Maria Syed

Machine Learning in Healthcare

Identifying Pneumonia with Artificial Intelligence

Helsinki Metropolia University of Applied Sciences

Bachelors of Engineering

Information Technology

Bachelor's Thesis

12 October 2018

Author(s) Title Number of Pages Date	Maria Syed Machine Learning in Healthcare 60 pages + 0 appendices 12 October 2018
Degree	Bachelors of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Janne Salonen, Supervisor

Artificial Intelligence has become popular in a multitude of everyday tasks, from self driving cars to predicting stock market fluctuations, however, it has most notably impacted the healthcare field where its usefulness is especially significant due to the nature of healthcare centers recording large volumes of health data.

Therefore, the applications of Artificial Intelligence in healthcare are endless, for instance, future health problems can be predicted and treatment plans can be devised instantly. However, this thesis aims to explore one of these applications, i.e. disease diagnosis, particularly the diagnosis of pneumonia.

Pneumonia is the infection of the lungs that affects about 400 million worldwide causing various health problems such as difficulty in breathing and uncontrollable coughing. This disease is typically diagnosed with the help of a chest X-ray along with other physical examinations. This thesis aims to build a machine learning model that is able to leverage computer vision to analyse chest X-ray images and identify features that indicate the presence of pneumonia. It aims to speed up the time taken by doctors to diagnose pneumonia and start a suitable treatment plan immediately, helping save lives.

The thesis will uncover the fundamentals of deep learning and computer vision to aid the understanding of how the practical work was done and at the same time explores different methods of developing a functional convolutional neural network. The outcomes of each model developed will be analysed and a conclusion drafted based on the analysis.

Keywords	pneumonia, machine learning, health, neural network, artificial intelligence

Table of Contents

List of Abbreviations

AI	Artificial Intelligence
GPU	Graphics Processing Unit
RML	Right Middle Lobe
ILI	Influenza-Like Illness
PCA	Principal Component Analysis
ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
CNN	Convolution Neural Network
3D	Three Dimensional
API	Application Programming Interface
TPU	Tensor Processing Unit

1 Introduction

Following the 31 year long Artificial Intelligence (AI) winter, AI was reborn in 1997 with the release of IBM's Deep Blue AI and in the years that followed it has spread into almost every aspect of human life. One of the biggest impact AI has made is undoubtedly in the healthcare sector with some even predicting that AI powered doctors will eventually succeed human doctors. This exaggeration may, to some extent, be true, however there is a higher possibility that in the near future AI may simply aid physicians to make critical decisions or substitute the decision making entirely (1.) Given the nature of healthcare centers accumulating large volumes of health data, AI is especially useful in this field since it can bring to light crucial pieces of information that would be otherwise be buried in heaps of data.

There are several startups already taking advantage of this circumstance by applying machine learning with Big Data to provide medical experts with more informative data to help them decide on the correct mode of action. For instance, Google's DeepMind Health project designs the best treatment plan for cancer patients by rapidly analyzing their medical test results and instantly referring them to the correct specialist, hence, creating a "test to treatment" flow which has been proved to be beneficial for the patients. (2) Similarly, the famous Watson supercomputer from IBM has been reapplied for use in oncology where it can leverage its learning on abundant medical research, textbooks, guidelines etc. to assess the medical records and medical evidence of a patient to present possible treatment options along with evidence to support them. An oncologist then solely needs to select the most appropriate option for the patient (3.)

The spike in the popularity of smartphones collecting various health data including but not limited to blood pressure, dietary habits and heart rate, means that the possibilities of not only keeping up with a user's health but also identifying patterns using machine learning models becomes significantly effortless. These models have several uses, from recommending healthier substitutes to sending out alerts when the level of healthiness could be a concern. (4 pp. 6)

It is evident that the possibilities for applications of machine learning in the healthcare field could be endless from personalized treatment to drug discovery to disease diagnosis. However, this thesis focuses on one of these applications, i.e. the diagnosis of pneumonia.

Pneumonia is an infection that affects one or both of the lungs commonly due to viruses or bacteria. Diagnosis of pneumonia typically involves a physical exam along with a chest X-ray analysis. The analysis of the chest X-ray can be aided by using machine learning techniques to rapidly produce a result and a certainty of it. The machine learning algorithm learns from about 5800 images of categorized patient chest X-rays and applies this learning to diagnose new images. The purpose for developing this model is to allow doctors to receive a quick and near accurate prediction of diagnosis and to possibly further examine the patient in need, in effect, allowing them to spend more time on patients who need it.

This research aims to create a solid machine learning model and describe the methods and tools used. Especially it tries to answer the following research question: How can machine learning can be used to most accurately diagnose pneumonia from the analysis of chest X-rays.

The scope of this thesis includes an introduction to theory about AI and machine learning and deep learning techniques relevant to the thesis practical work carried out along with an introduction to pneumonia as an infection, the causes and current medical efforts conducted for diagnosing it. Machine learning techniques were applied to a dataset fetched of about 5800 chest X-ray images of people categorized as either normal or diagnosed with pneumonia.

The criteria used for the evaluation of each test model was whether it could predict the set of validation data most accurately with least errors in a short period of time taking little Graphics processing unit (GPU) power. In addition, the final precision rate was taken into account.

The outcome of this thesis is a machine learning model that fulfils all the criteria and thus aids doctors with the diagnosis of the disease.

This thesis is divided into 7 sections. This is the first section introduces the thesis topic, the research question and outcome. The second, third and fourth sections add theoretical background to the thesis work, while the fifth and sixth sections are related to the implementation and analysis of the project work. The second section explores briefly the complications of pneumonia, the current efforts of doctors in its diagnosis and the benefits of using AI in this process. Next, the third section adds more theoretical background to the practical work carried out this in thesis by exploring the fundamentals of machine learning, AI, neural networks and deep learning and the fourth section introduces the tools and methods used in the project work. While the fifth section dives deep into the implementation of the project work describing the different methods of creating the machine learning model that were used, the sixth section analyzes the results of the implementation and finally the seventh section interprets the results from the research question point of view and draws the final conclusions.

2 Pneumonia

This section gives a brief introduction to pneumonia which is the leading cause of death in underdeveloped countries, elderly communities, infants and the ill. This section also describes the traditional methods of diagnosing pneumonia without the use of AI and then advances further into explaining the importance of using AI to help with the diagnosis of pneumonia.

To begin with, pneumonia, a disease infecting 450 million people worldwide (5), affects the lungs of the infected, inflaming the alveoli (air sacs) and discharging pus in them causing wracking cough with phlegm in addition to chills and uneasy inhalation and exhalation. (6) Figure 1 shows the difference between normal and pneumonia infected alveoli.

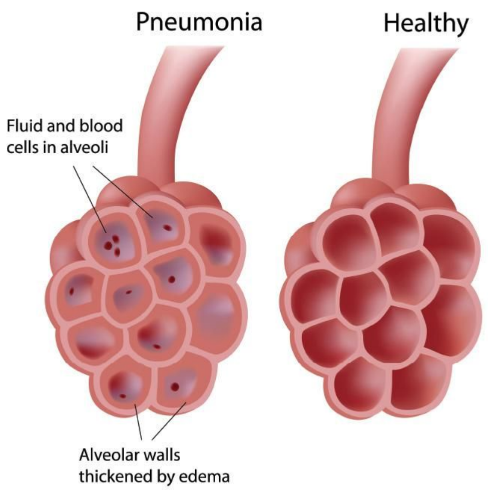

Pneumonia
Healthy
Fluid and blood cells in alveoli
Alveolar walls thickened by edema

Figure 1. How pneumonia affects the alveoli. (7)

As shown in Figure 1, alveoli affected with pneumonia are occupied with fluid and few blood cells while healthy alveoli are empty and able to hold enough air to carry out its normal functions.

While pneumonia can be caused due to several types of germs, it is typically caused by bacteria or viruses present in the inhaled air. In most cases the body can fight off these germs preventing any infection to the lungs, but in some cases the body's immune system is not strong enough (6.)

Pneumonia can be acquired in several ways, for example in hospitals or healthcare institutions where the patient comes into contact with the disease when visiting a hospital for another reason, such as when patients are on ventilators. Pneumonia acquired in this way are quite dangerous since the bacteria causing it might be resistant to antibiotics and moreover, the effects of the infection can be more serious since the patient's condition is considerably weak (6.)

On the other hand, pneumonia can also be acquired outside of hospitals/health care facilities. This acquisition is said to be community acquired and is the most common way of acquiring pneumonia. While the source can be bacteria which can effortlessly spread to a person infected with a common flu, certain fungus found in soil or faeces of birds can also be a cause for this type of pneumonia. Other sources include bacteria-like organisms and certain viruses which cause the common cold, but the infection caused by them are quite mild (6.)

Apart from the community and hospital/health care, pneumonia can also be acquired when food, fluids, saliva or vomit is accidentally inhaled into the lungs. Those with some disruption to their natural gag reflex are at the highest risk for this type of acquisition (6.)

Since pneumonia can be acquired in several ways, it can be difficult to assess the source of the infection. There are several diagnostic techniques administered to infer this information as well as correctly identify the presence of the infection itself. The next subsections will explore the traditional methods of diagnosis and how AI can intercept and help with the process.

2.1 Traditional Diagnosis

Traditionally, in order to diagnose pneumonia, a physical examination is performed which includes listening to the patient's lungs, with a stethoscope, for atypical sounds that indicate the presence of pneumonia. In the case of a suspicion, a series of tests are administered which include blood tests to confirm the presence of pneumonia and to identify the exact organism causing the pneumonia if it is present. A pulse oximetry may also be performed to determine the level of oxygen present in the blood since pneumonia can prevent lungs from moving much of the oxygen into the blood (8.)

The main method of diagnosis, however, is a chest X-ray. With this X-ray, the extent and location of pneumonia can be determined. Interpreting it involves searching for certain white spots called infiltrates in the X-ray of the lungs that show the presence of an infection (8.) Figure 2 shows a chest X-ray of a patient with typical right middle lobe (RML) pneumonia. The infection is highlighted with white arrows.

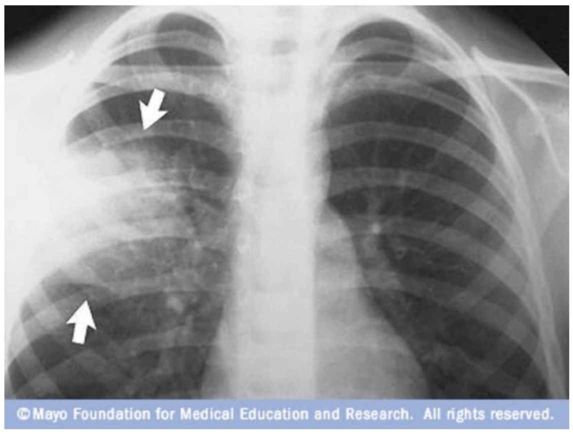

©Mayo Foundation for Medical Education and Research. All rights reserved.

Figure 2. RML pneumonia. (8)

To further identify the type of organism causing the infection and kind of acquisition, other tests can be performed such as the analysis of sputum which is the fluid expelled from the lungs after a deep cough. Analysis of the pleural fluid can also be carried out, where the fluid is extracted from the lungs with a sharp needle between the ribs (8.)

In a chest X-ray, the infiltrates (white spots) can be quite small and difficult to spot at one glance with the human eye, furthermore, other discrepancies may look like white spots resulting in a misdiagnosis. For this reason, a machine learning model with high accuracy would be optimal for spotting the infiltrates in the images. It not only spares time for the radiologists to spend on other important tasks, but is also able to spot very early stages of pneumonia before it has become critical.

2.2 Importance of AI in Diagnosis

As previously outlined, certain radiographs of patients with pneumonia can visually look normal, resulting in false negatives (situation where an ill person is incorrectly diagnosed as normal) which can be fatal for the patient since the right care is not given in time. According to study in Critical Care (9), almost 50% of initial stage pneumonia linked with influenza were not identified in chest X-rays. In addition, during the 2009 H1N1 infection pandemic, italian emergency departments were given 34 cases of pneumonia patients reporting influenza-like illness (ILI) and intense respiratory issues but a chest X-ray analyzed by a human eye failed to identify pneumonia in almost half of them (9.)

This is quite serious since H1N1 can drastically advance into an acute illness resulting in admission to an intensive care unit or in some cases, death. However, detecting pneumonia early enough can decrease the harshness of the infection and linked mortality by initiating the right antiviral treatment in time. In some cases, a chest X-ray may not be enough to diagnose, but a chest ultrasonography should also be taken (9.)

To summarize, the model produced by the outcome of this thesis does not aim to correctly diagnose patients, however, it aims to ease the decision making of radiologists by bringing to light certain features that may be missed by the human eye, therefore helping the detection of pneumonia in its early stages. The next section introduces machine learning, AI and deep learning and how they are related to one another.

3 Machine Learning

Computer science is a vast field with a large number of areas. This sections introduces AI as one of those areas and how it relates to machine learning and deep learning. Following that, this section further investigates the different types of machine learning and the challenges of building models and then dives deeper into neural networks, convoluted neural networks and the theory behind the approach and architectures used in the practical implementation.

In short, AI focuses on replicating human behaviour and decision making in machines in order to give them the ability to take note of their surroundings and then undertake tasks centered towards fulfilling its primary objective. In turn, AI has a number of applications with machine learning being one of them which emphasizes on building computer systems that assimilates data and its experiences to learn and improve itself without being specifically coded for it. Consequently, it makes these computer systems autonomous, i.e. be able to work in complicated surroundings with no continual direction provided by someone, and adaptive, i.e. be able to gain insight from its experience and advance its performance based on that (10.) Figure 3 shows a diagrammatic representation of the relationship between all the fields and subfields discussed with the addition of deep learning, a subfield of machine learning which is explained further in section 3.2.

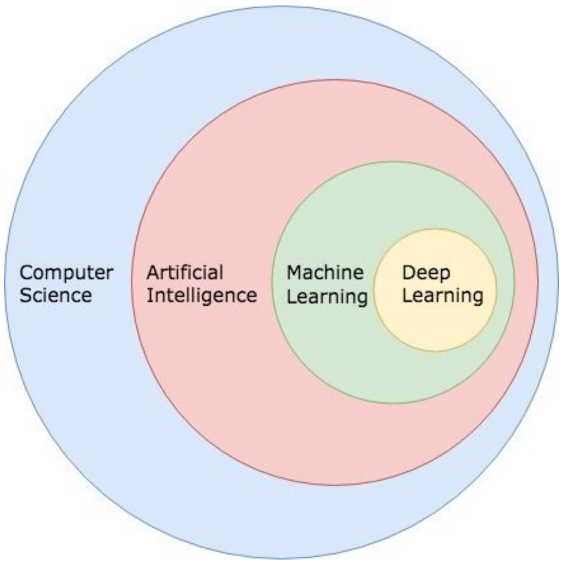


Figure 3. The relationship between computer science, AI, machine learning and deep learning

As can be seen in Figure 3, computer science encapsulates AI which in turn contains machine learning as a subfield and which finally includes deep learning. The practical work of this thesis incorporates deep learning techniques to develop the machine learning models. The next subsection provides a overview on the types of machine learning systems that can be created and explains in more detail the type of machine learning that was used in the practical work and why.

3.1 Machine Learning Systems

To begin with, there are three types of machine learning; supervised learning, unsupervised learning and reinforcement learning. Each type produces a model that learns and improves in different ways, whether it learns from labelled data, unlabelled data or from continuous trial and error.

Firstly, unsupervised learning begins with unlabelled data and attempts to find meaning in it using different machine learning techniques. These techniques either apply *clustering* which is the grouping of data with similar features into clusters and attempting to estimate which cluster the new data belongs to, or alternatively these techniques may *reduce the dimensionality* of the data to form a simple representation of the training data to answer questions about new data effortlessly (12 pp. 55.)

Secondly, reinforcement learning involves the improvement of the machine learning model through constant trial and error with the aim of earning maximum rewards in the long run. The model has no real training data, instead it learns from its own experience. The model, in practice, explores its surroundings and if the outcome of its exploration produces a positive result (reward), it learns from this and continues on this correct path. On the other hand, if the result of its actions results negatively then it tries to perform other actions to stray away from the incorrect path and the process repeats. In this way, the model is continuously learning and improving with every iteration (12 pp. 81.)

On the contrary, supervised learning begins with a set of labelled data and aims to classify new data after studying the relationship between the training data. In the practical work of this thesis, the training dataset fed to the machine learning model is labelled and so supervised learning falls in line with the type of learning that needs to be done.

A supervised learning model studies the relationship in the training dataset with their labels and then applies that knowledge to label new data. As a real world metaphor, this is the type of learning that is typically conducted in schools, i.e. the teacher presents objects to the students and informs them what they are then the students try to recognize these objects in different scenarios and guess what they are.

Supervised learning can perform two tasks; either regression i.e predicting an analogous variable by determining the relationships between the inputs or classification i.e. selecting a distinct label for the unknown input. (12 pp. 16-18)

The labelling of chest X-ray images carried out in the practical work however, falls into the category of classification since distinct labels need to be predicted for the given input image but more specifically, the practical work belongs to binary classification since there are two labels meaning the input image can be one or the other.

Softmax regression is a function used in the architecture of the project work model which initially calculates a score per class then it applies the *softmax function* to the scores to compute an approximate probability for each class. (11 pp. 139) The softmax function is as follows:

$$\widehat{P}_k = \sigma(s(x))_k = \frac{exp\,(s_k(x))}{\sum_{j=1}^{k} exp(s_j(x))} \qquad \text{(11 pp. 140)}$$

K is the number of classes,

$s(x)$ is the vector consisting of the scores pertaining to each class for x

$\sigma(s(x))_k$ is the approximate certainty that the instance x falls into the class k

The aim of these supervised learning models is to calculate parameters that reduce the value of the *cost function (or loss function)* with every iteration. A cost function is essentially a function that calculates how far the model's prediction is from the true values. (11 pp. 20) The idea is that a particular metric is calculated on the network's prediction errors. These errors are averaged over the whole dataset producing a value representing the proximity of the model to the optimal (24.) In the case of logistic and softmax regression, an ideal model predicts large certainty for the target class and a small certainty for the rest of the classes. (11 pp. 140)

There are different loss functions that can be used for the classification purposes. One is the *hinge loss function* which is used for distinct classification of the data, for example, 0 being normal chest X-ray and 1 being pneumonia. Alternatively, the two values could be -1 or 1 (24.)

Although diagnosing whether or not a person has pneumonia is quite distinct, i.e. a person either has or does not have pneumonia, the certainty of the machine learning model is also quite interesting, i.e. a person has 60% chance of having pneumonia. For this reason, a *binary cross entropy* loss function is selected. In this loss function, the output would be a value between 0 and 1 for each class representing the probability that the image belongs to that class. For this reason, logically the final layer in the neural network needs to be a softmax function which as explained, outputs value just as required. The value of this loss function increases as the model's output probability strays away from the actual probability (24.)

One approach to discover the lowest value of this cost function is *gradient descent.* It works by beginning with the cost function with random parameters (*random initialization*) and then it computes the cost function's local gradient at those parameters and selects new parameters which are further in the direction of the decreasing gradient until it reaches a point where the value of the gradient is zero. (11 pp. 111) The calculations converge as shown in Figure 4 and the most optimal parameters are found (12 pp. 24-25.)

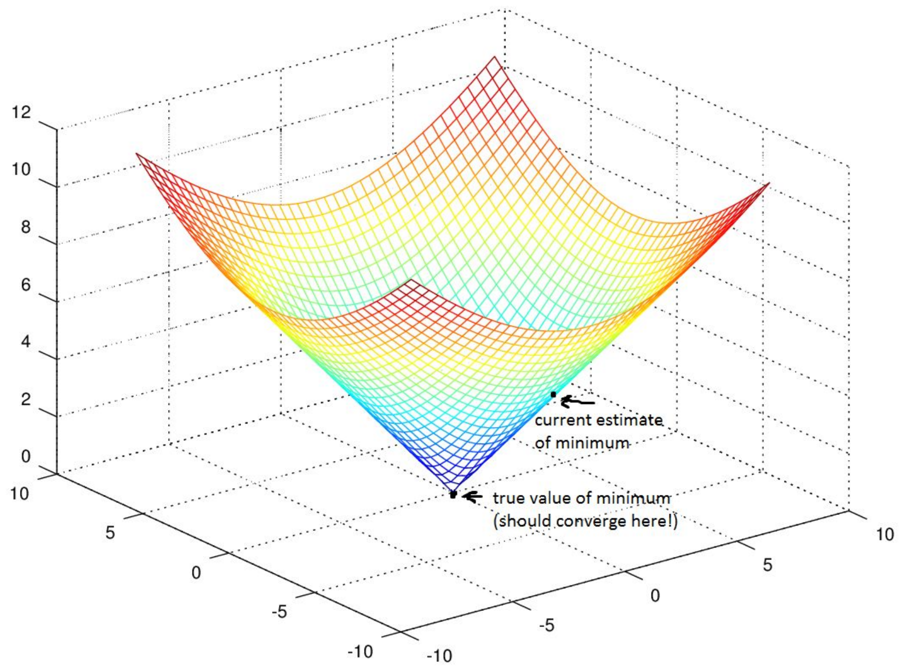


Figure 4. Gradient descent. (12 pp. 25)

The length of the steps taken in gradient descent during each iteration (called the *learning rate*) is quite critical since a low learning rate can take a considerable length of time to arrive at the minimum, however, a high learning rate can result in *overshooting*, i.e. crossing over the minimum to the other half of the graph resulting in a value for the cost function even higher than it was before. Hence, the learning rate needs to be adjusted to overcome these two issues. A good learning rate is one where the value of the cost function is always decreasing after every iteration.

On the contrary, gradient descent does not necessarily compute the *global minimum* i.e. the lowest value of the cost function, it can sometimes settle at the *local minimum* i.e. a point in the graph which is the lowest in its area but not the lowest for the whole function. However, with simple linear regression techniques with the *mean squared cost function* which is most common for this regression, the resulting graph is a shape like that of Figure 4, meaning there is solely one minima and that is the global minima. (12 pp. 112-113)

Bias-Variance Tradeoff

One of the most common problem in supervised machine learning is overfitting. Overfitting occurs when the model learns a function that exactly explains the training data and starts creating relationships or learning characteristics that do not exhibit the actual patterns. On the other hand, underfitting occurs when the model is not formed well enough to identify the actual trends and relationships in the dataset. (11 pp. 26)

This is where bias-variance tradeoff is useful. *Bias* is the magnitude of inaccuracies made when using a simple model to estimate real-world circumstances while *variance* is measure of how sensitive the test errors are to changes in the training data. As a model becomes more complex, its bias reduces but its variance rises i.e. it can explain the training data very well but cannot explain new data as well. Conclusively, an acceptable model should have low bias along with low variance (12 pp. 27-28, 8 pp. 126-127.) Figure 5 demonstrates the consequences of having different levels of bias and variance.

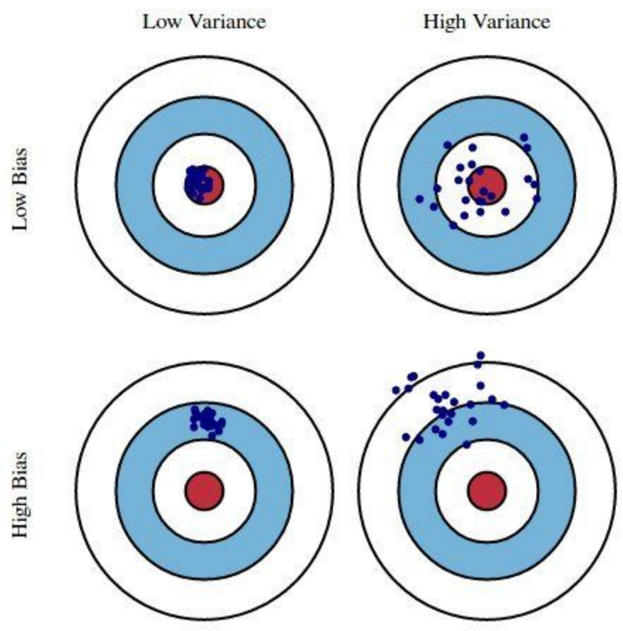


Figure 5. Bias-variance tradeoff consequences. (13)

As illustrated in Figure 5, the best balance of bias and variance is having a low level of each, too high levels in either can skew or scatter the results. In order to attain acceptable levels of bias and variance, a substantially large amount of data needs to be used to prevent inaccurate relationships being inferred and the loss functions needs to be regularized by adding a penalty for big coefficients to prevent over explaining any variable. Therefore the dataset chosen for this thesis is quite large with about 5800 images to use in order to prevent overfitting. This is important to keep a low bias and low variance in the produced model.

3.2 Deep Learning and Neural Networks

To begin explaining the fundamentals of deep learning and neural networks, the *biological neural network* must be explained since it has been studied extensively due to its phenomenal structure and has heavily inspired deep learning. At its core, It is made up of neurons which are interlinked with each other through structures called *dendrites*. Each neuron receives bursts of signals called *impulses* from interlinked neurons and if the number and frequency of these impulses exceed a set threshold, the neuron fires off its own impulse through a long structure called *axon* to the next neurons. This dense network of neurons is collectively called the human neural network and is found in any animal brain (9 pp. 255-256, 2 pp. 92) Figure 6 shows the structure of a singular neuron.

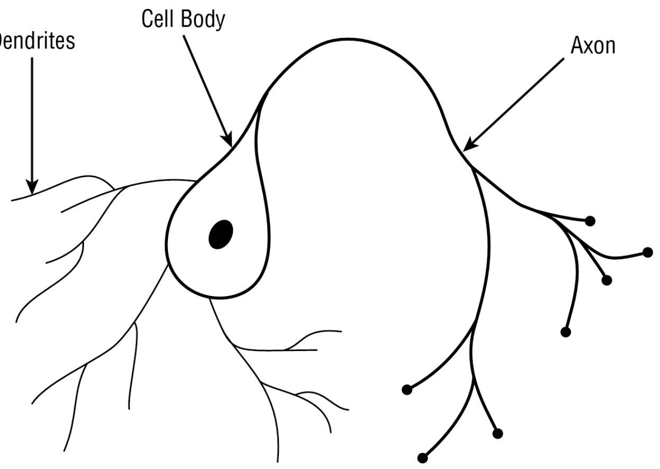


Figure 6. Model of a biological neuron. (4 pp. 92)

As illustrated in Figure 6, the dendrites are connected to the cell body so it can directly send the signals it receives to the cell body for processing and the axon branches off into smaller structures called *synapses* at which the dendrites of the next neuron are connected.

This complicated structure of the human brain has inspired a subfield of machine learning called deep learning to develop Artificial Neural Networks (ANNs) which involve simple processing components connected in a large network in multiple layers to study and model complicated patterns without requiring an unusually large volume of data. ANNs work in a similar manner as biological ones, i.e. it receives several inputs with weights (positive or negative), weighs them individually and sums them up in the summation processor, passes it through an *activation function* and finally fires an output. This configuration is called a Perceptron and is depicted diagrammatically in Figure 7 (14.)

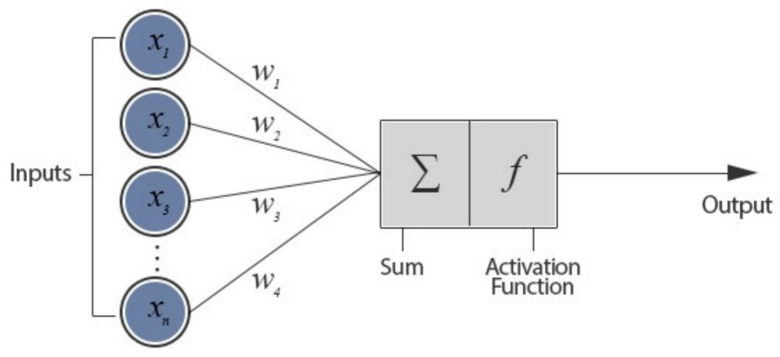


Figure 7. Model of an artificial neuron. (15)

The activation function essentially computes the addition of the weighted inputs and makes the decision of whether to send an impulse to the next neurons or not. (4 pp. 95)

There are several choices of activation functions with the most simplest being the step function which returns either 1 or 0 depending on whether the sum of the inputs are greater than a certain value. In theory, there are two main types of activation functions; Linear activation functions and non-linear activation functions. The former has the equation $f(x) = x$ which is simply a straight line graph with a constant slope hence it does not work well with the complex and multitude of parameters that are commonly input into a neural network. On the other hand, non-linear activation functions are more commonly used since they have a graph which could include curves making it adaptable and generalizable with a multitude of data. Non-linear functions are further divided based on their curves/ranges. Some of these include the sigmoid/logistic activation function and Rectified Linear Unit (ReLU) Activation Function which will be utilized in this thesis' project work (16.)

A sigmoid/logistic activation function is differentiable and monotonic (entirely non-increasing or non-decreasing). This method calculates the addition of the weighted input features along with a bias and returns the *logistic* of the result which is a *sigmoid function* returning a value between zero and one. (11 pp. 134) This logistic function is:

$$\sigma(t) = 1 \div (1 + e^{-t}) \qquad \text{(11 pp. 135)}$$

t is the input value

If the result of this logistic function is above a threshold then the next impulse is fired, otherwise it is not. Figure 8 shows this logistic function graph demonstrating the sigmoid shape of the function.

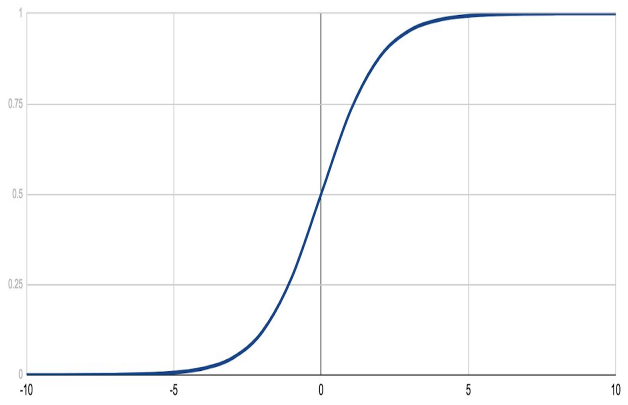


Figure 8. Logistic function graph. (11 pp.135)

On the other hand, ReLU activation function produces a graph, as shown in Figure 9, which has its bottom half rectified hence the y value is always zero if the x value is zero or below. This activation function is heavily used in Convolutional Neural Networks (CNNs) and deep learning (16.)

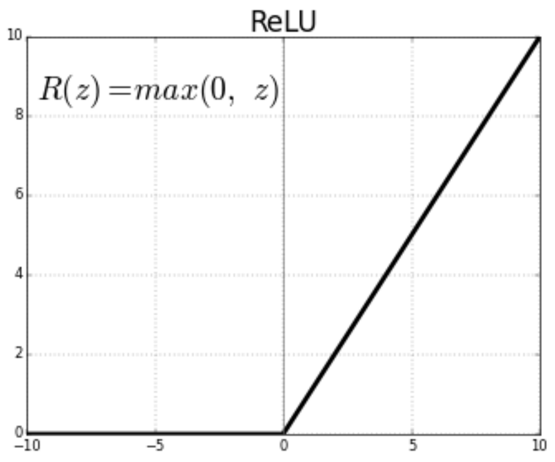


Figure 9. ReLU activation function graph. (16)

A neural networks consists of several these perceptrons connected in one or more layers with the output of one layer being the input of the next. Several neural networks are jointly called multi-layered neural networks as illustrated in Figure 10 which consists of 3 types of layers: Input, hidden and output where there may be several hidden layers which is more commonly found (17.)

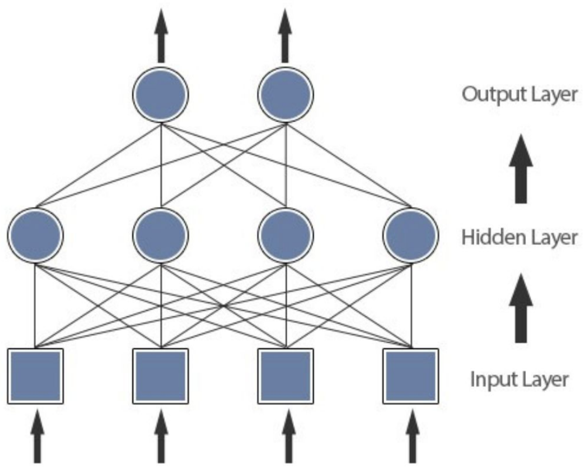


Figure 10. Multi-layered neural network. (15)

It can be seen in Figure 10 that the signals pass successively from the input layer to the hidden layer and finally to the output layer. The main functionality of the hidden layer is to identify the most critical information and patterns from the input layer and exclusively pass those onto the successive layer making the neural network swift and efficient by discarding useless data. (16)

In order to adjust the weights between the input and hidden layers, a method called *backpropagation* is applied making the ANN a *learning algorithm*. This is necessary since only the error margin in the output layer is known i.e. only the correct output from the training sets as well as the actual output from the neural network are known. In backpropagation, the errors are calculated in the final output layer then propagated backwards to the hidden layers which is in turn propagated back to the input layer. In this way, the weights are fine-tuned to reduce the error from the output layer. This process repeats til the weights are only changed very little in each step and the error shrinks to a very small value (16.)

Since majority of real-life machine learning problems involve inputs and outputs having a non-linear complex relationship, ANNs as a solution are consequently a great fit since they possess the ability to learn and model these types of relationships very well. Moreover, ANNs can be applied to a variety of purposes since once it is trained, it is able to deduce hidden relationships from new data that is has never encountered before. Furthermore, since ANNs do not have restrictions on the distribution of input variables and do not impose rigid relationships in data, they are very useful with highly volatile data e.g. stock prices (17.)

There are several types/variations of ANNs being used in Machine Learning each specialised for different uses. One of these is a convolutional neural network which is specialised for image classification and is consequently used in the project work.

3.3 Convolutional Neural Networks

The brilliance of the human vision is unfathomable, in nanoseconds, the eyes digest voxels of data which the human brain not only can identify and name objects in, but it can also differentiate colors, understand depth and distinguish shapes from its background. At the very core of this structure are neurons passing and processing signals with certain neurons in the eyes specialized to assimilate information from light and pass it to the brain for preprocessing and analysis. (18 pp. 85) Identifying images of chest X-rays is a computer vision problem where that exact behaviour of the eyes needs to be replicated in computers in order to process images. This can be carried out with a specialized form of ANN called CNN.

It can be argued that traditional machine learning algorithms such as *linear regression* or *k-means clustering* can be fed raw pixels of image data and it can attempt to classify them by identifying common features and their relationships. However, this would not work in practice since the signal to noise ratio would be extremely low for any proper learning. On the contrary, a further argument could be made that if a human could handpick important features from the images and feed that to a traditional machine learning algorithm, the model would perform better since the dimensionality of the problem would be lowered and the signal to noise ratio would increase. Although it may be true that the model would indeed perform better, it would not have really grasped the underlying idea of the image without requiring the entry of several thousands and thousands of handpicked features which in hindsight could be time consuming and difficult in complex problems. (18 pp. 87-88)

In comparison, deep learning would be suitable for image classification of chest X-rays since every layer of a neural network learns and builds up features from the input data it obtains. However, a vanilla/regular deep neural network would be unsuitable for this classification task since the chest X-ray images are quite large with some about 2090

× 1858 pixels large, so each neuron in a fully connected hidden layer would receive about 3,883,220 input weights and that would mean the network would need a very very large number of neurons over numerous layers. This makes it quite uneconomical and increases the likelihood of the model overfitting to the data. Evidently, vanilla deep neural networks have scalability issues as demonstrated in Figure 11 (18 pp. 89.)

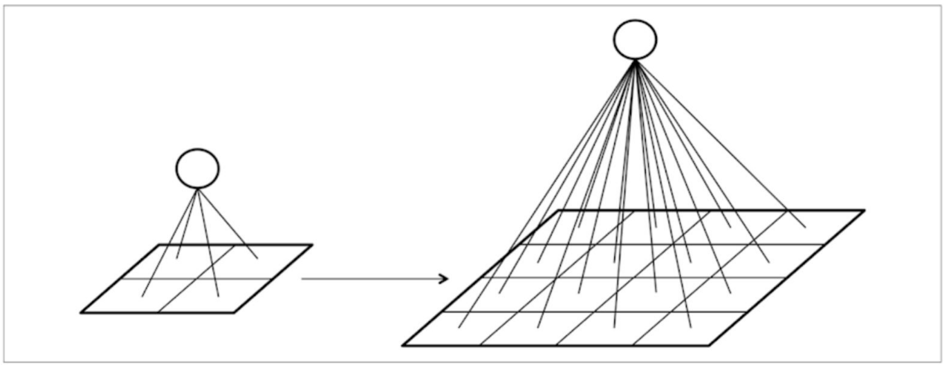


Figure 11. The number of connections in between layers escalates as the image size increases. (18 pp. 89)

On the other hand, CNNs remarkably decrease the amount of parameters by arranging the neurons in the layers in three dimensions, i.e. the layers have height, breadth as well as depth as seen in Figure 12. Consequently, CNNs are the most suitable architecture to be used in image classification. This architecture has been heavily inspired by how human vision works (18 pp 90.)

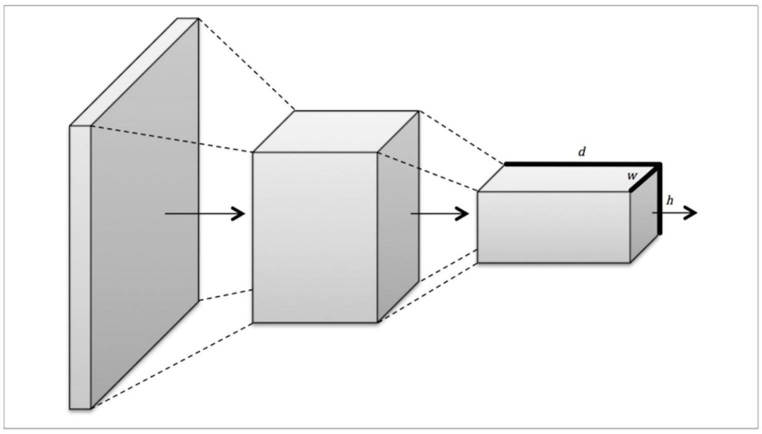


Figure 12. Neurons arranged in three dimensions to reduce number of parameters. (18 pp. 90)

CNNs essentially reshapes two dimensional image data to Three Dimensional (3D) data which includes the image width, height and the number of channels the image has, for example, a grayscale image has 1 channel while a full color image has 3 channels (red, green, blue). This is depicted in Figure 12 as 3D blocks of data passing through each 3D layer one after another. Simply, CNNs apply multiple filters to an image's raw pixel data to isolate and study higher-level features which the machine learning model can use for classifying. (19)

The architecture of a CNN is similar to an ANN containing an input layer and an output layer. However, instead of the hidden layers, CNNs consist of three important components; convolutional layers, pooling layers and dense layers. (19) A simple architecture of a CNN is showing in Figure 13.

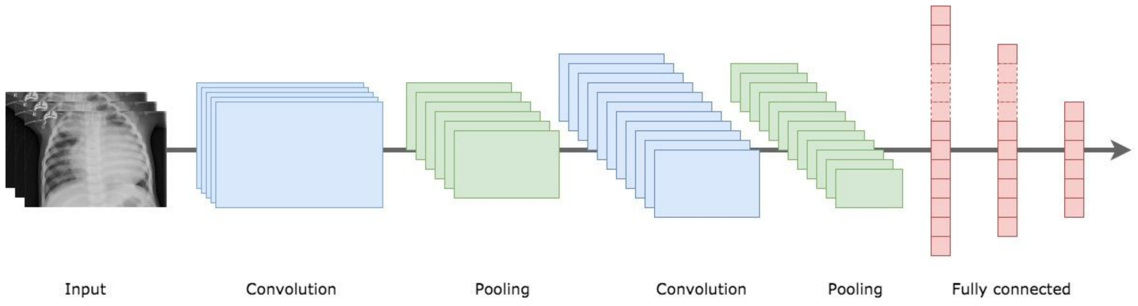


Figure 13. Common CNN structure. (11 pp. 367)

The CNN architecture illustrated in Figure 13 has an input layer followed by pair of convolution and pooling layers then again followed by a second pair of convolution and pooling layers and finally the fully connected layers which is connected to the output layer. Each layer has different functionalities.

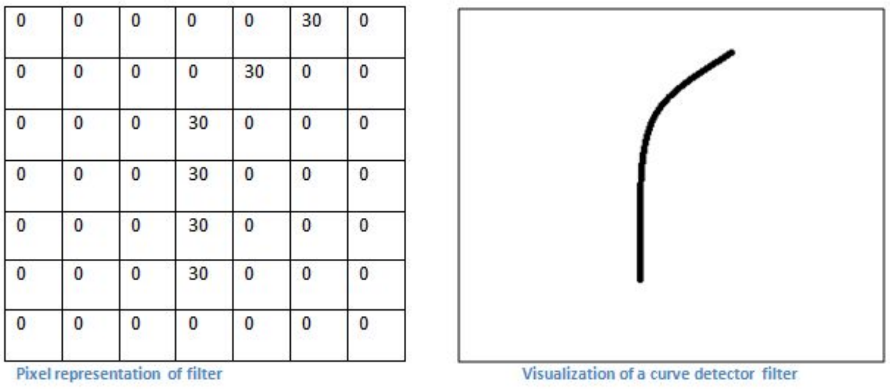
Convolutional Layers

The convolutional layers receive the 3D image data from the input layer and apply a set amount of *convolutional filters* to it to produce a 3D output *feature map*. Neurons in this layer are only connected to a little portion of the layer before it, making its efficiency far better than neurons that are fully connected. (18 pp. 94)

This layer's design was inspired by how human's visual cortex is arranged in layers where each layer is scattered with feature detectors replicated over the entire area to detect and expand upon the features identified in the preceding layers. (18 pp. 91)

In practice, convolutional layers consist of multiple layers of neurons in a feed-forward neural network and each applies a set of filters/kernels to identify certain features in the image. A filter is essentially a matrix of weights trained to identify specific features. It advances across every section of the image to check for the presence of particular features and it performs a *convolution operation* in order to produce a value of the measure of confidence it has that the feature exists. A convolution operation is essentially the multiplication and addition of two matrices and in the context of these filters, it produces a larger value if the feature is actually present in the segment of the image, otherwise it produces a low value (20.)

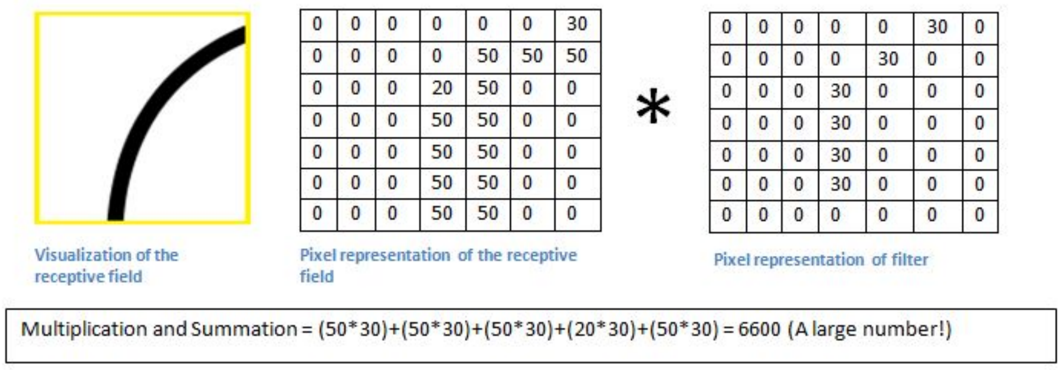
To illustrate the function of the convolutional filters, consider the pixel filter illustrated in Figure 14 with weights distributed specifically to identify a right curve.


Pixel representation of filter
Visualization of a curve detector filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Figure 14. Example of a convolutional filter and its visualization for a right curve feature. (21)

The pixel representation of the filter in Figure 14 has weights of 30 distributed in the shape of a right curve with weights of 0 surrounding it. The visualization of the filter shows the right curve it searches for. Figure 15 shows the result of applying this filter to a segment of the image that contains a right curve (20.)


Visualization of the receptive field
Pixel representation of the receptive field
*
Pixel representation of filter

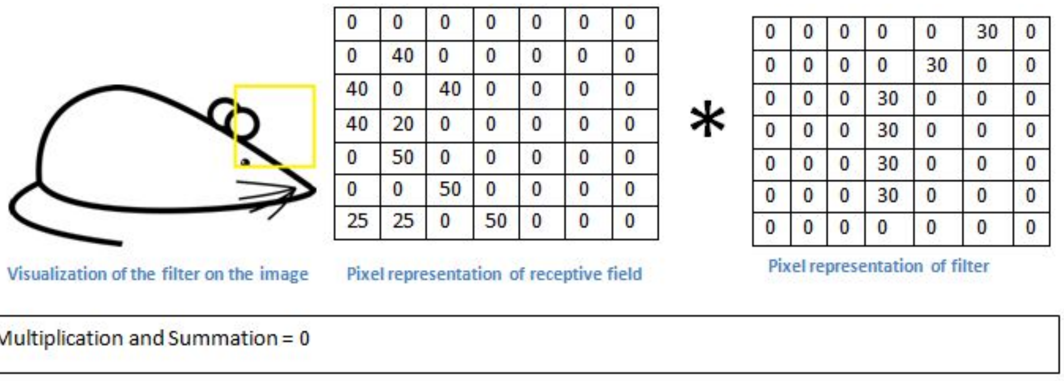
0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Figure 15. Application of the filter to a segment of the input image with a right curve. (21)

As shown in Figure 15, the product of the calculation between the input image's pixels and the convolutional filter's pixels is a very large number indicating the right curve feature is, in fact, present in this image segment. However, if the same filter was applied now to a different section of the image, one without a curve bending to the right, it outputs a low value for that section as shown in Figure 16 (20).


Visualization of the filter on the image
Pixel representation of receptive field
*
Pixel representation of filter

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Multiplication and Summation = 0

Figure 16. Application of the filter to a segment of image without any right curves. (21)

The calculation of the pixels in Figure 16 produce a value of 0 indicating that this image segment does not contain the right curve. Consequently, after applying this filter to all sections of the image, an output feature map is produced which consists of all the convolutions of the input image. It is important for the filter to contain the exact number of channels as the input so that the matrix multiplication can be applied. This is why since the chest X-rays used in this thesis are black and white with only one channel, the filters also have a single channel. Alternatively to passing the filter over every section of the image, it can also slide over intervals with a *stride* value (20.) A convolution with strides has output dimensions that can be calculated as:

$$n_{out} = floor(\frac{n_{in} - f}{s}) + 1 \qquad (20)$$

n_{out} is the output dimension

n_{in} is the input image dimension

f is the window size

s is the stride.

Figure 17 shows a complete 3D visualisation of how the convolutional filters extract the features from the 3D image data and finally produce the 1 by 1 by 1 feature map.

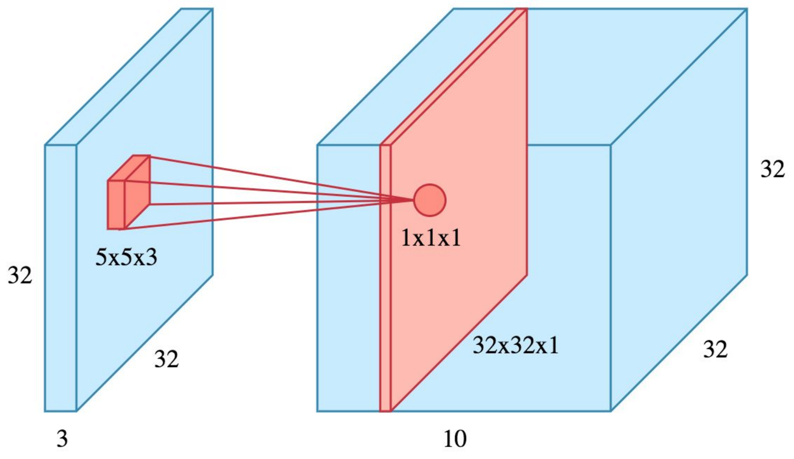


Figure 17. 3D representation of one convolutional filter being applied to image. (22)

Figure 17 shows the image data on the left with dimensions 32 by 32 by 3 meaning the images are 32 pixels wide, 32 pixels high and is full color RGB since it has 3 channels. The convolution filter has dimensions of 5 by 5 by 3, the depth of the filter must match the depth of the image. The filter produces an output feature map with dimensions 32 by 32 by 1 as shown in red in the right side of the figure. Each filter produces a feature map with a depth of 1. So 10 filters will produce a output feature with a depth of 10. The feature maps produced by each filter are stacked onto each other to produce the final output feature map (22.)

In order for the CNN to learn the weights of a filter, it must undergo some non-linear mapping. This is carried out by adding a bias term to every output produced by the filter and then passing through a non-linear activation function like the ReLU activation function in the case of the CNN used in this thesis. This non-linearity in the network is important since the model needs to account for the non-linearity in the input (20.)

Pooling Layers

Pooling layers are responsible for reducing processing time by downsampling the data it receives from the preceding convolutional layers in order to decrease the feature map's dimensionality and sharpen the identified features. Max pooling is a typically used algorithm for pooling. In this algorithm, a window moves over subregions of the feature map with a specified stride, keeping the highest value and discarding the rest shown in Figure 18 generating a compressed feature map (19.)

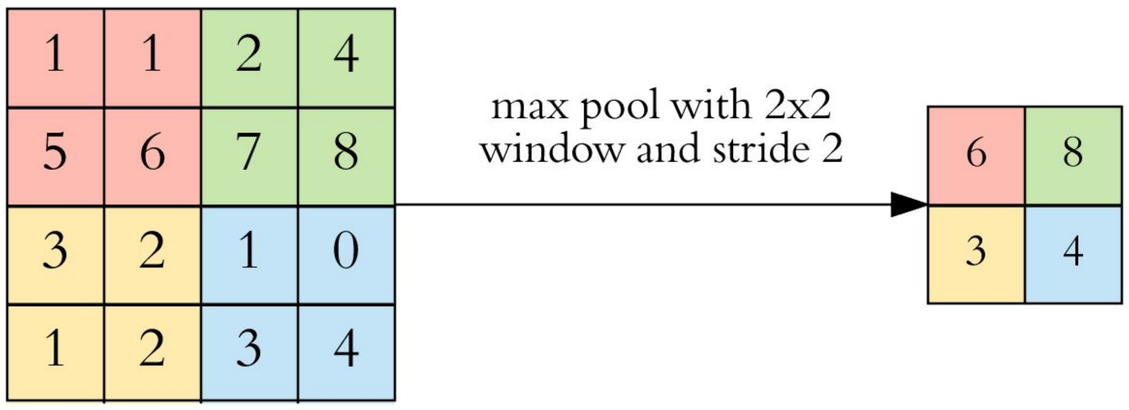


Figure 18. 2 by 2 max pooling. (22)

As is demonstrated in Figure 18, the 2 by 2 max pooling window first moves over the top left portion containing the 2 by 2 matrix of numbers in light red, of which 6 is the maximum number so it is recorded on the top left of the resulting matrix. Next, the max pooling window moves to the top right 2 by 2 portion of the image which has the maximum value of 8 and it is therefore recorded on the resulting matrix. The window then proceeds to move over the final bottom two portions of the image and record their maximum values. Finally the final resulting matrix containing the maximum numbers 6, 8, 3 and 4 is produced. (22)

Figure 19 shows a 3D representation of the pooling layers' functionalities. The depth of the final output is always equal to the depth of the input data. (22)

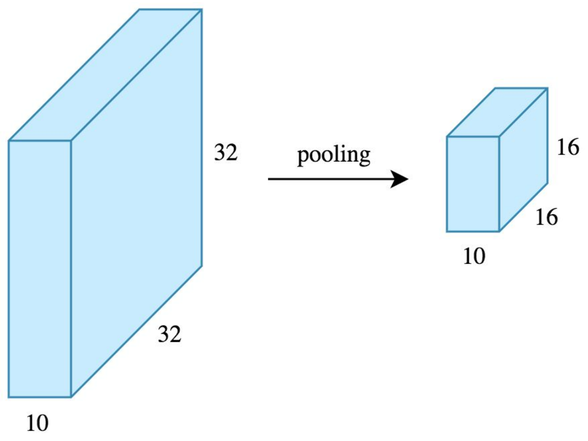


Figure 19. 3D representation of the reducing dimensionality feature of pooling. (22)

As is seen in Figure 19, pooling drastically decreases the volume of memory needed and the amount of operations carried out in the next processes of the network. In addition, it makes the network focus on only particular neurons reducing the chances of overfitting the data. The size of the output of a max pooling operation is calculated exactly the same way as convolutional output dimensions are calculated.

Furthermore, max pooling output remains the same even in the event of the input variables moving slightly. This is especially beneficial if the location of the feature is unimportant compared to the existence of that feature. On the contrary, a large amount of this local invariance could damage the ability of the network to hold critical information. Consequently, an efficient max pooling layer should have low spatial extent (18 pp 99.)

After the convolutional layers isolate the features and the pooling layers downsample them, the output is passed to the dense layers or fully-connected layers for *flattening* and classifying.

Dense/Fully Connected Layers

The dense layers are responsible for classifying the features into the possible classes. These layers are also called fully connected layers since all the nodes contained in it are connected with every node in the layer preceding it (19). In these layers, the input is *flattened* to produce a feature vector, this process is depicted in Figure 20, where every row in the input is chained together to form a long feature vector. More the number of layers, longer this flattened feature vector is (20).

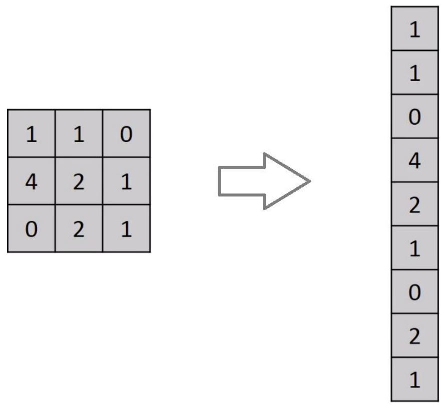


Figure 20. Flattening to produce a feature vector. (23)

As illustrated in Figure 20, the 3 by 3 matrix of numbers are sequentially added row by row to the feature vector chain forming a new vector with 9 rows. This feature vector passes through several dense layers, where in each layer, it is multiplied by specific weights, added to set biases and introduced non-linearities (20).

In the final output layer in the neural network, there exists one node per target class that the model can output along with a softmax activation function to calculate a number between zero and one for each class. This value generated can be interpreted as the probability that the image belongs to the corresponding class (19.)

3.4 Transfer Learning

Building a whole CNN architecture requires a large amount of time, GPU and data to find the correct weights for achieving the best results. However, there exists machine learning models developed pre trained on a gigantic dataset with a great amount of generalization i.e. it can be applied to big variety of datasets and still produce a good prediction. One alternative to constructing own model is to begin training with a model that has already been trained on other similar data and train that model on another dataset leveraging their feature extraction abilities. This process is called *transfer learning*. It performs well when the amount of data is in the thousands and it can be run in a very short period of time in a GPU-less machine (24.)

Transfer learning is possible since CNNs can learn perceptively generic features in the early layers and then steadily develop features particular to the given dataset in further layers. These features from the first few layers are the "building blocks of vision processing" (24).

Hence, an alternative method to building a model to identify pneumonia could be to apply transfer learning on an existing well trained generalized CNN model. There are several pretrained models available that are well able to learn with new data. This thesis explores transfer learning using the popular Inception v3 and the MobileNet v1 model.

3.4.1 Inception v3

Google's Inception v3 is a model for image recognition that was developed by the Google Brain Team for ImageNet's visual recognition challenge. It has proven to achieve an accuracy more than 78.1% on the Imagenet dataset which consists of a large number of images classified into a thousand classes. The concept of the model was developed over the years by several expert researchers (25.)

The paper "Rethinking the Inception Architecture for Computer Vision" published in 2015 was strongly used as the basis of the inception v3 model. (25)

This paper states that improvements in the performance of a classification model is transferable to a large multitude of computer vision tasks. Different CNN models created in 2014, for example VGGNet, needed a large amount of computation for the evaluation of the network. On the other hand, GoogLeNet architecture of Inception was constructed for functioning under rigid computational limitations. This special design resulted in using parameters amounting to only five million beating AlexNet being a twelve times reduction from it. In comparison, VGGNet used three times more parameters than AlexNet. This justified the use of Inception in big data where a large amount of data needs to be processed at a relatively low computational cost (26.)

The famously paper describes certain design strategies to optimize the design of Inception. Firstly, it describes that bottlenecks (which relates to the layer preceding the last output layer performing the classifications) can be avoided so the depicted size from the inputs reduces moderately down to the outputs prior to undertaking the last representation for the assignment given. This ensures the assessment of the data is not determined solely on the dimensions of the data which at best can provide only an approximate guess (26.)

Secondly, in order to train the CNN at a higher rate, the activations for each unit tile can be increased. Thirdly, to bring about quick learning, the input's dimensions can be decreased prior to carrying out spatial aggregation functions since there is very little loss in information when the dimensions are lowered. Finally, the paper suggests to harmonize the network's width as well as depth to achieve the best performance. It can be carried out by adjusting the magnitude of filters with respect to the network's depth (26.)

With all of these adjustments to the existing structure of CNNs at that time, including factorising the convolution layer by increasing the size of the filters and decreasing the size of the grids for efficiency. (26)

This architecture and all of the ideas put forth in the paper has revolutionized the Inception structure. Now Inception v3 is built with convolutions, average pooling, max pooling, concatenates, dropouts and fully connected layers with batchnorm being utilized heavily in the model and used on the activation inputs with the softmax activation function to calculate the loss. Figure 21 shows the high level model of the Inception v3 (25.)

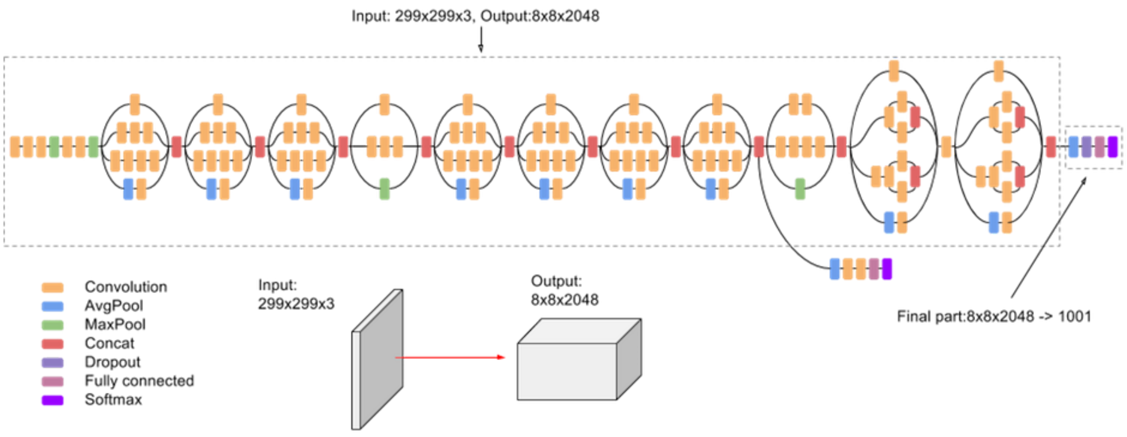


Figure 21. Inception v3 high level architecture. (25)

As is shown in Figure 21, the sample input to the model is a 299 by 299 image with 3 channels and the output is a three dimensional 8 by 8 by 2048. There are about 11 layers in the model where the final two layers can be retrained with new data and the model should be able to produce a mode with a good accuracy.

3.4.2 MobileNet v1

MobileNets are a new grade of models with high efficiency for "mobile and embedded vision applications". A MobileNet model makes use of factorized convolutions to form a depthwise convolution and a pointwise convolution which is basically a 1x1 convolution.

The combination of depthwise and pointwise convolution is called *depthwise separable convolution*. These models were built on the basis of utilizing "depthwise separable convolutions" in order to create deep learning neural networks that are lightweight (27.)

Essentially, a regular convolution layer performs both the filtering and merging inputs within a single step, however, the depthwise separable convolution splits this process into 2 layers, one for filtering and one for merging. This significantly decreases the size of the model as well as the computation needed (27.) Figure 22 shows the difference between a regular convolution versus a MobileNet's depthwise separable convolution.

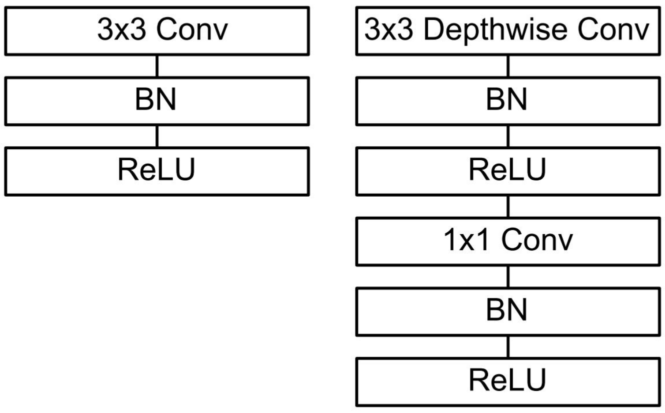


Figure 22. Left: A regular convolution layer. Right: Depthwise Separable convolution. (27)

The regular convolution layer shown in Figure 22 begins with a standard 3 by 3 convolution, followed by a batchnorm and finally a ReLU function. However, the MobileNet convolution layer begins with a depthwise convolution layer which is a factorized form of the standard convolution, followed by pointwise convolution an ReLU with batchnorms between all the layers.

Despite of utilizing this special type of convolution, a MobileNet model still keeps its first layer as a full convolution. Every layer of the MobileNet architecture has a batchnorm and ReLU function right after it except the last dense layer since it requires no nonlinearity, instead it is linked to a softmax layer to produce probabilities for classification (27.)

4 Project Tools and Methods

There are various deep learning tools that are available, out of these two, Keras and TensorFlow, were tested and used in this thesis, therefore, this section will focus on the fundamentals of these tools, the advantages they offer and why they were chosen. Furthermore, this section will also emphasize the method of building a machine learning model, more specifically the seven steps in building a machine learning model.

4.1 Keras

Keras is a simple high level application programming interface (API) for building and training neural networks with a significantly low learning curve. This library is written entirely in Python programming language and can allow other deep learning languages of a lower level run under it like TensorFlow, CNTK or Theano (28.)

Kera was built with user-friendly interface and quick experimentation in mind giving it an advantage over other deep learning tools. It provides a logical straightforward interface which reduces to a minimum the amount of actions required by the user for typical uses, as well as it gives feedback at the time of error that is comprehensible (28.)

Furthermore, since Keras is easily learnt and simple to use, it increases productivity of the user giving it a competitive edge in machine learning competitions and challenges. Not only is Keras user-friendly, but it is also transposable and extensible since their models are built by interlinking customizable units together with hardly any restrictions (28.)

Evidently, Keras is highly used in not only startups like Netflix and Uber focussing on deep learning but also in deep learning research communities making it the second most mentioned deep learning framework in scientific papers sent to arXiv.org as of October 2017 (shown in Figure 23). (28)

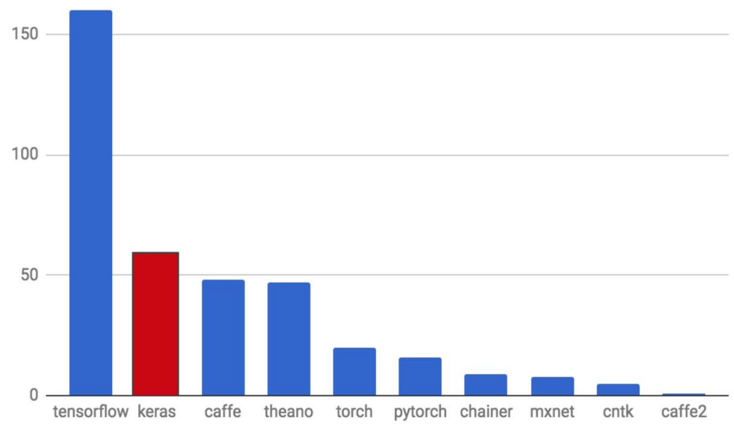


Figure 23. arXiv mentions in October 2017. (28)

Evidently, as seen in Figure 23, Keras has been mentioned second most in arXiv while TensorFlow takes the largest share being mentioned about 3 times as much as Keras since it is open sourced.

Further popular features of Keras include the capability of developing with various other deep learning backends with the added bonus that a model with only built-in layers can be moved easily across the backends, for example, a model can trained with a particular backend then loaded with a different one. The choices of backends are limited to TensorFlow, CNTK and theno backend with MXNet backend support currently in progress (28.)

Due to the ease of use, efficiency and popularity of Keras, it was selected as one of the tools used to develop a deep learning model for image classification.

4.2 TensorFlow

Originally used by developers in Google to build machine learning models for internal use, TensorFlow has since then in 2015 been released as an open source software library for developers to develop and train neural networks. This library is also written in Python where data is expressed as arrays with multiple dimensions called tensors. (18 pp. 39)

TensorFlow visualizes distinct calculations as a graph of *data flows* where each node stands for an arithmetic operation. When used in the context of deep learning, neural networks are represented as tensors giving the possibility to leverage state-of-the-art hardware for increased speed producing an elegant yet articulate method for the implementation of models (18 pp. 39-40.)

In order to supply inputs to a model produced by TensorFlow, a *placeholder* component is initialized every time the computational graph is instantiated. TensorFlow uses *sessions* to interact with the computational graph. A session's purpose is to initialize the graph and all of the variables and to run the graph as well. Consider a placeholder variable *a,* another TensorFlow variable *b* and a bias *c*. In a session where *a* and *b* are multiplied by TensorFlow operation `tf.Mul` and the bias *c* is added the result, the resulting computational graph generated would be Figure 24 (18 pp. 46-47.)

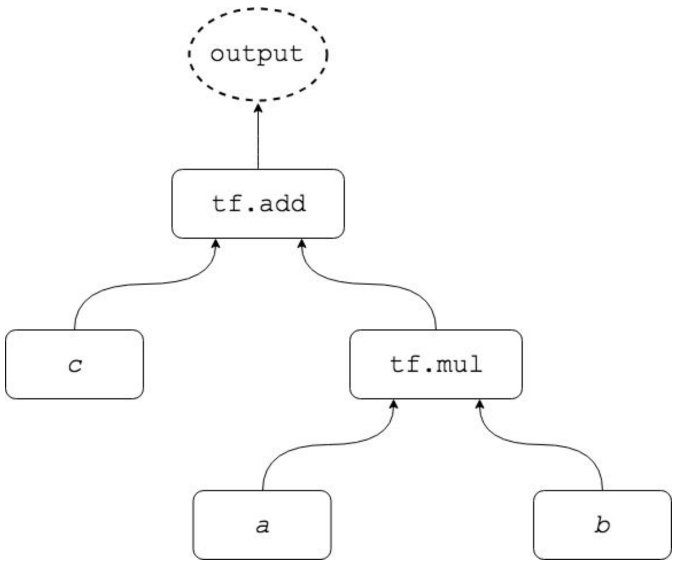


Figure 24. Sample computational graph in TensorFlow

Google released a new breed of processing units called Tensor processing unit (TPU) which is a hardware chip developed specifically for machine learning and was customized to work seamlessly with TensorFlow. This hardware chip was programmed to have highly optimized execution using little power for its machine learning tasks. Furthermore, Google has since then released a second generation of TPUs in mid 2017 which supply a performance of almost two hundred teraflops, moreover, if these TPUs are arranged in clusters, the performance could exceed 10 petaflops. Evidently, the performance of TensorFlow has been heavily invested on and so has been chosen as the second deep learning tool used in this thesis (29.)

This subsection gave a brief simplistic introduction to how Google's TensorFlow works under the hood, however, there are various other concepts to study related to TensorFlow and its use. TensorFlow was used in this thesis project to create the model produced from retraining the pretrained models which are described in detail in section 5.

4.3 Building a Model

A model in the context of machine learning is the product of training a machine learning algorithm on a training data set. This model can then be applied on new data to predict outcomes. There are 7 essential steps in build a machine learning model and thereby accurately identifying pneumonia in chest X-rays; data collection, data preparation, model selection, training, evaluation, hyperparameter tuning and finally prediction (30.)

The first step is to gather and prepare the data. This step is critical since the amount and the condition of the data can determine how successful the model is. The data for the purposes of this thesis was gathered from kaggle where the data was collected from children aged 1 to 5 from Guangzhou Women and Children's Medical Center in Guangzhou. These X-rays were all scanned during their scheduled clinical care. There are 5863 images of chest X-rays of two categories, normal or pneumonia, where the diagnosis for pneumonia was triple checked by 3 expert physicians before they were sent for training AI (30.)

After gathering the data, comes the next step of data preparation. At this stage, the data needs to be void of errors, labelled correctly and also needs to be segmented into three. The largest portion of this split is the training set, i.e. the data the model learns from. The second largest portion is the test set, i.e. the data the model assesses its loss from while it trains. Finally, the third and smallest portion is the evaluation set, i.e. the data we use to evaluate our final model to truly infer how accurate it is. It is important to separate this data since evaluating the model based on the training data is useless since the model already knows what class it falls in. Using new data for this purpose truly gives us the real world performance (31.)

In this case, the data came prepared from the kaggle dataset as separated in three folders, train, test and val (evaluation), where each of these had images separated in two folders representing their category, normal or pneumonia. Also, prior to the data being uploaded for AI use, low quality or incomprehensible images were removed during its quality control screening. (30)

The third step in machine learning is the selection of a model. This is the main portion of the practical work which was experimenting with different kinds of models and identifying which suits the best for our machine learning problem and dataset. Since this is an image classification task, a CNN is certainly used. The reasons for using a CNN was described in detail in section 3.3. Section 5.1 and 5.2 explores the different alternatives to CNN that can be used to solve this problem and then section 6 evaluates the two models based on certain metrics and section 7 presents the suitable model that has been selected (31.)

The fourth step is training. This is the step where the dataset prepared is fed into the model to iteratively improve its predictive abilities. The labelled training images are input to the CNN model, in which certain features of the image pertaining to their class are identified via the various convolutional filters which then is passed through some pooling layers to reduce its dimensionality and finally to the dense layers which pass it through a softmax function to output a probability value for each class, normal and pneumonia. During the training, the parameters of the neural network are tuned and optimized in each iteration in order to minimize the loss in the output layer which is calculated by a particular loss/cost function. While at first it performs poorly, the predictivity becomes better and better but must be stopped at a certain point to avoid unbalancing the bias and variance tradeoff (31.)

The fifth step is evaluation. Using our evaluation set extracted in data preparation step, we try to predict whether the given image has pneumonia or not. Outputting a higher probability value for the correct class more number of times confirms that the model has a good predictivity.

The sixth step in machine learning process is hyperparameter tuning. This is the process of tweaking our parameters of our model in order to improve our model (31.) This is carried out several times in sections 5.1 with details on experiments with different number of epochs. The aim is to improve the evaluation of the model. If the final evaluation improves then it shows the tuning is in the right direction. Otherwise, alternative tunings are attempted.

Finally, the seventh and final step is prediction. This is the stage when the machine learning model has graduated from its experimental stages and can now be applied in practice and answer real life questions (31.)

5 Implementation

This section focuses on the implementation of the project work, more particularly, the fourth, fifth and sixth steps in machine learning, starting with training CNN models, then evaluating the results and finally tuning certain parameters to aim to improve the model. There are two approaches to building a CNN that are used, first one being building the CNN from scratch with Keras, i.e. creating a machine learning model, adding the required layers, activation function and finally compiling with a certain loss function. While the second approach is applying the method of transfer learning. Both approaches were be tested and the results presented.

5.1 Building CNN with Keras

As described in detail in section 4.1, Keras is a simple to use, highly efficient deep learning API. The fourth step in machine learning is training, hence, this subsection explores the building and training of CNNs with Keras.

The first step is to import all the related libraries, as shown in Listing 1.

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```
Listing 1. Importing required libraries from Keras.

These packages include the *Sequential* Keras model which is a model built ahead where layers just need to be linearly stacked. It is useful in the modelling of this CNN with its many layers. Since the inputs are images with two dimensions, i.e. width and height, two dimensional convolution layers are used, hence, *Conv2D* is imported. While *MaxPooling2D* is imported for max pooling in two dimensions. Max pooling is selected for use in the pooling layers of this CNN. Finally, *Flatten* and *Dense* correspond to the flattening of the feature vector and the dense/fully connected layer respectively.

One the required packages have been imported, the machine learning convoluted neural network can begin to be built. To do this, the sequential model is initialized as in Listing 2

```
# Initialising the CNN
model = Sequential()
```
Listing 2. Initialising the CNN model.

Then a two dimensional convolutional layer is added as shown in Listing 3

```
model.add(Conv2D(32, (3,3), input_shape=(64,64,1), activation='relu'))
```
Listing 3. Adding convolutional layer to model.

As shown in Listing 3, a two dimensional convolutional layer is added to the model with 32 output filters, filters with shape of 3 x 3, 64 x 64 input images with one channel since the X-ray images are grayscale and it utilizes the ReLU activation function.

Next, a second convolutional layer is added and after that a two dimensional max pooling layer with the pool size of 2 x 2 as seen in Listing 4.

```
# Adding a second convolutional layer
model.add(Conv2D(32, (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
```
Listing 4. Adding second convolutional layer and max pooling layer to model.

Following that, the flattening is carried out to reduce the multi-dimensional feature vector into a long chain of vectors as in Listing 5.

```
# Flattening
model.add(Flatten())
```

Listing 5. Adding flattening layer to model.

And finally the fully connected layers are added with ReLU activation function and a sigmoid activation function. After that the classifier is compiled with the binary cross entropy loss function and calculate the loss based on accuracy metric. Listing 6 shows these additions.

```
# Full connection
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dense(units = 1, activation = 'sigmoid'))
# Compiling the CNN
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
```
Listing 6. Adding the dense layer to the model and compiling it.

Finally after the whole model is set up and compiled, the training and testing data can be prepared and fed into the model. First, the training and test set need to be rescaled and reshaped. This is carried out with the help of *ImageDataGenerator* where the images are scaled down by 255 and reshaped to 64 by 64. Listing 7 shows these steps.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rescale = 1./255)

# Fetching training data set
training_set = datagen.flow_from_directory('data/train',
target_size = (64,64),
batch_size = 32,
color_mode='grayscale',
class_mode='binary')

# Fetching test dataset
test_set = datagen.flow_from_directory('data/test',
target_size = (64,64),
batch_size = 32,
color_mode='grayscale',
class_mode='binary')
```

Listing 7. Resizing and reshaping train and test dataset.

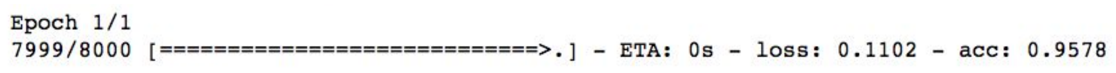
As seen in Listing 7, `datagen` is an instance of the `ImageDataGenerator` and is used to load the training and test set scaled down by 255 with target size 64 by 64 as grayscale and binary images. This makes all the data in the set of the same size and have the same shape keeping them uniform.

Finally we run the classifier with certain number of epochs with certain number of steps per epoch and number of validation steps as in Listing 8.

```
model.fit_generator(training_set,
steps_per_epoch = 8000,
epochs = 1,
validation_data = test_set,
validation_steps = 2000)
```

Listing 8. Fitting images to model.

As seen in Listing 8, 8000 training steps are set per epoch with 1 epoch defined and utilizing the test dataset with 2000 validation steps. Figure 25 shows the progress of this training on a Jupyter notebook.

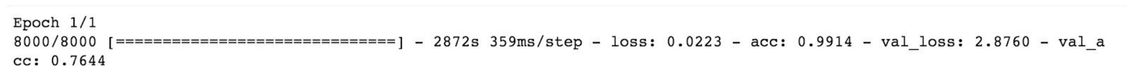


```
Epoch 1/1
7999/8000 [============================>.] - ETA: 0s - loss: 0.1102 - acc: 0.9578
```

Figure 25. One epoch running progress on Jupyter notebook

As shown in Figure 25, every iteration shows the loss and the accuracy which are based on the training data. At the end of one epoch, the validation accuracy and loss are shown which represent true accuracies since it is the value obtained from comparing the results on the test data that has been kept aside.

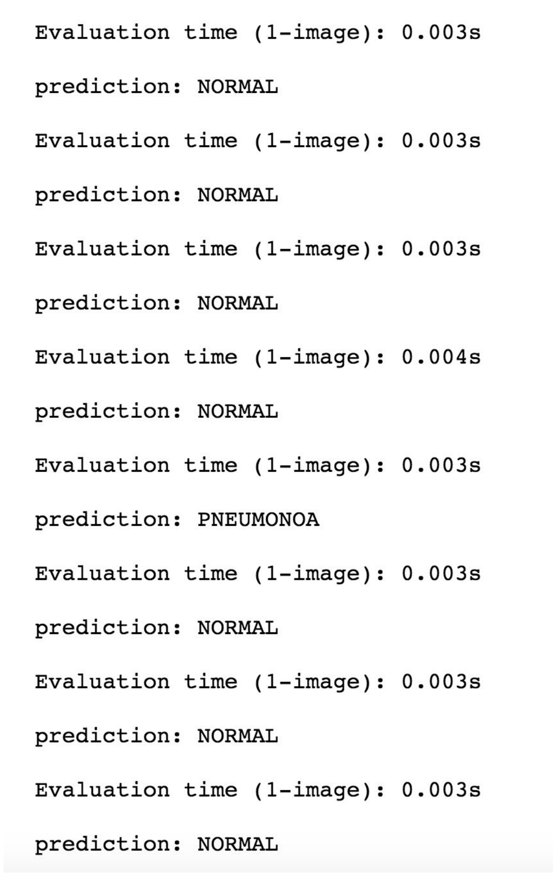
One epoch with 8000 training steps was run which took about 40 to 60 minutes to complete training. The results can be seen in Figure 26.



```
Epoch 1/1
8000/8000 [==============================] - 2872s 359ms/step - loss: 0.0223 - acc: 0.9914 - val_loss: 2.8760 - val_a
cc: 0.7644
```

Figure 26. Results of one epoch run.

The next step in building a machine learning model is evaluation. In here, the results of training the model with 1 epoch are evaluated. As shown in the figure, running one epoch produced a validation accuracy of only 76.4% with training accuracy of 99.1%. While 76.4% is not a very high number since it means that 1 in 4 patients are incorrectly diagnosed, an evaluation of the models in this thesis are carried out with the 16 evaluation images kept aside. These are new data the model has not been trained or tested on. At first, the model was made to predict the classes of 8 normal chest X-ray images. Figure 27 shows the results.

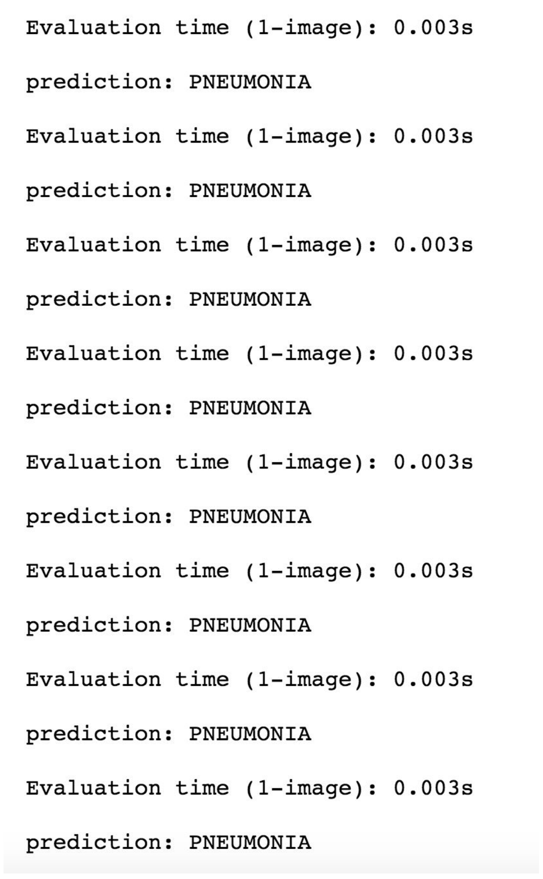


```
Evaluation time (1-image): 0.003s

prediction: NORMAL

Evaluation time (1-image): 0.003s

prediction: NORMAL

Evaluation time (1-image): 0.003s

prediction: NORMAL

Evaluation time (1-image): 0.004s

prediction: NORMAL

Evaluation time (1-image): 0.003s

prediction: PNEUMONOA

Evaluation time (1-image): 0.003s

prediction: NORMAL

Evaluation time (1-image): 0.003s

prediction: NORMAL

Evaluation time (1-image): 0.003s

prediction: NORMAL
```

Figure 27. Evaluation of model training with 1 epoch on normal chest X-ray images.

The results of the predictions shown in Figure 27 show all but one were correctly identified as normal and each prediction was completed in milliseconds which is exceptionally fast. The model seemed to get 7 out of 8 predictions correct meaning 87.5% correct with this small dataset.

Next 8 images of X-rays with pneumonia were fed to the model. Figure 28 shows the results.



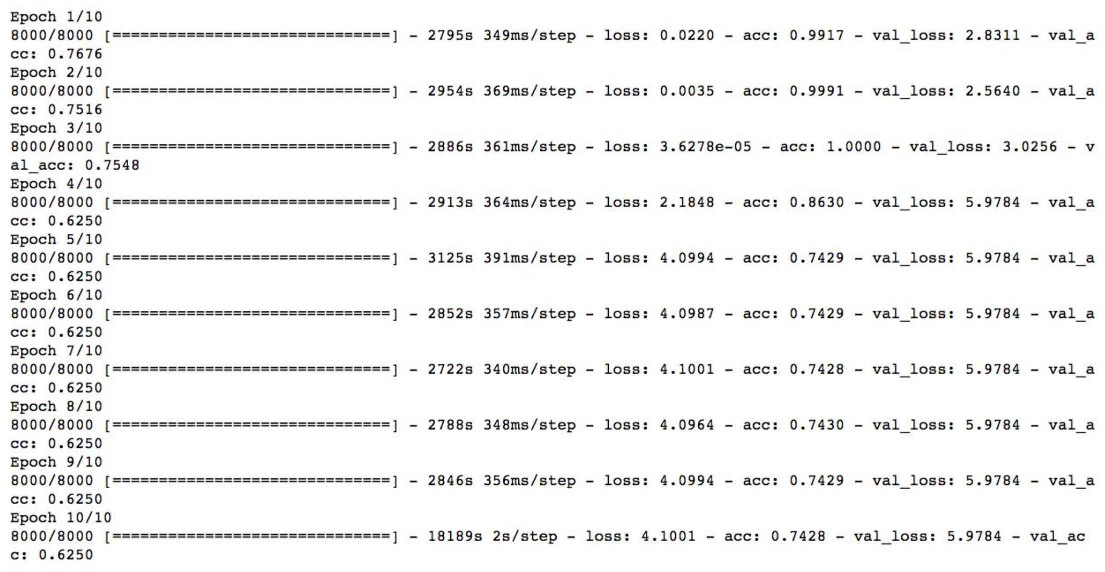
```
Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA

Evaluation time (1-image): 0.003s

prediction: PNEUMONIA
```

Figure 28. Evaluation of model training with 1 epoch on pneumonia chest X-ray images.

As displayed in Figure 28, all of the images were correctly classified within milliseconds, giving it a 8 out of 8 meaning 100% accurate.

In conclusion, the final tally for 8000 straining steps of a CNN built from scratch by Keras was 76.4% validation accuracy with 93.8% accuracy with evaluation data giving predictions in 3 milliseconds. While the results of the evaluation data is quite impressive already, the validation accuracy is still not satisfactory. However, certain parameters were tuned to further bring the results to what was expected.

The sixth step in machine learning after evaluation is hyperparameter tuning. Hence since the number of epochs in the last iteration was quite low, being only one, the value was increased to 10 epochs and the results collected. Figure 29 shows the final validation accuracy and loss after running 10 epochs.



```
Epoch 1/10
8000/8000 [==============================] - 2795s 349ms/step - loss: 0.0220 - acc: 0.9917 - val_loss: 2.8311 - val_a
cc: 0.7676
Epoch 2/10
8000/8000 [==============================] - 2954s 369ms/step - loss: 0.0035 - acc: 0.9991 - val_loss: 2.5640 - val_a
cc: 0.7516
Epoch 3/10
8000/8000 [==============================] - 2886s 361ms/step - loss: 3.6278e-05 - acc: 1.0000 - val_loss: 3.0256 - v
al_acc: 0.7548
Epoch 4/10
8000/8000 [==============================] - 2913s 364ms/step - loss: 2.1848 - acc: 0.8630 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 5/10
8000/8000 [==============================] - 3125s 391ms/step - loss: 4.0994 - acc: 0.7429 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 6/10
8000/8000 [==============================] - 2852s 357ms/step - loss: 4.0987 - acc: 0.7429 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 7/10
8000/8000 [==============================] - 2722s 340ms/step - loss: 4.1001 - acc: 0.7428 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 8/10
8000/8000 [==============================] - 2788s 348ms/step - loss: 4.0964 - acc: 0.7430 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 9/10
8000/8000 [==============================] - 2846s 356ms/step - loss: 4.0994 - acc: 0.7429 - val_loss: 5.9784 - val_a
cc: 0.6250
Epoch 10/10
8000/8000 [==============================] - 18189s 2s/step - loss: 4.1001 - acc: 0.7428 - val_loss: 5.9784 - val_ac
c: 0.6250
```

Figure 29. Progress of accuracies running 10 epochs.

As demonstrated in the figure, the first epoch produced a validation accuracy of 76.76% as expected, however, it can be seen that the validation accuracy begins dropping with every iteration, settling at 62.5% after the fourth iteration. On the other hand, the accuracy increases from 99.17% to 100% at the third epoch but then starts drastically reducing and finally settles at about 74%. These results are significantly worse than the results before and consequently the model performs poorly during the final evaluation.

It can be inferred from these results that the model could be overfitting to the data, hence it performs well with data it is familiar with and performs poorly with new data. The largest difference was in the third epoch, as can be seen in Figure 29, where the accuracy was 100% but validation accuracy only 75.48%.

All in all, it can be concluded that the CNNs did not perform up to expected, although, the model that ran for 1 epoch did significantly better and had a fairly good accuracy. The next subsection will explore a completely different approach of building CNNs.

5.2 Transfer Learning

As explained in section 3.4, an alternative to building a customized CNN is to apply transfer learning to existing CNN models. This thesis will explore retraining the Inception v3 model and the MobileNet v1 model.

Inception v3 Retraining

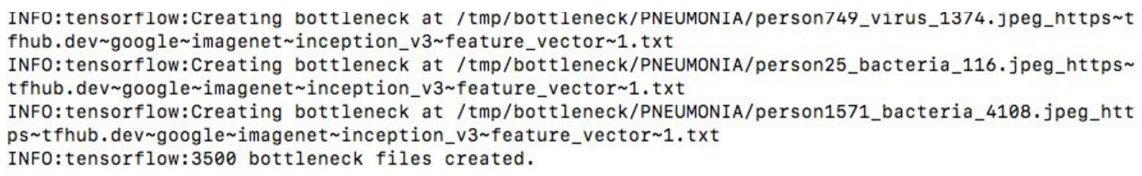
In order to retrain Inception v3 which has been pre-trained on ImageNet, TensorFlow Hub needs to be used to consume the modules which are essentially parts of a pre-trained model.

Quite simply, the first step to begin the retraining was to download the retraining code from GitHub (32) which was stored in file called *retrain.py*. Next to run the retraining process, code shown in Lising 9 was run in terminal on MacOS, where the last parameter points to the directory the images were stored.

```
python retrain.py --image_dir <path to image dir>
```

Listing 9. Command for retraining Inception v3.

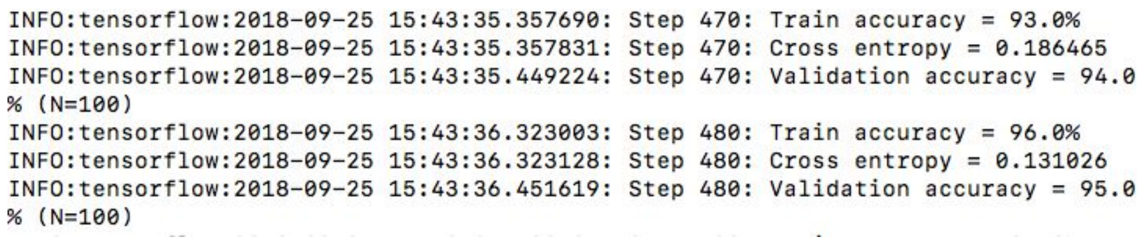
The very first step is computing the bottleneck values for all the images fed to the script and caching these values to be reused any time the same data is retrained. This layer is responsible for producing adequate values that the model can use for differentiating between the two classes. Therefore, it needs to produce a meaningful and information packed summary relating to the images. It has been seen that the data required to distinguish between the 1000 categories available in ImageNet is adequate to classify unfamiliar data. Figure 30 shows the logs relating to the creation of these bottlenecks (33.)



```
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/PNEUMONIA/person749_virus_1374.jpeg_https~t
fhub.dev~google~imagenet~inception_v3~feature_vector~1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/PNEUMONIA/person25_bacteria_116.jpeg_https~
tfhub.dev~google~imagenet~inception_v3~feature_vector~1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/PNEUMONIA/person1571_bacteria_4108.jpeg_htt
ps~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.txt
INFO:tensorflow:3500 bottleneck files created.
```

Figure 30. Creation of bottlenecks in the terminal.

As shown in the figure, there is a bottleneck that is created for each image and saved into a folder named tmp. After all the bottlenecks have been created, the training of the last few layers of the model commences. The number of training steps defaults to 4000 which is enough to produce adequate results. Figure 31 shows the logs relating to the training (26.)



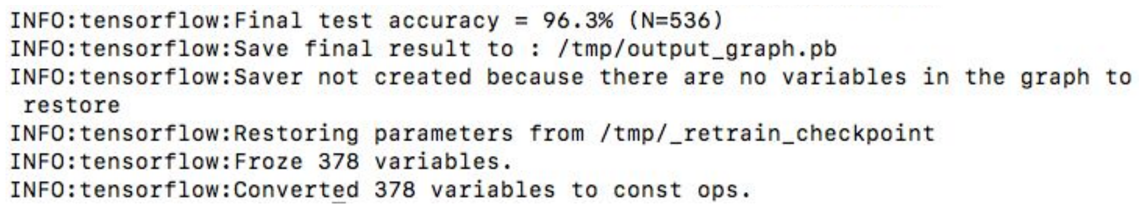
```
INFO:tensorflow:2018-09-25 15:43:35.357690: Step 470: Train accuracy = 93.0%
INFO:tensorflow:2018-09-25 15:43:35.357831: Step 470: Cross entropy = 0.186465
INFO:tensorflow:2018-09-25 15:43:35.449224: Step 470: Validation accuracy = 94.0
% (N=100)
INFO:tensorflow:2018-09-25 15:43:36.323003: Step 480: Train accuracy = 96.0%
INFO:tensorflow:2018-09-25 15:43:36.323128: Step 480: Cross entropy = 0.131026
INFO:tensorflow:2018-09-25 15:43:36.451619: Step 480: Validation accuracy = 95.0
% (N=100)
```

Figure 31. Logs related to the retraining process.

As displayed in Figure 31, there is a train accuracy, cross entropy and validation accuracy. While the train accuracy represents the share of the training images in the current set accurately labelled, the cross entropy shows how well the training is performing relative to the result of the cross entropy loss function and the validation accuracy relates to the accuracy on a batch of images selected at random from another set. This represents the legitimate accuracy since it is measured on new data the model has not learnt from. A low validation accuracy and high training accuracy indicates the model is overfitting to the data (26.)

This loop runs 4000 times and in each iteration it selects 10 images at random, send it to the last layer, compare the results with the correct labels, compute the loss and backpropagate to the previous layers to update their weights. In this way, the accuracy is improved at every step and at the last run, the model is evaluated on a batch of images stored separate from those learned from to present the final accuracy (33.)

With 4000 iterations, the final test accuracy was 96.3% as shown in Figure 32.



```
INFO:tensorflow:Final test accuracy = 96.3% (N=536)
INFO:tensorflow:Save final result to : /tmp/output_graph.pb
INFO:tensorflow:Saver not created because there are no variables in the graph to
 restore
INFO:tensorflow:Restoring parameters from /tmp/_retrain_checkpoint
INFO:tensorflow:Froze 378 variables.
INFO:tensorflow:Converted 378 variables to const ops.
```

Figure 32. Final test accuracy of the retraining.

This result is very great and is a value that was beyond the expected range of 90% - 95%. Figure 33 shows the changes in validation accuracy and cross entropy loss over training steps.

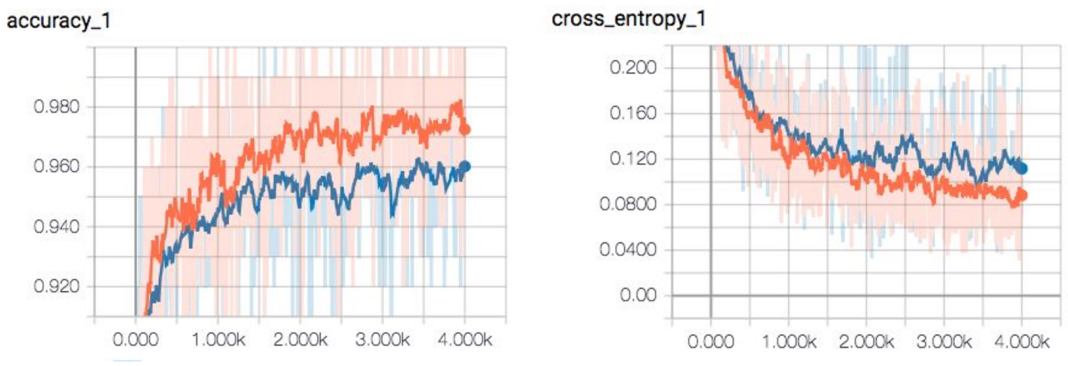
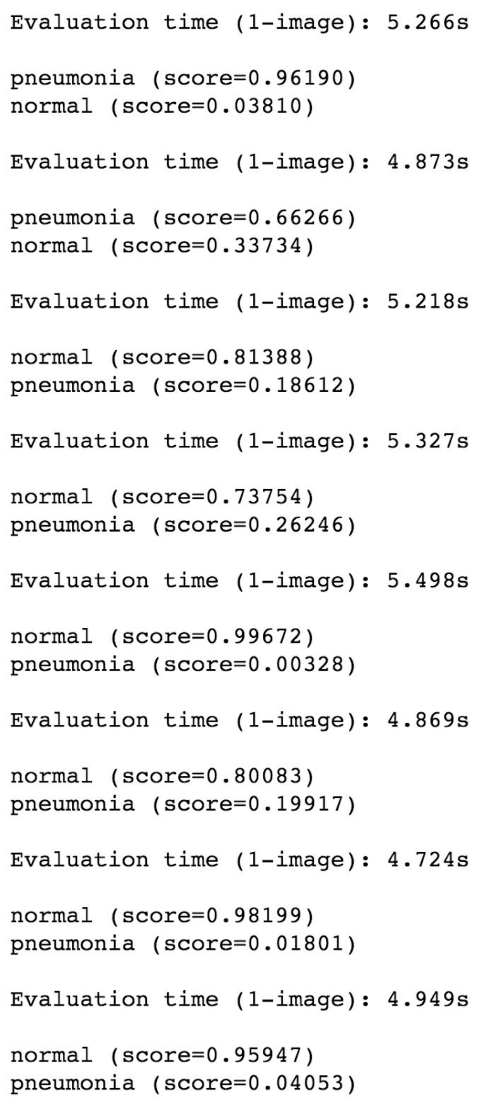


Figure 33. Tensorboard graphs for Inception v3 retraining.

In Figure 33, the tensorboard shows the graph of the validation (eval) in blue and that of the training in orange. Both plots are plotted with the number of training steps completed as the independent variable. In the first plot on the left, it can be seen that the validation and training accuracy increase at a similar rate, at first rapidly then gradually reaching an equilibrium. Although the validation accuracy graph is always slightly below that of the training. It can be concluded from this that there is not a significant amount of overfitting occurring. The second plot on the right shows the value of the cross entropy loss function. Both curves seem to decrease at a similar rate after each training step, first rapidly then eventually reaching some equilibrium at about 0.1. The value of the validation loss function seems to be only slightly higher than that of the training, indicating the overfitting occuring was at an acceptable level.

Finally, the output model was tested on the evaluation data, the 16 images not used in the training which was used as the final criteria for this thesis. Figure 34 shows the results of 8 images of normal chest X-rays.

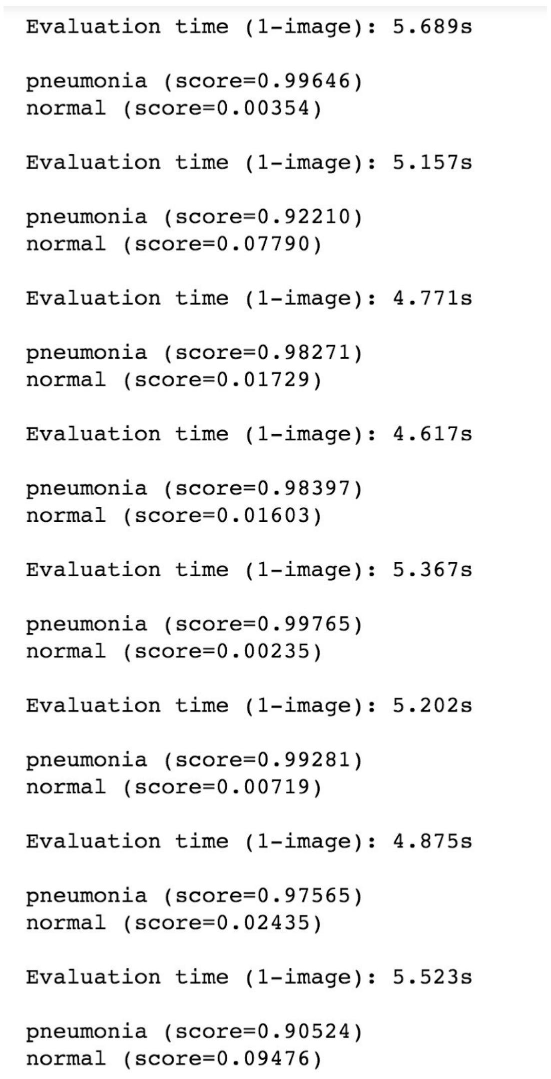


```
Evaluation time (1-image): 5.266s

pneumonia (score=0.96190)
normal (score=0.03810)

Evaluation time (1-image): 4.873s

pneumonia (score=0.66266)
normal (score=0.33734)

Evaluation time (1-image): 5.218s

normal (score=0.81388)
pneumonia (score=0.18612)

Evaluation time (1-image): 5.327s

normal (score=0.73754)
pneumonia (score=0.26246)

Evaluation time (1-image): 5.498s

normal (score=0.99672)
pneumonia (score=0.00328)

Evaluation time (1-image): 4.869s

normal (score=0.80083)
pneumonia (score=0.19917)

Evaluation time (1-image): 4.724s

normal (score=0.98199)
pneumonia (score=0.01801)

Evaluation time (1-image): 4.949s

normal (score=0.95947)
pneumonia (score=0.04053)
```

Figure 34. Evaluation of model retrained on Inception v3 with normal chest X-ray images.

As can be inferred from the figure, 6 out of 8 images were correctly identified which is 75% of the dataset and the percentages of those that were correctly identified as normal are quite high while the other two incorrectly identified outputted a high probability of being infected with pneumonia.

Figure 35 shows the results of the 8 images of chest X-rays of patients with pneumonia.



```
Evaluation time (1-image): 5.689s

pneumonia (score=0.99646)
normal (score=0.00354)

Evaluation time (1-image): 5.157s

pneumonia (score=0.92210)
normal (score=0.07790)

Evaluation time (1-image): 4.771s

pneumonia (score=0.98271)
normal (score=0.01729)

Evaluation time (1-image): 4.617s

pneumonia (score=0.98397)
normal (score=0.01603)

Evaluation time (1-image): 5.367s

pneumonia (score=0.99765)
normal (score=0.00235)

Evaluation time (1-image): 5.202s

pneumonia (score=0.99281)
normal (score=0.00719)

Evaluation time (1-image): 4.875s

pneumonia (score=0.97565)
normal (score=0.02435)

Evaluation time (1-image): 5.523s

pneumonia (score=0.90524)
normal (score=0.09476)
```

Figure 35. Evaluation of model retrained on Inception v3 with pneumonia chest X-ray images.

As seen in Figure 35, 8 out of 8 images were correctly identified as infected with pneumonia with very high probabilities, all certainty of pneumonia being above 90%.

All in all, Inception v3 has not out performed building a CNN from scratch with keras yet since a CNN with 2 convoluted layers and only 1 epoch performed slightly better already although its final validation accuracy was low compared to Inception v3 retrained.

Next, transfer learning process was carried out on another pretrained model, MobileNet v1 and the results from that are compared to the previous models.

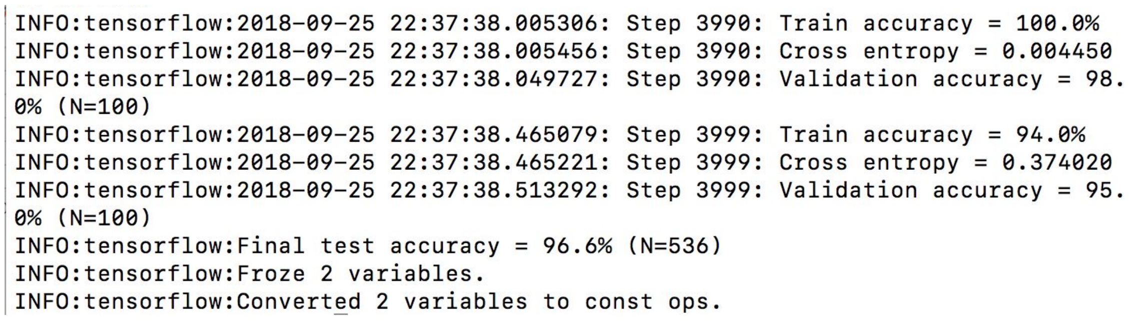
MobileNet v1 Retraining

For the purposes of this thesis, a pre-trained MobileNet_v1_1.0_224 model was retrained with images of chest X-rays from the dataset. This process was similar to the retraining of Inception v3 in the previous section, with the exception of the first script that was run. This script is shown in Listing 10.

```
python retrain.py --architecture="mobilenet_1.0_224" --image_dir <path to
image dir>
```

Listing 10. Command for retraining MobileNet v1.

The bottlenecks were also created similar those generated during the inception v3 retraining and following that, it was trained iteratively with train accuracy, validation accuracy and cross entropy values displayed after each iteration. And finally after the final iteration, the final validation accuracy was also displayed. The final accuracy was 96.6% which is 0.3% better than that returned by Inception v3 retraining. Figure 36 shows this.



```
INFO:tensorflow:2018-09-25 22:37:38.005306: Step 3990: Train accuracy = 100.0%
INFO:tensorflow:2018-09-25 22:37:38.005456: Step 3990: Cross entropy = 0.004450
INFO:tensorflow:2018-09-25 22:37:38.049727: Step 3990: Validation accuracy = 98.
0% (N=100)
INFO:tensorflow:2018-09-25 22:37:38.465079: Step 3999: Train accuracy = 94.0%
INFO:tensorflow:2018-09-25 22:37:38.465221: Step 3999: Cross entropy = 0.374020
INFO:tensorflow:2018-09-25 22:37:38.513292: Step 3999: Validation accuracy = 95.
0% (N=100)
INFO:tensorflow:Final test accuracy = 96.6% (N=536)
INFO:tensorflow:Froze 2 variables.
INFO:tensorflow:Converted 2 variables to const ops.
```

Figure 36. Final test accuracy of MobileNet v1 retraining.

The graph of validation accuracy and cross entropy changes, however, look quite different with lot of fluctuations with quite frequently the graph rising all the way to the top and bottom. This is shown in Figure 37.

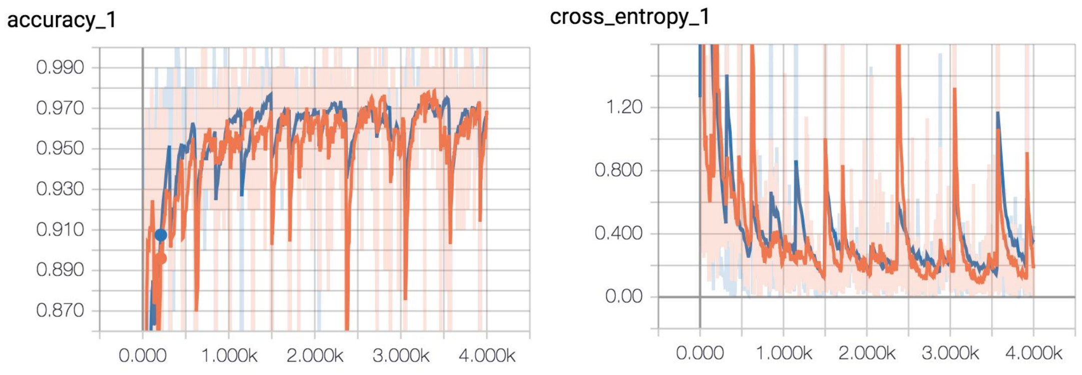


Figure 37. Tensorboard graphs generated for accuracy and cross entropy by training step.

In Figure 37, both graph plots have orange lines corresponding to the training process while the blue line corresponds to the validation process. The left graph plot shows the accuracy over training steps while the right plot shows the cross entropy loss function value over training steps. There were 4000 training steps in total. As can be seen, the accuracy for both the training and validation increase almost at the same rate while increasing rapidly initially and levelling out gradually. However, the training accuracy seems to reduce and increase drastically at few points resulting in long vertical lines on the graph. The validation accuracy does not seem to follow this trend. The cross entropy for both training and validation also decrease at the same rate but the traìning cross entropy value also has few spikes which match those in the accuracy graph. These spikes can be due to some anomaly, but skipping these anomalies the trend of the accuracy and training graphs being almost the same is a positive sign that there is very little overfitting to the data occuring.

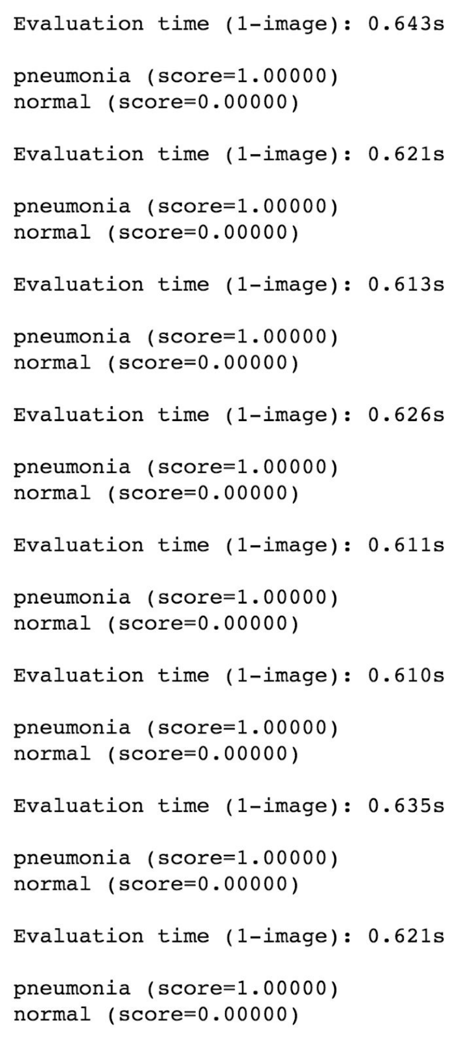
Although the validation accuracy of the retrained MobileNet model has achieved a significantly great accuracy, being the highest value yet, of 96.6%, the final evaluation of the model with the 16 evaluation images reveals really how well the model performs. Figure 38 shows the predictions of this model on the 8 normal evaluation images.

```
Evaluation time (1-image): 0.639s

normal (score=1.00000)
pneumonia (score=0.00000)

Evaluation time (1-image): 0.622s

normal (score=1.00000)
pneumonia (score=0.00000)

Evaluation time (1-image): 0.633s

normal (score=1.00000)
pneumonia (score=0.00000)

Evaluation time (1-image): 0.633s

pneumonia (score=0.95730)
normal (score=0.04270)

Evaluation time (1-image): 0.616s

normal (score=0.99935)
pneumonia (score=0.00065)

Evaluation time (1-image): 0.624s

normal (score=0.99138)
pneumonia (score=0.00862)

Evaluation time (1-image): 0.615s

normal (score=1.00000)
pneumonia (score=0.00000)

Evaluation time (1-image): 0.628s

normal (score=1.00000)
pneumonia (score=0.00000)
```

Figure 38. Evaluation of MobileNet v1 retrained model on normal chest X-rays

As seen in the figure, the model was able to predict 7 out of 8 images with the correct class and has predicted the correct class for all but one with a very high certainty percentage with percentages above 95% and 5 of them predicted with 100% probability. Furthermore, it has predicted them within a fraction of a second, which is about 8 times faster than Inception v3 retrained model.

Next, the pneumonia chest X-ray images were run through the model and the results are shown in Figure 39.



```
Evaluation time (1-image): 0.643s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.621s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.613s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.626s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.611s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.610s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.635s

pneumonia (score=1.00000)
normal (score=0.00000)

Evaluation time (1-image): 0.621s

pneumonia (score=1.00000)
normal (score=0.00000)
```

Figure 39. Evaluation of MobileNet v1 retrained model on pneumonia chest X-rays

As seen in Figure 39, this model has remarkably been able to predict all of the classes correctly with 100% certainty and also within a fraction of a second.

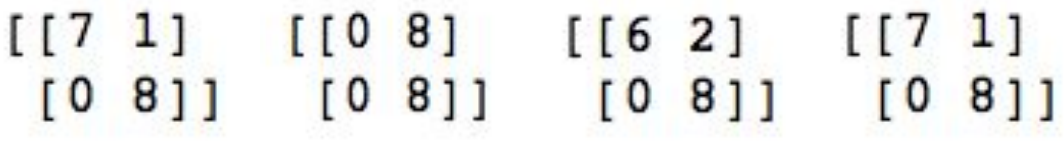
After all the models have been trained and used on new data, the results were analysed and compared against different metrics which is explored in the next section of this thesis.

6 Results and Analysis

This section covers the comparison and analysis in order to create full picture of how the different models performed against each other. This thesis explored two different methods of building a CNN model, either building one or applying transfer learning on an existing generalized pretrained network. There are several ways to measure performance of the models. One way is to make use of a confusion matrix, which is essentially a table with the number of rows corresponding to the number of actual classes and the number of columns corresponding to the number of predicted classes. Hence in this case, it will be a 2 by 2 table. Listing 11 shows how a confusion matrix can be generated with Keras.

```
# Step 1. Prepare the validation images

val_datagen = ImageDataGenerator()

val_dataset = val_datagen.flow_from_directory('data/val',
target_size = (64, 64),
batch_size = 32,
class_mode = 'binary',
color_mode='grayscale',
shuffle=False)

# Step 2. Create an array of predictions

y_pred = classifier.predict_generator(val_dataset)

# Step 3. Get array of correct classes
y_real = val_dataset.classes

# Step 4. Generate confusion matrix report
from sklearn import metrics

confusion_matrix_report = metrics.confusion_matrix(y_real, y_pred)
```
Listing 11. Generating confusion matrix

As can be seen in Listing 11, first the validation images are prepared by resizing to 64 by 64, second the classifier is used to generate predictions based on the validation dataset, third an array of the correct class labels are fetched and finally a confusion matrix is generated from the predicted classes and the actual classes. Figure 40 compares the generated confusion matrix for the classifiers built with Keras for 1 epoch and 10 epoch.



```
[[7 1]     [[0 8]     [[6 2]     [[7 1]
 [0 8]]     [0 8]]     [0 8]]     [0 8]]
```

Figure 40. Confusion matrices for, in order from left: 1 epoch, 10 epochs, Inception v3 retrained, MobileNet retrained.

The first row in each confusion matrix in Figure 40 represents the classes for the normal class (negative class), while the second row represents that for the pneumonia class (positive class). In each confusion matrix, the top left number corresponds to the true negatives, that is the number of images that were correctly classified as normal, while the top right corresponds to the false positives, that is the number of images wrongly classified as pneumonia. The number at the bottom left corresponds to false negatives, that is the number of images wrongly classified as normal while the number at the bottom right corresponds to the true positives, that is the number of images correctly classified as pneumonia (11 pp. 85.) The precision of these models can be calculated as:

$$precision = \frac{TP}{TP + FP} \qquad \text{(11 pp. 85)}$$

TP is the number of true positives,
FP is the number of false positives

As shown in the confusion matrices, the CNN that ran with one epoch had 7 true negatives while only 1 false positive while having no false negatives and 8 true positives. These results are great and the precision is calculated to be 0.89 which is 89% precise.

While the CNN that ran with 10 epochs has no true negatives, 8 false positives, no false negatives and 8 true positives. Hence, the precision is calculated to be 0.5 which is only 50% precise.

For Inception v3 retrained model, there were 6 true negatives, 2 false positives, no false negatives and 8 true positives, Therefore, the precision is calculated to be 0.8 making the model 80% precise according to its confusion matrix.

Finally, the model retrained on MobileNet v1 had the same confusion matrix and therefore the same precision as that of the model built with Keras running 1 epoch. Hence, it had a precision of 89% based on its confusion matrix.

To summarize, Table 1 shows the overview of the performance of each model showing its accuracy, validation accuracy, precision and average time taken to produce one prediction.

Table 1. Summary of the performance of all the models.

Method	Model	Accuracy	Validation Accuracy	Precision	Avg time for a prediction	Time to train
With Keras	1 epoch	99.14%	76.4%	89%	0.003s	47.9 min
	10 epochs	74.28%	62.5%	50%	0.002s	734.5 min
Transfer Learning	Inception v3	96.3%	96.3%	80%	5s	26 min
	MobileNet v1	96.6%	96.6%	89%	0.6s	21 min

As can be seen in Table 1, running 1 epoch with 8000 steps produced 76.4% validation accuracy while acquiring 89% precision taking 3 milliseconds for each prediction. While running 10 epochs produced only 62.5% validation accuracy acquiring only 50% precision taking 2 milliseconds for each prediction. On the other hand, retraining Google's Inception v3 model resulted in 96.3% validation accuracy when run with 4000 steps but was able to attain only 80% precision taking over 5 seconds for each classification. Finally, retraining MobileNet v1 model with 4000 training steps fared much better results with 96.6% final validation accuracy and at the same time attaining 89% precision taking only about half a second for each classification.

Additionally, it can be seen from Table 1 that the time taken to train a 1 epoch model was almost an hour, while the retraining completed within 20 to 26 minutes. On the other hand, running 10 epochs took over 12 hours to complete.

Inferring from these results, it can be concluded that the model built by retraining a pretrained model with MobileNet v1 architecture was best suited for the data since it produced the highest accuracy, validation accuracy, precision and completed training in the least amount of time. Moreover, it produced predictions in a fairly good amount of time, taking about half a second.

7 Conclusion

To summarize, this thesis emphasized the importance of artificial intelligence in the medical field and used the diagnosis of the pneumonia infection with machine learning as a use case of the vast applications of artificial intelligence in healthcare. The research aimed to answer how AI can be used to identify pneumonia in chest X-ray images. The concept of AI and its many subfields were introduced along with the concept of neural networks as a computational replica of the human nervous system created to answer questions. However, it was highlighted that a plain neural network is not enough to classify images due to scalability issues, hence convolutional neural networks which is new class of neural networks with better handling for image data was used.

Two different approaches to creating CNNs were highlighted, first approach was building a whole one, while the other method involved applying transfer learning method on a pre-trained neural network. After training and analyzing the results, a CNN developed from retraining MobileNet v1 model architecture presented the most optimal solution to the problem, generating the highest accuracy, precision and taking the least amount of time to train while producing predictions at a fairly good rate. This model can be used in the real world to identify pneumonia with accuracy of 96.6% and precision of 89%. Therefore the answer to the research question is that retraining a pre trained CNN produces results faster and more accurate than the other option, however the best pre trained CNN architecture should be selected.

All in all, with the applications of machine learning in healthcare rising to new heights in today's world, the tools with which these models can be made are improving and becoming simpler to use, unlocking a multitude of opportunities such as identifying pneumonia in chest X-rays within half a second, a task that would take hours for a physician to infer in the years before.

References

1 Jiang F, Jiang Y, Zhi H, Dong Y, Li H, Ma S et al. Artificial intelligence in healthcare: past, present and future. Stroke and Vascular Neurology. 2017;2(4):230-243

2 Welcome to DeepMind Health | DeepMind [Internet]. DeepMind. 2018 [cited 26 September 2018]. Available from: https://deepmind.com/applied/deepmind-health/

3 IBM Watson for Oncology - Overview - United States [Internet]. Ibm.com. 2018 [cited 20 September 2018]. Available from: https://www.ibm.com/us-en/marketplace/ibm-watson-for-oncology

4 Bell J. Machine Learning: Hands-On for Developers and Technical Professionals. Indianapolis: John Wiley & Sons, Inc.; 2015.

5 Lodha R, Kabra SK, Pandey RM (June 2013). "Antibiotics for community-acquired pneumonia in children". The Cochrane Database of Systematic Reviews. 6 (6): CD004874. doi:10.1002/14651858.CD004874.pub4. PMID 23733365

6 Pneumonia - Symptoms and causes [Internet]. Mayo Clinic. 2018 [cited 26 September 2018]. Available from: https://www.mayoclinic.org/diseases-conditions/pneumonia/symptoms-causes/syc-20354204

7 Pneumonia | Wikisickness [Internet]. Wikisickness. 2015 [cited 27 September 2018]. Available from: http://wikisickness.com/pneumonia.html

8 Pneumonia - Diagnosis and treatment - Mayo Clinic [Internet]. Mayoclinic.org. 2018 [cited 26 September 2018]. Available from: https://www.mayoclinic.org/diseases-conditions/pneumonia/diagnosis-treatment/drc-20354210

9 Normal-Looking Radiographs Can Delay Pneumonia Diagnosis [Internet]. Ebmedicine.net. 2018 [cited 26 September 2018]. Available from: https://www.ebmedicine.net/content.php?action=showPage&pid=18

10 How Should We Define AI? [Internet]. Elements of AI. 2018 [cited 26 September 2018]. Available from: https://course.elementsofai.com/1/1

11 Géron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow. 1st ed. [S.l.]: O'Reilly Media, Inc.; 2017.

12 Maini V, Sabri S. Machine Learning for Humans. 2017.

13 Mayo M. Understanding the Bias-Variance Tradeoff: An Overview [Internet]. Kdnuggets.com. 2018 [cited 6 October 2018]. Available from: https://www.kdnuggets.com/2016/08/bias-variance-tradeoff-overview.html

14 Advanced Neural Network Techniques [Internet]. Elements of AI. 2018 [cited 26 September 2018]. Available from: https://course.elementsofai.com/5/3

15 For Dummies — The Introduction to Neural Networks we all need ! (Part 1) [Internet]. Medium. 2016 [cited 26 September 2018]. Available from: https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb

16 Activation Functions: Neural Networks – Towards Data Science [Internet]. Towards Data Science. 2018 [cited 26 September 2018]. Available from: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

17 Introduction to Neural Networks, Advantages and Applications [Internet]. Towards Data Science. 2018 [cited 26 September 2018]. Available from: https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207

18 Buduma N, Locascio N. Fundamentals of deep learning. 1st ed. Sebastopol, CA: O'Reilly Media, Inc.; 2017.

19 Build a Convolutional Neural Network using Estimators | TensorFlow [Internet]. TensorFlow. 2018 [cited 26 September 2018]. Available from: https://www.tensorflow.org/tutorials/estimators/cnn

20 Convolutional Neural Networks from the ground up – Towards Data Science [Internet]. Towards Data Science. 2018 [cited 26 September 2018]. Available from: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

21 Deshpande A. A Beginner's Guide To Understanding Convolutional Neural Networks [Internet]. Adeshpande3.github.io. 2016 [cited 26 September 2018]. Available from: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

22 Dertat A. Applied Deep Learning - Part 4: Convolutional Neural Networks [Internet]. Towards Data Science. 2017 [cited 10 October 2018]. Available from: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

23 Introduction to Convolutional Neural Networks | Rubik's Code [Internet]. Rubik's Code. 2018 [cited 26 September 2018]. Available from: https://rubikscode.net/2018/02/26/introduction-to-convolutional-neural-networks/

24 Patterson J, Gibson A. Deep Learning. 1st ed. O'Reilly Media, Inc.; 2017.

25 Advanced Guide to Inception v3 on Cloud TPU | Cloud TPU | Google Cloud [Internet]. Google Cloud. 2018 [cited 27 September 2018]. Available from: https://cloud.google.com/tpu/docs/inception-v3-advanced

26 Szegedy C, Vanhoucke V, Loffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision [Internet]. 2015. Available from: https://arxiv.org/pdf/1512.00567.pdf

27 Howard A, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Internet]. Google Inc.; 2017. Available from: https://arxiv.org/pdf/1704.04861.pdf

28 Keras Documentation [Internet]. Keras.io. 2018 [cited 26 September 2018]. Available from: http://keras.io

29 TensorFlow [Internet]. En.wikipedia.org. 2018 [cited 10 October 2018]. Available from: https://en.wikipedia.org/wiki/TensorFlow

30 Chest X-Ray Images (Pneumonia) [Internet]. Kaggle.com. 2018 [cited 26 September 2018]. Available from: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

31 The 7 Steps of Machine Learning – Towards Data Science [Internet]. Towards Data Science. 2018 [cited 26 September 2018]. Available from: https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e

32 Github.com. 2015 [cited 27 September 2018]. Available from: https://github.com/tensorflow/hub/raw/master/examples/image_retraining/retrain.py

33 How to Retrain an Image Classifier for New Categories | TensorFlow Hub | TensorFlow [Internet]. TensorFlow. 2018 [cited 27 September 2018]. Available from: https://www.tensorflow.org/hub/tutorials/image_retraining