: Learning generative machines to generate efficient neural networks via generative synthesis. CoRR.* 2018, vol. abs/1809.05989. Available from arXiv: 1809.05989.

111. GUNRAJ, Hayden; WANG, Linda; WONG, Alexander. *COVIDNet-CT: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases From Chest CT Images. Frontiers in Medicine.* 2020, vol. 7, p. 1025. ISSN 2296-858X. Available from DOI: 10.3389/fmed.2020.608525.

112. WONG, Alexander; LIN, Zhong Qiu; WANG, Linda; CHUNG, Audrey G.; SHEN, Beiyi; ABBASI, Almas; HOSHMAND-KOCHI, Mahsa; DUONG, Timothy Q. *COVIDNet-S: Towards computer-aided severity assessment via training and validation of deep neural networks for geographic extent and opacity extent scoring of chest X-rays for SARS-CoV-2 lung disease severity.* 2020. Available from arXiv: 2005.12855 [eess.IV].

113. TAMBAD, Samarth; NANDWANI, Rohit; MCINTOSH, Suzanne K. *Analyzing programming languages by community characteristics on Github and StackOverflow.* 2020. Available from arXiv: 2006.01351 [cs.SE].

114. COHEN, Joseph Paul; MORRISON, Paul; DAO, Lan. *COVID-19 Image Data Collection.* 2020. Available from arXiv: 2003.11597 [eess.IV].

115. CHUNG, Audrey. *Figure 1 COVID-19 chest x-ray data initiative* [https://github.com/agchung/Figure1-COVID-chestxray-dataset]. GitHub, 2020 [visited on 2021-04-07].

116. CHUNG, Audrey. *Actualmed COVID-19 Chest X-ray Dataset Initiative* [https://github.com/agchung/Actualmed-COVID-chestxray-dataset]. GitHub, 2020 [visited on 2021-04-07].

117. Radiological Society of North America. *RSNA Pneumonia Detection Challenge* [online]. [N.d.] [visited on 2021-04-07]. Available from: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data.

118. CHOWDHURY, M. E. H.; RAHMAN, T.; KHANDAKAR, A.; ISLAM, K. R.; MAZHAR, R.; KADIR, M. A.; MAHBUB, Z. B.; KHAN, M. S.; IQBAL, A.; EMADI, N. A.; REAZ, M. B. I.; ISLAM, M. T. *Can AI Help in Screening Viral and COVID-19 Pneumonia? IEEE Access*. 2020, vol. 8, pp. 132665–132676. Available from DOI: 10.1109/ACCESS.2020.3010287.

119. TSAI, Emily B.; SIMPSON, Scott; LUNGREN, Matthew P.; HERSHMAN, Michelle; ROSHKOVAN, Leonid; COLAK, Errol; ERICKSON, Bradley J.; SHIH, George; STEIN, Anouk; KALPATHY-CRAMER, Jayashree; SHEN, Jody; HAFEZ, Mona; JOHN, Susan; RAJIAH, Prabhakar; POGATCHNIK, Brian P.; MONGAN, John; ALTINMAKAS, Emre; RANSCHAERT, Erik R.; KITAMURA, Felipe C.; TOPFF, Laurens; MOY, Linda; KANNE, Jeffrey P.; WU, Carol C. *Data from Medical Imaging Data Resource Center (MIDRC) - RSNA International COVID Radiology Database (RICORD) Release 1c - Chest x-ray, Covid+ (MIDRC-RICORD-1c)*. 2021. Available also from: https://doi.org/10.7937/91ah-v663.

120. LI, Lisha; JAMIESON, Kevin; DESALVO, Giulia; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. Journal of Machine Learning Research*. 2018, vol. 18, no. 185, pp. 1–52. Available also from: http://jmlr.org/papers/v18/16-558.html.

121. KARRAS, Tero; LAINE, Samuli; AILA, Timo. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. Available from arXiv: 1812.04948 [cs.NE].

122. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. Available from arXiv: 1703.10593 [cs.CV].

Appendix A

Acronyms

ANN Artificial Neural Network

AUC Area Under the Receiver Operating Characteristic Curve

CAD Computer-aided Diagnosis

CLAHE Contrast Limited Adaptive Histogram Equalization

CNN Convolutional Neural Network

COVID-19 Coronavirus Disease of 2019

CXR Chest X-Ray

DCGAN Deep Convolutional Generative Adversarial Network

GAN Generative Adversarial Network

ILSVRC ImageNet Large Scale Visual Recognition Challenge

ML Machine Learning

MLP Multi-layer Perceptron

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

TPR True Positive Rate

TNR True Negative Rate

UMAP Uniform Manifold Approximation and Projection

Appendix B

Contents of Enclosed SD Card

- `README.md` ..... markdown file with SD card contents description
- `README.pdf` ..... PDF file with SD card contents description
- `src` ..... implementation files and IPython notebooks
  - `model_architectures` ..... definitions of CNN and GAN models
  - `preprocessing` ..... image and dataset preprocessing files
  - `utils` ..... utilities and helper functions
- `thesis` ..... directory with the thesis
  - `thesis.zip` ..... LaTeX source codes and images used in thesis text
  - `BP_Chodounsky_Dominik_2021.pdf` ..... PDF with thesis text
- `environment.yml` ..... virtual environment specifications

Appendix C

Network Architectures

Table C.1: Summary of the Generator's architecture in our implementation of the DCGAN for generating 256×256 px colour images.

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 65536)             6553600
_________________________________________________________________
re_lu_5 (ReLU)               (None, 65536)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 32, 32, 256)       1638400
_________________________________________________________________
re_lu_6 (ReLU)               (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 64, 64, 128)       819200
_________________________________________________________________
re_lu_7 (ReLU)               (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 128, 128, 64)      204800
_________________________________________________________________
re_lu_8 (ReLU)               (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 256, 256, 32)      51200
_________________________________________________________________
re_lu_9 (ReLU)               (None, 256, 256, 32)      0
_________________________________________________________________
G (Conv2D)                   (None, 256, 256, 3)       2400
=================================================================
Total params: 9,269,600
Trainable params: 9,269,600
Non-trainable params: 0
_________________________________________________________________
```

Table C.2: Summary of the Discriminators's architecture in our implementation of the DCGAN for generating 256×256 px colour images.

```
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 256, 256, 256)     19456
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256, 256, 256)     0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 128, 128, 256)     1638656
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 256)     0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 64, 256)       1638656
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 64, 64, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 256)       1638656
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 256)       1638656
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 65536)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 65536)             0
_________________________________________________________________
D (Dense)                    (None, 1)                 65537
=================================================================
Total params: 6,639,617
Trainable params: 0
Non-trainable params: 6,639,617
_________________________________________________________________
```

Table C.3: Summary of our BaseNet prototype architecture.

Model: "BaseNet"

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 224, 224, 3)	0	
conv2d (Conv2D)	(None, 224, 224, 64)	4864	input[0][0]
conv2d_1 (Conv2D)	(None, 224, 224, 64)	102464	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0	conv2d_1[0][0]
dropout_3 (Dropout)	(None, 112, 112, 64)	0	max_pooling2d[0][0]
conv2d_2 (Conv2D)	(None, 112, 112, 128	73856	dropout_3[0][0]
conv2d_3 (Conv2D)	(None, 112, 112, 128	147584	conv2d_2[0][0]
conv2d_4 (Conv2D)	(None, 112, 112, 128	147584	conv2d_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0	conv2d_4[0][0]
dropout_4 (Dropout)	(None, 56, 56, 128)	0	max_pooling2d_1[0][0]
conv2d_6 (Conv2D)	(None, 56, 56, 256)	819456	dropout_4[0][0]
conv2d_7 (Conv2D)	(None, 56, 56, 256)	1638656	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 256)	0	conv2d_7[0][0]
dropout_5 (Dropout)	(None, 28, 28, 256)	0	max_pooling2d_3[0][0]
conv2d_8 (Conv2D)	(None, 28, 28, 256)	1638656	dropout_5[0][0]
conv2d_9 (Conv2D)	(None, 28, 28, 256)	1638656	conv2d_8[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 256)	0	conv2d_9[0][0]
dropout_6 (Dropout)	(None, 14, 14, 256)	0	max_pooling2d_4[0][0]
conv2d_10 (Conv2D)	(None, 14, 14, 128)	295040	dropout_6[0][0]
conv2d_11 (Conv2D)	(None, 14, 14, 128)	147584	conv2d_10[0][0]
conv2d_12 (Conv2D)	(None, 14, 14, 128)	147584	conv2d_11[0][0]
conv2d_5 (Conv2D)	(None, 56, 56, 128)	409728	dropout_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 128)	0	conv2d_12[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 128)	0	conv2d_5[0][0]
dropout_7 (Dropout)	(None, 7, 7, 128)	0	max_pooling2d_5[0][0]
add (Add)	(None, 7, 7, 128)	0	max_pooling2d_2[0][0] dropout_7[0][0]
residual (Conv2D)	(None, 7, 7, 128)	409728	add[0][0]
conv2d_13 (Conv2D)	(None, 7, 7, 128)	409728	residual[0][0]
conv2d_14 (Conv2D)	(None, 7, 7, 128)	409728	conv2d_13[0][0]
dropout_8 (Dropout)	(None, 7, 7, 128)	0	conv2d_14[0][0]
GAP (GlobalAveragePooling2D)	(None, 128)	0	dropout_8[0][0]
flatten_3 (Flatten)	(None, 128)	0	GAP[0][0]
dense_6 (Dense)	(None, 512)	66048	flatten_3[0][0]
dropout_9 (Dropout)	(None, 512)	0	dense_6[0][0]
dense_7 (Dense)	(None, 1)	513	dropout_9[0][0]

Total params: 8,507,457
Trainable params: 8,507,457
Non-trainable params: 0