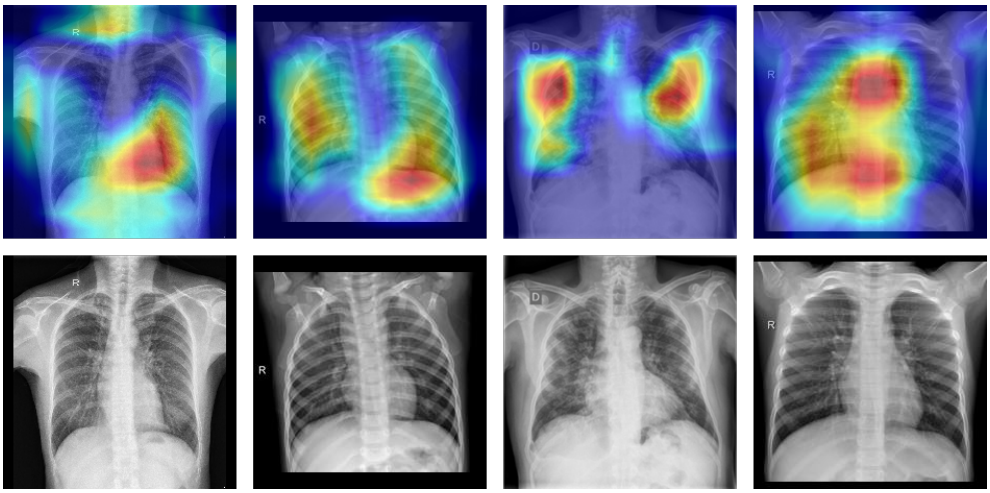
Robust Tuberculosis Detection in Chest X-ray Images Using a Convolutional Neural Network



Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East, Denmark
www.cs.aau.dk

Title

Robust Tuberculosis Detection in Chest X-ray Images Using a Convolutional Neural Network

Project type

8^{th} Semester Project:
Reliable Innovative Systems

Project period

Spring 2022

Project group

6

Participants

Nicolai Harbo Christensen
Simon Eliasen
Jakob Rønnest Sørensen
Dennis Tran
Kim Nguyen
Thomas Vinther

Supervisor

Martin Zimmermann

Number of pages: 102
Date of completion: May 25, 2022

Abstract

Tuberculosis (TB) is one of the leading causes of death by an infectious disease. It is most prominent in developing countries due to a lack of expertise. As such, this project investigates the use of Machine Learning (ML) for TB detection in Chest X-ray (CXR) images, which could be used in lieu of radiologists in developing countries, and furthermore, be used to aid radiologists in developed countries.
Most research only considers TB detection in a simplified setting using CXR images of only TB or healthy lungs. For this reason, this project examines the development of a robust ML solution, that is capable of performing well in a more nuanced and realistic setting, where other similar diseases are present. Furthermore, semantic segmentation using U-Net is explored, different data augmentation methods are evaluated, and Class Activation Maps is implemented to enable visualization of the solution's findings.
The project considers five state-of-the art Convolutional Neural Network architectures – VGG, Inception, DenseNet, ResNeXt and EfficientNet – where VGG showed to be the best performer.
The developed VGG model is compared to a baseline, that has been trained in a simplified setting akin to current papers. With a test set containing TB and similar diseases, the developed VGG model showed to significantly outperform the baseline in accuracy, precision, true positive rate, false positive rate, F1 score and AUC.

Preface

This semester project was written by a 8^{th} semester Computer Science project group at Aalborg University.

References follow the Harvard standard. Further reference information can be found in the bibliography provided in the end. The bibliography is sorted alphabetically by author. However, if the author is unknown, the publisher is used instead. If the reference is a website, the latest visiting date is also included. Figures and tables are sorted according to the chapters, e.g. the first figure in chapter 2 is labelled 2.1. Every figure found externally will have a source reference. Figures without a reference are created by the authors themselves. Numbers are written using periods as decimal separators and spaces as thousand separators.

Disclaimer: The authors of this project are not medical students or qualified doctors.

All source code is available in the following repository:
https://github.com/simoneliasen/x-map

All experiment results are available at:
https://wandb.ai/thebigyesman/X-map/sweeps

The authors of this project are:

Nicolai Harbo Christensen	Simon Eliasen	Jakob Rønnest Sørensen
Dennis Tran	Kim Nguyen	Thomas Vinther

Contents

Nomenclature

Abbreviations

TB	*Tuberculosis*
CXR	*Chest X-Ray*
CT	*Computerized Tomography*
MTB	*Mycobacterium Tuberculosis*
AFB	*Acid-Fast Bacilli*
ML	*Machine Learning*
SGD	*Stochastic Gradient Descent*
MLP	*Multilayer Perceptron*
CNN	*Convolutional Neural Network*
Tanh	*Hyperbolic Tangent*
ReLU	*Rectifier Linear Unit*
GAP	*Global Average Pooling*
CAM	*Class Activation Map*
IOU	*Intersection Over Union*
MOI	*Manifold of Interest*
VGG	*Visual Geometry Group*
FC	*Fully Connected*
FLOPS	*Floating Point Operations Per Second*
PPV	*Positive Predictive Value*
TPR	*True Positive Rate*
TNR	*True Negative Rate*
FPR	*False Positive Rate*
FNR	*False Negative Rate*
TP	*True Positive*
TN	*True Negative*
FP	*False Positive*
FN	*False Negative*
AUC-ROC	*Area Under the Curve of the Receiver Operating Characteristics*

Mathematical Notation

x	A scalar (integer or real)
$\mathbf{x}$	A vector
$\mathbf{X}$	A Matrix
$\mathbb{X}$	A set

1 Introduction

In 2020, Tuberculosis (TB) was one of the leading causes of death by a single infectious disease (among HIV-negative people). With 1.3 million deaths, it outranks HIV/AIDS, and is only surpassed by COVID-19. In this day and age, TB is largely curable, where 85% are able to recover with a six-month prescribed drug regimen. However, that is only the case if it is detected, which is usually done using Chest X-Ray (CXR) images by radiologists. Unfortunately, TB is primarily dominant in developing countries, and especially in rural areas, where there is a lack of expertise – in particular radiologists – to identify it [The World Bank, 2022; WHO, 2021, p. 11]. For this reason, TB largely goes untreated. The high demand for radiologists spans worldwide [AAMC, 2020]. Thus, doctors from poorer countries emigrate to richer countries, causing a brain drain, leaving poorer countries, with even less doctors. As an example, 50% of all Kenyan doctors now are practicing overseas [USAID, 2014, p. 28]. Due to the large societal toll enacted by a curable and preventable disease, WHO and the United Nations have set a goal of ending TB by 2039 [UN, 2020; WHO, 2021]. In this regard, a Machine Learning (ML) solution could be trained and used to detect TB using CXR images in lieu of radiologists in developing countries.

Medical diagnostics through CXR images using ML has shown to outperform or perform on par with human experts [Jonathan G. Richens and Johri, 2020]. Thus, besides having the potential to help developing countries, an ML solution also has promising prospects in developed countries. With human experts there is naturally a variation in what conclusions are drawn from CXR images, and human expert are undoubtedly subject to fatigue due to heavy workloads [Ul Abideen et al., 2020]. Currently, around 5% of adult US patients are expected to experience diagnostic errors [Singh et al., 2014]. Misclassification of TB can have negative repercussions, as it can lead to wrong medication and even worsening of health conditions. An ML solution could, on this matter, reduce the occurrences of misclassifications of TB in CXR images. Summarily, an ML solution could in developed countries aid the human experts, and as a result, reduce fatigue, interpretation differences, and misclassification. In doing so, the process of identifying TB could presumably also be expedited, save money, and ultimately save lives.

Looking at the common denominator of the accuracy metric from previous TB classification papers, which heavily utilize Convolution Neural Networks (CNNs), the metric was generally below 90% pre-2018 [Lopes and Valiati, 2017b; Rohilla et al., 2017b; Hooda et al., 2017b; Melendez et al., 2016]. In extension to the availability of larger datasets and a vast development in the field of ML, performance has improved greatly with approaches such as reducing the search space to the lungs only and generally more computational power. This enables more effective hyperparameter optimization and the usage of more

complex models. Performance in later years have reached up to 99% Accuracy [Abbas et al., 2020; Rahman et al., 2020].

Current research in using ML for TB detection in CXR images show promising results comparative to humans. In one particular survey, radiologists only correctly identified, 56% of CXR images with TB infected lungs, and 80% of the CXR images with non-TB infected lungs [Nash et al., 2020]. However, most research have only examined the problem in simplified settings, where only healthy lungs and TB infected lungs are considered (see Appendix A.2 for an overview of the 10 sampled papers). This is not reflective of real world settings, where other diseases are present. In this regard, pneumonia, lung cancer and COVID-19 have similar clinical symptoms and radiographic features as TB [Keikha and Esfahani, 2018; H. et al., 2018; Afum et al., 2022]. In particular, initial misclassifications of TB is 52% of the cases pneumonia and 20% of the cases lung cancer [Laushkina and Filimonov, 2012]. In light of all this, this project seeks to develop a well performing and robust ML solution that is correctly able to detect TB in a more nuanced and realistic setting with the presence of the three aforementioned similar diseases.

Tuberculosis 2

This chapter provides an overview of the most common manifestations of TB, as well as showcasing the different radiographic features that are used in order to identify instances of TB by radiologists. This is presented in order to better compare the visual interpretation of the computer vision tasks with actual radiographic imagery.

TB is an infectious disease, typically with pulmonary manifestation (occurrence in the lungs) most commonly caused by the bacteria, mycobacterium tuberculosis, which is spread through the air when an infected person coughs, sneezes or speaks.

Once a person encounters a bacterial invasion, and it develops into an infection, TB can either become active and develop into *primary TB*, or become latent and thus, asymptomatic, which in immunocompromised patients have the ability to later reactivate and develop into *post-primary TB*. Typical symptoms of active TB are chronic coughing, blood-containing mucus, fever, night sweats, and severe weight loss.

Active TB is most often discovered using CXR images, after which additional examination is required for a final diagnosis. In CXR images typically one or two radiographic features are visible, although there are times, especially in the case of early infections, where CXR images can appear normal.

As previously mentioned, there is a distinction between *primary* and *post-primary* TB. Primary infection can be anywhere in the lung in children, whereas the presence of infection typically takes place in the upper or lower zones in adults. Post-primary infections are typically placed in the upper zones of the lungs. Primary TB has historically been considered most prevalent in infants and children under the age of five. On the contrary, post-primary TB is mostly prevalent in adults, but due to the lack of exposure during childhood in western countries, adults only account for 23-34% of primary TB cases. Due to this, the disease is often not considered in adults, and can lead to misdiagnosis [Burrill et al., 2007]. In the next section the different radiographic features, that might suggest active TB, are presented.

2.1 Radiographic Features

The following section presents the primary radiographic abnormalities used to suggest further treatment for TB or to rule it out. The radiographic features for both primary and post-primary TB is presented together, due to a high overlap of features and a high difficulty in distinguishing the two at times. Thus, the focus is on confirming the diagnosis, rather than the subtype, as this will still allow for the initial step towards a correct treatment. When defining the different radiographic features, two identical pictures will

be presented, one with the feature highlighted and one without. The CXR images are posteroanterior, taken from back-to-front. Thus, as an example, when describing the left lung, it will be presented on the right. For reference, Figure 2.1 showcases a healthy lung.

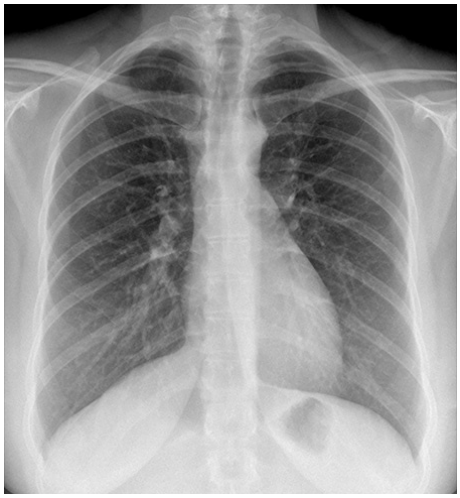


Figure 2.1. CXR image of healthy lungs [Lloyd-Jones, 2022].

Consolidation

Consolidation in the lungs refers to the airways of one's lungs being occupied by something else. For primary TB this consolidation can be found anywhere in the lungs, ranging from too small to detect, to patches or even lobar consolidation – taking up an entire section of the lung. It can also be found in post-primary TB, but is less common. The typical appearance of post-primary TB is patchy consolidation or poorly defined lines and node opacities on the CXR image, as shown in Figure 2.2 [Burrill et al., 2007].

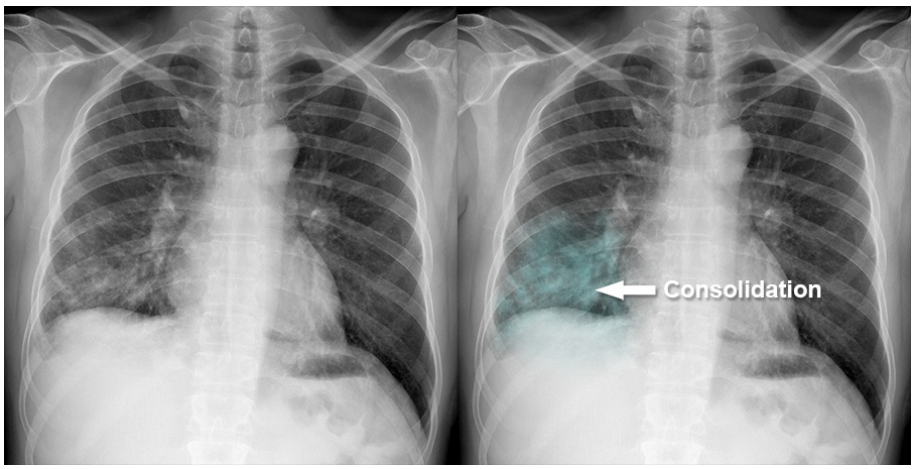


Figure 2.2. CXR image with patchy consolidation in the right lower zone [Lloyd-Jones, 2022].

In Figure 2.2, a patchy consolidation can be seen in the right lower zone of the lungs, which is primarily indicative of post-primary TB, but might also classify under the description of primary TB, as previously stated.

Lymphadenopathy

Lymphadenopathy refers to an abnormal size or consistency of the lymph nodes, caused by a disease in them. This feature is seen in approximately 96% of children with primary TB, but only 43% of adults with primary TB. For post-primary TB, they are only observed a third of the time [Burrill et al., 2007]. In Figure 2.3, an enlarged lymph node can be observed, presumably due to lymphadenopathy, as well as consolidation in the upper right lung.

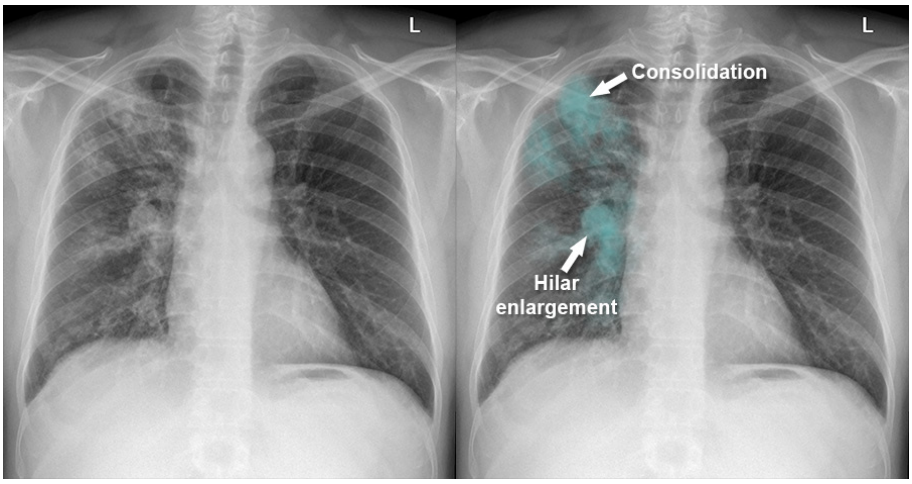


Figure 2.3. CXR image with enlargement of lymph node (hilar) and consolidation [Lloyd-Jones, 2022].

Pleural effusion

Pleural effusion refers to a built-up of excess fluid between the layers of pleura – that surrounds the outside of the lungs – and the lungs. The pleura is a thin membrane covering the lungs that help the lungs lubricate and facilitates breathing. Pleural effusion is seen in approximately one-fourth of patients with primary TB and approximately 18% of patients with post-primary TB [Burrill et al., 2007]. From Figure 2.3, it can be observed that almost the entire pleura of the left lung is filled with excessive liquid, it should be noted that this is quite an extreme case and that occurrences are often a lot smaller than in Figure 2.3.

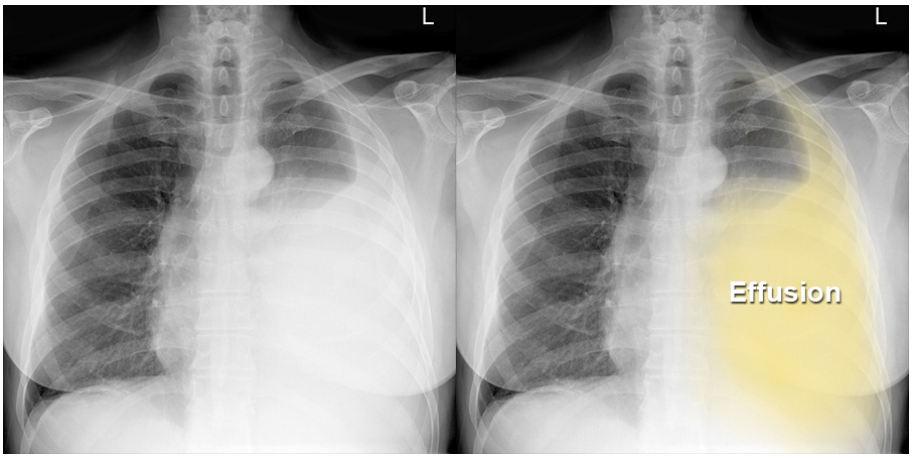


Figure 2.4. CXR image with pleural effusion [Lloyd-Jones, 2022].

Tuberculomas

Tuberculomas or tuberculous granulomas are well-defined lumps located in the lungs, as a result of TB infection, and are well-known to be hard to distinguish from radiographic features associated with lung cancer. Tuberculomas are associated with post-primary cases, but only occur in about 5% of cases. They appear as well-defined round masses, usually only a single one in 80% of occurrences, and can measure up to 4 cm in size. In the remaining 20% of cases with tuberculomas, a multitude of them are detected[Burrill et al., 2007]. Figure 2.5 showcases a case with a clear, well-defined area of high opacity located in the left lung, which correlates with the presence of a tuberculoma.

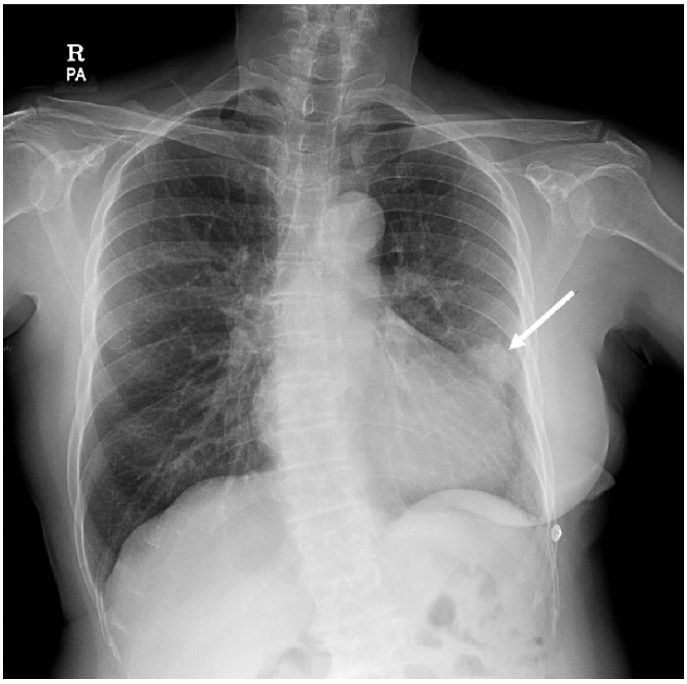


Figure 2.5. CXR image with a single tuberculoma [Lloyd-Jones, 2022].

Cavitation

Cavitation refers to dead or destructive tissue being broken down. In TB this is due to the breakdown of a tuberculoma, which usually eventually calcifies. Cavitation occurs in 20-45% of post-primary cases and in 10-30% of primary TB cases [Burrill et al., 2007]. Figure 2.6 showcases a cavity caused by cavitation, which is a gas-filled space in the lung surrounded by a border as a result of a TB infection.

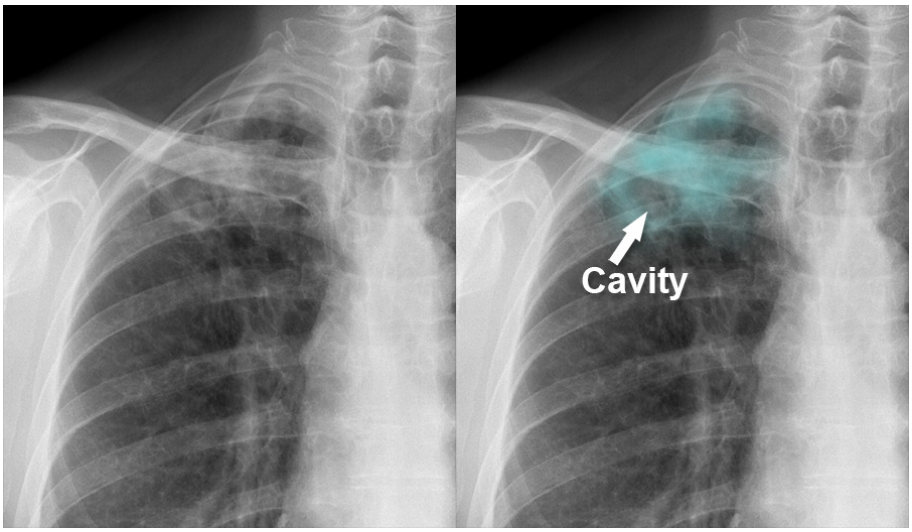


Figure 2.6. CXR image with cavitation [Lloyd-Jones, 2022].

Miliary Disease

Miliary disease can follow both primary and post-primary TB, but is rare and only affect 1-7% of patients with all forms of TB. Thus, often resulting in a poor prognosis. Miliary disease is reflective of an uncontrolled TB infection, where 1-3 mm diameter nodules which are uniform in size, are distributed throughout the whole body, but often detected through CXR as it is easier to capture [Burrill et al., 2007]. Figure 2.7 shows the nodules distributed throughout the patient's body in a CXR image. With proper treatment, the nodules should resolve within two to five months, without any lasting damage [Burrill et al., 2007].

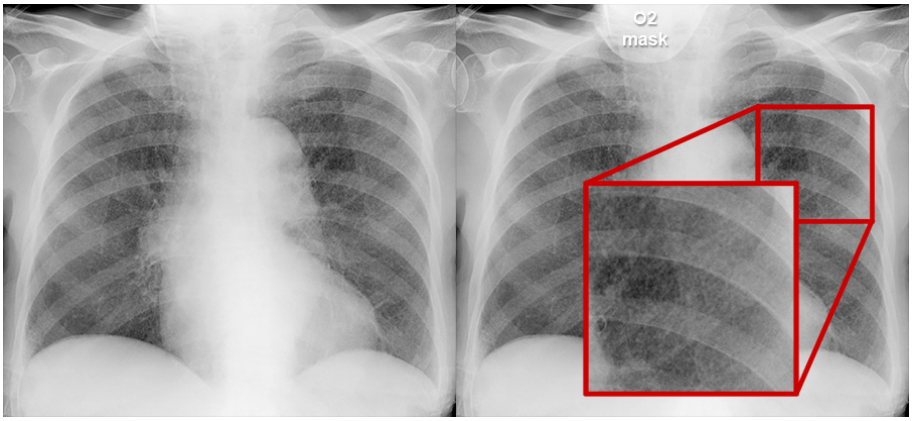


Figure 2.7. Uniform miliary nodules, distributed throughout both lungs [Lloyd-Jones, 2022].

2.2 Identification and Treatment Guidelines

The following section is based on WHO's "Chest Radiography in Tuberculosis Detection" [WHO, 2016] and follows the structural path of Figure 2.8.

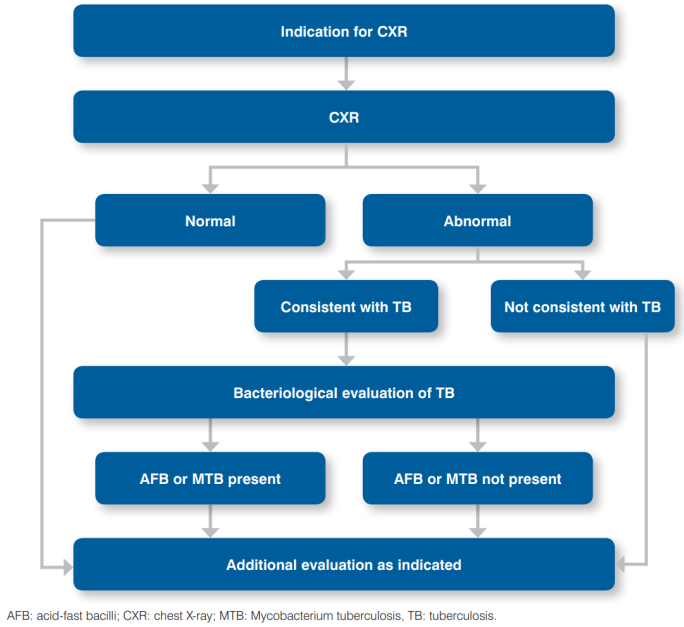


Figure 2.8. WHO recommendations and guidance on programmatic approaches to TB detection in CXR images [WHO, 2016].

In an effort to streamline the detection of TB, WHO outlines a programmatic approach to achieve efficient and correct diagnostics. The foundation of the approach is using CXR images, as they are well suited to highlight pulmonary TB. However, TB detection using CXR images lack exactness, due to a large majority of the radiographic abnormalities being similar with other pulmonary diseases. Thus, CXR images are used indicatively. They are followed up with a bacteriological confirmation, where a bacterial culture is gathered from the patient and is analyzed through a Mycobacterium tuberculosis(MTB) assay and an Acid-Fast Bacilli(AFB) smear microscopy. The MTB in short studies the bacterial culture on a molecular level and the AFB smear microscopy is done through a microscopy, by identifying Mycobacterium Tuberculosis colonies. If either of the tests come back positive, the patient is given the TB diagnosis.

The recommended treatment regimen for TB is isoniazid, rifampicin, pyrazinamide, ethambutol and streptomycin, with a variation in dosage depending on the patient being either a child or adult, and if the treatment is consumed daily or three times weekly. Furthermore, treatment is categorized based on if the patient previously has been treated or untreated, as drug resistance is more likely to occur in previously treated cases. The drug regimen primarily last six months and consists of an initial and continuation phase, which typically each last three months. It distributes out the previously mentioned five treatments according to what group of classification that the patient might fall in to, according to the previously mentioned characteristics [WHO, 2008].

Multilayer Perceptron 3

Currently, Machine Learning (ML) methods – and in particular Convolutional Neural Networks (CNNs) – are predominantly used in the domain of image processing and medical diagnostics. In this regard, Multilayer Perceptrons (MLPs) have played a fundamental role in forming the basis of CNNs. As such, this chapter provides an overview of MLPs and introduces the preliminary knowledge needed in order to understand CNNs, which are subsequently presented in chapter 4. This chapter is a direct excerpt from the authors' previous 7^{th} semester project, Christensen et al. [2021, p. 20-25], with some alterations.

An MLP have loosely been inspired by the findings of neuroscience. Roughly speaking, it can be viewed as a superficial and largely simplified mathematical model of the biological brain [Goodfellow et al., 2016, p. 169]. An MLP is a network of *fully connected* units, called *neurons*. In its essence, an MLP can be viewed as a directed acyclic graph arranged in layers, where information is only passed forward from one immediate layer to another. The architecture of an MLP is visualized in Figure 3.1.

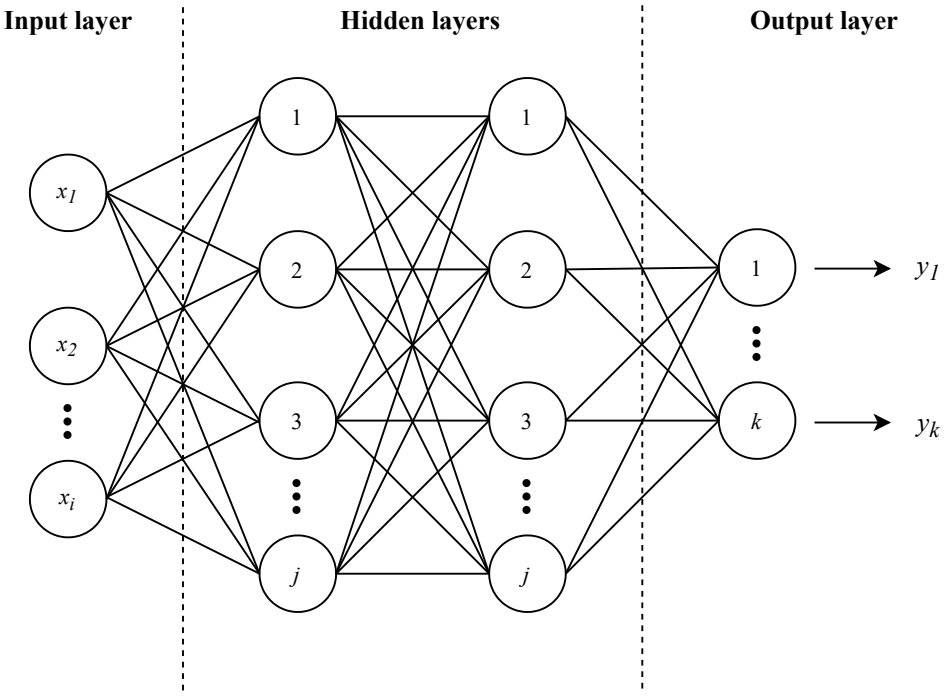


Figure 3.1. The architecture of an MLP with a *forward pass* from left to right. Each neuron has an activation, a, and each connection between two neurons has an associated weight, w, to define the connection's strength. The first layer is called the *input layer* as it takes in an input vector, $\boldsymbol{x}$. The last layer is called the *output layer* as it produces an output vector, $\boldsymbol{y}$. The intermediary layers are called *hidden layers*, and can consist of one or more layers. The figure is from one of the authors' previous works [Bonvang et al., 2020, p. 34].

3.1 Hidden Neurons

Each hidden neuron acts as a vector-to-scalar function, where each hidden neuron takes as input the weighted sum of activation values of all the neurons from the preceding layer along with a bias, b. E.g. for a hidden neuron, n_j, in the earliest hidden layer of the MLP shown in Figure 3.1, the input is [Russell and Norvig, 2010a, p. 727]:

$$in_j = \sum_{i=1}^{J} w_{i,j} \cdot a_i + b_j \tag{3.1.1}$$

Thereafter, in order to make the hidden neuron's activation relatively binary, in which it is either active or inactive, a differentiable non-linear *activation function*, h, is applied to Equation (3.1.1). As a result, the activation of a hidden neuron is expressed as [Russell and Norvig, 2010a, p. 727]:

$$a_j = h(in_j) \tag{3.1.2}$$

The introduction of non-linearity expands the MLP's modelling capacity, and makes it possible for MLPs to recognise patterns beyond linear relationships [Goodfellow et al., 2016, p. 169]. There are many types of activation function available, of which the three most commonly used are shown in Figure 3.2.

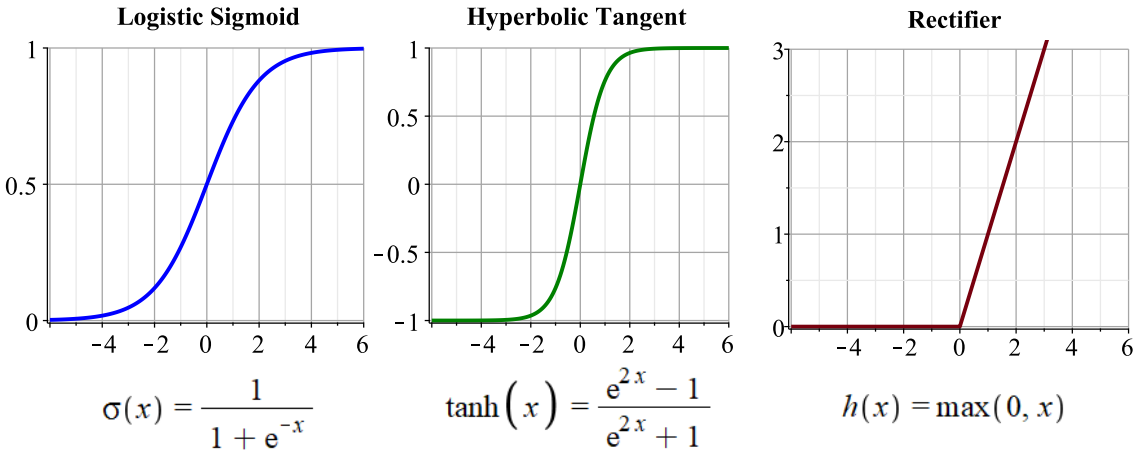


Figure 3.2. The three most commonly used activation functions for hidden neurons. The figure is from one of the authors' previous works [Bonvang et al., 2021, p. 10].

Naturally, the question which arises from this, is what activation function to use. In general, there is not a single ideal activation function. It heavily depends on the particular task, and finding the most suited for a given task is a process of trial-and-error [Goodfellow et al., 2016, p. 191 - 192]. However, two general points can be said.

First, non-linear activation functions with linear characteristics are usually more favored, as models are easier to train when their behaviour resembles linear models. For instance, between the two sigmoidal functions – the hyperbolic tangent (Tanh) and the logistic sigmoid – the Tanh usually performs better than the logistic sigmoid, despite them having similar curve characteristics. This is because the Tanh resembles the identity function more closely near $x = 0$, which makes it easier to train [Goodfellow et al., 2016, p. 194].

Second, activation functions that do not saturate usually perform better. An inherent downside of both the logistic sigmoid and the Tanh is that they both saturate; The logistic sigmoid is said to saturate at 0 and 1, as it maps large negative input values to 0 and maps large positive input values to 1. Similarly, the Tanh does this as well, but maps to -1 and +1 instead. Saturation can make learning difficult, as gradients will be close to zero, for large negative input values and large positive input values. This especially becomes a problem for deep models with many layers. When gradients are propagated back through many hidden layers, they will eventually vanish in size, making it unclear to know which direction the parameters should be adjusted in order to improve the objective function. As such, this problem is known as the *vanishing gradient problem* [Murphy, 2022, p. 423].

In light of the two mentioned points above, the Rectified Linear Unit (ReLU) is usually the most preferred among the three activation functions shown in Figure 3.2. This is due to mainly two reasons. First, among the three it bears the closest resemblance to a linear function, as it consists of two linear pieces. Second, it does not saturate for large positive input values, and as such, it is not as susceptible to the vanishing gradient problem.

3.2 Goal

The overall goal of any ML model is that it is capable of *generalizing* what is has learned to *unobserved inputs*, which it has not been trained with. This is done using a *test set*. The ML model has to appropriately fit in the middle between two ends of a spectrum, *underfitting* and *overfitting*, which concepts are illustrated in Figure 3.3.

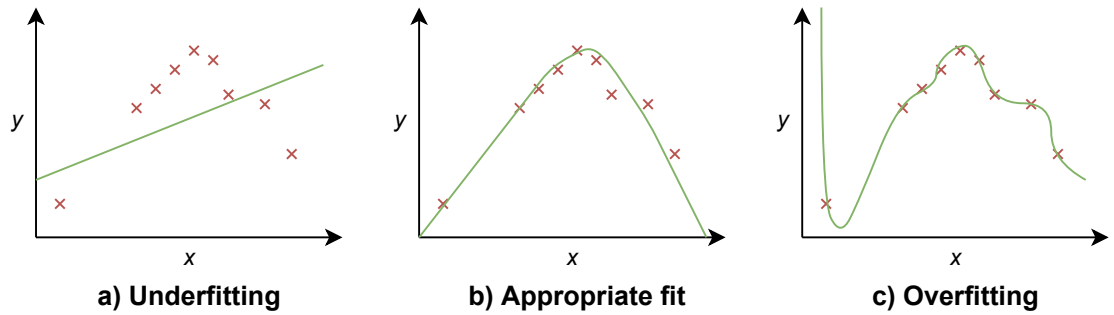


Figure 3.3. Conceptualization of three types of fittings: Underfitting, an appropriate fit and overfitting. The figure is from one of the authors' previous works [Bonvang et al., 2021, p. 9].

With underfitting, the model is unable to fit the complexity of the given task, resulting in a generally incapable model with an unsatisfactory *training error* (error is also known as loss). On the other hand, with overfitting, the model is capable of fitting to the complexity of task. However, the model fits the training set too closely, and its generalization abilities suffers as a consequence. The gap between the training error and *generalization error* (also known as test error) is too large. Thus, a model does not perform well, if it solely has a low training error. Instead, it has to have a low generalization error as well [Goodfellow et al., 2016, p. 111].

Training Goal

From Figure 3.1, an MLP can be viewed as a series of functional transformations, which defines a deterministic mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$ [Goodfellow et al., 2016, p. 183]:

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{3.2.1}$$

As such, the goal of training an MLP is for it to learn the values of the parameter (weights and biases), $\boldsymbol{\theta}$, that produces the best function approximation. Exactly how the values of $\boldsymbol{\theta}$ are adjusted depends on the *objective function*, which is aimed to be either minimized or maximized. Maximization is achieved by minimizing $-f(\boldsymbol{x}; \boldsymbol{\theta})$.

In supervised learning, the goal is generally to minimize the objective function, where the principles of *maximum likelihood estimation* are applied. Maximum Likelihood Estimation (MLE) is an estimate of $\boldsymbol{\theta}$ where the empirical data is most likely to occur [Pishro-Nik, 2014, section 8.2.3]. Let $\mathbb{X} = \{x_1, ..., x_m\}$ be a sample drawn independent and identically distributed (i.i.d) from the true data distribution, $p_{\text{data}}(\boldsymbol{x})$, and let $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ be the model distribution which estimates $p_{\text{data}}(\boldsymbol{x})$. MLE is then defined as [Goodfellow et al., 2016, p. 131-132]:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \; p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{3.2.2}$$

$$= \arg\max_{\boldsymbol{\theta}} \; \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}_i; \boldsymbol{\theta}) \tag{3.2.3}$$

$$= \arg\max_{\boldsymbol{\theta}} \; \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}_i; \boldsymbol{\theta}) \tag{3.2.4}$$

$$= \arg\max_{\boldsymbol{\theta}} \; \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} [\log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})] \tag{3.2.5}$$

The product in Equation (3.2.3) can be prone to numerical underflow – a rounding error where numbers near zero are rounded to zero. For this reason, a logarithmic probability, Equation (3.2.4), is much more practical, as it conveniently transforms it into a sum instead, which also makes it faster to compute. Finally, MLE can be expressed as an expectation (by dividing with m) as shown in Equation (3.2.5) [Goodfellow et al., 2016, p. 132]. Most commonly for supervised learning, the goal is to estimate a conditional probability, where $\boldsymbol{y}$ has to be predicted given $\boldsymbol{x}$. The conditional MLE is expressed as [Goodfellow et al., 2016, p. 133]:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \; \mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \sim p_{\text{data}}} [\log p_{\text{model}}(\boldsymbol{y} | \boldsymbol{x}; \boldsymbol{\theta})] \tag{3.2.6}$$

With the principles of MLE, the goal of training an MLP can be viewed as an attempt to make the model distribution match the empirical distribution (defined by the training set). As a result, there is a need for quantifying the loss between them, in order to minimise their dissimilarities. In this regard, *cross entropy*, H, can be used as a measurement [Goodfellow et al., 2016, p. 179]:

$$H(p_{\text{data}}, p_{\text{model}}) = -\mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \sim p_{\text{data}}} [\log p_{\text{model}}(\boldsymbol{y} | \boldsymbol{x}; \boldsymbol{\theta})] \tag{3.2.7}$$

In light of this, finding the MLE is equivalent to minimizing the cross entropy. As such, in supervised learning the objective function, in a general sense, is the cross entropy between the two probability distributions, and the goal is to minimise it in order for the MLP to learn the appropriate values of the parameter $\boldsymbol{\theta}$.

3.3 Output Neurons

The particular form of the objective function depends on how the output should be represented, and the choice of how to represent the output depends on the problem that the MLP is tasked to solve. An MLP can be tasked to solve many different problems, of which the two most common are:

- Regression: Predict one or more numerical values from an input vector $\boldsymbol{x}$ [Bishop, 2006, p. 137].
- Classification: Assign an input vector $\boldsymbol{x}$ to one of K discrete classes [Bishop, 2006, p. 179].

Accordingly, the output layer has the purpose of providing a final transformation, such that the model's output coheres with the intended task [Goodfellow et al., 2016, p. 181].

If it is a regression problem, a linear activation function is usually used in the output layer. Here, no transformation is applied, and each output neuron's activation is simply their weighted sum. As the model predicts a real-value, the model can be considered to predict the mean of a Gaussian distribution. As such, the objective function is the cross entropy between the data distribution and a Gaussian distribution. In particular, the objective function J can be expressed as the Mean Squared Error (MSE) [Goodfellow et al., 2016, p. 132-134]:

$$J^{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2 \tag{3.3.1}$$

Where:

y_i	The predicted value
t_i	The observed target value
N	The number of training examples

On the other hand, if it is a classification problem, the MLP is tasked to output either a Bernoulli distribution (two classes) or Multinoulli distribution (more than two classes). For a binary classification problem the logistic sigmoid activation function is used, and only a single neuron in the output layer is needed, as a Bernoulli distribution is defined by a single probability. The objective function is, as a result, the cross entropy between the data distribution and a Bernoulli distribution, called the *binary cross entropy*.

For more than two classes, the softmax activation function is used to represent a probability distribution over n classes. Here, the number of output neurons corresponds to the number of classes. The softmax function is a generalization of the logistic sigmoid. It produces a vector, where each element is between 0 to 1, and the entire vector sums to 1 [Goodfellow

et al., 2016, p. 182, 184]. The objective function is, consequently, the cross entropy between the data distribution and a Multinoulli distribution, called the *categorical cross entropy*. Intuitively, a softmax activation function and a categorical cross entropy can also be used for a binary classification. However, one should keep in mind that such a model would require two output neurons, whereas a model with a logistic sigmoid and a binary cross entropy would only need one output neuron. Thus, using a softmax activation function and a categorical cross entropy for binary classification would require twice as many weights in the output layer.

3.4 Learning

Initially, the parameter $\boldsymbol{\theta}$ is set randomly, after which the values of $\boldsymbol{\theta}$ are adjusted iteratively. This is usually done by the use of either *gradient descent* or *gradient ascent* – both of which are iterative optimization algorithms. Gradient descent aims to minimize the objective function. As shown in Equation (3.3.1), this is achieved by minimizing the loss between the current output and its target value. In brief, gradient descent finds a local minimum by computing the gradient of the objective function with respect to $\boldsymbol{\theta}$. For each iteration, the values in $\boldsymbol{\theta}$ are then adjusted with a step of size α (also known as the learning rate) in the direction opposite of the gradient – the direction which minimises the objective function. In other words, a step is taking 'downhill' for each iteration. Conversely, gradient ascent does the exact opposite. Instead, a step is taken 'uphill' for each iteration, in order to find a local maximum [Goodfellow et al., 2016, p. 82-86].

As previously mentioned, an MLP can be viewed as a series of functional transformations. For instance, a four-layered MLP, as shown in Figure 3.1, can be presented as follows [Goodfellow et al., 2016, p. 168]:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_4(f_3(f_2(f_1(\boldsymbol{x})))) \tag{3.4.1}$$

with f_1 as the input layer, f_2 as the subsequent layer, and so forth. As such, the layers are chained together and the output layer, f_4, depends on all the layers prior to it. Deducing how f_4 depends on f_3 can be done by calculating the gradient of the objective function. After which, computing how f_4 depends on f_2 is done by the use of the *chain rule* of calculus. Subsequently, computing how f_4 depends on f_1 can be done by applying the chain rule again. Summarily, deducing how f_4 depends on all the layers prior to it, is a matter of recursively applying the chain rule. For this reason, the method for computing the gradients for the whole MLP is known as *backpropagation*, as the gradients are computed in a *backward pass*, from back to front [Goodfellow et al., 2016, p. 205-208].

In order for an MLP to become sufficiently capable in generalization, it has to be trained with a relatively large training set. As a result, gradient methods are usually quite computationally expensive and time-consuming, as a *training pass* consists of a forward pass with the whole training set, followed by a backward pass. As such, gradient methods have by and large been replaced with more computationally efficient sampling-based versions known as *stochastic gradient methods*. Here, a gradient is considered an expectation, which has statistically been estimated using a set of samples, known as a *mini-batch*. In order to obtain an unbiased estimate of the gradient, each sample has to be

drawn i.i.d. In practice, this is usually achieved by shuffling the training set once for each epoch – a full pass through the entire training set. Summarily, with *stochastic gradient methods* the computation time for a training pass is no longer bound to the size of the training set, but instead depends on the size of the minibatch [Goodfellow et al., 2016, p. 151-153, 280].

However, the random sampling inevitable introduces noise. As the gradient is estimated from a set of randomly picked samples, it will no longer converge directly towards a local optimum. Instead, it will converge in an oscillating manner, as shown in Figure 3.4.

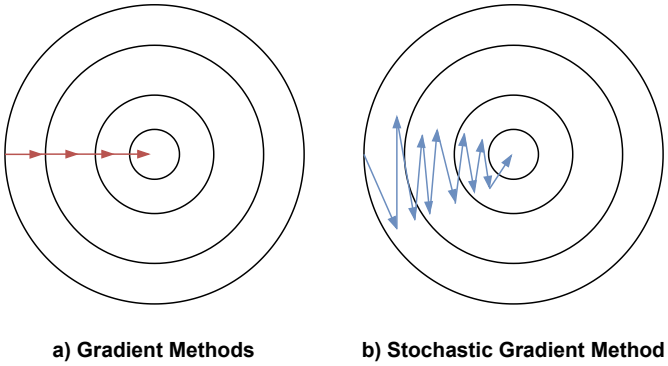


Figure 3.4. Conceptualization of the convergence behavior of gradient methods compared to stochastic gradient methods. The figure is modified from one of the authors' previous works [Bonvang et al., 2021, p. 39].

Consequently, the computational speed gained with stochastic gradient methods is at the expense of a slower convergence rate. Several methods exist to mitigate this issue [Goodfellow et al., 2016, p. 294]. One way to mitigate the noise, introduced by stochastic methods, is by using *momentum*, which, in short, is a method to accelerate learning when the convergence rate is impeded by noise. Furthermore, the learning rate has a substantial influence on the MLP's performance. A smaller learning rate will result in more precise weight adjustments, but at the cost of slower convergence. However, if it is too small the model is at risk of getting stuck at a suboptimal local optimum. In contrast, a larger learning rate will produce a faster convergence, but at the cost of accuracy. If it is too large, on the other hand, the model might never converge to any optimum [Goodfellow et al., 2016, p. 429]. One way to address this issue is by using a *learning rate scheduler*, that reduces the learning rate according to some form of schedule. An example could be a constant learning rate scheduler, that decays the learning rate by a constant factor for each epoch. Another way to address the learning rate issue, is through the use of stochastic gradient methods with *adaptive learning rates*, such as *RMSprop* or *Adam*, where each model parameter has its own learning rate, which is adjusted adaptively [Goodfellow et al., 2016, p. 306-308].

Convolutional Neural Networks 4

With the preliminaries of ML and CNNs outlined in chapter 3, this chapter ensuingly provides an overview of CNNs and why they are particularly suited for image processing. This chapter uses "Deep Learning" by Goodfellow et al. [2016] and "Probabilistic Machine Learning: An Introduction" by Murphy [2022] as its primary sources of reference.

The architecture of a CNN is visualized in Figure 4.1.

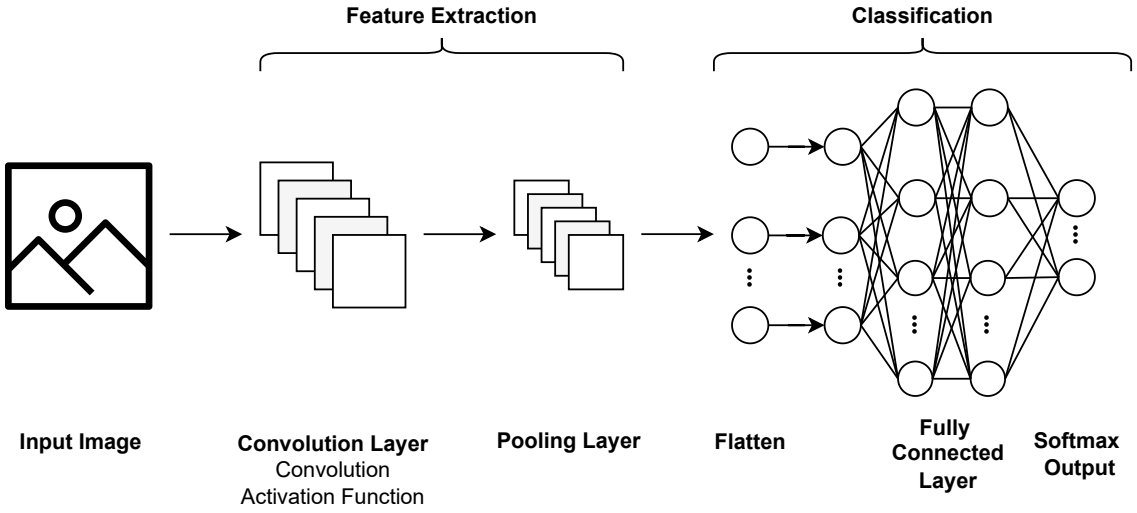


Figure 4.1. The architecture of a simple CNN with a convolutional layer, a pooling layer and a fully connected layer. The CNN takes an image as input, extracts features using a convolution layer and a pooling layer. Subsequently, it uses a fully connected layer for classification, where it outputs a probability distribution over n classes using a softmax activation function. However, for a binary classification a sigmoid activation function is most commonly used.

As seen in Figure 4.1, a CNN comprises of two main parts: Feature extraction and classification. The two feature extraction elements – Convolution layer and pooling layer – are mainly the concepts that make a CNN particularly suited for image processing. As such, these two concepts are first explored. Subsequently, the classification elements are presented. Finally, regularization strategies – methods for reducing overfitting – that are commonly used in CNN architectures are presented.

4.1 Convolutional Layer

The traditional MLPs, presented in Chapter 3, are not particularly suited for processing and recognising patterns in spatial data. With an MLP, a pattern learned at one location may not be recognized at any other locations. An MLP has an inherent inability to do this, as its network consists of tied weights, meaning each weight is only used once and not shared across locations. As a result, in order for an MLP to recognize the same pattern at different locations, it has to learn a different set of parameters for each location. To solve these problems, the core idea behind CNNs is to replace the matrix multiplication operation used in MLPs with a convolution operation [Murphy, 2022, p. 461].

Convolution

A convolution operation, denoted with *, is defined as the integral of the product of two functions, f and g, where the function g has been reversed and shifted with τ. For a one-dimensional space, the operation can be described as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{4.1.1}$$

The integral is evaluated for all values of shifts, thus, producing a third function, $(f * g)$, known as the convolution function. For integer values i, the discrete convolution operation can be expressed as:

$$(f * g)[i] = \sum_{n=-\infty}^{\infty} f[n]g[n-i] \tag{4.1.2}$$

Removing the negative integer indices and replacing the functions with finite-length vectors, $\mathbf{x}$ and $\mathbf{w}$, subsequently yields:

$$(\mathbf{x} * \mathbf{w})[i] = \sum_{n} x_n w_{i-n} \tag{4.1.3}$$

An example of a discrete convolution using Equation (4.1.3) is shown in Table 4.1.

	1	2	3	4		
6	5					$(\mathbf{x} * \mathbf{w})[1] = x_1 w_2 = 5$
	6	5				$(\mathbf{x} * \mathbf{w})[2] = x_1 w_1 + x_2 w_2 = 16$
		6	5			$(\mathbf{x} * \mathbf{w})[3] = x_2 w_1 + x_3 w_2 = 27$
			6	5		$(\mathbf{x} * \mathbf{w})[4] = x_3 w_1 + x_4 w_2 = 38$
				6	5	$(\mathbf{x} * \mathbf{w})[5] = x_4 w_1 + x_5 w_2 = 24$

Table 4.1. The discrete convolution operation of two vectors: An input vector $\mathbf{x} = [1, 2, 3, 4]$ and a weight vector $\mathbf{w} = [5, 6]$, known as a kernel. The discrete convolution operation consists of reversing $\mathbf{w}$, and then step-wise moving it from left to right over $\mathbf{x}$. At each step, values of $\mathbf{x}$ within the kernel window are element-wise multiplied and the results are summed.

A closely related operation is called *cross correlation*, denoted with $\star$, where the kernel is not reversed:

$$(\mathbf{x} \star \mathbf{w})[i] = \sum_{n} x_{i+n} w_n \tag{4.1.4}$$

Thus, if the kernel is symmetric, e.g. $\mathbf{w} = [1, 0, 1]$, then a cross correlation and a convolution will produce the same result. As it is often the case that kernels are symmetric in image processing, a cross correlation is usually implemented in machine learning libraries instead of a convolution, but is still referred to as a convolution [Goodfellow et al., 2016, p. 333].

Naturally, when working with spatial data a convolution is often applied to more than one dimension. A two-dimensional cross correlation – from now on referred to as a convolution – with the input matrix $\mathbf{X}$ and kernel matrix $\mathbf{W}$, can be expressed as:

$$(\mathbf{X} \star \mathbf{W})[i, j] = \sum_{m}^{H} \sum_{n}^{W} x_{i+m,j+n} w_{m,n} \tag{4.1.5}$$

where i is the row element, j is the column element, and the kernel matrix has height H and width W. If the input matrix $\mathbf{X}$ is 3×3 and the kernel matrix $\mathbf{W}$ is 2×2, then a two-dimensional convolution operation yields the output $\mathbf{Y}$:

$$\begin{aligned}
\mathbf{Y} &= (\mathbf{X} \star \mathbf{W})[i, j] \\
&= \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \star \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \\
&= \begin{bmatrix} x_1 w_1 + x_2 w_2 + x_4 w_3 + x_5 w_4 & x_2 w_1 + x_3 w_2 + x_5 w_3 + x_6 w_4 \\ x_4 w_1 + x_5 w_2 + x_7 w_3 + x_8 w_4 & x_5 w_1 + x_6 w_2 + x_8 w_3 + x_9 w_4 \end{bmatrix}
\end{aligned} \tag{4.1.6}$$

An illustration of a two-dimensional convolution is shown in Figure 4.2.

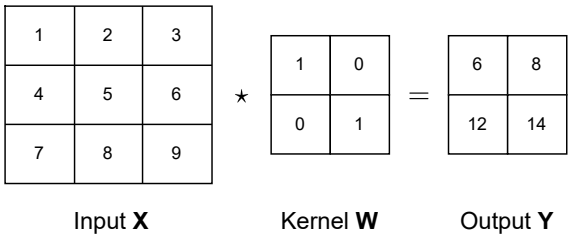


Figure 4.2. A two-dimensional convolution operation with a 3×3 input matrix $\mathbf{X}$ and a 2×2 kernel matrix $\mathbf{W}$, resulting in a 2×2 matrix output $\mathbf{Y}$.

Roughly speaking, the output of a convolution at point (i, j) will be large, if the corresponding input at point (i, j) is similar to the kernel. With this in mind, a kernel can be thought of as a template, that is used to search for a particular pattern. Convolving the kernel over an image will cause regions with patterns that match the template to be, in a sense, highlighted in the output, as these regions will have larger values. Thus, convolution acts as a form of feature detection, and as such, the output of a convolution is called a *feature map*.

To illustrate the key differences between the convolution operation of a CNN and the matrix multiplication of an MLP, the convolution operation, shown in equation (4.1.6), is converted to a matrix-vector multiplication between a matrix $\mathbf{C}$ derived from the kernel

$\mathbf{W}$, and a flatten input vector $\mathbf{x}$ [Murphy, 2022, p. 465]:

$$\mathbf{y} = \mathbf{C}\mathbf{x} = \left[\begin{array}{ccc|ccc|ccc} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{array}\right] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} \tag{4.1.7}$$

From Equation (4.1.7) it can be seen that the matrix $\mathbf{C}$ has a sparse structure. Consequently, it has a significantly reduced number of parameters, compared to the dense matrix multiplication of MLPs, where each entry has a separate weight. As such, a CNN has reduced memory requirements, and requires fewer operations for computing an output. Furthermore, the same weights are applied multiple times in a convolution, whereas each weight is only used once in the dense matrix multiplication of MLPs. This form of parameter sharing enables the convolutional layer to be *equivariant to translation*. Broadly speaking, equivariance to translation means that if the input changes, the output changes correspondingly. In the case of a convolution with an image, it means that if an object is moved in the input, its feature representation is moved by the same amount in the output [Goodfellow et al., 2016, p. 338-339]. As a result, the position of a feature does not have to be fixed, in order for a CNN to detect it. Instead, the feature detection will work equally well across all positions.

Zero Padding

As illustrated in Figure 4.2, a convolution produces an output of a smaller size than the input. Such a convolution is called a *valid convolution*, as the kernel window is always entirely within the image. In general, for a $k_h \times k_w$ kernel matrix and a $x_h \times x_w$ input matrix, a valid convolution produces an output of size [Murphy, 2022, p. 465]:

$$(x_h - k_h + 1) \times (x_w - k_w + 1) \tag{4.1.8}$$

The reduction in size for each convolution layer, restricts the number of convolution layers there can be in a network. With valid convolutions the representation reduces in size for each layer. Eventually, the spatial dimension of a network will reduce to 1×1, where additional convolutional layers no longer can be as considered meaningful convolutions. Thus, valid convolutions can limit the capabilities of a network. A way to circumvent this is through the use of *zero padding*. Here, the input is expanded by a border of zeroes. This ensures that the output has the same size as the input. As such, convolution with zero padding is called a *same convolution* [Goodfellow et al., 2016, p. 349]. A same convolution is illustrated in Figure 4.3.

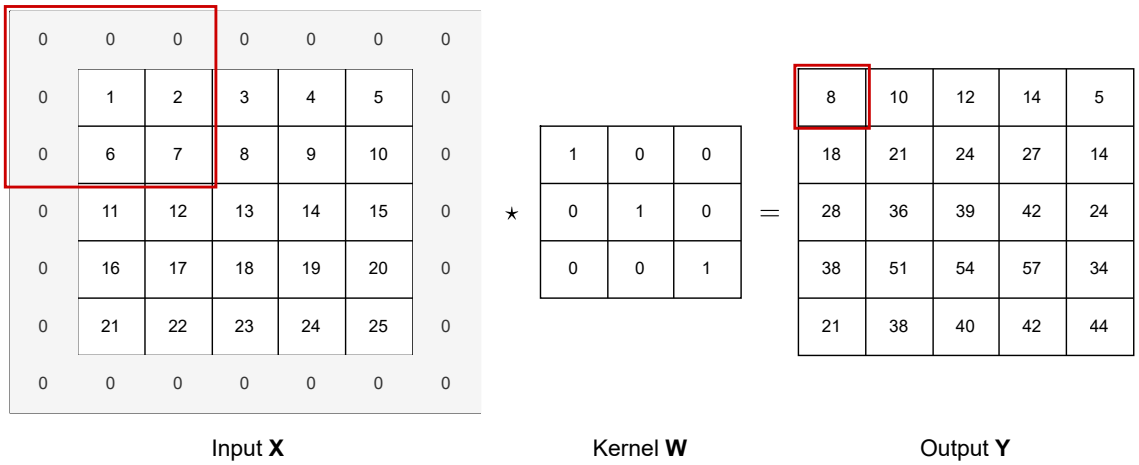


Figure 4.3. A two-dimensional same convolution with a 5×5 input matrix $\mathbf{X}$ and a 3×3 kernel matrix $\mathbf{W}$, resulting in a 5×5 output matrix $\mathbf{Y}$. The red square indicates the initial kernel window and the corresponding output. Due to zero padding the output does not reduce in size.

With a padding on each side of p_h and p_w, a same convolution produces an output of the following size [Murphy, 2022, p. 466]:

$$(x_h + 2p_h - k_h + 1) \times (x_w + 2p_w - k_w + 1) \tag{4.1.9}$$

With same convolutions there is no limit to how many convolutional layers there can be. However, one should keep in mind, that between valid convolutions and same convolutions, the optimal amount of zero padding, usually lies somewhere in between, and finding it is a matter of trial-and-error [Goodfellow et al., 2016, p. 349].

Strided Convolution

Naturally, the neighboring outputs of a convolution will have values akin to each other, as some inputs will be overlapping. In this regard, this redundancy can be reduced by convolving with a stride s larger than one, where the kernel window skips past s inputs. Accordingly, this is called a *strided convolution*, and can be thought of as a convolution that downsamples the input. For a strided convolution with strides of size s_h and s_w, the output has the following size [Murphy, 2022, p. 467]:

$$\left\lfloor \frac{x_h + 2p_h - k_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{x_w + 2p_w - k_w + s_w}{s_w} \right\rfloor \tag{4.1.10}$$

The reduced computational cost gained from using strided convolutions, will however be at the expense of less fine feature extractions [Goodfellow et al., 2016, p. 348].

Multichannel Convolution

As of now, only two-dimensional convolutions have been considered, where the input is a grid of real values (e.g. a grayscaled image input). On the other hand, when working with images, the input is usually a grid of vector values, as they usually consist of multiple

channels C (e.g. RGB). Thus, the kernel matrix needs to have an additional dimension, as it needs to have a separate kernel for each channel. Moreover, a single kernel can only extract one feature. As such, in order to extract multiple features, the kernel matrix needs to be a *kernel stack* of multiple kernels for each channel, where each kernel has learned to detect a different feature. In particular, the kernel stored at $\mathbf{W}_{:,:,c,d}$ has in channel c learned to detect feature d. In light of this, the two-dimensional convolution, expressed in Equation (4.1.5), can be extended as follows [Murphy, 2022, p. 467]:

$$z_{i,j,d} = (\mathbf{X} \star \mathbf{W})[i,j,d] = \sum_{m}^{H} \sum_{n}^{W} \sum_{c}^{C} x_{si+m,sj+n,c} w_{m,n,c,d} \tag{4.1.11}$$

where the subscript s is the stride. Thus, when working with multichannel images, the input and output of the convolution is a three-dimensional tensor, where the two first indices are spatial coordinates, and the third index is a channel [Goodfellow et al., 2016, p. 348].

Pointwise Convolution

In some cases, it may be desirable to only convolve across channels (depth) but not space (height and width). For such cases, a *pointwise convolution* can be used. Here, a 1×1 kernel is used to calculate the weighted sum of the channels for each point in the image:

$$z_{i,j,d} = \sum_{c}^{C} x_{i,j,c} w_{1,1,c,d} \tag{4.1.12}$$

As a result, using a pointwise convolution changes the number of channels from C to D channels [Murphy, 2022, p. 469].

Depthwise Separable Convolution

A regular multichannel convolution requires the use of a four-dimensional kernel matrix of size $H \times W \times C \times D$, which can require many computations [Murphy, 2022, p. 482]. For instance, a regular multichannel convolution between a $20 \times 20 \times 3$ input image and $5 \times 5 \times 3 \times 100$ kernel matrix, produces a $16 \times 16 \times 100$ output (assuming it is a valid convolution, see Equation (4.1.8)). Such an operation requires:

$$5 \cdot 5 \cdot 3 \cdot 100 \cdot 16 \cdot 16 = 1\,920\,000 \text{ computations}$$

A way to reduce the number of computations, is by using a *depthwise separable convolution*, which first convolves each input channel with a two-dimensional kernel $\boldsymbol{w}$, and subsequently uses a pointwise convolution with a 1×1 kernel, $\boldsymbol{w'}$ [Murphy, 2022, p. 482]:

$$z_{i,j,d} = w'_{c,d} \sum_{c}^{C} \left(\sum_{m}^{H} \sum_{n}^{W} x_{si+m,sj+n,c} w_{m,n} \right) \tag{4.1.13}$$

Using the same example as before, a depthwise separable convolution first convolves across space, but not channels. The $20 \times 20 \times 3$ input image is convolved with a $5 \times 5 \times 1 \times 1$ kernel matrix, which produces a $16 \times 16 \times 3$ output. After this, a pointwise convolution is applied,

that convolves across channels but not space, with a $1 \times 1 \times 3 \times 100$ kernel matrix, which produces a $16 \times 16 \times 100$ output. As such, the depthwise separable convolution produces an output of the same size as a regular convolution, and the number of computations required are far less:

$$5 \cdot 5 \cdot 3 \cdot 16 \cdot 16 + 1 \cdot 1 \cdot 3 \cdot 100 \cdot 16 \cdot 16 = 96\,000 \text{ computations}$$

Thus, a depthwise separable convolution is computationally more efficient, requiring both fewer parameters and computations, than its counterpart [Murphy, 2022, p. 482].

Learning

The kernel's weights in a CNN are not set manually. Instead, similar to an MLP's weights, the kernel's weights are initialized randomly and then learned using a gradient method, as described in Section 3.4. Thus, during training, a CNN learns what features its convolutional layer(s) should search for.

4.2 Pooling Layer

A desirable property to have for image processing is the ability to identify a feature irrespective of transformations such as changes in rotation, size or location. Convolution, however, does not introduce any form of invariance to transformations. As such, a pooling operation is used to introduce a learned invariance to different transformations. The two most common pooling operations are *max pooling* and *average pooling*. Max pooling computes the maximum of values within its pooling window, whereas average pooling computes the mean of values within its pooling window [Murphy, 2022, p. 469]. An illustration of both max pooling and average pooling is shown in Figure 4.4.

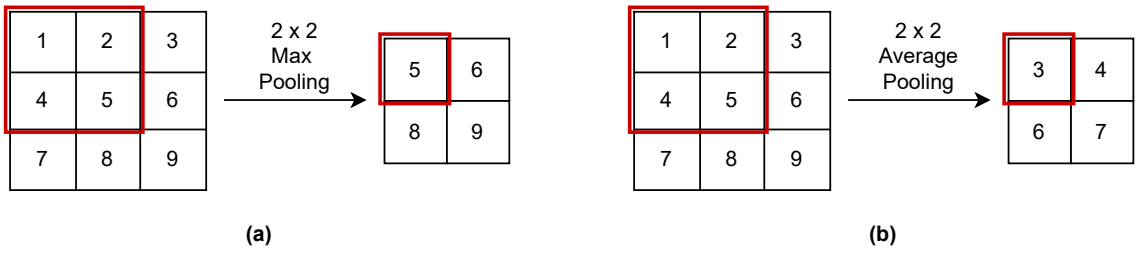


Figure 4.4. (a) A 2×2 max pooling operation. (b) A 2×2 average pooling operation. In both cases it is with a stride of 1. The red square indicates the initial pooling window and the corresponding output.

In general, both pooling operations, when done over spatial regions, introduce invariance to small translations of the input. This means that if the input is translated by a small amount, it will not affect most of the outputs of the pooling operation. Furthermore, if the inputs within the pooling window were to switch places, the pooling operation will still produce the same outputs. Invariance to small translations is a useful property, when it is more of interest to know whether or not a feature is present, rather than to know its exact position. For instance, if the goal is to determine the presence of a face in an image, it is not of importance to know the exact locations of the eyes. It is instead adequate only

to know that there are eyes on both sides of the face within sensible regions [Goodfellow et al., 2016, p. 339, 342].

Besides translation invariance, pooling can also introduce invariance to different transformations by pooling over the outputs of separately parameterized convolutions [Goodfellow et al., 2016, p. 342]. An illustration of how rotational invariance can be achieved using pooling is shown in Figure 4.5.

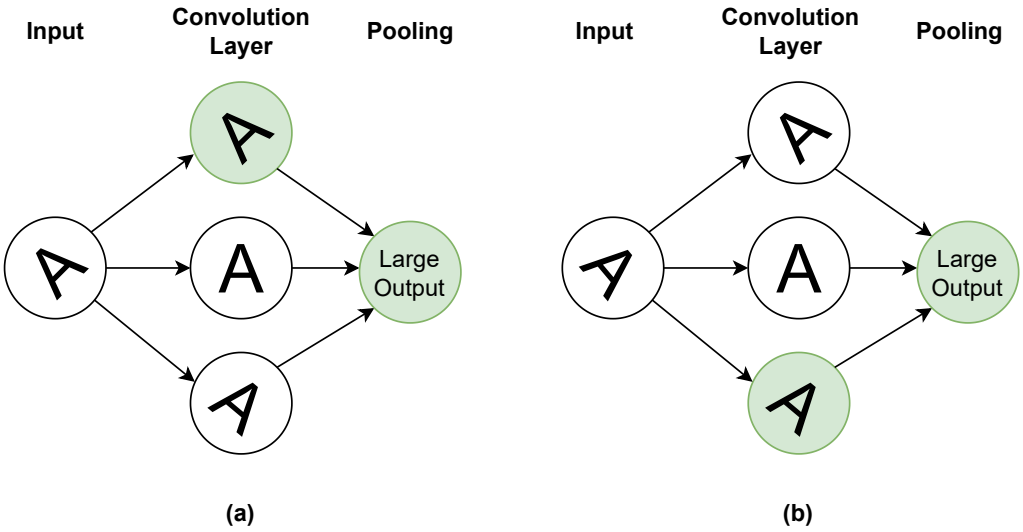


Figure 4.5. A high-level illustration of a learned rotational invariance using pooling. Here, a pooling operation has been applied to the output of three different convolutions, that each have their own set of parameters. All three convolutions have learned to detect the letter "A" with different rotations: The top convolution has learned to detect an "A" rotated counter-clockwise, the middle convolution has learned to detect an "A" without any rotation, and the bottom convolution has learned to detect an "A" rotated clockwise. In (a) the network has been fed an "A" rotated counter-clockwise, resulting in an activation of the top convolution unit. In (b) the network has been fed an "A" rotated clockwise, resulting in an activation of the bottom convolution unit. Nonetheless, the pooling unit has the same large activation, despite the different inputs. The figure has drawn inspiration from [Goodfellow et al., 2016, p. 344].

Pooling can also be used as a downsampling method. E.g. when pooling with a window width of three and a stride of two, the representation size is reduced by a factor of roughly around two. As such, pooling can also be used to improve the computational efficiency, as the next layer has fewer inputs to process. Furthermore, if the next layer is Fully Connected (FC) the reduced input size will also reduce the memory needed for storing the parameters of the FC layer [Goodfellow et al., 2016, p. 342, 344].

Finally, the ability to process images of varying sizes can be achieved using pooling. The FC layer used for classification requires an input of a fixed size. In this regard, pooling can be used to downsample any arbitrary input to the fixed size by changing the stride of the pooling operation correspondingly.

4.3 Fully Connected Layer

After feature extraction has been done using a combination of convolutional layers and pooling layers (of which there exist a plethora of ways), the output is flattened and inserted into a FC layer for classification. The architectural considerations that one should keep in mind here, are described in Chapter 3.

4.4 Regularization

CNN architectures are expressive models that naturally have a high representational capacity – the ability to learn a wide array of functions – and as such they are capable of learning complicated relationships and solving complex tasks. However, an inherent downside of this is that they will have a tendency to overfit and learn properties during training that are not generalizable. To address this issue, one strategy could, for instance, build into the ML model innate preferences towards simpler solutions by e.g. penalizing more complex solutions. Such a strategy is called *regularization*. Generally speaking, regularization is any strategy intended to reduce generalization error, but not training error [Goodfellow et al., 2016, p. 120]. There exists a large variety of regularization strategies. Some involve modifying the network architecture of the model, while others involve modifying the training procedure. The two most common regularization strategies for CNNs are *batch normalization*, *dropout* and *parameter regularization*. As such, these three regularization strategies are presented in this section.

Weight Decay

One way to reduce the capacity – the range of functions that the model can approximate – of a model is by adding some form of penalty term, $\Omega(\boldsymbol{\theta})$, to the objective function, J. In doing so, the regularized objective function $\tilde{J}$ is as follows [Goodfellow et al., 2016, p. 230]:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta}) \tag{4.4.1}$$

where $\lambda \in [0, \infty)$ defines the strength of the penalty term. There are many variants of penalty terms. In this project, *weight decay* is considered, as it is the simplest and most commonly used penalty term. It reduces the capacity of the model by penalizing the size of a model's weights. A model using large weights is more complex than a model using smaller weights. Thus, large weights could indicate overfitting. Furthermore, large weights make a model unstable, as it will be more sensitive to the input, and small changes in the input will produce large changes in the output. Weight decay has the following expression [Goodfellow et al., 2016, p. 231]:

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}|\mathbf{w}|_2^2 \tag{4.4.2}$$

where $|\mathbf{w}|_2$ is called the L^2-norm of the weight vector $\mathbf{w}$, which is the sum of the squared values of $\mathbf{w}$:

$$|\mathbf{w}|_2 = \sqrt{\sum_{k=1}^{n} x_k^2} \tag{4.4.3}$$

With weight decay, the size of the weights will impact the loss score, where larger weights will result in a larger loss score. As a result, weight decay encourages weights to decay towards zero, and thus, ensures that the model's weights do not grow to unfavorable sizes [Bishop, 2006, p. 144].

Dropout

Dropout is a stochastic regularization strategy, that can be added to any layer in a network. With dropout, a layer's neurons are randomly deactivated during training. Specifically, for each training pass, each neuron in the layer has a probability p of being temporarily deactivated. A deactivated neuron, along with its incoming and outgoing connections, are for the given training pass temporarily removed. This means that information cannot pass through it during this forward pass, and weight updates are not applied to it during this backward pass. As such, dropout is a way of regularizing the network by adding random noise. In doing so, neurons are made more robust, as they are motivated to learn useful features on their own, and not be overly dependent on the other neurons in the network [Srivastava et al., 2014].

Batch Normalization

During the course of training, the update of a layer's parameters is done under the assumption that the other layers' parameters will remain unchanged. However, in practice, this is a false assumption as all layers' parameters are updated simultaneously. Inevitably, the parameter updates will have an inherent unreliability and not always go as expected, as the update of a layer's parameters have been computed under the assumption, that preceding layers' parameters would stay the same [Goodfellow et al., 2016, p. 317]. In other words, when doing an update of a layer's parameters, it is done assuming a fixed distribution of inputs. However, as the distribution of inputs will change after each training pass, the layer needs to continuously re-adapt to a new distribution of inputs. In order to accommodate this, lower learning rates has to be used, which inevitably makes training a slow affair [Ioffe and Szegedy, 2015, p. 1].

Batch normalization has been designed to address this issue. Batch normalization standardizes each layers' input for each training pass. In other words, each layers' inputs are rescaled to have a mean of zero and a standard deviation of one. In doing so, the distribution of inputs between each training pass will not change as dramatically. They will be closer to a fixed distribution, and consequently, the layers do not need to re-adapt as drastically to the new distribution of inputs. As a result, batch normalization stabilizes the learning process, and allows for the use of higher learning rates, which makes training converge faster. Furthermore, the stability also makes the model less sensitive to the initial parameters. Lastly, batch normalization acts as a regularization strategy, and has shown to reduce or eliminate the need for using dropout [Ioffe and Szegedy, 2015, p. 5].

Design and Development 5

The ML solution is iteratively developed through the parts shown in Figure 5.1

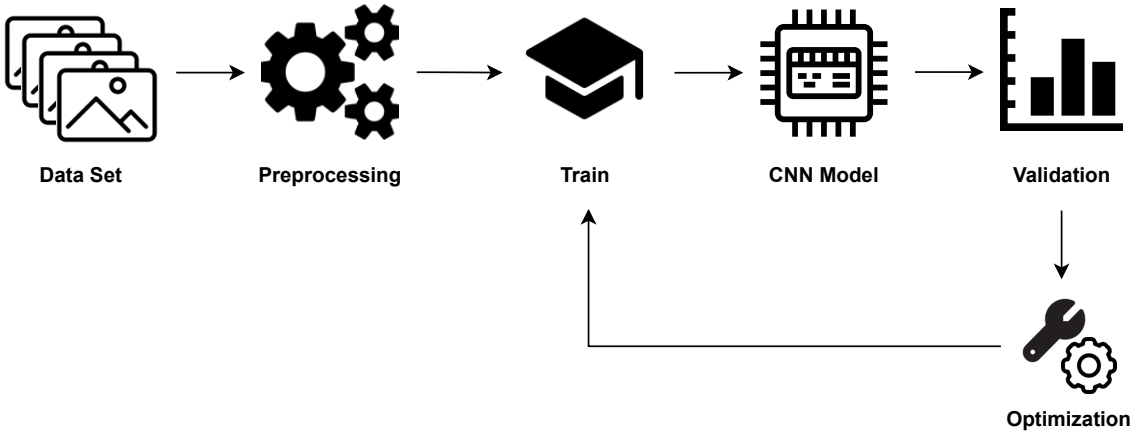


Figure 5.1. The development process of the ML solution.

As shown in Figure 5.1, the development consists of having a dataset that is preprocessed, and then used for training the CNN model. Subsequently, the CNN model's performance is validated. The results are then used to optimize different parts of the solution (e.g. the preprocessing or the CNN model) such that performance can be improved further.

The development of the ML solution mainly involves design considerations of four parts:

- Dataset
- Preprocessing
- CNN model
- Optimization

As such, the design considerations of each of the four parts is presented and discussed in the following sections of this chapter.

5.1 Dataset Design

The TB CXR images have been gathered from three public sources, with the corresponding contents shown in Table 5.1.

Dataset	CXR Images	TB	Healthy	Split [%]
Shenzhen	662	336	326	51/49
Montgomery	138	58	80	42/58
Belarus	1050	1050	0	100/0
Total	1850	1444	406	78/22

Table 5.1. The baseline dataset containing CXR images of TB and healthy lungs. The CXR images have been gathered from three public sources: The Montgomery dataset [Jaeger et al., 2014], the Shenzhen dataset [Jaeger et al., 2014] and the Belarus dataset [NIH, 2022].

In order to create a robust solution capable of detecting TB in a realistic setting, it has to be trained with a extended dataset containing not only healthy and TB infected lungs, but also other similar diseases, that share similar radiographic features. For this reason, pneumonia, COVID-19 and lung cancer CXR images have been added, due to their high similarity as presented in Chapter 1. Furthermore, additional CXR images of healthy lungs have been added by uniformly sampling from all the previously mentioned datasets (see Appendix A.1), such that the dataset consists of 1/3 TB CXR images, 1/3 healthy CXR images and 1/3 similar diseases CXR images. Table 5.2 shows the added content.

CXR Images	TB	Healthy	COVID	Cancer	Pneumonia	Split [%]
4331	1444	1444	675	93	675	34/34/16/2/16

Table 5.2. The dataset containing CXR images of TB, similar diseases and additional healthy lungs. The pneumonia CXR images are from Kermany et al. [2018], the COVID-19 CXR images are from Chowdhury et al. [2020]; Rahman et al. [2021] and the lung cancer CXR images are from JSRT [2022]. All sources are publicly available. The low count of lung cancer cases is due to lack of public availability.

As the goal is to provide a binary classification of either TB or not TB, the dataset, shown in Table 5.2, is accordingly reduced to two classes as shown in Table 5.3.

CXR Images	TB	NOT TB	Split [%]
4331	1444	2887	33/66

Table 5.3. The extended dataset used for training the model.

5.2 Preprocessing

Preprocessing is intended to make the input more digestible, with the aim of improving performance. This is done either by normalizing the input to similar configurations or by varying the input to generate additional training data for a dataset. In this project, preprocessing entails normalizing the inputs. Furthermore, data augmentation is used to

generate additional data. At last, U-Net is described, which seeks to mask the lungs of a CXR image, thus reducing the search space for the model in regard to classification of TB.

5.2.1 Normalization

In object recognition and detection, preprocessing methods can be utilized to normalize varied input. Images mostly have pixels that lie in the integer range [0,255], where 0 represent black, and 255 represent white. It is usually required to standardize the images, so their pixel ranges are normalized – usually in the decimal range of [0,1] or [-1,1] for the entire dataset. Reason being, that the activation functions (e.g. Sigmoid, ReLU) usually work with decimals between 0 and 1, resulting in smaller corresponding weights when training. This creates for a more stable environment when training, as bigger weights makes for unstable training. As a result, neurons in the network will experience a decreased learning time [BrownLee, 2019]. Moreover, many ML architectures require image inputs to be of a fixed resolution. As such, images are usually re-scaled as a preprocessing step for CNNs. However, the use of re-scaling depends on the CNN architecture, as each architecture handles the input size differently from one another; Some CNN architectures accept varying input sizes and adjust the size of their pooling regions dynamically to keep the output size constant [Goodfellow et al., 2016, p. 453].

5.2.2 Grayscaling

In this project, grayscaling is applied as a preprocessing method to standardize the color representation in all images, as CXR-images have different standards in terms of coloring (e.g. blue). For reference, see Figure 5.2. Therefore, grayscaling all the CXR images as a preprocessing step may reduce the generalization error.

5.2.3 Data Augmentation

More data usually leads to a model with better generalization. A classifier needs to learn how to detect different features irrespective of transformations such as changes in rotation, scale or location. In this regard, data augmentation is a way to add different transformations to the current training set. In doing so, there is artificially more data to train the model with, and it increases the probability of the model learning invariance to different forms of transformations [Goodfellow et al., 2016, p. 240].

One must be particularly careful when selecting which transformations to perform, as a transformation could impact the classification. For instance, an optical character classifier would need to recognize the difference between the characters 'q' and 'p', and the difference between '9' and '6'. In this regard, the image augmentation operations, such as horizontal flips and 180 degree rotations, would not be the appropriate operations for augmenting datasets of these types of tasks. Data augmentation is domain dependent and cannot be applied to all domains in the same way [Goodfellow et al., 2016, p. 453-454].

In this project five different data augmentation operations are considered: Reflection, rotation, scaling, translation and color jittering. The five operations are shown in Figure 5.2.

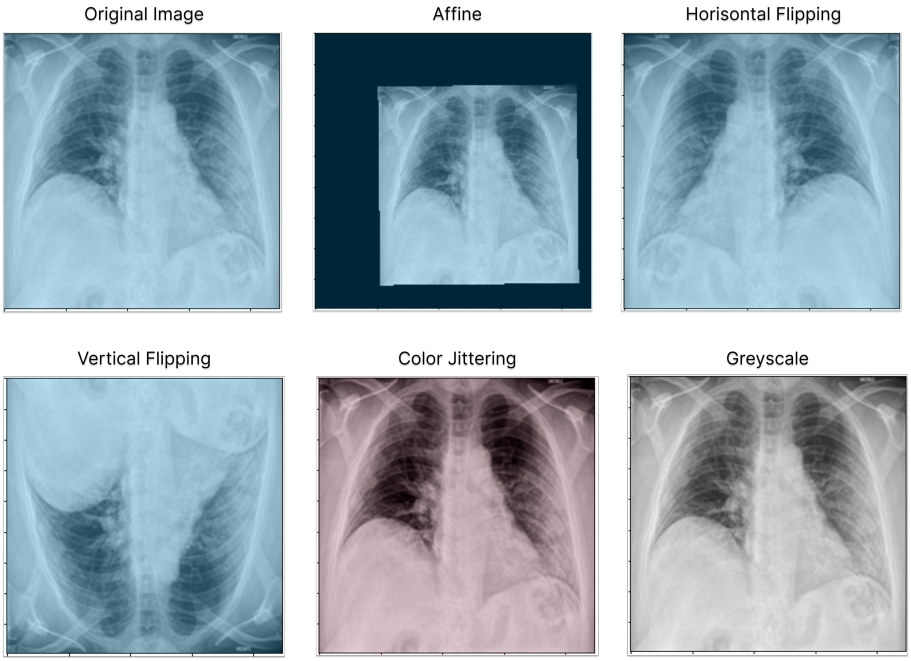


Figure 5.2. Image augmentations that can be used for CXR images to detect bacterial disease in lungs to artificially extend the existing training dataset, in order to reduce the generalization error. Affine entails both translation and rotation of an image. Color Jittering entails brightness, saturation, contrast and hue of an image.

The first four operations are considered, as these operations have proven to be beneficial in the context of CXR images [Elgendi et al., 2021]. Besides, one can also imagine radiographic features to vary in these ways, and as such it would be beneficial for the developed model to have a learned invariant to these transformations. The fifth operation, color jittering, which transforms the color of the CXR image, is also considered, as real life CXR images vary in color.

Reflection

By flipping the image on either the x- or y-axis, the image will be *reflected*. CXR images that are upside down are highly unusual and can lead to unnecessary noise for learning [Elgendi et al., 2021]. However, the addition of reflection in the y-axis – vertical flipping – may be beneficial for learning, if the radiographic features for TB are similar even when being upside down.

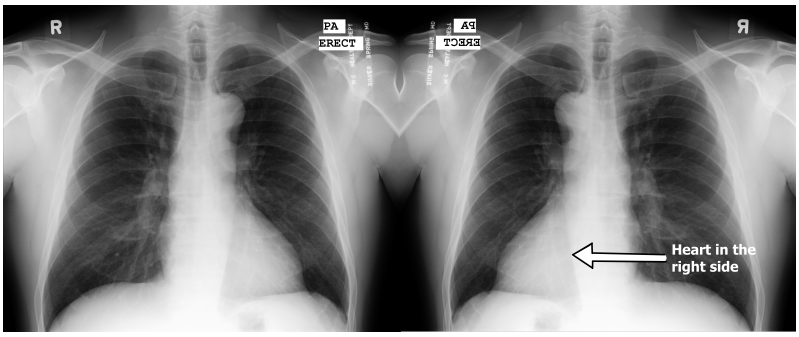


Figure 5.3. Reflection in the x-axis – Horizontal flipping in CXR images will not be consistent with physiologic standards. For instance, the heart will be in the right side of the chest, rather than the left, after this operation. This CXR image is taken from the Montgomery dataset [Jaeger et al., 2014].

.

Horizontal flipping – reflection in the x-axis – will lead to unusual physiologic images, namely the heart being on the right side of the chest, as illustrated in Figure 5.3. However, the lungs' outline will remain the same. Utilization of horizontal flipping as data augmentation is contested amongst papers [Elgendi et al., 2021] [Heidari et al., 2020, p. 4].

Rotation

To detect radiographic features in a CXR image, it could be beneficial to *rotate* it. However, the range of rotation has to be appropriate in order to prevent unnecessary noise. Severe rotations can be harmful, whereas slight rotations between -5 and 5 degrees typically are seen in clinical practice. Slight rotation can improve learning, as CXR images have slight variance in rotation [Elgendi et al., 2021]. The range will be between -30 and 30 degrees in order to explore a possible point of diminishing returns.

Color Jittering

A way to modify an image, without changing the dimensionality of objects, is by jittering its colors. Color jittering tweaks the brightness, contrast, saturation and hue of an image. Brightness is the overall intensity of lightness and darkness in an image. Contrast is the difference in brightness between regions. Saturation is the degree of depth and intensity of color. Hue defines the difference between primary colors: Red, green and blue.

Translation

By moving the image along the x- or y-axis the image will get *translated* into a direction. This augmentation technique is useful for CXR images to avoid regional overfitting. Reason being, that the CXR images does not always have lungs in the center. This could be due to the radiographic unit or the patient's position. As such, translating the image could lead to more robust classification of the lungs.

Scaling

Scaling an image refers to rescaling it to a higher or lower pixel size. Scaling revolves around multiplying the amount of pixels with a scalar **a**. An image can be scaled in the x-axis, y-axis or both. It will invariably stretch the augmented CXR image, when scaling up a CXR image by increasing the number of pixels in either direction (a>1). Meanwhile, scaling down the image by decreasing the number of pixels in either direction (a<0), the size of the augmented image will be less than the original image. Clinically, it is not recommended to scale the CXR image in only the x- or y-axis. However, equal scaling of the image in both directions is acceptable [Elgendi et al., 2021]. In order to combat missing parts of the object in a rotated and/or translated image, the image will be re-scaled.

Overview of Data Augmentation Operations

Finally, to create an overview of the augmentation operations selected for detecting TB in CXR images, Table 5.4 provides the probabilities and ranges associated with them. The ranges for each operation have been selected based on intuition and what has been done in similar papers [Heidari et al., 2020, p. 4]. Color jittering, namely brightness, contrast, saturation and hue has a 100% probability of occurring, but notice that their ranges all start from 0 (non-changes), which means that each sub-operation of color jittering has a probability of not being utilized.

Image Augmentation Operation	Probability	Range
Vertical flipping	20%	N/A
Horizontal flipping	20%	N/A
Scaling	100%	$[0.7, 0.8]$
x-axis translation	100%	$[-0.1 \cdot W, 0.1 \cdot W]$
y-axis translation	100%	$[-0.1 \cdot H, 0.1 \cdot H]$
Rotation	100%	$[-30°, 30°]$
Brightness	100%	$[0.0, 0.4]$
Contrast	100%	$[0.0, 0.4]$
Saturation	100%	$[0.0, 0.4]$
Hue	100%	$[0.0, 0.2]$

Table 5.4. Each operation has a probability of being used for each image in the dataset. Probability denotes the chance of each image to be augmented, and the ranges denotes the range that operation, by probability, is being used. Every operation is in decimal range, except rotation which is in integer range.

5.2.4 U-Net: Semantic Segmentation of Lungs

In an effort to improve the classification of TB in CXR images, previous papers have applied masks of lungs over the CXR image, such that the classification and localization of TB is limited to the lungs [Jaeger et al., 2012; Candemir et al., 2012]. This is effectively improving the performance by limiting the search space for the classification. Due to the successful application of this method, also in relation to TB, it is explored and experimented with.

Masks of lungs can be obtained, either by having them supplied by professionals or by generating them. As masks are not supplied with the dataset utilized in this project, the

focus will be on generating masks. The generation of masks relates to the task of semantic segmentation, utilized in computer vision, which clusters parts of images, which belong to the same object class. In this case, the result is intended to be a single cluster in each image containing the lungs of a patient.

The generation of masks will be based on learning a model to segment lungs on a dataset, which has professionally masked lungs, and then transfer the learning to the project's dataset.

In biomedical imagery, the model, U-Net [Ronneberger et al., 2015], has displayed favorable results, also more specifically in semantic segmentation of lungs for TB classification [Rahman et al., 2020]. The model works well for smaller datasets, as well as both working for binary and multiple classes. U-Net, as depicted in Figure 5.4, is a CNN, which consists of a contracting part (left side) and an expansive part (right side), creating a U-shaped architecture.

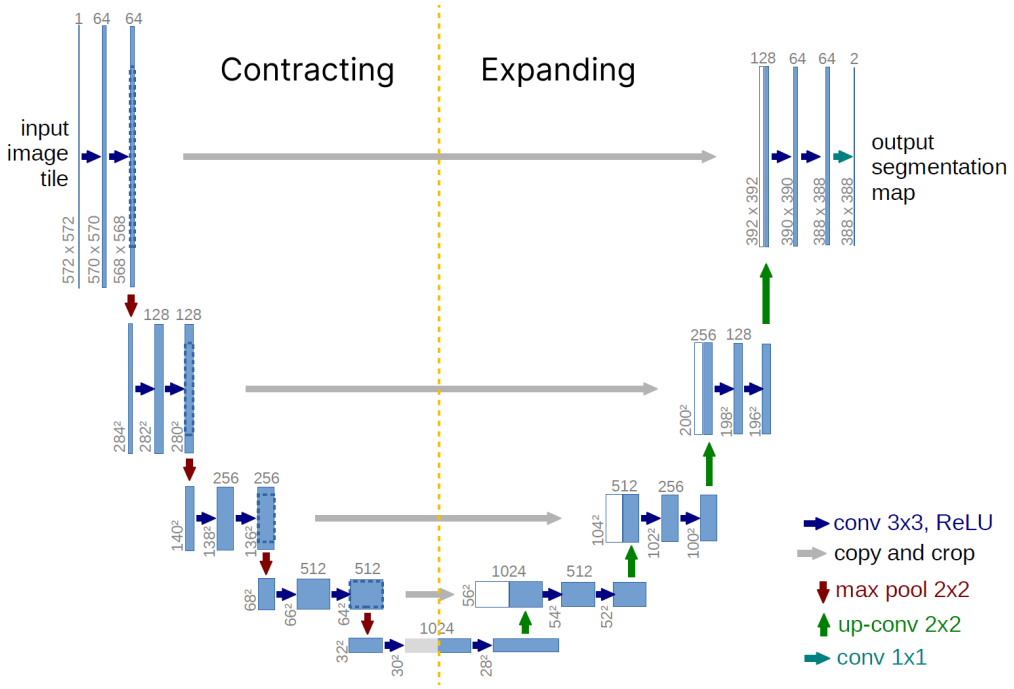


Figure 5.4. The U-Net architecture. The blue squares represent a feature map with multiple channels, which amount is written on the top of each square. Additionally, the x-y size is written on the lower left of the square. The white squares represent a copy of a feature map. The arrows reflect the different operations [Ronneberger et al., 2015].

The contracting part performs a CNN classification, by using repeated application of 3×3 valid convolutions, each followed by a ReLU activation function and a 2×2 max-pooling, with a stride of two. Additionally, the number of feature channels is doubled at each down-sampling. The CNN architecture in the contraction can be exchanged for a range of alternate CNN architectures.

The expansive part utilizes 2×2 convolutions (up-convolutions). This halves the number of feature channels, which subsequently is concatenated with the corresponding cropped feature map from the contracting path. This is followed by two 3×3 convolutions, each followed by a ReLU activation function.

The cropping is necessary due to the loss of border pixels in every convolution. At the final layer, a 1×1 convolution is used to map each 64-component feature vector to the desired number of classes. In total, the network has 23 convolutional layers. Effectively, the result is a segmented image – a mask – that provides the localization of the lungs in the image. The mask can both limit the search space in regard to classification, but also ensure that the classifier does not get access to text that might suggest diagnoses on the corners of the CXR images. In addition, the mask would ensure more precise object localization, when using interpretation methods. An example of a U-Net output can be seen depicted in Figure 5.5.

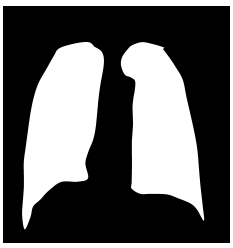


Figure 5.5. A mask of a set of lungs, supplied by semantic segmentation, using U-Net.

5.3 CNN Architectures

In extension to the theoretical background of CNNs provided in chapter 4, a specific CNN architecture needs to be selected, with the goal of achieving the best possible performance. Five architectures will be thoroughly tested and optimized for the specific case.

Only architectures with pre-trained model implementations would be up for selection, as they ensure out-of-the-box performance. These models have significantly better initial performance due to transfer learning and are as such also much faster to train. Limiting the selection to architectures with pre-trained model are of little concern, as papers in the TB Classification domain often use the very same models [Guo et al., 2020; Showkatian et al., 2022; Sathitratanacheewin et al., 2020].

Several state-of-the-art architectures, typically not used for TB classification, have emerged in recent years, boasting better performance on benchmarks such as ImageNet paperswithcode [2022]. These new architectures are, however, not proven to provide better performance within the TB classification domain, but do show promise due to their high performance on general image classification.

In light of the above, five architectures are considered, where three of them have shown great performance in the domain of TB classification, and the latter two have shown state-of-the-art results on ImageNet, or are newer iterations of architectures with great performance in TB classification.

The three architectures selected from TB-papers are: VGG, DenseNet and Inception-V3. The two state-of-the-art architectures have been selected based on different criteria. The first architecture, EfficentNet, is picked due to its high performance on ImageNet paperswithcode [2022]. The second architecture, ResNeXt, is picked as it is a newer variant of ResNet, which has shown to perform almost as well on TB-classification, as VGG, DenseNet and Inception-V3. ResNeXt shows potential as it has scored better than

ResNet on the ImageNet benchmark and incorporates some architectural inspirations from newer architectures. As such, ResNeXt shows potential to outperform its predecessor.

Summarily, the five architectures considered in this project are, VGG, Densenet, Inception-V3, EfficentNet, and ResNeXt.

5.3.1 VGG

This CNN architecture was developed and proposed in 2015 for improving accuracy in object detection in images. While it is the oldest CNN architecture that is considered in this project, the Visual Geometry Group (VGG) architecture utilizes methods that greatly increases performance from the classical CNN architecture [Simonyan and Zisserman, 2015].

VGG specifically focuses on the depth design of the CNN architecture. In this regard, VGG increases the depth of the network by adding more layers. In the original paper from Simonyan and Zisserman [2015], different configurations of the architecture are experimented with. These configurations differ in the amount of layers and the type of kernels that are added. In general, the most viable kernels that are added in the layers are small (3×3). The configuration of the FC layers are the same in all three layers. The architecture features many configurations, wherein two primary configurations with the best performance are the most notable: VGG16 and VGG19. The only difference is the number of layers in the architecture [Simonyan and Zisserman, 2015]. For instance, a VGG16 consists of 13 convolutional layers and three FC layers, which can be seen in Table 5.5.

Stage i	**Operator** F_i	**Resolution** $H_i \times W_i$	**no. Layers** L_i
1	3×3 Conv	224×224	2
2	3×3 Conv	112×112	2
3	3×3 Conv	56×56	3
4	3×3 Conv	28×28	3
5	3×3 Conv	14×14	3
7	FC layers	1×1	2
8	FC layer	1×1	1

Table 5.5. A depiction of the VGG16 architecture that contains 16 layers (13 convolutional layers and three FC layers). It takes an input image of size 224×224. There exists a max pooling layer with size 2×2 with a stride of two in between the stages. Three FC layers are followed after a stack of convolutional layers (i.e. 13 convolutional layers for VGG16). The last FC layer's channels contain the amount of classes, i.e. ImageNet has 1 000 classes. All hidden layers are equipped with the activation function ReLU [Simonyan and Zisserman, 2015, p. 2].

VGG utilizes multiple 3×3 kernels throughout the whole net as a substitution for the relatively large kernels (e.g. 11×11 and 7×7). The kernels (3×3) is the smallest size possible to capture the notion of direction. For instance, a stack of three 3×3 convolutional layers with ReLU (also called a rectification layer), instead of a single 7×7 layer with ReLU, will make the decision function more discriminative.

Moreover, assuming that a three-layer 3×3 convolutional stack has C channels and has $3(3^2C^2) = 27C^2$ parameters, whereas a single 11×11 convolutional layer would require $11^2C^2 = 121C^2$ parameters, i.e. 348% more. Furthermore, a single 7×7 convolutional layer ($7^2C^2 = 49C^2$) requires 81% more parameters. The number of parameter decreases with the usage of a 3×3 convolutional stack instead, thus being more beneficial to utilize than its respective counterparts [Simonyan and Zisserman, 2015, p. 3].

Although the increase of the CNN's depth expands the architecture's ability to fit to more complex functions, the drawback is the additional computational time required to train it; Recall from chapter 3 that during backpropagation, the weights are updated proportional to the gradient for each training pass. An initial problem with adding more layers and increasing the depth of the network, is that the gradient can become too small (≈ 0) and gradually diminish during backpropagation. This prevents the model from learning efficiently (updating its weights), and eventually making minuscule changes. Thus, resulting in a significant increase in training time. This is known as the *vanishing gradient* problem. On the other hand, when the gradient value is too large (> 1), the gradient will increase exponentially, and thus explode in size. Thus, making unreasonable updates to the weights. This is referred to as the *exploding gradient* problem. A common approach to prevent both problems is through the utilization of batch normalization. With batch normalization, the distribution of inputs between each training pass will not change dramatically (see Section 4.4 [Basodi et al., 2020, p. 197]. Originally, VGG did not use batch normalization, as VGG was introduced prior to batch normalization's development. However, VGG with batch normalization shows better performance. As such, this project considers VGG with batch normalization.

5.3.2 DenseNet

As CNNs have become increasingly large, there have been some problems that have become increasingly prevalent with them. Specifically, as mentioned in the VGG architecture, the *vanishing gradient* problem. This is the main problem that the DenseNet architecture tries to solve [Huang et al., 2017, p. 1]. The primary way in which it does so, is by having all layers connected with each other. More specifically, a layer is connected to all previous layers, making all layers directly affected by the loss function. An overview of the DenseNet121 architecture can be seen in Table 5.6. The DenseNet161 and 201 are not added here, as they are architecturally similar to 121, but simply with more convolutional layers within the dense block.

Stage i	Operator F_i	Operations	Resolution $H_i \times W_i$	no. Layers L_i
1	Conv	k7×7	224×224	1
2	Dense block	$6 \cdot [1 \times 1, 3 \times 3]$	56×56	12
3	Transition layer	k1×1	56×56	2
4	Dense block	$12 \cdot [1 \times 1, 3 \times 3]$	28×28	24
5	Transition layer	k1×1	28×28	2
6	Dense block	$24 \cdot [1 \times 1, 3 \times 3]$	14×14	48
7	Transition layer	k1×1	14×14	2
8	Dense block	$16 \cdot [1 \times 1, 3 \times 3]$	7×7	32
9	Classification	Softmax	7×7	

Table 5.6. A table of the layers in DenseNet121 [Huang et al., 2017, p. 4]. The architecture consists of dense blocks and transition layers. Each Dense block consists of multiple conv layers.

As seen in Table 5.6 the DenseNet architecture primarily consists of dense blocks and transition layers. The purpose of the transition layers is to limit the number of channels that the dense block outputs and feed them to all the proceeding layers. This enables dense connectivity, where each block gets all the feature maps as input from all previous blocks. The channels of the output are reduced by a pointwise convolutional as well as a pooling operation in the transition layer. Each Dense block consists of a number of convolutional layers, and each convolutional layer in a dense block outputs a fixed number of feature maps. These are, along with the output from the other convolutional layers in the block, concatenated in the output of the block. A visualization of the dense block is shown in Figure 5.6.

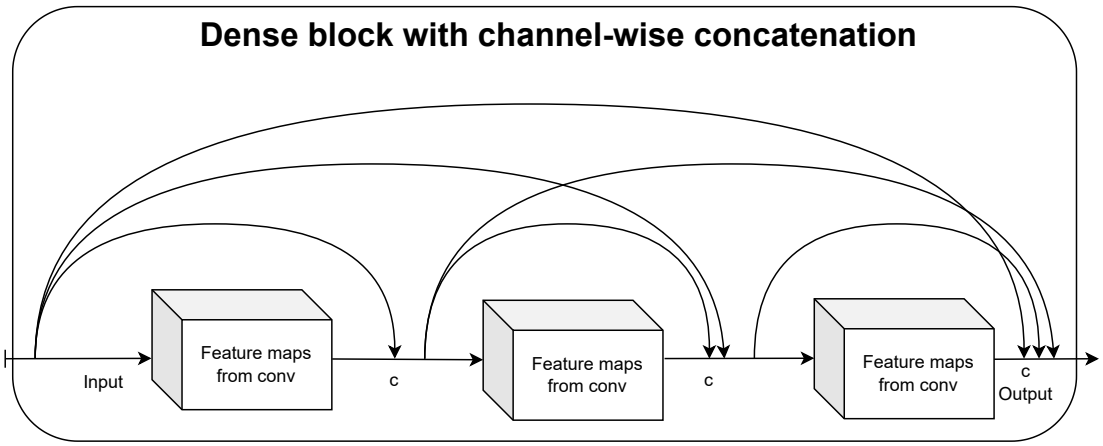


Figure 5.6. An overview of the inner workings of the dense block, in this case in the middle of the DenseNet architecture. The input of the block is a transition layer that lies between this block and the previous block. Each cube represents the feature maps outputted by each convolutional layer in the block. As is apparent from the figure, each of these outputs along with the input is fed to all proceeding layers. This implicates that all feature maps that have been generated are concatenated at the end of the dense block. This output is then given to the next transition layer, and this is repeated until the FC layer at the end of the architecture [Huang et al., 2017, p. 1].

The architecture of DenseNet with dense blocks and transition layers comes with a couple of desirable benefits compared to deeper, less connected networks.

The first, as a consequence of dense connectivity, is implicit deep supervision, which means that activations and feature maps in all layers are directly affected by the loss function, mitigating the effect of the *vanishing gradient problem* [Huang et al., 2017, p. 2].

The second is an efficient use of features by means of feature reuse, which eliminates the replication of unnecessary features in the next layer, as well as more diverse features between blocks [Huang et al., 2017, p. 8].

The DenseNet architecture does, however, create a couple of concerns, which need to be addressed. One of such is that one could imagine if a dense block receives its input from all previous layers, there would be a very large inputs near the end of the network. However, this problem is handled in a few different ways. First, each convolutional layer is limited to k-features, which means that all convolutional layers output the same number of features [Huang et al., 2017, p. 3]. The value of k is referred to as the growth rate of the network and keeps the number of features in check. This combined with diverse features and reuse of these, as well as the reduction of channels in the transition layers creates an architecture with fewer parameters than competing architectures [Huang et al., 2017, p. 5].

5.3.3 Inception

The Inception v3 network architecture is described in this section instead of its baseline Inception v1 for two reasons: First, in contrast to the baseline of the other architectures, Inception v3 is the only variant of Inception, which has a publicly available pre-trained model. Second, where the other architectures mainly increase in size, Inception, to a larger degree, changes the architecture itself between versions, making differences larger.

Inception-v3 was developed in 2015 and is named after the movie Inception, which is a joke referring to the fact that this architecture is very deep. Its main purpose was to add computational efficiency to support the increasing depth and complexity of CNNs by means of reduction [Szegedy et al., 2015, p. 1]. However, despite its name, which implies a heavy focus on depth, the creators argue that in order to achieve optimal performance, one has to increase both depth and width in parallel [Szegedy et al., 2015, p. 2]. These combination of factors have led the paper's authors to create the inception block, which can be seen in the network architecture in Table 5.7 [Szegedy et al., 2015, p. 6].

Stage i	Operator F_i	Operations	Resolution $H_i \times W_i$
1	Conv	k3×3	299×299
2	Conv	k3×3	149×149
3	Conv padded	k3×3	147×147
4	Conv	k3×3	73×73
5	Conv	k3×3	71×71
6	Conv	k3×3	35×35
7	Inception block	$3 \cdot [4 \cdot 1 \times 1, 4 \cdot 3 \times 3]$	35×35
8	Inception block	$5 \cdot [3 \cdot 7 \times 7, 3 \cdot 1 \times 7, 4 \cdot 1 \times 1]$	17×17
9	Inception block	$2 \cdot [4 \cdot 1 \times 1, 1 \cdot 3 \times 3, 2 \cdot 1 \times 3, 2 \cdot 3 \times 1]$	8×8
10	Classification	Softmax	8×8

Table 5.7. A table of the Inception-v3 architecture [Szegedy et al., 2015, p. 6]. The input is an image of size 299x299 pixels. There are three varying types of inception blocks. Each of the inception blocks contain multiple convolutional layers and a single pooling layer. The last stage is a classification layer with a softmax activation function. Inception-v3 has a total of 42 layers [Szegedy et al., 2015, p. 6]

The main idea of the inception block is to not have to choose a convolutional layer with a specific size or a pooling layer, but instead do all of them in one block and then concatenate the result. This makes all the information available to the model, so that it can choose which layer provides the most information in a given instance. The logic behind this is that it can be hard for a human to know which type of layer provides the most useful information to the model, as this often varies from case to case. So providing information of multiple different types of layers to the model within the same block, and allowing it to pick what information it thinks is relevant, can be very giving.

An instance of the inception block is shown in Figure 5.7.

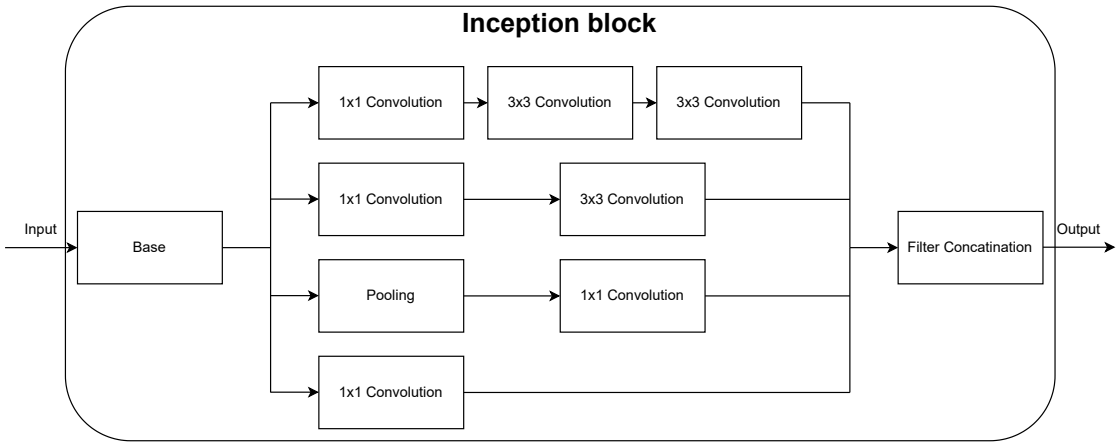


Figure 5.7. An overview of the layers within an inception block, which employs the split-transform-merge strategy. The base represents the input to the block, where it is then split to four different operation tracks. Each of these tracks transforms the input and then merges them all in the filter concatenation block. This concatenation of all operations, that have been deployed in the inception block, is the output of the inception block [Szegedy et al., 2015, p. 4]

As shown in Figure 5.7, the base layer of the block branches out to multiple different operations, within the Inception block, where a couple of noteworthy steps happen. First, all the convolutions larger than 1×1, are first subject to a pointwise convolution [Szegedy et al., 2015, p. 2]. This significantly cuts the number of computations for larger convolutions.

Second, to the far left in Figure 5.7, one can see that a 1×1 convolution is followed by two 3×3 convolutions, which is a factoring technique for a 5×5 convolution. The two 3×3 convolutions are a replacement for a single 5×5 convolution because as the kernel of convolutions become larger they become disproportionately more expensive. So replacing a 5×5 convolution with two 3×3 convolutions yields a 28 % reduction in computations, while still maintaining the same expressiveness [Szegedy et al., 2015, p. 3].

Third, a pointwise convolution is applied after, instead of before the pooling layer, which can be seen on the right-hand side of Figure 5.7. This pointwise convolution is again used to cut the number of channels, as the max pool layer has the same number of channels as the entire output from the previous layer. As such, it would take up too much of the concatenated output without this factorization.

All of these branches are then concatenated and given as the output of the entire inception block, which is then forwarded as input into the next inception block. Seemingly all of these operations would make the network computationally infeasible. However, the Inception network is both more efficient and have fewer parameters than VGG [Szegedy et al., 2015, p. 6].

There are two more variants of the inception modules, which uses reduction as described for the variant shown in Figure 5.7. The other variations use them in slightly different ways, however, this will not be described in detail at this time [Szegedy et al., 2015, p. 5].

Other than the Inception modules themselves, the Inception-v3 network also introduces two auxiliary classifiers to two of the Inception blocks. The purpose of these is to combat the vanishing gradient problem, by providing an additional loss function called the auxiliary loss. This gives information about the activations and feature maps in the middle of the network, to help the network better update during back propagation. The auxiliary classifiers are only used during training [Szegedy et al., 2015, p. 5].

5.3.4 ResNeXt

VGG, ResNet (ResNeXt's predecessor) and Inception all show promising results, but are still limited in terms of adaptability. They are well-suited for general datasets, but due to the numbers of computations and hyperparameters, adapting them to new datasets is a difficult task. To overcome this problem, the ResNeXt architecture from 2017 takes advantage of the Inception's split-transform-merge strategy, while keeping the depth strategy of repeating blocks with the same topology from the aforementioned VGG and ResNet in mind [Xie et al., 2017, p. 2]. In order for ResNeXt to have more layers it adopts the residual connections from Resnet, which solves the problem of an increasing training error when a network becomes too deep (See Figure 5.8) [He et al., 2015, p. 1]. The problem is referred to as the degradation problem. The basic principle of a residual connection, is that an input to a layer, is also directly added to the output of a layer deeper in the

network. The reason behind this additional connection is due to the fact that values in a channel can approximate to 0 or above 1, which introduces the aforementioned exploding or vanishing gradient problem. The residual connection solves this, as in the case nothing valuable is learned from n-layers, the input to these layers is still added to the output of n-layers ahead. Essentially meaning that no information is lost, as the features learned before is still passed forward in the network [He et al., 2015, p. 2-3].

Recall that, in the inception architecture, the amount of branches is limited by four kernel window sizes: 1×1, 3×3 and 5×5 branches and a single 3×3 pooling branch. By utilizing the depth strategy, which secures branches with identical topologies, unlocks the architecture's ability to not be limited by the kernel window sizes and include as many branches as deemed appropriate. The size of the set of transformations – the number of branches – in the architecture is denoted as the *cardinality*. The ResNeXt block from Figure 5.8, showcase a cardinality of 32. The set of transformations with the same topology are concatenated before the output.

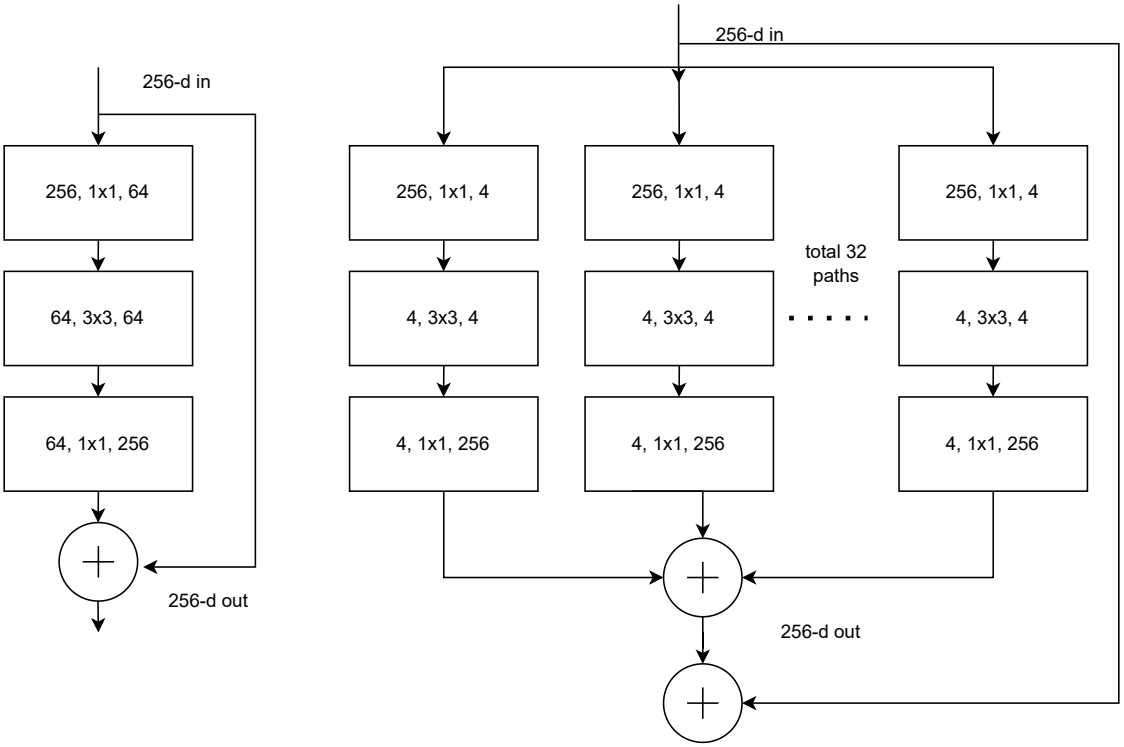


Figure 5.8. Comparison between ResNet and ResNeXt blocks. Left side is a block of ResNet[He et al., 2015]. Right side is a block of ResNeXt. The arrow going directly from the input to the output is the residual connection (identity function), where the input is directly added to the output.

The ResNeXt architecture is defined by two rules: First, if the blocks outputs feature maps of the same size, they share the same width and kernel size. Second, if all the feature maps are downsampled by a factor of two, the width of the block is multiplied by a factor of two [Xie et al., 2017, p. 2]. Table 5.8 showcase the ResNeXt50 architecture, which is an extension of the ResNet50 architecture, the only addition in RexNeXt being cardinality [Xie et al., 2017, p. 2]. Another variation of ResNeXt is ResNeXt101, the difference between ResNeXt50 and ResNeXt101 is the amount of residual layers.

Stage i	Operator F_i	Operations	Resolution $H_i \times W_i$	no. Layers L_i
1	Conv	k7 × 7	224 × 224	1
2	Residual layer	3 · [1 × 1,3 × 3 (C=32),1 × 1]	112 × 112	9
3	Residual layer	4 · [1 × 1,3 × 3 (C=32),1 × 1]	56 × 56	12
4	Residual layer	6 · [1 × 1,3 × 3 (C=32),1 × 1]	28 × 28	18
5	Residual layer	3 · [1 × 1,3 × 3 (C=32),1 × 1]	14 × 14	9
6	Classification	1 × 1, 1000-d fc, softmax	7 × 7	1

Table 5.8. ResNeXt50 with a $32 \times 4d$ template. Takes an input of 224 × 224 image size. The brackets denotes a residual block. Outside the brackets is the amount of stacked blocks on a stage. "C=32" denotes that there are 32 groups in the grouped convolution. [Xie et al., 2017, p. 3]

5.3.5 EfficientNet

EfficientNet was first introduced in 2020, and the core idea behind EfficientNet was to design a method to scale up networks in three dimensions simultaneously, namely depth, width, and resolution [Tan and Le, 2019, p. 1]. Scaling up networks has been widely used to enhance the performance of CNNs, but the approaches so far have mainly been focused on scaling one or at most two dimensions [Tan and Le, 2019, p. 1]. As explained in Section 5.3.3, Inception, for instance, focused on scaling depth and width in parallel. Even though it is possible to arbitrarily scale up a network on three dimensions simultaneously, it can lead to suboptimal accuracy and efficiency [Tan and Le, 2019, p. 1]. This is why EfficientNet has introduced a compound scaling method, which simultaneously scales and balances all three dimensions [Tan and Le, 2019, p. 1]. The compound scaling method is defined as follows [Tan and Le, 2019, p. 5]:

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

where ϕ is a compound coefficient, and $\alpha, \beta^2, \gamma^2$ are constants that can be determined by a small grid search. The constants essentially determine how the resources should be allocated between the three dimensions, and ϕ determines how many resources are available in regard to Floating Point Operations Per Second (FLOPS). The constraint, $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$, is included due to the fact that doubling the network's width or resolution, will roughly increase the FLOPS by four times, which is why these two constants are squared by two. As the constraint expression approximately evaluates to two, it means that the total amount of FLOPS will approximately increase by 2^ϕ. E.g. $\phi = 1$ will approximately increase the total amount of FLOPS by a factor of two [Tan and Le, 2019, p. 5].

To test the performance of the proposed scaling method, they designed a baseline network EfficientNet-B0 for scaling, which created a family of architectures ranging from B0-B7. The baseline network was constructed by the use of a multi-objective neural architecture search, which essentially optimizes both accuracy and FLOPS [Tan and Le, 2019, p. 5]. The architecture of EfficientNet-B0 is illustrated in Table 5.9.

Stage i	Operator F_i	Resolution $H_i \times W_i$	no. Layers L_i
1	Conv3 $\times$ 3	224×224	1
2	MBConv1, k3 $\times$ 3	112×112	1
3	MBConv6, k3 $\times$ 3	112×112	2
4	MBConv6, k5 $\times$ 5	56×56	2
5	MBConv6, k3 $\times$ 3	28×28	3
5	MBConv6, k5 $\times$ 5	14×14	3
5	MBConv6, k5 $\times$ 5	14×14	4
5	MBConv6, k3 $\times$ 3	7×7	1
6	Conv1 $\times$ 1/Pooling/FC	7×7	1

Table 5.9. An overview of the EfficientNet-B0 architecture. It contains 17 layers, but since a layer can consist of different blocks with multiple layers itself, the EfficientNet-B0 has a total of 237 layers. The input is an image of size 224×224. The first stage is a convolution with size 3×3, otherwise the other stages except the last consists of MBConv blocks. The last stage consists of a pointwise convolutional layer, a pooling layer, and an FC layer.

The main building block of the architecture is an MBConv (Mobile inverted Bottleneck), which was introduced in the MobileNetV2, which is a network created to run on mobile devices[Sandler et al., 2018]. The MBConv block is illustrated in Figure 5.9.

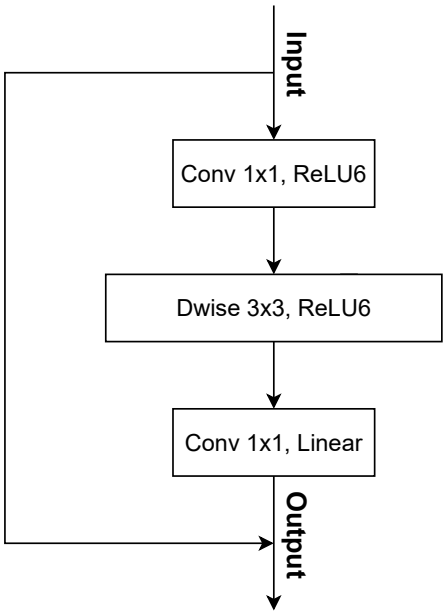


Figure 5.9. Illustration of the MBConv block architecture. The input is of low dimensionality and temporarily linearly transformed. In the first layer, the input is scaled by a 1×1 convolution, and then transformed by the non-linear activation function ReLU6. ReLU6 is very similar to ReLU, but values larger than 6 are capped. In the second layer, a depthwise convolution 3×3 is applied, followed by the ReLU6 again. In the last layer, the input is again downscaled by a 1×1 convolution and linearly transformed. Giving an output of low dimensionality again. The arrow from the input to the output, visualize the residual connection between the low dimensional bottlenecks.

The blocks consist of two parts, namely linear bottlenecks and inverted residuals. The main idea behind linear bottlenecks is that an output is temporarily linearly transformed, and then in an adjacent layer non-linearly transformed. The reason behind the use of bottlenecks is due to a phenomenon introduced by the founders of MobileNetV2, called *Manifold of Interest* (MOI) [Sandler et al., 2018, p. 2]. Translating to that only a subset of an input $H \times W \times C$ is of interest in order to detect an object, where H is height, W is width and C is the number of channels. This means that the MOI in theory can be represented as a low dimensional subspace (an input with few channels) [Sandler et al., 2018, p. 2]. However, reducing the dimensions of an input leads to a problem, which is loss of information when applying a non-linear activation function (i.e. ReLU, Tanh or Sigmoid) to an input with low dimensionality, which is why the bottlenecks are introduced. The authors argue that this is not problematic when applying a non-linear activation function to a high-dimensional input [Sandler et al., 2018, p. 3]. The solution to the problem is to add a 1×1 convolution layer that expands the input by a factor of t, and then apply a non-linear transformation. Afterwards, a depthwise separable convolution is applied, which is an efficient replacement of a convolution layer (See Section 4.1). The last layer, in the depthwise separable convolution, downscales the input again by a pointwise convolution and produces an output that is linearly transformed, mitigating the information loss problem [Sandler et al., 2018, p. 2].

The inverted residual in the MBConv has the same function as a normal residual block, which is to mitigate the degradation problem when stacking multiple layers (see Section 5.3.4). The only difference is that normally the residuals are connected to layers with a high number of channels, whereas in a MBConv they are connected to bottlenecks consisting of a small number of channels [Sandler et al., 2018, p. 3]. The particular placement of the residuals on the bottlenecks means that only the data from the bottleneck layers are needed in order to perform inference. Essentially meaning that the combined amount of memory needed is all the inputs and outputs from the bottleneck layers, the ones with low dimensionality, where layers with high dimensionality can be discarded [Sandler et al., 2018, p. 5-6]. Essentially leading to a very memory efficient architecture, which was showcased by comparing it to other CNNs memory usage [Sandler et al., 2018, p. 5].

Besides the standard MBConv building block, the founders of the EfficientNet architecture also included squeeze and excitation [Hu et al., 2020]. In short terms, squeeze and excitation is a technique that learns the independencies between features in convolutions, and by these learned independencies performs a recalibration. The recalibration essentially squeezes less informative features, and highlights more informative features [Hu et al., 2020, p. 1].

To test the performance of the baseline network EffiicenNet B0, and the scaled EfficientNets, the authors tested the network on the ImageNet benchmark. The EfficicentNet-B0 achieved an accuracy of 77.1%, whereas ResNet-50 scored 76.0%, and DenseNEt-169 scored 76.2%. The interesting observation, besides the slightly better performance, is the fact that the ResNet-50 used 4.9x more parameters, and 11x more FLOPS, whereas DenseNet-169 used 2.6x more parameters and 8.9x more FLOPS [Tan and Le, 2019, p. 6]. In essence, illustrating that the BO-network primarily constituted of MBConv blocks is very efficient in regard to memory and computations. Examining

the scaled networks, the same trend is evident, where networks with similar performance used significantly more FLOPS. The highest scaled network EfficientNet-B7 achieved a state-of-the-art performance on the ImageNet benchmark, with a accuracy of 84.3%, while utilizing 8.4x fewer parameters than the best CNN at the time [Tan and Le, 2019, p. 1].To further prove that the compound scaling method achieves higher accuracy, they scaled existing networks (ResNet-50, MobileNet V1/V2) on one dimension at a time, and with the compound scale method. In all cases, the compound scaling method achieved a higher accuracy while using roughly the same amount of FLOPS [Tan and Le, 2019, p. 5-6].

5.4 Interpretation Methods

A common problem in regard to ML and CNNs is deducing the reasoning behind the model's decisions. Thus, it can be desirable to have interpretation methods, in order to gain further insight into the primary deciding factors behind the models classifications. This section provides an overview of ways to highlight pixels that are relevant for classification outcomes and a methodological walkthrough of choosing a method for interpretation.

5.4.1 Class Activation Maps

Zhou et al. [2015] proposes a technique for generating class activation maps (CAM) in CNNs. A Class activation map is a weighted activation map that for any category indicates the discriminative image regions used by the CNN to identify that category. This practically highlights the areas of interest in a CNN model. This effectively provides a free object localization, which should roughly translate to the class label, that the model is supposed to identify, in the case of object detection. This localization is possible without supplying any explicit bounding-box annotations.

Class Activation Mapping is calculated by projecting back the weights of the output layer onto the convolutional feature maps, as seen in Figure 5.10. Global average pooling (GAP) outputs the spatial average of the feature map of each unit at the last convolutional layer [Lin et al., 2013]. A weighted sum is then calculated for the feature maps of the last convolutional layer to obtain the CAM.

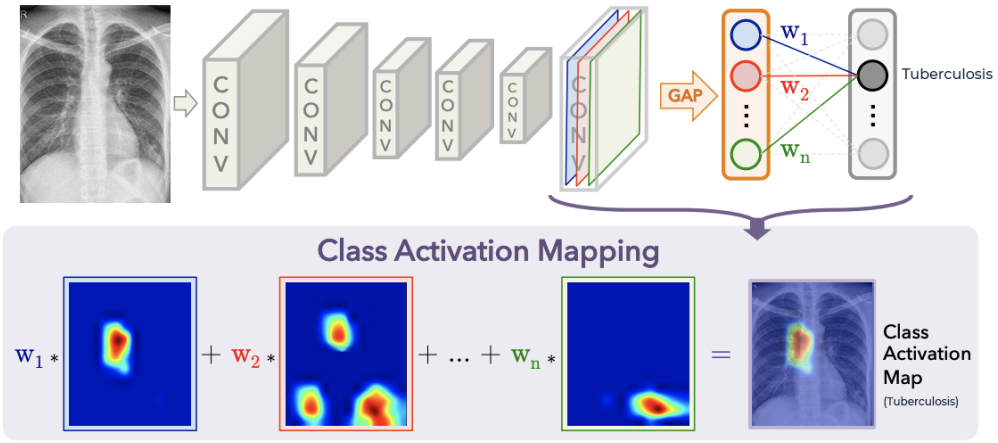


Figure 5.10. Integration of GAP and CAM in a model architecture. The figure is modified from Zhou et al. [2015].

GAP is a strategy to replace the FC layers of the CNN, to get a classification that is more native to the convolutional structure. This effectively enables a direct correspondence between feature maps and categories. This, in turn, increases interpretability, by using the feature maps of each category as confidence maps, as depicted in Figure [Lin et al., 2013]. Meanwhile, there are no real parameters connected to this approach, which in turn, could avoid overfitting, related to the structure of the FC layer. Instead, the resulting vectors from the average of the feature maps are being fed directly into the softmax layer.

CAM requires GAP to be present in the model architecture. More specifically, the FC layer is replaced by feature maps, representing each category, as well as a soft max layer directly preceding them. In order to make this work in architectures, such as VGG, layers have to be removed, and the model re-trained. Thus, CAM is more easily pertained to a specific model architecture.

5.4.2 GradCAM

Instead of CAM, GradCAM can be utilized. GradCam propagates the model architecture with respect to a target class, while storing the gradient and output at a given layer, typically the final convolution [Selvaraju et al., 2016]. Afterwards, a global average of the saved gradient is applied, keeping the channel dimension in order to get a one-dimensional input. This is done in order to correspond to the final layer of the CNN, which will represent the importance of each channel in the target convolutional layer. Each element of the convolutional layer's output is then multiplied by the averaged gradients to create the GradCAM. In contrast to CAM, this implementation is architecture independent, and this is the favored option in this project [Selvaraju et al., 2016].

In addition to CAM and GradCAM, a large array of additional methods are available, such as: GradCAM++, XGradCAM, AblationCAM, ScoreCAM, EigenCAM, EigenGradCAM, LayerCAM, FullGrad. In this project, it has been decided not to further investigate this, as it would take a professional radiologist, to provide useful feedback of the interpretation results. Although, if the model went into production, this should be prioritized.

5.5 Optimization

5.5.1 Hyperparameter Optimization

Hyperparameters are parameters that define the overall structure of the model (such as the number of hidden layers), or control the learning process of the model (such as the learning rate of the gradient method used). Thus, hyperparameters have a high influence on a model's performance. The optimal hyperparameter configuration depends on a given task, and consequently, finding the most suited configuration must be done through some form of search method. There exist many different automatic search methods, of which the three most common are: Grid search, random search and Bayesian search. As such, these three search methods are introduced in this section. The choice of search methods for this project are presented and discussed in Chapter 6.

Grid Search

The grid search is the most simplistic of the three. For each hyperparameter a discrete search space of values should be given, after which the grid search algorithm trains the model with all possible configurations. The configuration that yields the lowest validation error is deemed to be the optimal configuration. With grid search, each training and validation run are independent of one another. As such, multiple runs can be run in parallel to expedite the search process. If given sufficient time, resources, and appropriate search space values, the grid search algorithm will always eventually find the optimal configuration. However, grid search suffers from the curse of dimensionality, and its computational cost grows exponentially with the number of hyperparameters. In particular, for m hyperparameters and each with at most n search space values, the required number of training and validation runs grows as $O(n^m)$. For this reason, despite its ability to run in parallel, grid search is only applicable in practice when using limited search spaces [Goodfellow et al., 2016, p. 432-433].

For the best results, the grid search should be executed multiple times, where the values initially are set to be roughly on a logarithmic scale. E.g. for the number of hidden units in each layer the search space could be $\{50, 100, 200, 500, 1000\}$. Subsequent runs should then refine upon the results of the previous grid search run. For instance, if the grid search algorithm determined that the optimal number of hidden units in each layer is 100, then the following run should have a more refined search space of e.g. $\{80, 90, 100, 110, 120\}$ [Goodfellow et al., 2016, p. 434]. Naturally, this process of stepwise refining the search space could continue until satisfactory results have been produced.

Random Search

With random search, a probability distribution should be provided for each hyperparameter. The common choice is usually a uniform or log-uniform distribution. In contrast to the grid search, the search space values for a random search should not be discretized. Instead, only a range should be provided, as this allows for the exploration of a larger set of values without being at the expense of additional computational resources. In this regard, it has been found that the random search generally is more effective than the grid search, as it generally requires fewer runs to reach a satisfactory configuration. Primarily,

this is due to the extensive nature of the grid search. The grid search has to unnecessarily conduct runs with similar configurations, as it has to check all configurations. On the other hand, a random search has a much smaller risk of doing so given its explorative nature. Similar to a grid search, each run in a random search is independent of one another. Thus, a random search can also take advantage of parallelization to accelerate the search process. Likewise, it is also best to execute a random search multiple times and step-wise refine the search space based on previous runs [Goodfellow et al., 2016, p. 435].

Bayesian Search

In contrast to the other search methods, the Bayesian search is an informed search method. Here, the past results are used to build a probabilistic model, that is used to guide the search. With Bayesian search, the search for the optimal hyperparameter configuration is viewed as an optimization problem, where the cost to be optimized is the validation set error [Goodfellow et al., 2016, p. 435]. It is a *Bayesian optimization* approach, which aims to find the extrema of an objective function. The objective function, however, is assumed to be a *black box* function. As a result, a probabilistic model is used to model the objective function, which for this reason is called a *surrogate model*. As the name implies in Bayesian optimization, the principles of Bayes' Theorem are used to incrementally update the surrogate model [Brochu et al., 2010, p. 5]:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{5.5.1}$$

where:

P(A|B)	Posterior probability
P(B|A)	Likelihood
P(A)	Prior probability
P(B)	Evidence

The *posterior* is the updated beliefs about the objective function. As such, the surrogate model – the beliefs about the objective function – is updated using the *prior* beliefs about the objective function, and *evidence* gathered from interacting with the objective function. Naturally, the quality of each surrogate model update is heavily dependent on where the evidence has been sampled. Thus, in order to determine where to sample evidence most optimally, an *acquisition function* is used. The acquisition function uses the posterior to determine where to acquire the next sample. It balances the trade-off between exploration and exploitation. With exploration, information is gathered by exploring configurations with a high uncertainty, where it is unknown whether they will perform poorly or well. With exploitation, configurations with high certainty of performing well are used [Brochu et al., 2010, p. 5].

Summarily, in the context of hyperparameter optimization a Bayesian optimization can be summarized as follows:

1. A configuration is selected using the acquisition function given the current beliefs.
2. The hyperparameter configuration is evaluated and evidence – validation error – is gathered.

3. The surrogate model – that is, the beliefs about what can lead to a lower validation error – is updated.

The steps above can then be repeated until satisfactory results has been achieved.

In general, the Bayesian search requires fewer runs to find the best configuration compared to grid search and random search. However, one inherent drawback is that it is difficult to parallelize due to its model-based and sequential nature. Another drawback to keep in mind, is that, it generally requires more computational resources than its counterparts [Yu and Zhu, 2020, p. 18, 22].

5.5.2 Hyperparameters

Now with the different search methods outlined, the hyperparameters that have been chosen to be optimised are as follows:

- *Optimizer:* The stochastic gradient method used for learning. Two methods are considered in this project, Stochastic Gradient Descent (SGD) and RMSprop, as they currently are among the most popular methods. Adam, which is an equally predominant method, was initially also considered. However, preliminary testing showed that Adam was lackluster for this task. The main difference between SGD and RMSprop is that RMSprop makes use of individual adaptive learning rates, meaning that each parameter in the model has its own learning rate, which is adjusted adaptively during the course of training.

- *Learning rate:* The step size of the stochastic gradient method. As such, it sets the model's convergence speed. As presented in Section 3.4, it has a significant influence on the model's performance, as both a too high and too low learning rate will produce suboptimal results.

- *Learning rate scheduler:* Gradually reduces the learning rate during training in a fixed manner. This is useful, as higher precision and smaller steps are usually preferred later in training. Otherwise, the model might run the risk of missing a local minimum when its steps are too large. An exponential scheduler is in this project used, as it is recommended to update the learning rate on a logarithmic scale. The schedule exponentially decays the learning rate by γ for every epoch. The exponential schedule starts out with strong changes, but gradually becomes more gentle as the model approaches convergence [Yu and Zhu, 2020, p. 5,7].

- *Momentum:* Both SGD and RMSprop have been implemented with momentum to accelerate learning in an environment with stochastic noise. Momentum uses the hyperparameter $\alpha \in [0, 1)$, that dictates how quickly the contributions of previous gradients exponentially decay [Goodfellow et al., 2016, p. 296].

- *Batch size:* The sample size used to statistically estimate the gradient for each training pass. The batch size are set in powers of two, 2^n, as this usually yields faster runtime when using GPUs for training [Goodfellow et al., 2016, p. 279].

- *Weight decay:* A regularization strategy that adds a penalty term to the objective function, which limits the sizes of a model's weights. Weight decay has the hyperparameter, λ, which controls the strength of the weight decay.

- *Dropout rate:* Three of the five considered CNN architectures – namely VGG, Inception and EfficentNet – makes use of a dropout in their network as a regularization strategy. The dropout rate is the probability p that a neuron – in a layer with dropout – is deactivated for a given training pass.

Search spaces

Now that the hyperparameters have been outlined, their search spaces needs to be defined. The ranges are chosen on the basis of the papers of the five architectures, VGG, Inception V3, DenseNet, ResNeXt and EfficientNet, presented in Section 5.3. For each of the hyperparameters, the ranges used have been collected from the five architectures, where the minimum and maximum values found define the range for that hyperparameter. The only exception to this is the very large batch size of 4096 found for some EfficentNet variants, which is infeasible to run given the resources available. The second largest batch size was 256. Thus the upper limit was set to 256 instead. The hyperparameter search spaces are shown in Table 5.10.

Hyperparameter	**Search Space**
Batch size	$[32 - 256]$
Optimizer	[SGD, RSMProp]
Exponential Scheduler	$[0.01 - 0.05]$
Momentum	[0.9]
Learning rate	$[0.001 - 0.1]$
Weight Decay	$[0.000005 - 0.0001]$
Dropout rate	$[0 - 0.5]$

Table 5.10. The hyperparameter search spaces.

5.5.3 Early Stopping

One challenge with training an ML model, is naturally how long to train it. Too little training leads to underfitting and too much training leads to overfitting, where the latter is usually the most common problem. In this regard, *early stopping* is a regularization strategy to address this issue, by stopping training earlier, such that overfitting is avoided. Thus, early stopping can be viewed as a hyperparameter optimization tool that dynamically sets the epoch amount. With early stopping, a validation set is needed, in order to determine when overfitting arises during training. Using a validation set, overfitting is detected when training error continues to decrease, but validation error begins to increase [Goodfellow et al., 2016, p. 246-247].

The model's parameter settings are at their most optimal when validation error is at its lowest. Thus, with early stopping, every time validation error improves over a pre-specified *save threshold*, the model's parameter settings are saved. Then, when validation error has not improved for some pre-specified number of epochs, called *patience*, training

is terminated. The model is then set to the latest saved parameter settings [Goodfellow et al., 2016, p. 246-247].

For this project early stopping is done with the following settings:

- A validation checkpoint occurs every third epoch.
- The save threshold is set to 10^{-4}.
- The patience is set to five.

Evaluation 6

The following chapter is devoted to evaluate the five chosen CNN models with the use of the extended dataset, the implemented optimization methods and to evaluate how diseases with similar radiographical features impacts a TB classification problem. Each part of the evaluation is explained below:

- **Part 1 - U-Net:** Is evaluated, to assess if lung segmentation can increase the model performance in regard to classification of TB CXR images.

- **Part 2 - Model Selection:** The five models are trained and hyperparameter optimized on the extended dataset with similar diseases. The model which achieve the lowest validation loss after hyperparameter optimization is chosen going forward in the evaluation.

- **Part 3 - Data Augmentation:** The best-performing model is evaluated with different combinations of data augmentation methods.

- **Part 4 - Extended Hyperparameter Optimization:** A more extensive hyperparameter optimization is conducted, where more specific ranges are utilized and optimization is given a longer runtime.

- **Part 5 - CAM:** Is evaluated, where generated heat maps are presented and interpreted.

- **Part 6 - Baseline Comparison:** The final model is compared to a baseline model. The baseline model has been selected based on related research papers, where the model with the best performance has been selected. Furthermore, the baseline model has been trained in a simplified setting, with only CXR-images of healthy lungs and TB-infected lungs (akin to current papers). Both the final model and the baseline model are evaluated on two test sets. A baseline test set containing CXR-images of only healthy and TB infected lungs. An extended test set containing CXR-images of healthy, TB-infected, pneumonia, lung cancer and COVID-19 lungs.

6.1 Evaluation Methodology

In the following sections the metrics and methods used to execute the evaluation parts are presented.

Performance Metrics

A model's performance on the test sets are evaluated using eight metrics: Accuracy, Positive Predictive Value (PPV), True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), F1-score and Area Under the Curve of the Receiver Operating Characteristics (AUC-ROC). A model's performance prior to the final baseline comparison tests, is evaluated using validation error.

Accuracy is a commonly used metric for classification, and is used in this project to easily gauge the performance difference between models. It is simply the percentage of all predictions that is correct [Hossin and M.N, 2015, p. 1-2]. The other metrics have been added to further evaluate the final model's performance and how it compares to the baseline model. As an example, this allows for examining the percentages of images that were miss-diagnosed as not having TB. These cases are not always directly evident from the accuracy, as the accuracy is based on all predictions made, and not specific for positive predictions as an example.

To calculate the aforementioned metrics, four values are used, namely [Hossin and M.N, 2015, p. 3]:

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

To illustrate how the values are calculated, a confusion matrix is showcased on figure 6.1, which for binary classification problems is simply a 2×2 matrix [Hossin and M.N, 2015, p. 3]. To get the sum of FN's, it is simply just cross-referencing the cases where the image label is negative, and the predicted label by the model is positive. In fact these cases are the most undesirable outcome in the given context, as they would have an immense consequent for a patient in the real world.

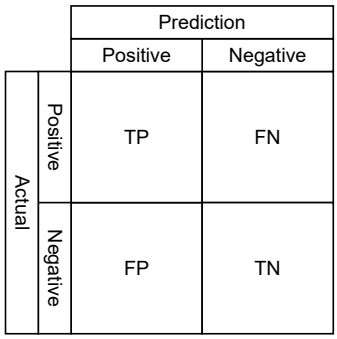


Figure 6.1. Confusion matrix for binary classification. The four parameters are calculated by cross-referencing the predicted class label with the actual class label for an image.

Using the four metrics, the eight selected performance metrics can be calculated. For readability reasons the results in Section 6.2 are presented in percentage by multiplying the value with 100, rather than presenting them as rates. The 8 utilized metrics are defined below [Hossin and M.N, 2015, p. 4]:

Accuracy
Outputs the proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + FN} \tag{6.1.1}$$

PPV
Outputs the proportion of all positive predictions that were correct:

$$PPV = \frac{TP}{TP + FP} \tag{6.1.2}$$

It can indicate the model's quality of predicting images classified as having TB (it is also referred to as precision).

True Positive Rate (TPR)
Outputs the probability of a model classifying an image as TB, given that the image is classified as TB (it also referred to as sensitivity):

$$TPR = \frac{TP}{TP + FN} \tag{6.1.3}$$

True Negative Rate (TNR)
Similar to TPR, but for the cases where the images are classified as not having TB (it is also referred to as specificity):

$$TNR = \frac{TN}{TN + FP} \tag{6.1.4}$$

False Positive Rate (FPR) and False Negative Rate (FNR)
Similar to TPR and TNR, but outputting the rate or frequency of a false positive/negative case occurring:

$$FPR = 1 - TNR \tag{6.1.5}$$

$$FNR = 1 - TPR \tag{6.1.6}$$

F1-score
Takes both TPR and PPV into account and computes the harmonic mean of the two. The essential logic behind is that, if a model achieves a high TPR it means that the model is good at predicting positives cases, but it does not take FPs into account. This means that a model can achieve a high TPR, while still misclassifying a large proportion of the images as being positives. This is naturally undesirable, which is why the F1-scores also includes PPV, which takes FPs into account. In this way, the F1-score only returns a high output if a model is both good at predicting positives values (low number of FN), and it is precise in its positive predictions (low number of FP). The F1-score is defined as follows:

$$\text{F1-score} = \frac{2 \cdot (TPR \cdot PPV)}{TPR + PPV} \tag{6.1.7}$$

AUC-ROC

The AUC-ROC is essentially used to visualize and measure how well a classifier is at discriminating classes in a classification problem, which in this case is a binary classification [Fan et al., 2006, p. 1]. AUC-ROC consist of two parts, namely Receiver Operating Characteristics curve (ROC curve) and Area Under The Curve (AUC). The ROC curve plots FPR on the x-axis and TPR on the y-axis at various thresholds (a.k.a. cut-off point), based on the probabilities from the positive predictions only. The thresholds determine at which point a prediction should be labelled as positive. As an example, if a classifier predicts a positive class with a probability of 0.6, and the threshold is set to 0.7, the prediction will be labeled as negative. Thus, with lower thresholds, more predictions are labelled as positive. This will likewise result in more FPs and TPs, as the degree of certainty declines relative to the threshold. Subsequently, this results in an increase for both the TPR and FPR as the FP increases. The concept is illustrated in Figure 6.2. By examining different FPR and TPR values at various thresholds, one can decide on an ideal threshold value. This is naturally case depend and is a trade-off between higher TPR or lower FPR [Fan et al., 2006, p. 1].

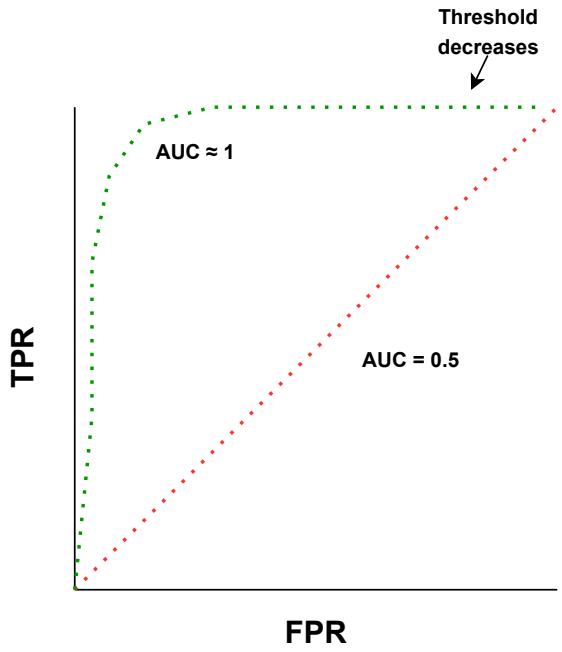


Figure 6.2. The green ROC curve illustrates a optimal curve, with a AUC approximately equal to one. The optimal ROC curve has a TPR constant equal to one, and FPR increasing relative to the decrease in threshold. The red ROC curve showcases a curve with a AUC equal to 0.5. This is the case where a classifier is not able to separate between two classes. The TPR and FRP values are based on various thresholds, where the plots to the right on the x-axis correspond to low threshold values.

The AUC is calculated based on the ROC curve, which determines a classifier's capability to discriminate between two classes. A score equal to one means perfect separability, a score of 0.5 means no class separation capacity (random), and a score of 0 is the reversed where a classifier always predicts the false class. The desired outcome is naturally a score close to 1, which correspond to a ROC curve which has a constant TPR value of 1 and

an FPR which increases relative to the decrease in thresholds [Fan et al., 2006, p. 1-2] (see Figure 6.2). The logic behind this is that if the TPR remain constantly high through various thresholds, only the number of FP vary, but the level of FN remain constant low. This means that the classifier has perfect separability, because at high threshold all the probabilities above are mapped to TNs, showcasing that the classifier can correctly classify positive cases. Secondly, all the probabilities below are correctly mapped to the negative cases, thus not producing any FN. Likewise, when the threshold is very low all the predictions below the threshold are mapped to TN and only the FP is increasing, thus showing perfectly capability to identify negative cases.

6.1.1 *K*-fold Cross Validation

In order to evaluate the generalization performance of the developed model, there needs to be a test set of unobserved data that have not been seen during training. However, this test set can only be used once for the final evaluation. If, for instance, the hyperparameter optimization is validated using the test set, the hyperparameter optimization will be biased towards performing well on the test set. In doing so, the test set losses its purpose, as it is no longer measures generalization performance on unobserved data. Thus, in order to validate the performance of a model during training, or validate the performance of a hyperparameter setting, one needs use some of the training data as a validation set. As such, a dataset is usually split intro three sets: Training set, validation set and test set.

However, a problem that arises in this regard, is that a model's performance is dependent on how the training set and validation set are split. In this regard, k-fold cross validation is used to get a more accurate estimate of a model's performance. K-fold cross validation splits the training set into k subsets. Subsequently, the model is trained k *folds*. For each fold a model is trained on $k-1$ subset, after which it is validated using a single subset. However, for each fold a different subset is used. In doing so, different parts are used for training and validation for each fold [Russell and Norvig, 2010b, p. 708]. Finally, an estimate of the model's performance is computed by averaging the average validation loss of all folds. A k-fold cross validation is visualized in Figure 6.3.

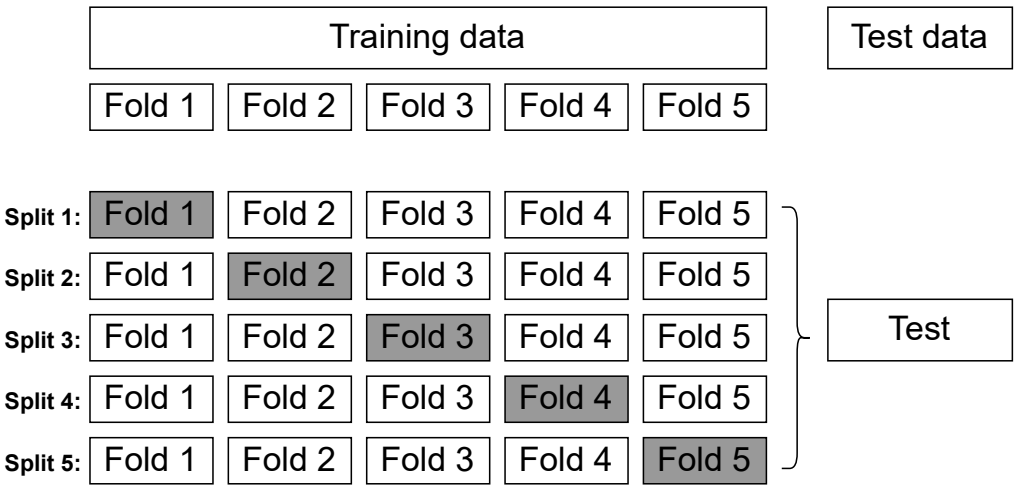


Figure 6.3. K-fold cross validation where k = 5, and the training/test split is 80/20. The training data is divided into 5 folds and splits. The gray box in each split marks the validation set for that individual split. Each split provides a fully trained model.

In this project, k is set to three, despite the normal range of folds being five to ten. This is due to restricted time and resources, and cutting k from five to three shortens the time for each k-fold run by approximately 40 %. This results in vastly faster evaluations and is estimated to be worth the trade-off in regard to accuracy of the models performance.

6.2 Experiments

Google Colab is used for all experiments conducted in this project. Google Colab is an online service, which provides access to a GPU server (Tesla T4 GPU or P100). Using Google Colab a total of 580 hours were used. The training set accounts for 80% of the dataset, and thus the test set accounts for 20%. From the 80% of the training set, 20% is used as a validation set in accordance with k-fold cross validation, using three k-folds, with a more in-depth description provided in Section 6.1.1. While training, early stopping is used, with the configuration presented in Section 5.5.3. For all the experiments, the extended dataset (see Table 5.3) is used. An exception is the baseline comparison in Section 6.2.6. In this section, a baseline model and dataset is used to gauge the comparative performance between the project and status quo in literature. Hyperparameter optimization is implemented using the Weights and Biases library [Biases, 2022]. Both random, grid and Bayesian search were used under different circumstances, as explained in their respective experiments. The source code for the experiments is available from the project's GitHub, which is linked in the preface.

6.2.1 U-Net

After the initialization of U-Net, a series of tests were run with the dataset with associated masks. In extension to this, U-Net is applied to the TB dataset constructed in this project, by transfer learning.

In order to successfully evaluate the predicted masks, the metric *Intersection Over Union* (IOU), is used to measure the successful application of predicted masks, comparatively to the ground truth mask supplied by radiologists. IOU works by dividing the area of overlap with the area of union, as visualized in Figure 6.4. For now, further metrics are not considered, as the initial stage solely will focus on a successful implementation of the model, that is somewhat comparative to other papers' performances.

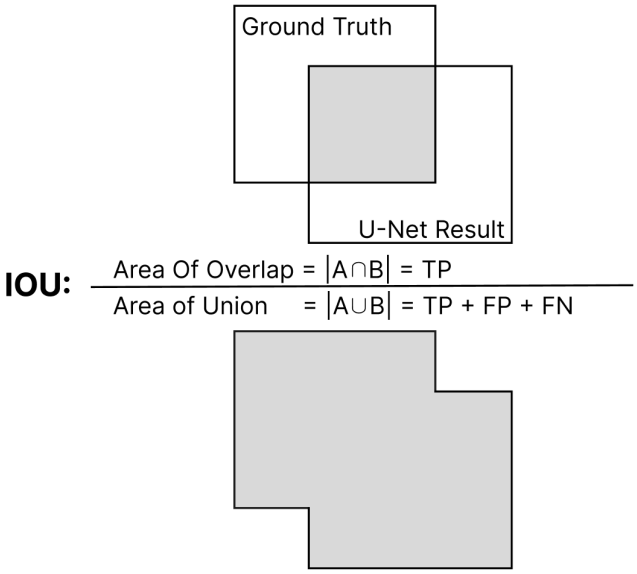


Figure 6.4. IOU Score calculation

As seen in Figure 6.5, after training, the performance in the validation accuracy converged around 96.8-98.2% accuracy and the loss around 0.2, which shows great potential for using it in order to increase the performance of the TB classification. The IOU score converges around 0.91, which is similar to the 0.92 score presented in the original U-Net paper [Ronneberger et al., 2015]. Although targeted towards segmentation of cells, the results still hints at a successful implementation. To truly gauge the performance, comparative results in Rahman et al. [2020] on TB CXR images reports an IOU score of 92.4, which is a comparative performance to the implementation in this project.

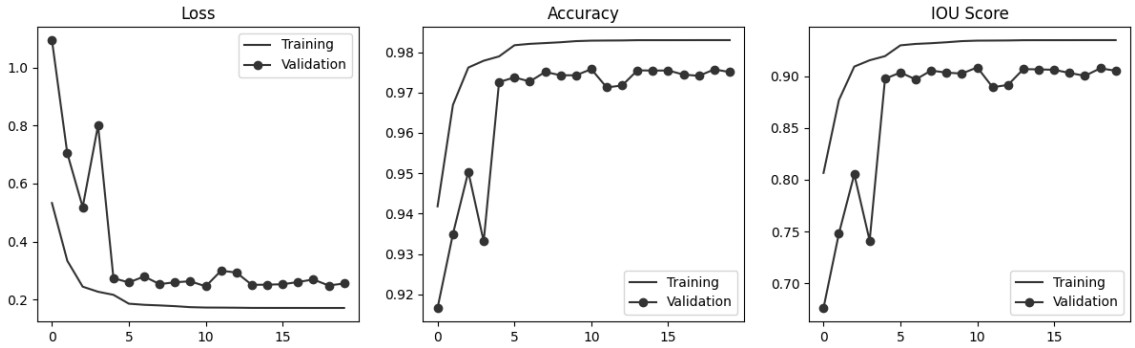


Figure 6.5. Results of training and validation of U-Net, including Loss, Accuracy and IOU score.

Figure 6.6 showcases an example of an original CXR image and ground truth mask presented along with the predicted mask. These results seem to correlate with the training performance in Figure 6.5, as the ground truth mask and predicted mask seem to overlap to a high degree.

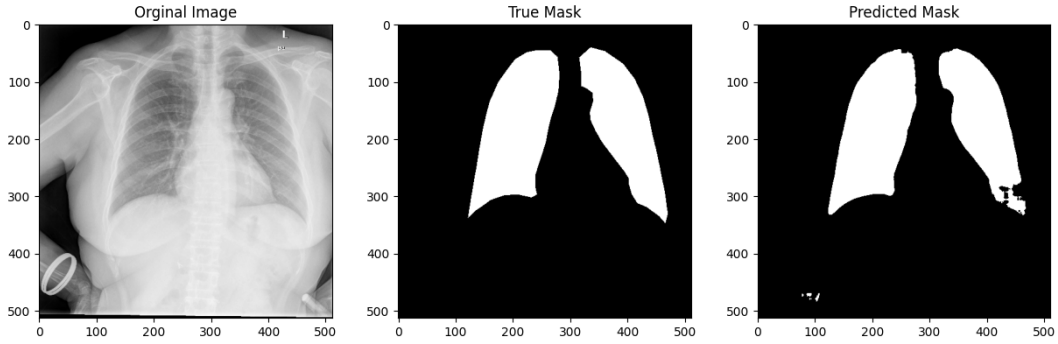


Figure 6.6. Prediction mask on masked dataset.

As most datasets of lungs do not have associated masks with them, transfer learning is used between the dataset with associated masks, and the extended dataset. Upon implementation, a large discrepancy seems to occur. As seen in Figure 6.7, a large amount of mask – small spots of lungs or entire lungs – are non-masked. From visual inspection, it can be deduced that this is nowhere near the performance of 96.8% to 98.2% accuracy reported in Figure 6.5.

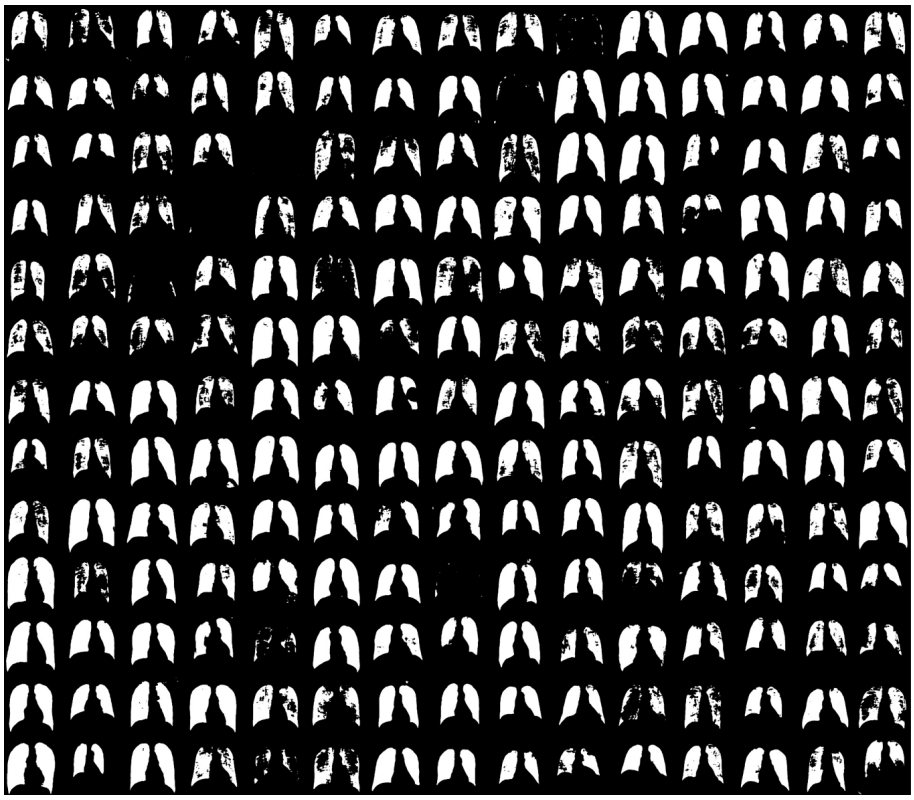


Figure 6.7. Transferability of performance on non-masked tuberculosis dataset.

From this experiment, it can be concluded that the variance between the dataset with associated masks and the project's extended dataset is too great. This results in unsuccessful transfer learning between the two. Thus, the usage of U-Net for enhancing performance, is not feasible in the scope of this project, which could be due to lack of

variance in the training set, or could perhaps be mitigated by using another classifier in the U-Net, as seen in a multitude of other papers [Iglovikov and Shvets, 2018; Diakogiannis et al., 2019; Jha et al., 2019].

Based on the presented result, it is possible to selectively use successfully segmented mask. It is also possible to fill out small missing spots. The aforementioned approaches would, however, compromise the generalizability and possibility of actual deploying the model. As it would require human inspection, and therefore make the implementation redundant, thus U-Net is not utilized in the final implementation.

6.2.2 Model Selection

In this section, the five architectures presented in Chapter 5.3 are evaluated, in order to find the best performer. The reasoning behind evaluating the models and not simply picking the best performer in a paper or of the ImageNet challenge, is the fact that the extended TB detection presents a new case, where an unexpected architecture potentially could perform better.

All variants of the five architectures were not tested as this would be too extensive, EfficentNet alone has seven variations. This resulted in the choice of limiting the amount of variations of each architecture. For VGG, DenseNet and EfficientNet three variations for each was picked; The smallest, a median and the largest variants. For Inception and ResNeXt have fewer than three variations with pre-trained models, therefore, all variations of these will be tested. This resulted in 12 different variations of the architectures, an overview of which can be seen in Table 6.1.

Model	Max Batch Size	Image Size	Avg. Val. Acc.	Avg. Val. Loss	Runs
VGG11_bn	64	224	95.369%	0.1059	54
VGG16_bn	64	224	96.382%	0.08923	40
VGG19_bn	**64**	**224**	**96.932%**	**0.0827**	**32**
Inception V3	128	299	96.566%	0.08593	25
DenseNet 121	64	224	87.496%	0.3202	52
DenseNet 161	32	224	94.153%	0.1487	27
DenseNet 201	32	224	96.758%	0.0887	32
ResNeXt 51	64	224	87.582%	0.3256	20
ResNeXt 101	32	224	96.613%	0.09047	11
EfficientNet B0	128	224	96.469%	0.0972	65
EfficientNet B4	16	380	96.006%	0.1119	14
EfficientNet B7	2	600	68.442%	1.425	1

Table 6.1. Results of the initial hyperparameter optimization, along with the maximum batch sizes and ideal image sizes of the models.

All 12 variations were hyperparameter optimized using Bayesian search, with the search spaces defined in Section 5.5. The only exception to this was the batch size, where some models could not run with the higher values in the search space on the utilized Google Colab GPUs. These models were tested for an upper limit with which they could run, and the value found was set as the maximum batch size for that variation. These upper limits can be viewed in the Table 6.1.

The hyperparameter optimization of each variation was stopped after a duration of 24 hours, due to time constraints. This meant that the number of runs each model could reach within the time-frame varied quite drastically. Each model was configured with the optimal input image size, as described in Section 5.3. The results of the hyperparameter optimization are shown in Table 6.1.

The results were close with at least one variant of each architecture hitting an average validation accuracy of 96 %. The average is taken from the validation accuracy of the three k-fold runs. VGG19 provides the best performance with an average validation accuracy of 96.932 % and an average loss of 0.0827. As such, the VGG19 architecture is selected with the hyperparameters shown in Table 6.2.

Hyperparameter	**Value**
Batch size	32
Optimizer	SGD
Exponential Scheduler	0.02837
Momentum	0.9
Learning rate	0.002987
Weight Decay	0.00009917
Dropout rate	0.05234

Table 6.2. Developed model hyperparameters

An observation from the results is that larger models ended up not being able to run large batch sizes and had a fewer runs than the smaller models, since they take a longer time to train, potentially making further gains possible if more time was allocated to training.

This also seems to have affected the results of EfficentNet B7, as this architecture only had one run within 24 hours, as most of it's training runs crashed due to it surpassing the allowed GPU allocation time on Google Colab. This resulted in resulting in EfficentNet B7 getting the worst score of all architectures.

One could hypothesize that had EfficentNet B7 been tested with a larger batch size and for more runs, it could potentially have outscored the other models. As it seems from all the other models that the larger variants of the architecture overall scored better than the smaller variants. This is not too far of a leap, since the smallest model-variant EfficentNet B0 scored just 0.463 % lower in accuracy than the highest performer, VGG19. However, as it was not possible to test EfficientNet B7 under ideal circumstances because of its large size and slow training time. Thus, it was decided to go with the VGG 19 architecture.

6.2.3 Data Augmentation

Proceeding from the model selection, the optimal data augmentation is evaluated with the use of the VGG19 model, with the same hyperparameter configuration presented in subsection 6.2.2. The four data augmentation methods and ranges can be seen in Section 5.2.3. The preprocessing method, grayscale, has been included in the experiment as well, to see if it affects model performance. This gives five methods in total to test. They are all applied to the training set only, to ensure that the model is not validated on non-clinical

images. All combinations of the methods are evaluated with use of grid search, which gives a total of $2^5 = 32$ runs. The top five runs can be seen in Table 6.3, which have been sorted based on the average validation loss. The run with only grayscale enabled performed the best, with an average validation loss of 0.0848 and an average validation accuracy of 96.452%. The loss is marginal higher compared to the VGG19 run model selection (see Section 6.2.2) with a difference of 0.021. The reason behind is due to a implementation difference which had to be created for this experiment, meaning that the data was shuffled differently from the Model Selection.

Model Performances						
Grayscale	**Horizontal Flip**	**Vertical Flip**	**Color Jitter**	**Affine**	**Avg. Val. Acc.**	**Avg. Val. Loss**
true	false	false	false	false	0.08487	96.452%
true	true	false	false	false	0.08507	96.699%
false	false	false	true	false	0.08663	96.424%
false	false	false	false	false	0.08933	96.644%
true	false	true	true	false	0.09170	96.369%

Table 6.3. Top five performing data augmentation combinations. The methods used are showcased by the corresponding name, and true/false values. The test are sorted based on the average validation loss.

Comparing the grayscale run to the run were no methods are used, the grayscale run scores only 0.0046 lower in the validation loss, and actually achieved a lower accuracy of 0.192%. Essentially a marginal improvement, which only indicate that grayscale enhance the performance, but not to a determinable degree. However, this is due to an implementation error, where grayscaling only was applied to the training set. As grayscaling an image does not change the clinical structure of an image, it can naturally be applied to the validation set as well. Due to this a second run with grayscaling was conducted, which resulted in a loss of 0.0816 and an accuracy of 97.029%. Giving an additional decrease of 0.0032 (3.92%) in validation loss compared to the run where grayscale was only applied to the training set. However, compared to the baseline run from this experiment with no data augmentation methods applied, it is a decrease of 0.0077 (8.62%). Based on these findings, grayscaling is implemented as a preprocessing method of the whole data set.

Examining the general pattern of the 32 runs, they all scored closely to the grayscale run or slightly worse, except runs using affine. With affine enabled, they all scored significantly worse, which is illustrated in Figure 6.8. This can indicate that the ranges configured for the affine method are not set up optimally. For instance, this can be due to the ratio between the scaling of the image and the rotation degree. This can have resulted in excess image distortion, essentially leading to information loss, which can have compromised the model's capabilities to classify an image. Due to limited resources this is not investigated further, but is something which can be investigated in future works.

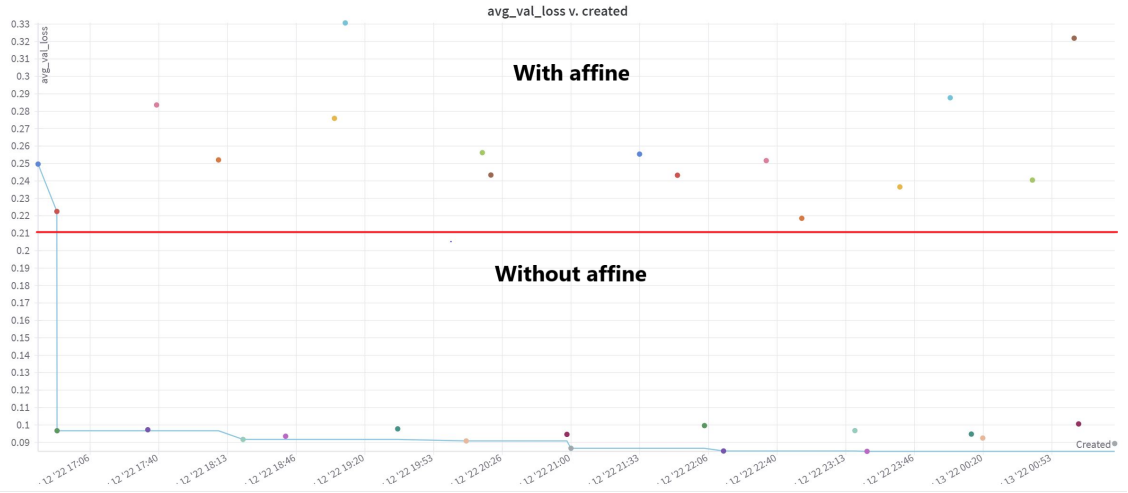


Figure 6.8. An illustration of runs with and without affine. The runs with affine enabled are above the red line. The y-axis shows the average validation loss, and the x-axis shows the timestamp for the specific run.

6.2.4 Extended Hyperparameter Optimization

Proceeding with the results from the data augmentation, the next step is further hyperparameter optimization on the model VGG19. In this regard, the parameters, ranges, and probability distributions are reevaluated based on the prior hyperparameter results. In this section, the prior hyperparameter results are therefore analyzed and used to determine the configuration for the new hyperparameter optimization. Finally, the new results are examined.

Search Spaces

The prior hyperparameter optimization of VGG19 had very varying scores of average validation loss, as illustrated in Figure 6.9.

Figure 6.9. Average validation loss for the first iteration of hyperparameter optimization. The x-axis is the time of execution.

Figure 6.9 shows that the worst loss was higher than 1.1e+9 (> 1,100,000,000). It should also be noted that the runs in the bottom seem to perform very similar, but this is only due to the very high values on the y-axis. Figure 6.10 shows the same results but with a y-axis in the range of 0 to 1.

Figure 6.10. Average validation loss for the first iteration of hyperparameter optimization, with a range of 0 to 1 in the y-axis. The x-axis is the time of execution.

Figure 6.10 shows that for the 29 runs of hyperparameter optimization for VGG19, only 11 runs had an *average validation loss* below one. Furthermore, the figure illustrates a cluster of five runs - those with a score below 0.2 - that noticeable outperforms the other runs. These runs will therefore lay the foundation for the next hyperparameter optimization configuration, as they are the top performers. The hyperparameter configuration for these five runs is shown in Figure 6.11.

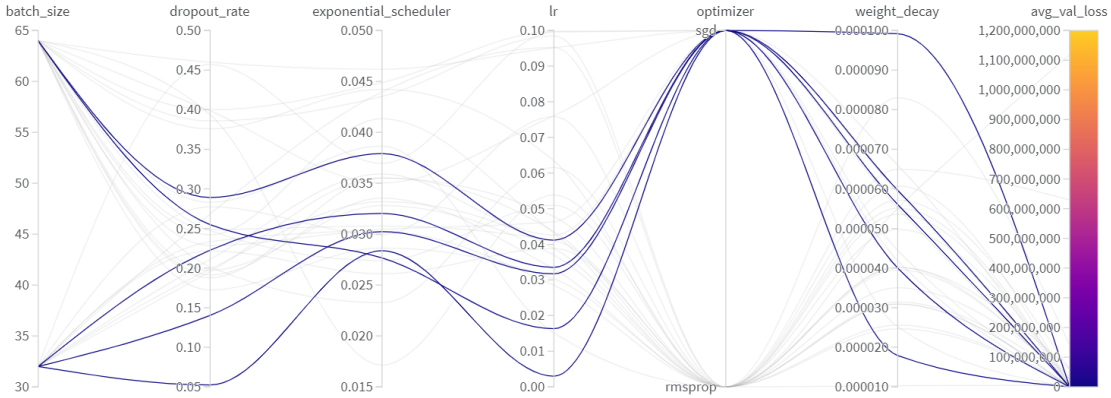


Figure 6.11. Hyperparameters for the top five runs of the first iteration of hyperparameter optimization (blue lines). The curves illustrates the hyperparameters' configuration for each run as well as the resulting *average validation loss*. The gray lines are the remaining runs.

Figure 6.11 will determine the new ranges for the hyperparameters in the new optimization; I.e. optimizer, batch size, dropout rate, exponential scheduler, learning rate, and weight decay.

Optimizer
Evident from the fifth column in Figure 6.11, the top five runs all used the SGD optimizer. For this reason, only this optimizer will be used in the next hyperparameter optimization.

Batch size
Figure 6.11 shows in the first column that the batch size was quite evenly distributed. Two of the top performers had a batch size of 64, and the three others had a batch size of 32. The best run had as well a batch size of 32. However, Figure 6.12 shows that there is a small linear correlation between a higher batch size and a lower average validation loss.

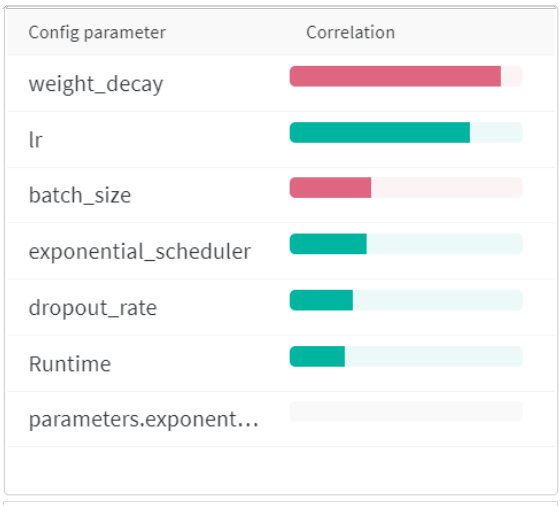


Figure 6.12. Correlation for hyperparameters for the top five runs. A red correlation, means a negative correlation. E.g. an increase in weight decay is highly linearly correlated with a decrease in average validation loss.

For these reasons, the ranges for the batch size will not be changed, as there is no unambiguous results for the best batch size.

Dropout rate
Figure 6.11 shows in the second column that the top five runs all had a dropout rate below 0.3. Furthermore, Figure 6.12 showcases that there is a weak positive linear correlation between *dropout rate* and the average validation loss. I.e. an increase in *dropout rate* is weakly linearly correlated with an increase in *average validation loss*. For these reasons, the range for dropout rate will be decreased from [0 - 0.5] to [0 - 0.3].

Exponential scheduler
Figure 6.11 shows in the third column that the top five runs all approximately had an exponential scheduler rate between 0.025 to 0.04. Therefore, this will be the new range.

Learning Rate
Examining the fourth column in Figure 6.11, the top five runs all had a learning rate approximately below 0.04. Therefore, 0.04 will be the new maximum value. Furthermore, Figure 6.12 shows that there is a high positive linear correlation for the learning rate and average validation loss. I.e. a lower learning rate is linearly correlated with a lower average validation loss. Therefore, a uniform probability distribution for the learning rate in the hyperparameter optimization, might not be optimal, as values closer to 0 outperforms

values closer to the max value of 0.4. As a result, a log-uniform probability distribution is applied instead such that the learning rate has a higher probability of being close to 0. Specifically, this distribution returns a learning rate, *lr*, such that log(lr) is uniformly distributed between log(min) and log(max), where the minimum is 0.0001 and maximum is 0.4.

Weight decay
Figure 6.12 shows in the correlation column that the top five runs had a very linear negative correlation between the weight decay and the average validation loss. I.e. a higher weight decay increases the performance of the model. The initial range for weight decay was [0.000005 - 0.0001] and Figure 6.11 shows that the best run had a weight decay very close to the maximum value of 0.0001. Therefore, the natural change in the weight decay range, would be to increase it. However, choosing a new range for the weight decay is quite difficult, as no upper bound can be concluded from the results; It is completely unknown if the linear negative correlation for example continues for 10x, 100×, 1 000×, or even 10 000× of the original max value. Therefore, this has been tested, and the results are shown in Table 6.4, where the first row is the initial max value for weight decay.

Weight Decay	Weight Decay Increase Factor	Avg. Val. Acc.	Avg. Val. Loss
0.0001	1	96.645%	0.0862
0.001	10	96.48%	0.08687
0.01	100	97.002%	0.08463
0.1	1 000	96.562%	0.08803
1	10 000	68.482%	0.656

Table 6.4. Results of the weight decay increase factor test.

Table 6.4 shows that the average validation accuracy and the average validation loss is very similar for the 1, 10, 100, and 1 000 increase factor. However, the 10 000 increase definitely worsens the performance, as both the loss and accuracy is significantly worsened.
These results indicate almost zero linear correlation for the 10×, 100×, and 1 000× increase in weight decay, which is opposed to the indications from Figure 6.12, that showed a large linear correlation. The increased ranges from the weight decay test is, however, quite significant, as it is increased 10× each time. Therefore, the optimal value, may be somewhere in between, that have not yet been tested. For these reasons, a broad range for weight decay have been chosen going forward: [0.001 (10×) - 0.1 (1 000×)].
As the weight decay value increases, so does the corresponding uncertainty in model performance. For this reason, most of the runs should have a weight decay value closer to the minimum rather than the maximum. Therefore, this parameter also utilizes the log-uniform probability distribution, which the learning rate also utilizes.

Search Method
The search method has also been revised. Overall, the results from the first iteration of hyperparameter optimization (Appendix C) indicate that the advantages of using Bayesian search might not be present in this case, as many of the best runs were found in the initial, randomly assigned first configuration. This is, for example, the case for the best run in the prior hyperparameter optimization, as seen in Figure 6.10. This indicates that with

the chosen search space and time limit, Bayesian search have not provided any significant advantage. For these reasons, a random search is used instead. Furthermore, a random search have the benefit of being able to run in parallel. The random search is therefore run with four parallel searches, which enables the hyperparameter optimization to run approximately four times more runs in the same time frame.

Table 6.5 shows an overview of the new setup for the new hyperparameter optimization.

Hyperparameter	Range
Optimizer	SGD
Batch size	[32, 64]
Dropout rate	[0.0 - 0.3]
Exponential scheduler	[0.025 - 0.04]
Learning rate	[0.0001 - 0.04] (log uniform)
Weight decay	[0.001 - 0.1] (log uniform)
Search method	Random

Table 6.5. Setup for the extended hyperparameter optimization.

Results

By running the random search in four parallel sessions for approximately 48 hours (including GPU time-outs), 175 runs with different configurations have been tested. Figure 6.13 illustrates all the configurations that have been tested.

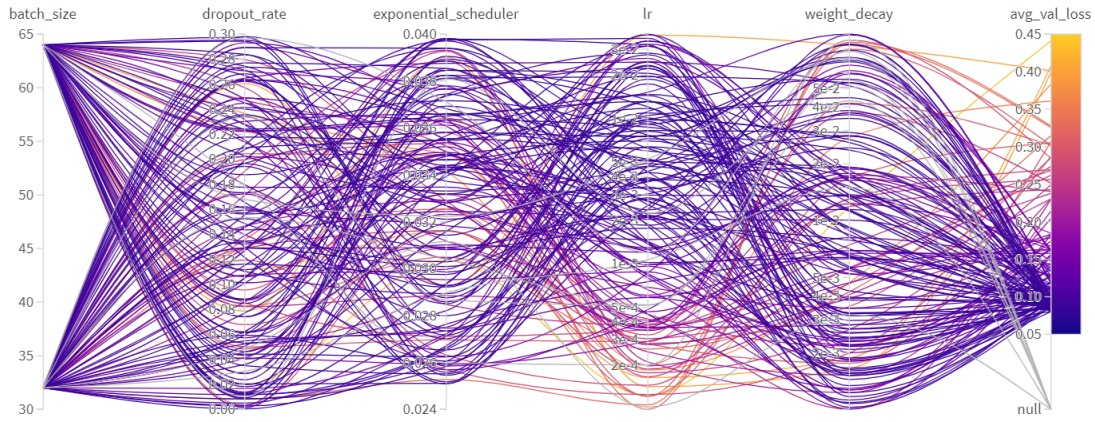


Figure 6.13. All configurations that have been run in the extended hyperparameter optimization. The runs with a avg_val_loss of *null* have been stopped before completion due to GPU-timeouts in Google Colab.

Figure 6.13 shows that the search have been very extensive, as the figure is almost completely purple; There is very little white space in the parameter ranges. However, it is impossible to conclude what the best hyperparameter configuration is from this figure, as there are simply too many curves. Figure 6.14 shows the curve for the best configuration, where the rest are grayed out.

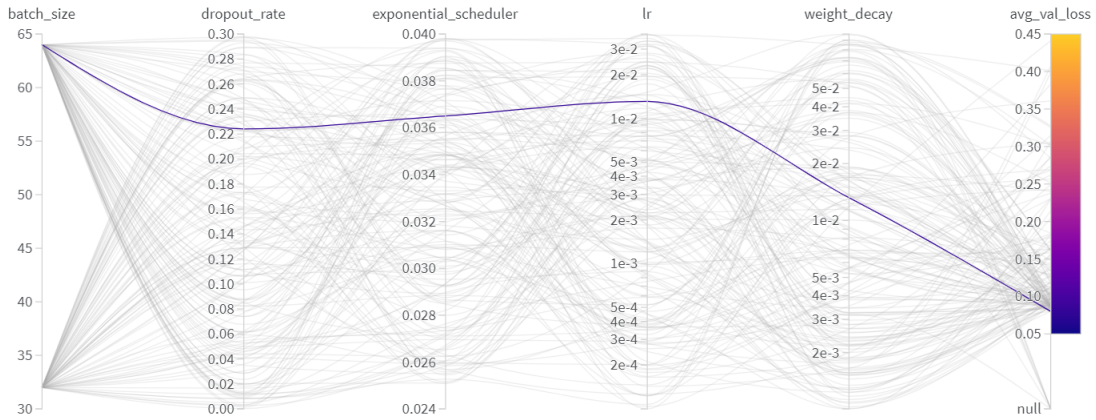


Figure 6.14. The best configuration of the extended hyperparameter optimization. The runs with an avg_val_loss of *null* have been stopped before completion due to GPU-timeouts in Google Colab.

The precise configuration for the best run is shown in Table 6.6.

Hyperparameter	**Value**
Batch size	64
Dropout rate	0.224
Exponential scheduler	0.037
Learning rate	0.013
Weight decay	0.013

Table 6.6. The best hyperparameter configuration of the extended hyperparameter optimization.

6.2.5 CAM

As alluded to in Section 5.4, the developed model has GradCAM implemented with it, which allows for visual interpretation of the results. In this way, radiologists would, in theory, be able to use the discriminative image regions, highlighted as a heatmap, and identify the radiographic features that correlates with the TB diagnosis. Thus, GradCAM enables the developed model to be used to aid radiologists. In Figure 6.15 the output of the GradCam can be seen.

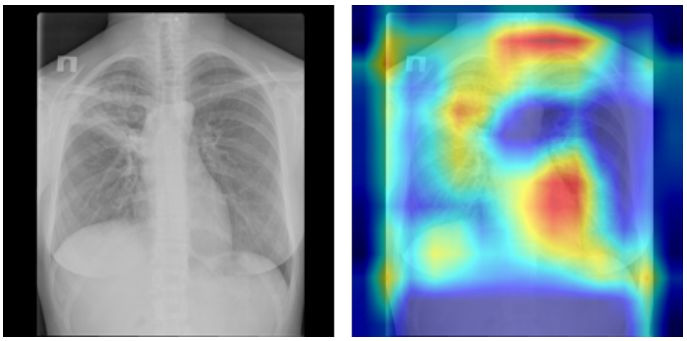


Figure 6.15. GradCAM output from the projects model.

In contrast to the radiographic features provided in Chapter 2, these CXR images are not used for educational purposes. Therefore, as seen in Figure 6.15 an enlarged uncertainty occur when seeking out a fitting diagnostic using GradCAM. As the area around the lymph nodes is highlighted, one might suggest Lymphadentapoty with hilar enlargement. However, there is a high uncertainty related to this guess as non-radiologists. Below in Figure 6.16 a wider array of GradCAMs with associated CXRs are presented.

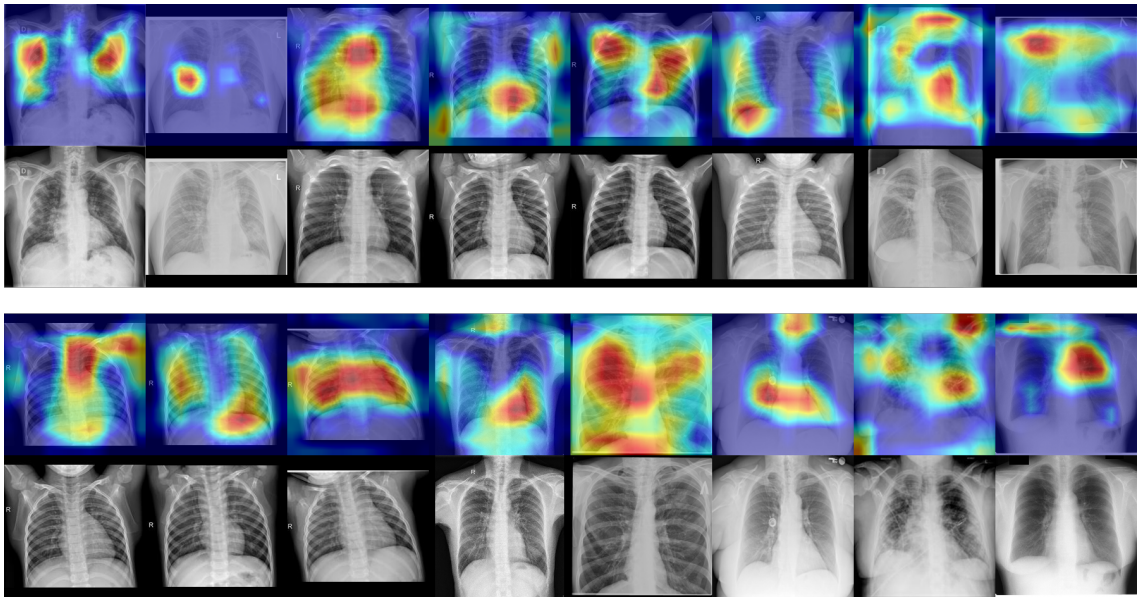


Figure 6.16. Overview of a series of GradCAM heatmaps and corresponding original CXR images.

The only thing that can be somewhat concluded about the implementation, is that the GradCAM retains to areas that are expected. Namely, in the lungs as seen in Figure 6.16. Additionally, the primary heatmap is centered around placements, where radiographic features are contained, such as the lymph nodes and upper or lower lungs. Based on this, radiographic features and diagnostics are assigned as seen in Figure 6.17.

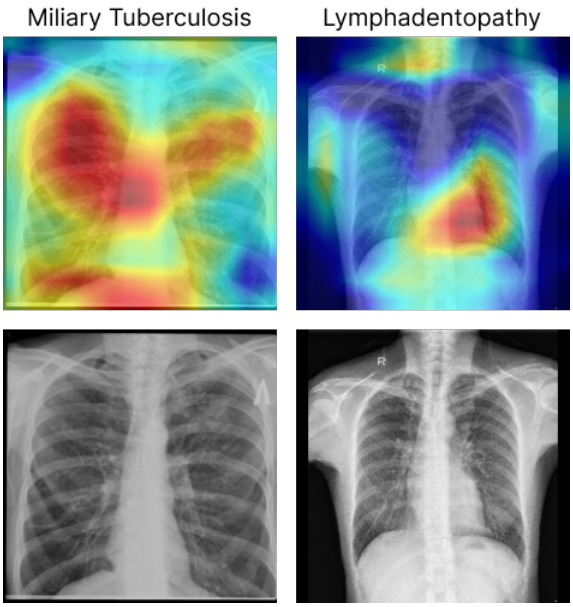


Figure 6.17. GradCAM interpretation of TB test images.

Based on Figure 6.17 it must be acknowledged that, in order to successfully evaluate the performance of the CAM implementations, consultation with radiologists is needed. Additionally, they could provide feedback in using alternative CAM models, that might increase interpretability of the heatmap, such that it could be used as a computer-aided tool.

6.2.6 Baseline Comparison

The purpose of this part is to compare the developed model with a baseline. Most research has only examined the use of a binary TB classifier in simplified settings with only health and TB infected lungs. As such, a baseline model is made to reflect this. In this regard, the baseline model is from the paper: "Reliable Tuberculosis Detection Using Chest X-Ray With Deep Learning, Segmentation and Visualization" by Rahman et al. [2020]. This was chosen as the baseline, because this the paper had the best possible score among all the considered with an accuracy of 98.6 %, and was tested with an adequate dataset. Furthermore, it was published by IEEE, validating their results.

From Rahman et al. [2020] their best performing model with segmentation is DenseNet201. However, this is not replicable as segmentation was implemented unsuccessfully in this project. As such, their best performing model without masking is considered instead. In this regard, CheXNet is the best performing model. Unfortunately, this is also not replicable, as an open source pre-trained model of CheXNet could not be found. As a result, their second best model without segmentation, VGG19, is chosen as the baseline model for this project.

Both models are tested on two test sets: A baseline test set with only healthy and TB infected lungs, and a extended test set with healthy, TB, pneumonia, lung cancer and COVID-19 infected lungs. Thus, through the baseline comparison it can be ascertained whether or not the developed model is more robust and well performing than a baseline

model. A graphic representation of the baseline comparison is shown in Figure 6.18.

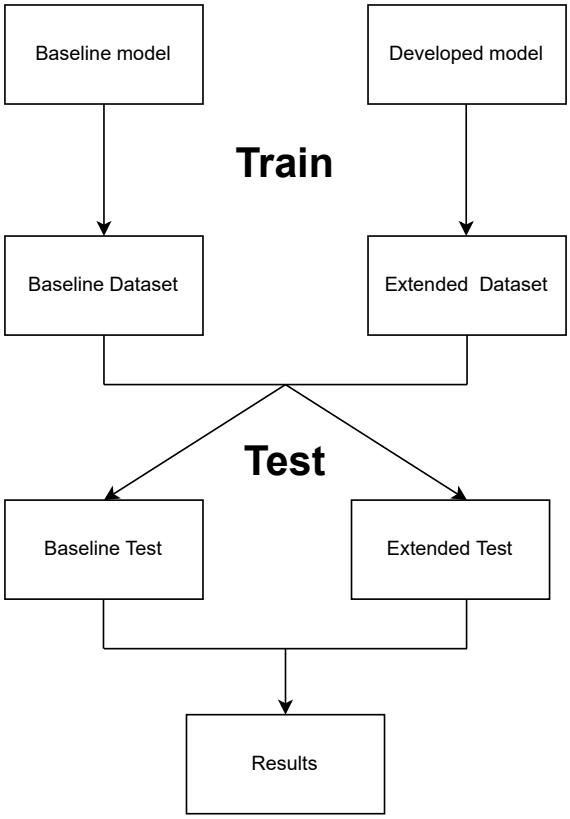


Figure 6.18. Graphic representation of the baseline comparison.

It is hypothesized that the baseline model will score lower on the extended test set, as it is not trained to classify COVID-19, pneumonia or lung cancer. This could potentially result in a higher FPR, as it might misclassify these diseases with similar radiographic features to be TB even though they are not. This would support the claim that training in a simplified setting is not a robust approach applicable for real world use.

Baseline Model Hyperparameters

Using Rahman et al. [2020], the hyperparameter settings of the baseline model is shown in Table 6.7.

Hyperparameter	**Value**
Batch size	32
Learning rate	0.001
Epochs	15
Epochs patience	3
Stopping criteria	5
Optimizer	SGD

Table 6.7. The hyperparameter settings of VGG19 from Rahman et al. [2020].

Datasets and test sets

The dataset presented in Rahman et al. [2020] consists of some images that this project does not have access to, so this needs to be replicated in some way Rahman et al. [2020]. The main trait of their dataset is that it only consists of CXR-images of healthy and TB infected lungs with a ratio of 50/50. Since this project only has access to a total of 1440 TB images, it was decided to combine these with 1440 normal images for a total of 2880 pictures to create a 50/50 ratio.

This split entails that some of the same pictures are used in both the baseline dataset and extended dataset. This resulted in randomly sampling images from each class, and manually doing the training/test split once to ensure that no model would be tested on an image on which it had trained. The relation between the test and datasets are shown in Figure 6.19.

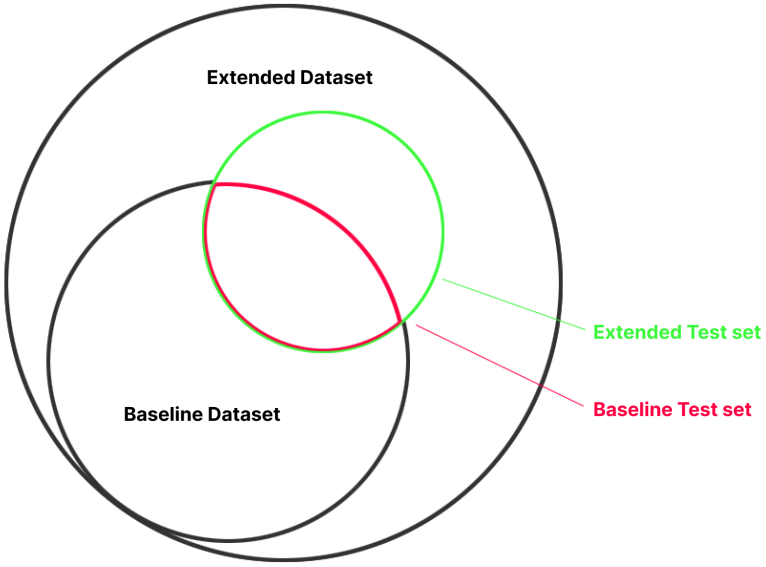


Figure 6.19. Dataset and test set distribution amongst the extended and baseline dataset.

Baseline Model Data Augmentation

The only information on data augmentation presented inRahman et al. [2020] is only that it is used, but without any further details, which makes it impossible to replicate their results. However, since it is stated that it is used, it was decided to apply the same data augmentation used in the developed model.

Baseline comparison results

Based on the presented approach, the training and tests has been conducted, where the confusion matrix results can be seen in Table 6.8 and 6.9.

Baseline Test Set

		Predicted	
		Positive	Negative
Target	Positive	TP: 263	FN: 26
	Negative	FP: 8	TN: 281

Extended Test Set

		Predicted	
		Positive	Negative
Target	Positive	TP: 263	FN: 26
	Negative	FP: 8	TN: 567

Table 6.8. Confusion matrix for the developed model.

Baseline Test Set

		Predicted	
		Positive	Negative
Target	Positive	TP: 266	FN: 23
	Negative	FP: 12	TN: 277

Extended Test Set

		Predicted	
		Positive	Negative
Target	Positive	TP: 266	FN: 23
	Negative	FP: 145	TN: 430

Table 6.9. Confusion matrix for the baseline model.

These values showcase the values used for the calculations of the metrics. The results from the eight metrics are summarized in Table 6.10.

Configuration		**Results**								
Model	Test set	Loss	Acc	TPR	PPV	TNR	FNR	FPR	F1 score	AUC
Baseline	Baseline	0.13	93.95	92.04	95.68	95.85	7.96	4.15	93.83	0.99
Baseline	Extended	0.54	80.56	92.04	64.72	74.78	7.96	25.22	76.00	0.95
Developed	Baseline	0.14	94.12	91.00	97.05	97.23	9.00	2.77	93.93	0.98
Developed	Extended	0.10	96.07	91.00	97.05	98.61	9.00	1.39	93.93	0.99

Table 6.10. In the table, all the results from the four conducted tests are presented by the 8 utilized metrics.

The developed model, on the extended test set, achieves an accuracy of 96.07% and a loss of 0.10. This showcase that the model is generally good at classifying TB images in a simulated realistic setting with similar diseases. At the same time, it shows improvements are needed to approach the 100% accuracy. Examining the metrics which takes FP into account, the developed model achieves a TNR of 98.61% and a FPR of 1.39%. This means that out of all the negative images, the model has a 98.61% probability of correctly classifying it as negative, and only in 1.39% of the cases the model wrongly classifies it as positive. Essentially, showcasing that the model is very precise at classifying the non-TB images.

Examining the metrics, which takes FN into account, they showcase that the model is less capable of classifying TB images. This is evident from the TPR and FNR, with a score of respectively 91.00% and 9.00%. I.e. 9% of the TB positive images are missclassified as being negative. This is also evident from the F1-score of 93.93%, which is negatively influenced by the FN cases. These results are less desirable, as these cases has an immense consequence for a patient in the real world. In this area, further improvements can be made to reduce the number FN and ultimately increase the accuracy of the model.

The developed model, on the baseline test set, achieves an accuracy of 94.12% and a loss of 0.14, which is a drop in accuracy of 1.95 percentage point compared to the developed model tested with the extended dataset. This is expected since the developed model is trained with the extended dataset and accordingly optimized for the extended test set. However, the model still shows that it is very capable of classifying TB in a dataset consisting of only TB and normal images, even though it has been trained on the extended dataset. Examining the other metrics, the same trends are evident again, where improvements can be made in regard to the FN cases. In fact, it is exactly the same scores in the metrics accounting for FN cases (FNR, PPV, TPR and F1-score). This is probably a result of the baseline and extended test sets containing exactly the same TB images, making the positive classes between the two test sets identical. The extended dataset has added more diseases, meaning that only the TB negative class differs between the two test sets.

The baseline model, on the baseline test set, achieves an accuracy of 93.95% and a loss of 0.14. The accuracy is roughly 1.85 percentage point lower than the score from the original paper [Rahman et al., 2020, p. 10]. This is probably due to the previously mentioned conditions, that the test had to be conducted with a different dataset and data augmentation method. However, this is not problematic, as the purpose is to evaluate how the baseline model performs on the extended test set that is more close to a real world application. With these clarifications, the results from the baseline model tested on the baseline and extended test set have now been compared.

The baseline model, on the extended test set, achieves an accuracy of 80.56% and a loss of 0.54. This is a significant drop in accuracy of 13.39 percentage point, and essentially means that 19.44% of all predictions are wrongly classified. This confirms that the utilization of a dataset consisting of only healthy and TB infected lungs, is inadequate for a real world application. A further examination of the metrics accounting for the FPs, gives an even better illustration of the problem. On the baseline test set, the baseline model achieves an FPR of 4.13%, compared to an FPR of 25.22% on the extended test set. This is a significant increase of approximately 510% in FPR. This means that the baseline model would in 25% of all the negative cases, wrongly classify negative cases as being TB. This clearly illustrates an inadequate ability to distinguish between TB and diseases with similar radiographic features. Comparing the results to the developed model tested on the extended test set, it showcases a even greater difference in performance, with a difference in accuracy of 15.51 percentage point. Compared to the FPR it is a difference of 23.8 percentage point. I.e. the baseline model tested on the extended test set produces 23.8% more FP cases (isolated to the sets of negative images), compared to the developed model. Ultimately, these results illustrates that the developed model is superior in distinguishing between TB and similar diseases.

ROC curves

To provide a graphical representation of the performance difference between the four conducted tests, the ROC curves are visualized in Figure 6.20. Examining the results from the AUC in Table 6.10, the developed model tested on the extended test set achieves the highest AUC of 0.99, which is in line with previous results. This essentially means that the model is close to a perfect separation between the two classes in the extended

test set. This is also evident from the corresponding ROC curve, which continuously has a high TPR and an FPR with an increasing threshold. However, it is also evident from the figure that the TPR increases from around 0.8 to 0.985, which illustrates that the model is not perfectly capable of identifying positives cases. Comparing it to the baseline model tested on the extended test set, it achieves an AUC of 0.95 which is 0.04 lower than the developed model. The difference in performance is better visualized from the ROC Curves in Figure 6.20, where the ROC curves between the two tests clearly illustrates that the developed model has a superior class separation capacity.

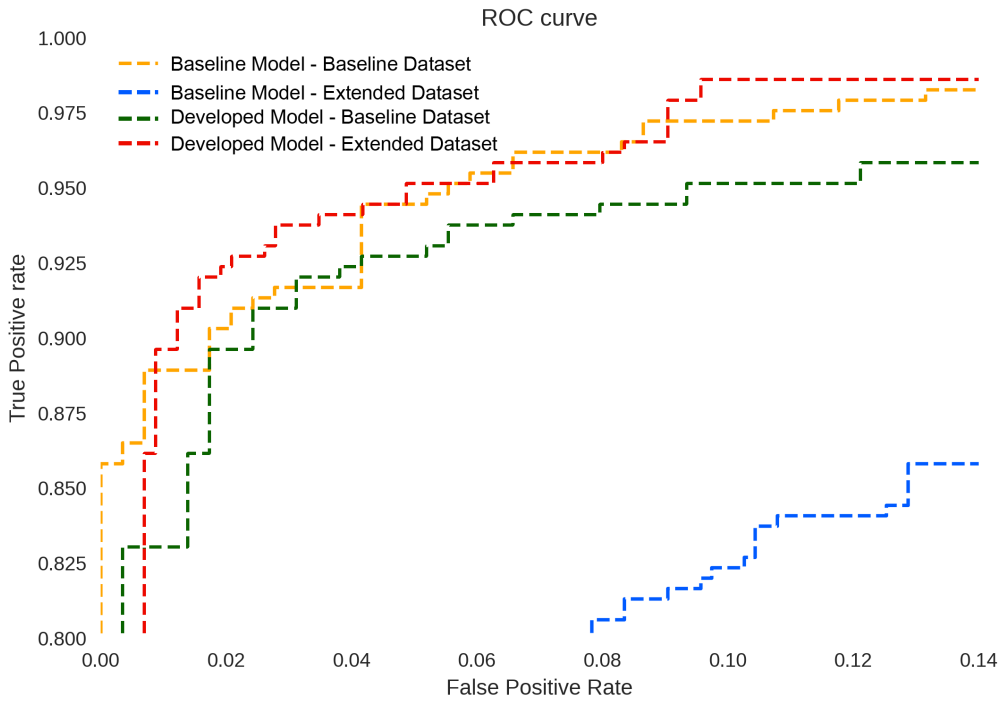


Figure 6.20. ROC curves of the four test conducted in the baseline comparison, with a TPR on the y-axis from range 0.8-1 and FPR on the x-axis from range 0-0.14. The Baseline model tested on the extended dataset (blue curve) is starting at an FPR at 0.08, because no points had an FPR below 0.08

The illustrated ROC curves can also be used to determine the optimal value for the classification threshold. Naturally, the most preferable threshold, is the one that has a corresponding point on the graph as close to the upper left as possible, as this would result in the highest TPR and lowest FPR. Following this logic, the threshold used to generate the blue point in Figure 6.21 might be the most preferable, as it is closest to the upper left corner for the full curve.

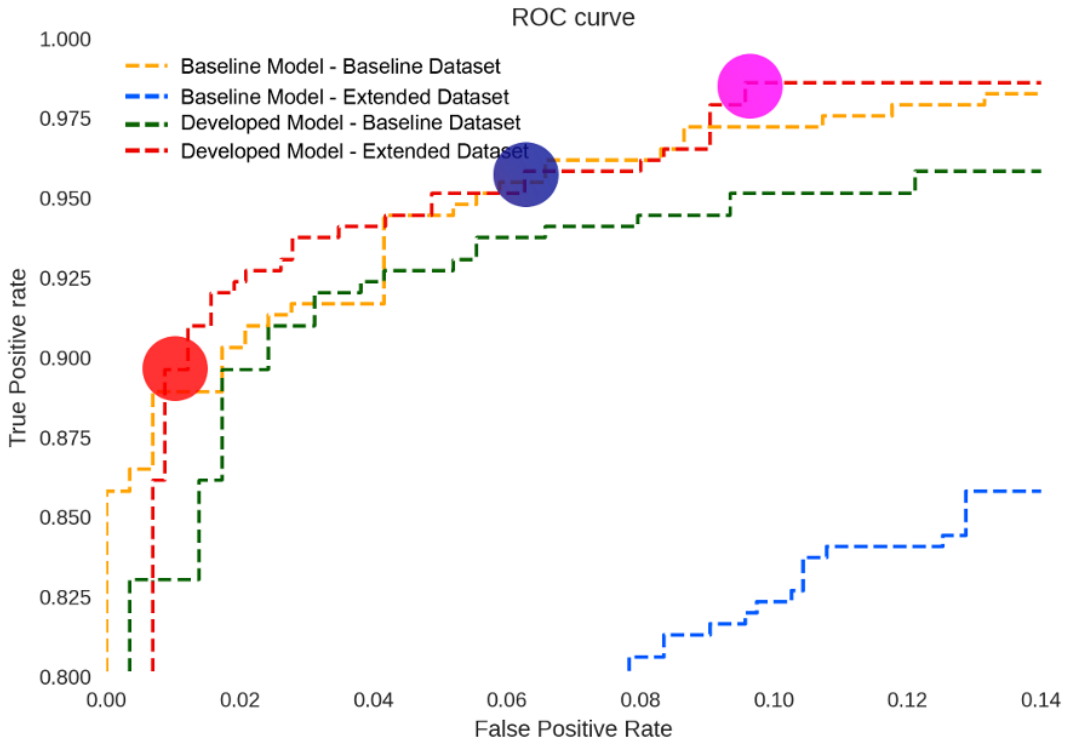


Figure 6.21. ROC curves of the four test conducted in the baseline comparison, with a TPR on the y-axis from range 0.8 to 1 and FPR on the x-axis from range 0 to 0.14. The Baseline model tested on the extended dataset (blue curve) starts at an FPR of 0.08, because no points had an FPR below 0.08.

Note that the gap for the range on the y-axis (1 - 0.8 = 0.2) is higher than the gap on the x-axis (0.14), which is why the blue point is placed a bit more to the right. However, the selection of the optimal threshold of course depends on the use case. As mentioned earlier, the developed model has a higher FNR of 9 %, which is very alarming, as a FN means that people that actually have tuberculosis are miss-diagnosed as not having it. The ultimate consequence of this, is that people could die. For this reason, it might be preferable to select a threshold with a higher TPR, at the cost of a higher FPR. If for example the corresponding threshold for the pink point in Figure 6.21 was selected, then the TPR would approximately be above 0.98, which is very preferable.

However, the potential real-world users of this model has no insight in how confident the model is in its prediction. For this reason, a third class could be introduced: Undecided. Based on the ROC curve, two points could be chosen, for example the pink and the red. Then their corresponding thresholds could be used as a range for determining when the model should classify a CXR as undecided. This would increase the credibility of the model, as the new TPR would be approximately 0.98 (pink point y-value), and the new FPR would approximately be 0.01 (red point x-value).

However, these are just examplified ways to determine the optimal threshold(s) for the model, and in a real-world application, they should naturally be selected in cooperation with radiologists.

Misclassification Distribution Among Diagnoses

The baseline model misclassifies similar diseases as being TB. However, the FPR may vary for each specific similar disease. It is, therefore, interesting to examine which diseases have the highest FP rate. This is shown in Table 6.11.

		FP	**TN**	**FP rate**
Baseline Model	Normal	63	327	0.16
	COVID-19	73	11	0.87
	Pneumonia	2	82	0.02
	Lung Cancer	7	10	0.41
Developed Model	Normal	8	382	0.02
	COVID-19	0	84	0.00
	Pneumonia	0	84	0.00
	Lung Cancer	0	17	0.00

Table 6.11. FP comparison with the extended test set between the baseline model and the developed model.

Table 6.11 shows that the baseline model have a significantly higher FPR than the developed model. The developed model have a FPR of zero for the similar diseases and almost zero (0.02) for the normal lungs, which are very satisfying results. Conversely, the baseline model have significantly high FPR, especially for COVID-19 and lung cancer, as these rates were respectively 0.87 and 0.41. As a result, the baseline model is most likely not great at classifying TB in a real-world application, even though it showed very promising results in Rahman et al. [2020].

These results indicate that promising results in simplified environments, might not result in adequate performance in a real world applications. For this reason, it is most likely preferable to train on datasets that represent the real world application as close as possible.

7 Discussion

This chapter discusses the primary shortcomings, opportunities, interests and ideas in relation to the project.

Improvements to Model and Methodology

As the model utilizes binary classification, it would be logical to assume that it uses binary cross entropy with a Bernoulli distribution and a sigmoid activation function. However, upon implementation, unforeseen performance limitations occurred, which dropped the Accuracy significantly. This was against expectation, as this is the recommended setup. As the performance dropped significantly, it is expected that it was due to an implementation error. In response to this, the models were initialized with category cross entropy and a Multinoulli distribution. Thus, it would be advisable to change this in the future, although the influence on performance can be discussed.

An additional oversight was in regard to the test regarding the weight decay range for the extended hyperparameter optimization (section 6.2.4). Here, weight decay was explored with a 1×, 10×, 100×, 1 000× and 10 000× factor increase. However, the 1× relation was accidentally not included in the extended hyperparameter optimization. On the other hand, in an experimental test using 1×, the model performed very similarly disregarding the change in weight decay increase factors, so it is expected that it would not have made much of a change. This is a methodological error, as the global minimum for average validation loss hypothetically could have been achieved using a weight decay increase factor in the range of [1× - 10×]. However, the performance was approximately the same for 10×, 100× and 1000×, so it is expected that it would not have made much of a change.

An attempt to improve the model, that did not pan out was the use of U-Net. This was implemented to decrease the search space for the classification task, and potentially increase performance. In revisits of this implementation, it might be advisable to use a training set with much higher variance in images. This would possibly improve the transferability and generalizability to the extended dataset. Furthermore, the classification network could be changed in the U-Net, potentially with VGG19, as it shows promise in regard to classification in CXR images.

Widening the Search Space

When working with neural networks, it is very easy to succumb to search space paralysis, due to a vast amount of parts to tweak, such as: model variations, data augmentation, hyperparameter optimization and so on. Thus, it is important to carefully reduce the

search space. As such, this section presents some search spaces, that would be beneficial to extend in future iterations of development.

One of the most impactful factors is the model selection, as it lays the primary foundation of which all other development is done. Although there were little difference between model performances, it would be beneficial to look at a wider array of models, especially in a scenario with more computational power, such that it could be analyzed if model size and performance scale accordingly. More specifically, in the likes of EfficientNet B7, as well as the impact of larger batch sizes, which have had to be limited throughout the project. Another possibility, with more computational resources, would be to extend the k-fold cross validation from 3 to 10 folds, thus giving a better estimate of the model's performance.

With additional computational resources, the hyperparameter optimization could also be extended with the addition of varying model architecture hyperparameters, such as kernel size and stride, to potentially unlock more performance improvements. In reality, this could be extended to a more rigorous look through the model architecture, changing the layer composition and occurrences to seek optimization of the model and potentially the creation of a new one.

The last suggested change in regard to search space expansion is a revisit of the CAMs. In this regard, a collaboration with radiologists would improve this process. This would in effect enable estimations of current performance and provide valuable feedback. Additionally, also in relation to implementation of additional CAM variations. and their ability to focus on the correct radiographic features.

Investigations to Conduct

The following presents some other interesting aspects worth of an inquiry, which has not been done in this project.

First, it would be interesting to analyze what effect the input size of images has on the model and how it would affect performance. Second, experiments could be conducted to analyze what impact a larger dataset would have, and when diminishing returns would occur in relation to the developed model.

Furthermore, the dataset properties' effect on performance, could be analyzed. This could either be by changing the split of TB, healthy and other diseases in the dataset. The current being 33/33/33 and 33/66 for only TB-infected and healthy lungs. One possible optimization could be to create a more balanced dataset with a 50/50 split between TB-infected and healthy lungs, as this could make the model estimations more precise. The developed model is optimized using validation loss. However, the loss is estimated using 66% negative cases and only 33% positive TB cases. This could lead to unrepresentative results, as training is only based on the 33% TB cases. Thus, it would be interesting, to see how the balance of TB and NOT TB, would affect performance. However, this would require more TB CXR-images. Similar to the distribution of classes and images in the dataset, variations in the training, validation, and test-split could be explored.

The model performance for each specific dataset could also be examined. Hypothetically, if for example the model's accuracy is 99% for the Shenzhen dataset, then the manufacturer of the applied radiographic unit could be contacted, and the model could be distributed to the users of this specific radiographic unit. These users would therefore be 99 % certain in the prediction from the model, which is very significant. Furthermore, if the threshold for classification is also optimized, then the model could have a very beneficial impact in clinical practice.

Another interesting topic is the somewhat surprising hyperparameter tuning results in regard to data augmentation operations, which resulted in the optimal data augmentation being, not to apply it. This logically seems odd, as so many papers lay a high emphasis on the specific subject. Based on this a visual inspection could be constructed, such that visual feedback is supplied, meanwhile undergoing an in-depth data augmentation of ranges of rotation, translation and parameters of all the utilized image augmentation operations. It could also be beneficial to see if online or offline data augmentation enhances run time. For instance, resizing online was very time-consuming, which was alleviated, once it was done offline.

One topic that is also worth investigating is pre-training. Currently, all models have been pretrained on ImageNet. However, it has not been investigated whether or not a model trained tabula rasa would perform better.

Finally, the three hyperparameter optimization search methods should be explored further, as some concerns arise for each approach, grid-search being too time-consuming and needing to rely on discrete value, Bayesian search not exploring enough initially and the random search, being too random. Thus, a concoction of the former could be advised. For instance, a possible approach is to first apply a parallelized random search with broad ranges, which then could identify more optimal hyperparameters values. Based on this, a new improved range could be found. The new selected ranges are then applied as an input to Bayesian search, which in an informed manner finds a new set of optimal hyperparameter values. Having conducted these two steps, the hyperparameter values can be narrowed down to discrete values and given as an input to the grid search method. The grid search then as a final step identify the optimal values within these. This could potentially lead to hyperparameter values which are close to be global optimal, this of course needs to be tested in potential future work.

Future Work

In the future, the model could be tried with other diagnoses, to see if the performance successfully applies to the likes of cancer and pneumonia, which are detectable with CXR images. In this regard, ensemble learning could be utilized. This would enable the model to undergo in a larger model, that is selectively used in accordance to a set of conditions, which theoretically should be able to generalize and specialize at the same time across many diagnoses.

Conclusion 8

This project has investigated the use of machine learning for tuberculosis detection in Chest X-ray images. In particular, the aim was to develop a well performing and robust ML solution that is correctly able to detect TB in a more nuanced and realistic setting with the presence of pneumonia, COVID-19 and lung cancer. The developed model uses a VGG Convolutional Neural Network architecture, that has been trained with an extended dataset more closely related to a real world setting, and uses Class Activation Maps (CAM) to visualize the model's findings.

Overall, the developed model outperformed the baseline in terms of classification accuracy, with a value of 94.12% and 96.07% for the baseline test set and the extended test set, respectively. However, the developed model approximately had a 1 % point higher *False Negative Rate (FNR)* with a value of 9 %. In a real-world setting, this value is not satisfactory, as 9 out of 100 patients with TB would be misclassified as not having TB, which ultimately could result in misclassified people dying. However, it is worth noting, that the developed ML-solution greatly outperforms real-world radiologists, in terms of detecting TB in CXR images. The misclassifcation problem can be accommodated by decreasing the classification threshold, which would result in a lower FNR, with the trade-off being a higher *False Positive Rate (FPR)*. This trade-off might be preferable to some degree, as the consequence of a *False Negative (FN)* is more severe than a *False Positive (FP)*. This is due to the facts that an FN might result in a person's death, and an FP might be corrected in the later steps of finalizing the TB diagnosis.

The developed model shows a surprisingly impressive performance of 100%, in regard to distinguishing TB, from similar diseases, namely pneumonia, COVID-19 and lung cancer. To further validate these results, a larger dataset, should be employed. In concession, the models' performance could be improved, by utilizing semantic segmentation of the lungs, and an even larger dataset. Furthermore, based on the reported results, the developed solution could be implemented – with some small adjustments – in a clinical setting. In this case, CAM, with some fine-tuning by radiologists, could be used as a computer aided diagnostics tool.

In conclusion, the results strongly indicates that the utilization of datasets with a higher reflection of the context wherein they are used, comes with considerable performance improvements.

Bibliography

AAMC, **2020**. AAMC. *The Complexities of Physician Supply and Demand: Projections From 2018 to 2033*, 2020. URL "https://www.aamc.org/system/files/2020-06/stratcomm-aamc-physician-workforce-projections-june-2020.pdf". Date: 10/03/2022.

Abbas et al., **2020**. Asmaa Abbas, Mohammed M. Abdelsamea and Mohamed Medhat Gaber. *DeTrac: Transfer Learning of Class Decomposed Medical Images in Convolutional Neural Networks*. IEEE Access, 8, 74901–74913, 2020. doi: 10.1109/ACCESS.2020.2989273.

Afum et al., **12 2022**. Theophilus Afum, Prince Asare, Adwoa Asante-Poku, Isaac Darko-Otchere, Portia Abena Morgan, Edmund Bedeley, Diana Asema Asandem, Abdul Basit Musah, Ishaque Mintah Siam, Phillip Tetteh, Yaw Adusi-Poku, Rita Frimpong-Manso, Joseph Humphrey Kofi Bonney, William Ampofo and Dorothy Yeboah-Manu. *Diagnosis of tuberculosis among COVID-19 suspected cases in Ghana*. PLOS ONE, 16(12), 1–8, 2022. doi: 10.1371/journal.pone.0261849. URL https://doi.org/10.1371/journal.pone.0261849.

Mostofa Ahsan, Rahul Gomes and Anne Denton, 2019. Mostofa Ahsan, Rahul Gomes and Anne Denton. Application of a Convolutional Neural Network using transfer learning for tuberculosis detection. In *2019 IEEE International Conference on Electro Information Technology (EIT)*, pages 427–433, 2019. doi: 10.1109/EIT.2019.8833768.

Basodi et al., **2020**. Sunitha Basodi, Chunyan Ji, Haiping Zhang and Yi Pan. *Gradient amplification: An efficient way to train deep neural networks*. Big Data Mining and Analytics, 3(3), 196–207, 2020. doi: 10.26599/BDMA.2020.9020004.

Biases, **2022**. Weights & Biases. *The developer-first MLOps platform*, 2022. URL https://wandb.ai/site.

Bishop, **2006**. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN 0-387-31073-8.

Bonvang et al., **2020**. Gustav Bonvang, Nicholas Hansen, Sune Krøyer, , Jonathan Sørensen and Mathias Thorsager. *Real Time Computer Vision Control System for Autonomous Vehicles*, 2020. Semester project, Aalborg University.

Bonvang et al., **2021**. Gustav Bonvang, Frederik Fagerlund, Sune Krøyer, Kim Nguyen and Mathias Thorsager. *Optimising Battery Charging Patterns in Satellites Using Recurrent Neural Networks*, 2021. Bachelor Thesis, Aalborg University.

Brochu et al., **2010**. Eric Brochu, Vlad M. Cora and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User*

Modeling and Hierarchical Reinforcement Learning, 2010. URL https://arxiv.org/abs/1012.2599.

BrownLee, **2019**. Jason BrownLee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. 2019. URL https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

Burrill et al., **2007**. Joshua Burrill, Christopher J. Williams, Gillian Bain, Gabriel Conder, Andrew L. Hine and Rakesh R. Misra. *Tuberculosis: A Radiologic Review*. RadioGraphics, 27(5), 1255–1273, 2007. doi: 10.1148/rg.275065176. URL https://doi.org/10.1148/rg.275065176. PMID: 17848689.

Sema Candemir, Stefan Jaeger, K. Palaniappan, Sameer Antani and George Thoma, *01 2012*. Sema Candemir, Stefan Jaeger, K. Palaniappan, Sameer Antani and George Thoma. Graph cut based automatic lung boundary detection in chest radiographs. 01 2012.

Chowdhury et al., **2020**. Muhammad E. H. Chowdhury, Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandakar Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al Emadi, Mamun Bin Ibne Reaz and Mohammad Tariqul Islam. *Can AI Help in Screening Viral and COVID-19 Pneumonia?* IEEE Access, 8, 132665–132676, 2020. doi: 10.1109/ACCESS.2020.3010287.

Christensen et al., **2021**. Nicolai Harbo Christensen, Simon Eliasen, Jakob Rønnest Sørensen, Dennis Tran, Kim Nguyen and Thomas Vinther. *Multi-Agent Reinforcement Learning in a Railway Traffic Management System*, 2021. 7th Semester Project, Aalborg University.

Diakogiannis et al., **2019**. Foivos I. Diakogiannis, François Waldner, Peter Caccetta and Chen Wu. *ResUNet-a: a deep learning framework for semantic segmentation of remotely sensed data*. CoRR, abs/1904.00592, 2019. URL http://arxiv.org/abs/1904.00592.

Elgendi et al., **03 2021**. Mohamed Elgendi, Muhammed Umer Nasir, Qunfeng Tang, David Smith, John-Paul Grenier, Catherine Batte, Bradley Spieler, William Donald Leslie, Carlo Menon, Richard Ribbon Fletcher, Newton Howard, Rabab Ward, William Parker and Savvas Nicolaou. *The Effectiveness of Image Augmentation in Deep Learning Networks for Detecting COVID-19: A Geometric Transformation Perspective*. 2021. URL https://www.frontiersin.org/articles/10.3389/fmed.2021.629134/full.

Lucas Gabriel Evalgelista and Elloá B. Guedes, *2018*. Lucas Gabriel Evalgelista and Elloá B. Guedes. Computer-Aided Tuberculosis Detection from Chest X-Ray Images with Convolutional Neural Networks. In *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pages 518–527, Porto Alegre, RS, Brasil, 2018. SBC. doi: 10.5753/eniac.2018.4444. URL https://sol.sbc.org.br/index.php/eniac/article/view/4444.

Fan et al., **2006**. Jerome Fan, Suneel Upadhye and Andrew Worster. *Understanding receiver operating characteristic (ROC) curves*. Canadian Journal of Emergency Medicine, 8(1), 19–20, 2006. doi: 10.1017/S1481803500013336.

Goodfellow et al., 2016. Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Guo et al., 2020. Ruihua Guo, Kalpdrum Passi and Chakresh Kumar Jain. *Tuberculosis Diagnostics and Localization in Chest X-Rays via Deep Learning Models*. Frontiers in Artificial Intelligence, 3, 2020. ISSN 2624-8212. doi: 10.3389/frai.2020.583427. URL https://www.frontiersin.org/article/10.3389/frai.2020.583427.

H. et al., 2018. Naderi H., Sheybani F., Erfani SS., Amiri B and Nooghabi MJ. *The mask of acute bacterial pneumonia may disguise the face of tuberculosis*. Electron Physician, 2018. doi: doi:10.19082/3943. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5407226/?report=classic.

He et al., 12 2015. Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. page 12, 2015. URL https://arxiv.org/abs/1512.03385.

Heidari et al., 2020. Morteza Heidari, Seyedehnafiseh Mirniaharikandehei, Abolfazl Zargari Khuzani, Gopichandh Danala, Yuchen Qiu and Bin Zheng. *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, 144, 104284, 2020. ISSN 1386-5056. doi: https://doi.org/10.1016/j.ijmedinf.2020.104284. URL https://www.sciencedirect.com/science/article/pii/S138650562030959X.

Hooda et al., 2017a. Rahul Hooda, Sanjeev Sofat, Simranpreet Kaur, Ajay Mittal and Fabrice Mériaudeau. *Deep-learning: A potential method for tuberculosis detection using chest radiography*. 2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), pages 497–502, 2017a.

Rahul Hooda, Sanjeev Sofat, Simranpreet Kaur, Ajay Mittal and Fabrice Meriaudeau, 2017b. Rahul Hooda, Sanjeev Sofat, Simranpreet Kaur, Ajay Mittal and Fabrice Meriaudeau. Deep-learning: A potential method for tuberculosis detection using chest radiography. In *2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 497–502, 2017b. doi: 10.1109/ICSIPA.2017.8120663.

Hossin and M.N, 03 2015. Mohammad Hossin and Sulaiman M.N. *A Review on Evaluation Metrics for Data Classification Evaluations*. International Journal of Data Mining & Knowledge Management Process, 5, 01–11, 2015. doi: 10.5121/ijdkp.2015.5201.

Hu et al., 2020. Jie Hu, Li Shen, Samuel Albanie, Gang Sun and Enhua Wu. *Squeeze-and-Excitation Networks*. IEEE Trans. Pattern Anal. Mach. Intell., 42(8), 2011–2023, 2020. doi: 10.1109/TPAMI.2019.2913372. URL https://doi.org/10.1109/TPAMI.2019.2913372.

Gao Huang, Zhuang Liu, Laurens van der Maaten and Kilian Q. Weinberger, 2017. Gao Huang, Zhuang Liu, Laurens van der Maaten and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages

2261–2269. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.243. URL https://doi.org/10.1109/CVPR.2017.243.

Iglovikov and Shvets, **2018**. Vladimir Iglovikov and Alexey Shvets. *TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation.* CoRR, abs/1801.05746, 2018. URL http://arxiv.org/abs/1801.05746.

Ioffe and Szegedy, **2015**. Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. URL https://arxiv.org/abs/1502.03167.

Jaeger et al., **08 2012**. Stefan Jaeger, Alexandros Karargyris, Sameer Antani and George Thoma. *Detecting tuberculosis in radiographs using combined lung masks.* Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference, 2012, 4978–81, 2012. doi: 10.1109/EMBC.2012.6347110.

Jaeger et al., **2014**. Stefan Jaeger, Sema Candemir, S. Antani, Yì-Xiáng J Wáng, Pu-Xuan Lu and George R. Thoma. *Two public chest X-ray datasets for computer-aided screening of pulmonary diseases.* Quantitative imaging in medicine and surgery, 4 6, 475–7, 2014.

Debesh Jha, Pia H. Smedsrud, Michael A. Riegler, Dag Johansen, Thomas De Lange, Pål Halvorsen and Håvard D. Johansen, 2019. Debesh Jha, Pia H. Smedsrud, Michael A. Riegler, Dag Johansen, Thomas De Lange, Pål Halvorsen and Håvard D. Johansen. ResUNet++: An Advanced Architecture for Medical Image Segmentation. In *2019 IEEE International Symposium on Multimedia (ISM)*, pages 225–2255, 2019. doi: 10.1109/ISM46123.2019.00049.

Jonathan G. Richens and Johri, **07 2020**. Ciarán M. Lee Jonathan G. Richens and Saurabh Johri. *Improving the accuracy of medical diagnosis with causal machine learning.* page 9, 2020. URL https://www.nature.com/articles/s41467-020-17419-7.pdf. Date: 10/03/2022.

JSRT, **2022**. JSRT. *Tuberculosis (TB) data science for public health impact.* 2022. URL http://db.jsrt.or.jp/eng.php.

Keikha and Esfahani, **2018**. Masoud. Keikha and Bahram. Esfahani. *The Relationship between Tuberculosis and Lung Cancer.* Advanced Biomedical Research, 7(1), 58, 2018. doi: 10.4103/abr.abr_182_17. URL https://www.advbiores.net/article.asp?issn=2277-9175;year=2018;volume=7;issue=1;spage=58;epage=58;aulast=Keikha;t=6.

Kermany et al., **2018**. Daniel S. Kermany, Michael Goldbaum, Wenjia Cai, Carolina C.S. Valentim, Huiying Liang, Sally L. Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, Justin Dong, Made K. Prasadha, Jacqueline Pei, Magdalene Y.L. Ting, Jie Zhu, Christina Li, Sierra Hewett, Jason Dong, Ian Ziyar, Alexander Shi, Runze Zhang, Lianghong Zheng, Rui Hou, William Shi, Xin Fu, Yaou Duan, Viet A.N. Huu, Cindy Wen, Edward D. Zhang, Charlotte L. Zhang, Oulan Li, Xiaobo Wang, Michael A. Singer, Xiaodong Sun, Jie Xu, Ali Tafreshi, M. Anthony Lewis, Huimin Xia

and Kang Zhang. *Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning.* 2018. URL https://www.cell.com/cell/fulltext/S0092-8674(18)30154-5. Date: 04/05/2022.

Laushkina and Filimonov, **2012**. Zhanna Laushkina and Pavel Filimonov. *Misdiagnosed pulmonary TB: Influencing factors and diagnostic chances in TB hospital.* European Respiratory Journal, 40(Suppl 56), 2012. URL https://erj.ersjournals.com/content/40/Suppl_56/P2720.

Lin et al., **2013**. Min Lin, Qiang Chen and Shuicheng Yan. *Network In Network*, 2013. URL https://arxiv.org/abs/1312.4400.

Lloyd-Jones, **2022**. Graham Lloyd-Jones. *radiology masterclass.* 2022. URL https://www.radiologymasterclass.co.uk/gallery/chest/pulmonary-disease/normal_x-ray_comparison.

Lopes and Valiati, **2017a**. U.K. Lopes and J.F. Valiati. *Pre-trained convolutional neural networks as feature extractors for tuberculosis detection.* Computers in Biology and Medicine, 89, 135–143, 2017a. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2017.08.001. URL https://www.sciencedirect.com/science/article/pii/S0010482517302548.

Lopes and Valiati, **2017b**. U.K. Lopes and J.F. Valiati. *Pre-trained convolutional neural networks as feature extractors for tuberculosis detection.* Computers in Biology and Medicine, 89, 135–143, 2017b. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2017.08.001. URL https://www.sciencedirect.com/science/article/pii/S0010482517302548.

Melendez et al., **04 2016**. Jaime Melendez, Clara Sánchez, Rick Philipsen, Pragnya Maduskar, Rodney Dawson, Grant Theron, Keertan Dheda and Bram Ginneken. *An automated tuberculosis screening strategy combining X-ray-based computer-aided detection and clinical information.* Scientific Reports, 6, 25265, 2016. doi: 10.1038/srep25265.

Meraj et al., **10 2019**. S. S. Meraj, R. Yaakob, A. Azman, S. N. Rum, A. Shahrel, A. Nazri and N. F. Zakaria. *Detection of pulmonary tuberculosis manifestation in chest X-rays using different convolutional neural network (CNN) models.* page 6, 2019. URL https://www.ijeat.org/wp-content/uploads/papers/v9i1/A2632109119.pdf.

Murphy, **2022**. Kevin P. Murphy. *Probabilistic Machine Learning: An introduction.* MIT Press, 2022. URL probml.ai.

Nash et al., **Jan 2020**. Madlen Nash, Rajagopal Kadavigere, Jasbon Andrade, Cynthia Amrutha Sukumar, Kiran Chawla, Vishnu Prasad Shenoy, Tripti Pande, Sophie Huddart, Madhukar Pai and Kavitha Saravu. *Deep learning, computer-aided radiography reading for tuberculosis: a diagnostic accuracy study from a tertiary hospital in India.* Scientific Reports, 10(1), 210, 2020. ISSN 2045-2322. doi: 10.1038/s41598-019-56589-3. URL https://doi.org/10.1038/s41598-019-56589-3.

NIH, **2022**. NIH. *Tuberculosis (TB) data science for public health impact.* 2022. URL https://tbportals.niaid.nih.gov/.

paperswithcode, **2022**. paperswithcode. *Image Classification on ImageNet*, 2022. URL https://paperswithcode.com/sota/image-classification-on-imagenet. Date: 05/11/2022.

Pasa et al., **April 2019**. F Pasa, V Golkov, F Pfeiffer, D Cremers and D Pfeiffer. *Efficient Deep Network Architectures for Fast Chest X-Ray Tuberculosis Screening and Visualization.* Scientific Reports, 9(1), 6268, 2019.

Pishro-Nik, **2014**. Hossein Pishro-Nik. *Introduction to probability, statistics, and random processes.* Kappa Research LLC, 2014. https://www.probabilitycourse.com/.

Rahman et al., **2020**. Tawsifur Rahman, Amith Khandakar, Muhammad Abdul Kadir, Khandaker Rejaul Islam, Khandakar F. Islam, Rashid Mazhar, Tahir Hamid, Mohammad Tariqul Islam, Saad Kashem, Zaid Bin Mahbub, Mohamed Arselene Ayari and Muhammad E. H. Chowdhury. *Reliable Tuberculosis Detection Using Chest X-Ray With Deep Learning, Segmentation and Visualization.* IEEE Access, 8, 191586–191601, 2020. doi: 10.1109/ACCESS.2020.3031384.

Rahman et al., **2021**. Tawsifur Rahman, Amith Khandakar, Yazan Qiblawey, Anas Tahir, Serkan Kiranyaz, Saad Bin Abul Kashem, Mohammad Tariqul Islam, Somaya Al Maadeed, Susu M. Zughaier, Muhammad Salman Khan and Muhammad E.H. Chowdhury. *Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images.* Computers in Biology and Medicine, 132, 104319, 2021. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2021.104319. URL https://www.sciencedirect.com/science/article/pii/S001048252100113X.

Rohilla et al., **08 2017a**. A. Rohilla, R. Hooda and A. Mittal. *Tb detection in chest radiograph using deep learning architecture.* page 12, 2017a. URL https://www.xilirprojects.com/wp-content/uploads/2019/11/5ea45d81d45ddafbc2b915671e44ebabddd5.pdf.

Anuj Rohilla, Rahul Hooda and Ajay Mittal, *2017b*. Anuj Rohilla, Rahul Hooda and Ajay Mittal. TB Detection in Chest Radiograph Using Deep Learning Architecture. 2017b.

Ronneberger et al., **2015**. Olaf Ronneberger, Philipp Fischer and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* CoRR, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Russell and Norvig, **2010a**. Stuart Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach.* Pearson, 2010a. ISBN 978-9332543515.

Russell and Norvig, **2010b**. Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010b. ISBN 978-0-13-207148-2. URL http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen, *2018*. Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*

2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00474. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

Sathitratanacheewin et al., 2020. Seelwan Sathitratanacheewin, Panasun Sunanta and Krit Pongpirul. *Deep learning for automated classification of tuberculosis-related chest X-Ray: dataset distribution shift limits diagnostic performance generalizability.* Heliyon, 6(8), e04614, 2020. ISSN 2405-8440. doi: https://doi.org/10.1016/j.heliyon.2020.e04614. URL https://www.sciencedirect.com/science/article/pii/S2405844020314584.

Selvaraju et al., 2016. Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh and Dhruv Batra. *Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization.* CoRR, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

Showkatian et al., 02 2022. Eman Showkatian, Mohammad Salehi, Hamed Ghaffari, Reza Reiazi and Nahid Sadighi. *Deep learning-based automatic detection of tuberculosis disease in chest X-ray images.* page 7, 2022. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8906182/pdf/PJR-87-46383.pdf.

Karen Simonyan and Andrew Zisserman, 2015. Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Yoshua Bengio and Yann LeCun, editor, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

Singh et al., 2014. Hardeep Singh, Ashley N D Meyer and Eric J Thomas. *The frequency of diagnostic errors in outpatient care: estimations from three large observational studies involving US adult populations.* BMJ Quality & Safety, 23(9), 727–731, 2014. ISSN 2044-5415. doi: 10.1136/bmjqs-2013-002627. URL https://qualitysafety.bmj.com/content/23/9/727.

Niharika Singh and Satish Hamde, 2019. Niharika Singh and Satish Hamde. Tuberculosis Detection Using Shape and Texture Features of Chest X-Rays. In H. S. Saini, R. K. Singh, Girish Kumar, G.M. Rather and K. Santhi, editors, *Innovations in Electronics and Communication Engineering*, pages 43–50, Singapore, 2019. Springer Singapore. ISBN 978-981-13-3765-9.

Srivastava et al., January 2014. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* J. Mach. Learn. Res., 15(1), 1929–1958, 2014. ISSN 1532-4435.

Szegedy et al., 2015. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision.* CoRR, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

Mingxing Tan and Quoc V. Le, 2019. Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Kamalika Chaudhuri

and Ruslan Salakhutdinov, editor, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

The World Bank, **2022**. The World Bank. *"Physicians (per 1,000 people): World Health Organization's Global Health Workforce Statistics, OECD, supplemented by country data."*, 2022. URL "https://data.worldbank.org/indicator/SH.MED.PHYS.ZS?most_recent_year_desc=true". Date: 10/03/2022.

Ul Abideen et al., **2020**. Zain Ul Abideen, Mubeen Ghafoor, Kamran Munir, Madeeha Saqib, Ata Ullah, Tehseen Zia, Syed Ali Tariq, Ghufran Ahmed and Asma Zahra. *Uncertainty Assisted Robust Tuberculosis Identification With Bayesian Convolutional Neural Networks*. IEEE Access, 8, 22812–22825, 2020. doi: 10.1109/ACCESS.2020.2970023.

UN, **2020**. UN. *Progress towards the achievement of global tuberculosis targets and implementation of the political declaration of the high-level meeting of the General Assembly on the fight against tuberculosis*, 2020. URL "https://documents-dds-ny.un.org/doc/UNDOC/GEN/N20/240/09/PDF/N2024009.pdf". Date: 10/03/2022.

USAID, **2014**. USAID. *"Workforce Connections: Kenya Youth Assesment"*, 2014. URL "https://d3n8a8pro7vhmx.cloudfront.net/fhi360/pages/348/attachments/original/1427227963/Kenya_Youth_Assessment_Final_Report.pdf?1427227963". Date: 10/03/2022.

WHO, **2008**. WHO. *Implementing the WHO Stop TB Strategy: A Handbook for National Tuberculosis Control Programmes*. 2008. URL https://www.ncbi.nlm.nih.gov/books/NBK310759/#!po=12.5000.

WHO, **2021**. WHO. *Global tuberculosis report 2021*. World Health Organization, 2021.

WHO, **2016**. WHO. *CHEST RADIOGRAPHY IN TUBERCULOSIS DETECTION - Summary of current WHO recommendations and guidance on programmatic approaches*. 2016. URL https://apps.who.int/iris/bitstream/handle/10665/252424/9789241511506-eng.pdf.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu and Kaiming He, *2017*. Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5987–5995. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.634. URL https://doi.org/10.1109/CVPR.2017.634.

Ojasvi Yadav, Kalpdrum Passi and Chakresh Kumar Jain, *2018*. Ojasvi Yadav, Kalpdrum Passi and Chakresh Kumar Jain. Using Deep Learning to Classify X-ray Images of Potential Tuberculosis Patients. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2368–2375, 2018. doi: 10.1109/BIBM.2018.8621525.

Yu and Zhu, **2020**. Tong Yu and Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*, 2020. URL `https://arxiv.org/abs/2003.05689`.

Zhou et al., **2015**. Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva and Antonio Torralba. *Learning Deep Features for Discriminative Localization*, 2015. URL `https://arxiv.org/abs/1512.04150`.

Datasets A

A.1 Dataset Normal Image Distribution

TB Datasets	
Dataset	Normal images
Shenzhen	326
Montgomery	80
Pneumonia	472
COVID	472
Cancer	93
Total	1443

Table A.1. Distribution of the 1443 normal images across multiple datasets, to limit overfitting on a singular dataset

A.2 Sample of TB Papers With Binary Datasets.

#	**Author**	**Dataset(s) used**
1	Hooda et al. [2017a]	Montgomery and Shenzhen
2	Lopes and Valiati [2017a]	Montgomery and Shenzhen
3	Showkatian et al. [2022]	Montgomery and Shenzhen
4	Evalgelista and Guedes [2018]	Montgomery, Shenzhen and JSRT
5	Singh and Hamde [2019]	Montgomery and Shenzhen
6	Yadav et al. [2018]	Shenzhen
7	Pasa et al. [2019]	Montgomery and Shenzhen
8	Ahsan et al. [2019]	Montgomery
9	Rohilla et al. [2017a]	Montgomery and Shenzhen
10	Meraj et al. [2019]	Montgomery and Shenzhen

Table A.2

Confusion Matrixes for baseline comparison

B

Baseline dataset

		Predicted	
		Positive	Negative
Target	Positive	TP: 266	FN: 23
	Negative	FP: 12	TN: 277

Extended dataset

		Predicted	
		Positive	Negative
Target	Positive	TP: 529	FN: 46
	Negative	FP: 73	TN: 216

Table B.1. Baseline Model test results

Baseline dataset

		Predicted	
		Positive	Negative
Target	Positive	TP: 263	FN: 26
	Negative	FP: 8	TN: 281

Extended dataset

		Predicted	
		Positive	Negative
Target	Positive	TP: 523	FN: 52
	Negative	FP: 4	TN: 285

Table B.2. Developed Model test results

Initial Hyperparameter tuning

C

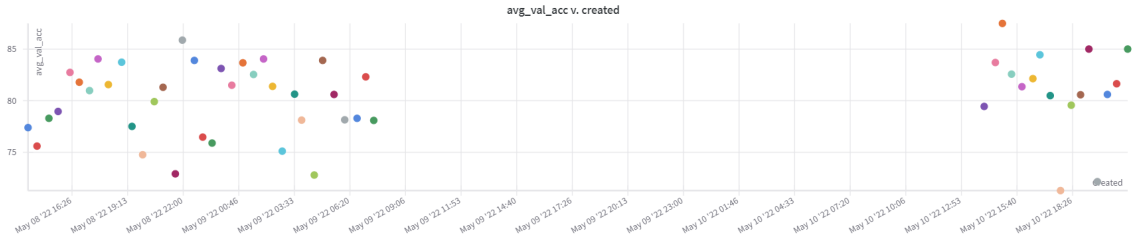


Figure C.1. DenseNet-121

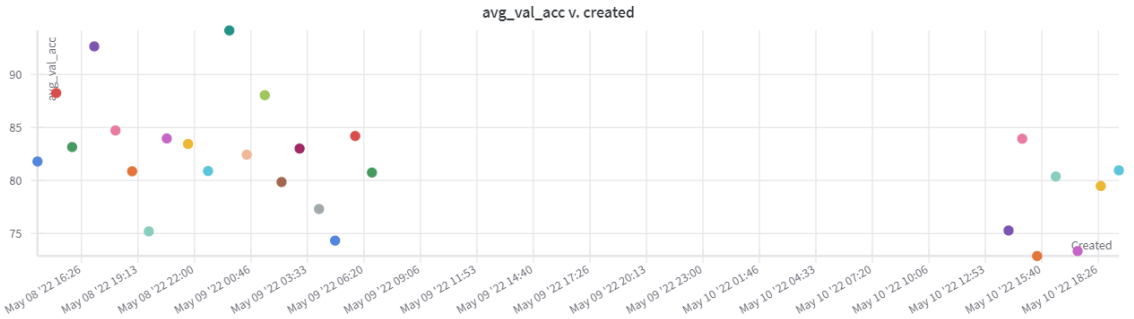


Figure C.2. DenseNet-161

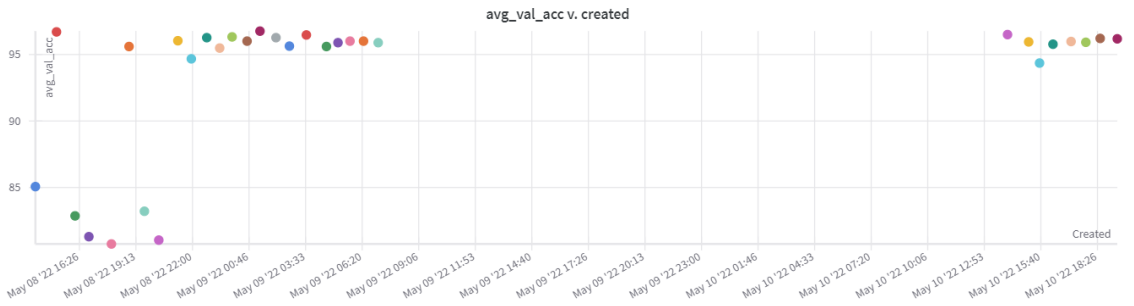


Figure C.3. DenseNet-201

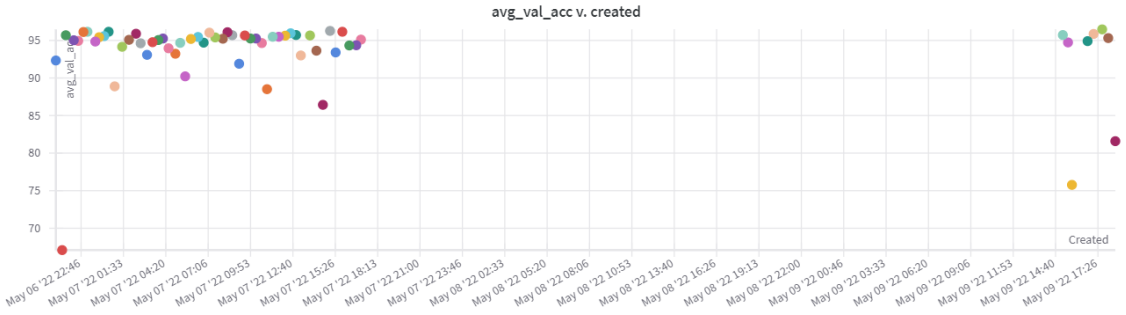


Figure C.4. EfficientNet B0

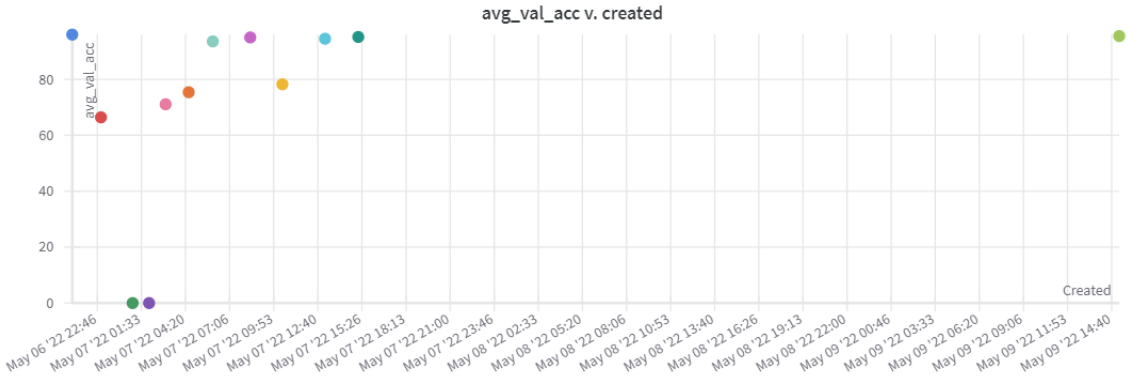


Figure C.5. EfficientNet B4

Figure C.6. EfficientNet B7

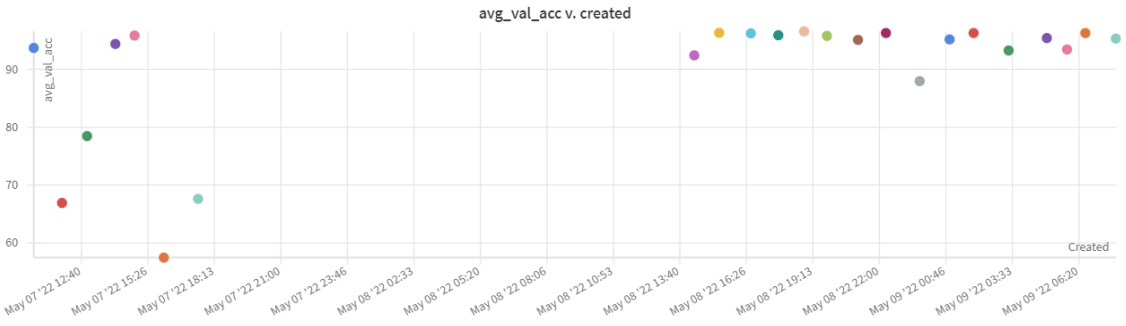


Figure C.7. Inception V3

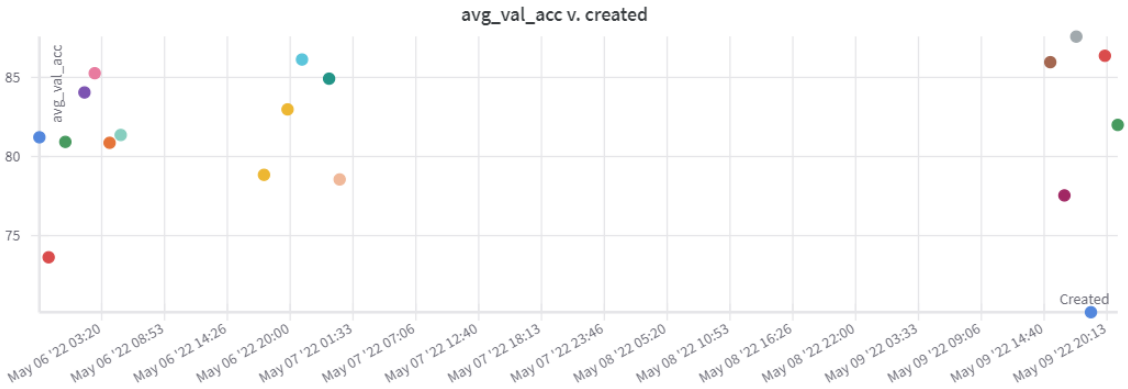


Figure C.8. ResNeXt-51

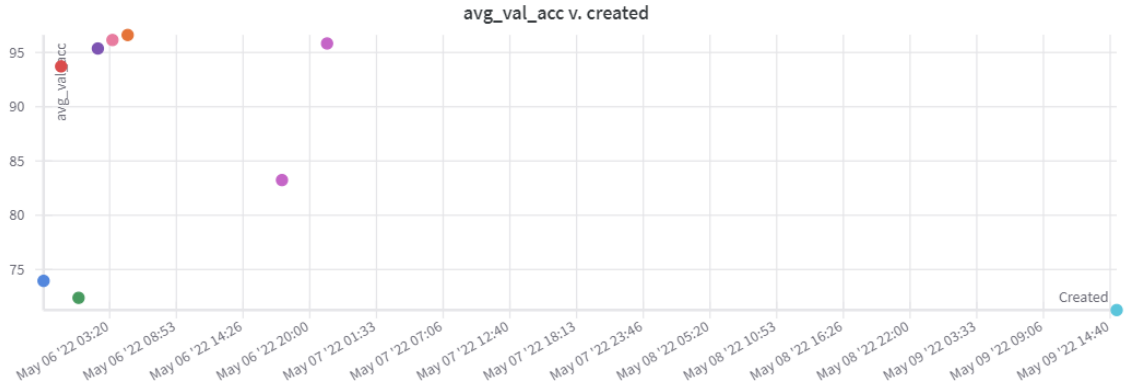


Figure C.9. ResNeXt-101

Figure C.10. VGG-11 w/ Batch normalization

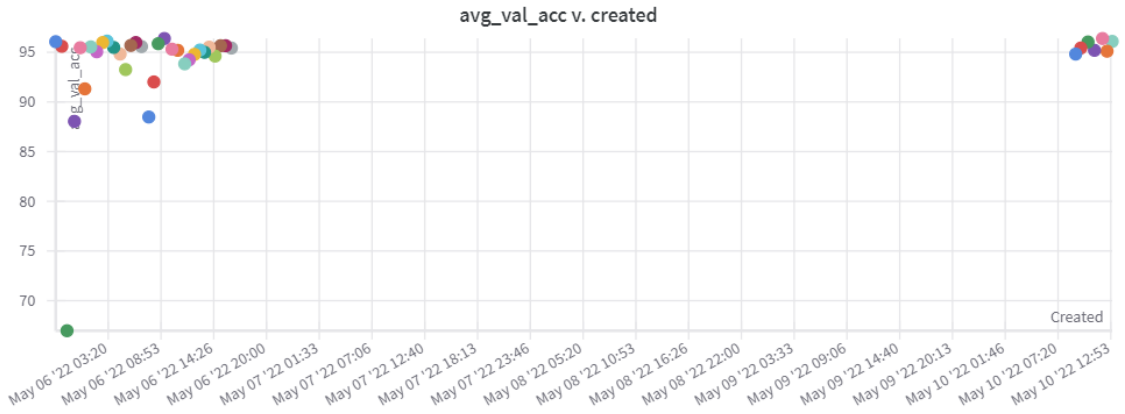


Figure C.11. VGG-16 w/ Batch normalization

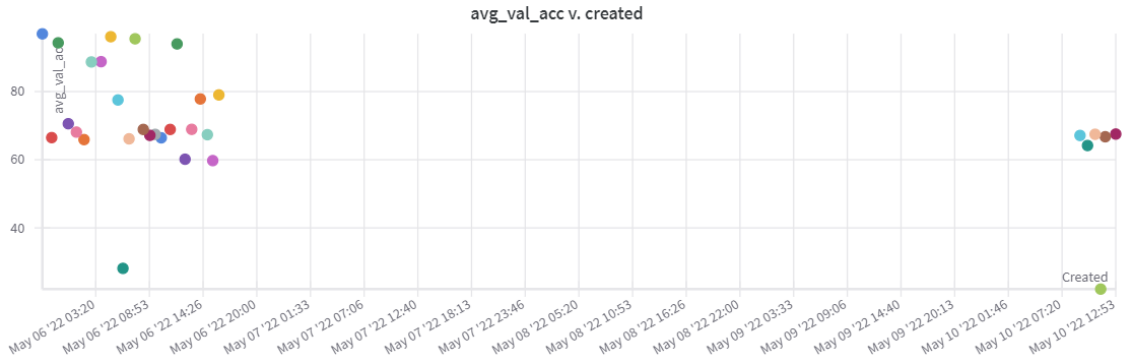


Figure C.12. VGG-19 w/ Batch normalization