Application of Deep Learning on CXR Data to Signal Cases of Pneumonia

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Electrical and Automation Engineering
Valkeakoski

Autumn, 2019

Nurbek Saidnassim and Flokart Blaku

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Electrical and Automation Engineering
Valkeakoski

Authors Nurbek Saidnassim and Flokart Blaku **Year** 2018

Title Application of Deep Learning on CXR Data to Signal Cases of Pneumonia

Supervisor(s) Juhani Henttonen

ABSTRACT

The recent wave of developments in artificial intelligence and data science inspired us to work on this thesis topic that challenged us and developed our skills.

The aim of this thesis was to implement an algorithm that would detect the presence of pneumonia in patients. This project was proposed by The Radiological Society of North America as an open source challenge on the renown data science platform Kaggle. The content of the thesis is mainly focused on the implementation, theory, methodology and results. Additionally, some obstacles that came up during the process are included.

Our efforts resulted in a system with an accuracy of 82% and the designed detector ranked us on the top 15% solutions in Kaggle.

Keywords Pneumonia, Deep Learning, Machine Learning, RSNA, Data Science

Pages 30 pages including appendices 1 page

CONTENTS

TABLE OF FIGURES 3

1 INTRODUCTION 1

2 PRELIMINARY MEDICAL RESEARCH 2

2.1 Pneumonia 2
2.2 DICOM Files 3

3 PROJECT OUTLINE 4

4 METHODOLOGY 5

4.1 Exploratory Data Analysis 5
4.1.1 Data Pre-processing 5
4.2 Biomedical Data Analysis 6
4.2.1 Data Exploration 6
4.3 Machine Learning 7
4.3.1 Data Transformations 8
4.3.2 Model Selection 8
4.3.3 Training 11
4.3.4 Testing 13
4.3.5 Ensembling 13
4.4 Neural Networks 15
4.4.1 Backpropagation and SGD 16
4.4.2 Convolutional Neural Networks 19
4.4.3 Object Detection 21
4.4.4 Regional Models 21
4.4.5 One Shot Models 23

5 IMPLEMENTATION 24

5.1 Classifier 24
5.2 Detector 28

6 RESULTS AND CONCLUSIONS 28

6.1 Possible Improvements 29

REFERENCES 30

APPENDICES 32

Appendices
Appendix 1 GitHub repository

TABLE OF FIGURES

Figure 1 Lungs (American Cancer Society, 2014) 2
Figure 2 DICOM content (Radiological Society of North America, 2018) 3
Figure 3 Trello Board 4
Figure 4 One-Hot Encoding..... 5
Figure 5 Pneumonia DICOM (Radiological Society of North America, 2018)..... 6
Figure 6 Non-pneumonia DICOM (Radiological Society of North America, 2018)..... 6
Figure 7 Homoscedasticity (Fox Jr. & Weisberg, 2011) 8
Figure 8 Logistic Regression (Cournapeau, 2007) 9
Figure 9 Support Vector Machine (Cournapeau, 2007) 9
Figure 10 k-Nearest Neighbours (Cournapeau, 2007) 10
Figure 11 Decision Trees (Cournapeau, 2007) 10
Figure 12 Bayes Theorem (Miller & Childers, 2012)..... 11
Figure 13 Bayes Theorem Visualized (DataMax, 2014) 12
Figure 14 Bayesian Optimization (Pohang University of Science and Technology, 2017) 12
Figure 15 Stacking (Güneş, 2017) 14
Figure 16 Random Forest (Verikas, Vaiciukynas, Parker, Gelzinis, & Olsson, 2016) 14
Figure 17 Gradient Boosting (Rogozhnik, 2016)..... 15
Figure 18 Artificial Neuron (Aleksander & Morton, 1996) 15
Figure 19 Neural Network (Nielsen, 2015) 17
Figure 20 Gradient Descent (Ng, 2018) 17
Figure 21 ANNs vs. CNNs (Karpathy, 2018) 19
Figure 22 CNN architecture (Karpathy, 2018) 19
Figure 23 ReLU activation function (Sarkar, 2018)..... 19
Figure 24 Max Pooling (Karpathy, 2018) 20
Figure 25 Classifier Model 20
Figure 26 Fast RCNN (Hui, 2018) 22
Figure 27 Faster RCNN (Hui, 2018) 23
Figure 28 SSD 23
Figure 29 YOLO 23
Figure 30 Feature Pyramid Network(Hui,2018) 24
Figure 31 File paths..... 25
Figure 32 Classifier functions..... 25
Figure 33 Pre-processing 25
Figure 34 Model definition 26
Figure 35 Training stage 26
Figure 36 Model Results 26
Figure 37 Ensembling code..... 27
Figure 38 Confusion Matrix 27
Figure 39 YOLO setup 28
Figure 40 Ranking in Kaggle..... 28

1 INTRODUCTION

Pneumonia is a disease that affects many people across the globe. Approximately 16% of all deaths under the age of 5, are related to, or caused by pneumonia. Statistics from 2013 shows that 935,000 children of this age group died from pneumonia. From 120 million reported cases of pneumonia, 10% progress to complications. Pneumonia does not affect all age groups equally. Adults have a less risk of a pneumonia diagnosis, but it remains a threat. Older people have a higher risk, and the fatality rate increases if diagnosed with pneumonia. (American Thoracic Society, 2015)

For this thesis, it was very important to examine and understand how pneumonia develops. It was also necessary to be able to distinguish between pneumonia and other similar ailments. This is discussed in more detail further on. The Radiological Society of North America, further on regarded as RSNA, provided a large dataset on the Kaggle platform. The dataset itself contained approximately 30,000 files in the DICOM format. Each file contained the data of the patient and his/her chest x-ray scan. There is a dedicated section for the DICOM format, later in the report. Our task initially was to do exploratory data analysis. That meant, excluding unnecessary data and extracting the crucial data that our system needed. (Radiological Society of North America, 2018)

Additionally, a thorough process of testing and selecting various deep learning methods was a necessity for this project. We decided to split the task in several parts, so that it is more digestible for us and the reader. The two core parts are the classification and detection tasks. The part of the system which classifies the patient image as a pneumonia or non-pneumonia x-ray scan is conventionally called, the classifier. And the detector, specifically locates the pneumonia regions in the x-ray scan. There are limitations to these techniques, we discussed them in-depth and added a section about it.

Considering the computational power required to build a system of this kind, we decided to utilize the free Colaboratory platform that Google provides. Colaboratory is a cloud-based environment that doesn't require users to set it up. The programming language used is Python. Expect to see code snippets in this report, the code is explained in the snippets and throughout this report.

A major challenge was to keep track of the progress and work with self-imposed deadlines. Trello was a big help in managing the timeline of the project and storing all the links that we later source.

2 PRELIMINARY MEDICAL RESEARCH

2.1 Pneumonia

There is no doubt that pneumonia is a very serious and life-threatening disease, but to fully create a solution we need to understand a problem at a deeper level. First, let us understand how pneumonia develops and how it affects the lungs.

When humans breathe, air reaches the lungs, as seen in Figure 1, by flowing down the trachea, then it continues through the bronchi and the bronchioles and finishes in the alveoli. The alveoli are tiny little air sucks that are wrapped up in capillaries. This is where the most of gas exchange occurs in the lungs. Oxygen leaves the air in the alveoli and enters the bloodstream, while carbon dioxide leaves the bloodstream and is exhaled out of the lungs.

In addition to inhaling air, sometimes other stuff is inhaled, like microbes. However, our immune system is very good at protecting our health in these kinds of situations. For instance, our organisms have mechanical techniques like coughing or special microorganisms like macrophages that in the alveoli and ready to protect it from anything.

Interesting things happen when some microbes succeed in colonizing the bronchioles or alveoli. When this happens, you have got pneumonia.

Pneumonia is a disease that is caused by bacterial or viral infection of the lungs. This causes the air sacs to fill up with fluid and substantially affects breathing.

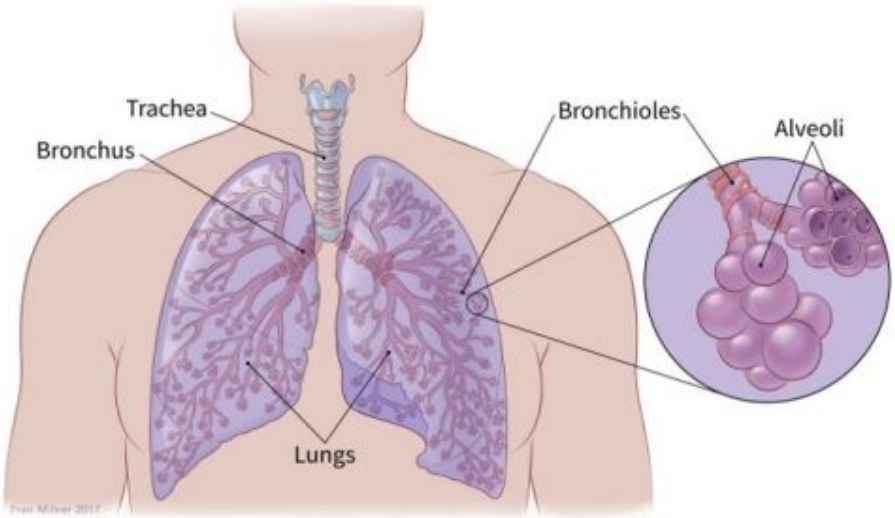

Figure 1 Lungs (American Cancer Society, 2014)

2.2 **DICOM Files**

One of the ways to diagnose pneumonia is to analyze the Chest X-Ray (further on CXR). Kaggle platform provided us with about 30 000 CXR images in dicom format. Now, we need to understand what it is and how to utilize it.

DICOM – Digital Imaging and Communications in Medicine is known as an international standard for medical images and everything that is related to them. DICOM images are known to have high quality, since the diagnosis requires as clear information as possible. Nowadays, this format is implemented in almost all medical domains like radiology, cardiology, radiotherapy devices, ophthalmology and even dentistry.

Since 1993, when DICOM was introduced, it has revolutionized radiology, allowing practitioners to switch from X-Ray films to digital format. (National Electrical Manufacturers Association , 1993)
In Figure 2, you can see the specific data inside a DICOM file provided to us by the radiological community in the USA. It contains several important bits of information, such as the unique patient ID, the view position of the body when the scan was taken, the gender and age of the patient. All this information can be used to further explore the possible solutions to the problem.

```
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.151
7874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.151
7874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.151
7874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: '1'
(0020, 0013) Instance Number                     IS: '1'
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: ['0.14300000000000002', '0.14300000000000
```

Figure 2 DICOM content (Radiological Society of North America, 2018)

3 PROJECT OUTLINE

Due to the framework of this thesis being project-based, it was necessary to make a short description of the tools and techniques used to manage the project. The aim of this section is to suggest a workflow template for future pursuers of similar projects.

Keeping track of deadlines and storing sources was done through an online tool, Trello. Most of the large files and the dataset were stored in Google Drive, to also have an easy access with Colaboratory. Keeping track of deadlines was very simple in Trello. Trello also allowed us to pin stickers on our project board, making the workflow much smoother. The timeline of the project stretched through three months, with the first month being dedicated to research, planning, material gathering and coding. Second month was mainly dedicated to coding and trying as many approaches considered during the first month. During the third month, coding continued along with writing and some finishing touches.

A necessary adjustment that could be added to similar projects in the future, would be to engage more than two or three machines for the model training stage. This would facilitate acquiring better results and would make the overall progress much faster. Considering the 12-hour computational limit on Colaboratory, it is also suggested to consider buying extra computational power.
Our main deadlines in this project were to complete the classifier and the detector. Everything else was built around these two tasks. In Figure 3, you can see a snapshot of our project board from Trello:

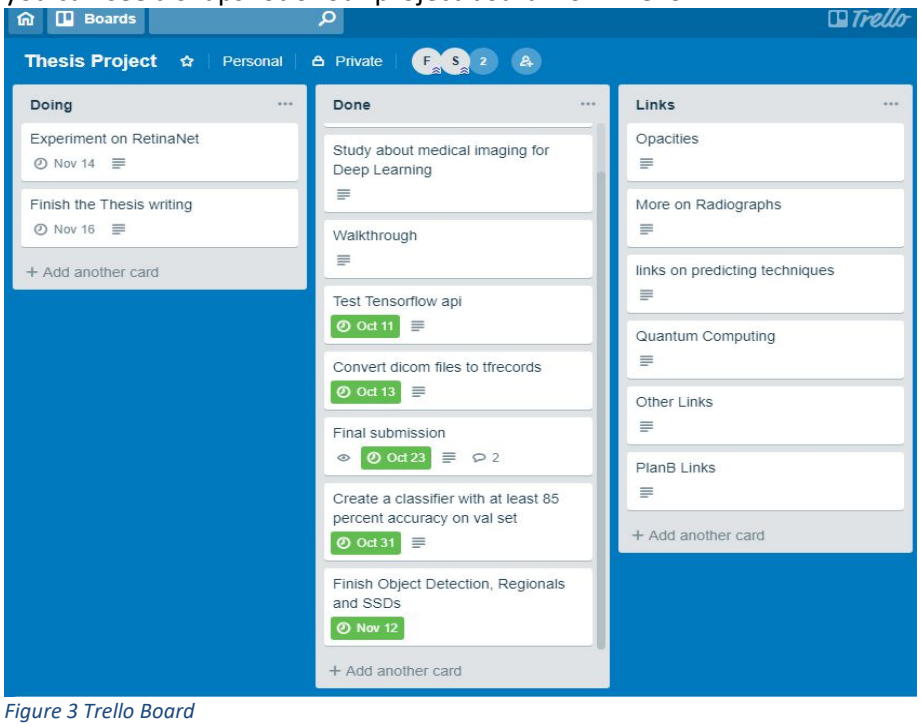

Figure 3 Trello Board

4 METHODOLOGY

4.1 Exploratory Data Analysis

4.1.1 Data Pre-processing

The most important part of data science and machine learning is data and data must be neat and clean for models to process it. Data pre-processing methodologies vary with regards to the data types. It can be tabular data, sequential data (text, music, etc.) or images.

In case of tabular data, there are several essential steps to take care of. The first one is handling missing values. For many reasons like privacy or absence of information in general some data could be missing. There are several techniques to fix that like substituting numerical values (-1, -999, -9999, etc.), deleting the column that has a lot of missing values or assigning some variable to indicate that the data is missing. The next step is to manipulate the categorical data, because some models cannot understand that the variable is represented as a category of some class. There are several techniques like one-hot encoding or label encoding. One-hot encoding, as illustrated in Figure 4, is basically binarization of a class. For example, if there are 5 classes in our dataset (guns, rifles, knives, shotguns and revolvers), each class is represented by assigning 1 to the class and 0 to others.

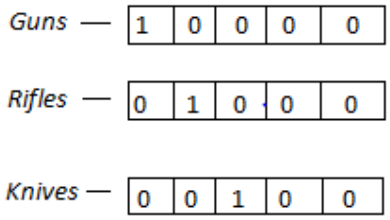

Figure 4 One-Hot Encoding

There are also other steps like handling outliers, which will not be discussed in this thesis.

When it comes to image data, there are other preparation methods. Every state-of-the-art model was built with some assumptions about the data like the size or the number of channels. So, the dataset needs to be resized, or the architecture of the network changed. The image can be resized to get rid of unnecessary information in the background.

When these steps are done, data needs to be analysed using various statistical methods to get some insights. Usually scatter plots and histograms help a lot.

4.2 **Biomedical Data Analysis**

4.2.1 Data Exploration

Now, let us dive deeper into the data. As it has been said earlier, the data comes in DICOM format. The image below is a part of a DICOM file.

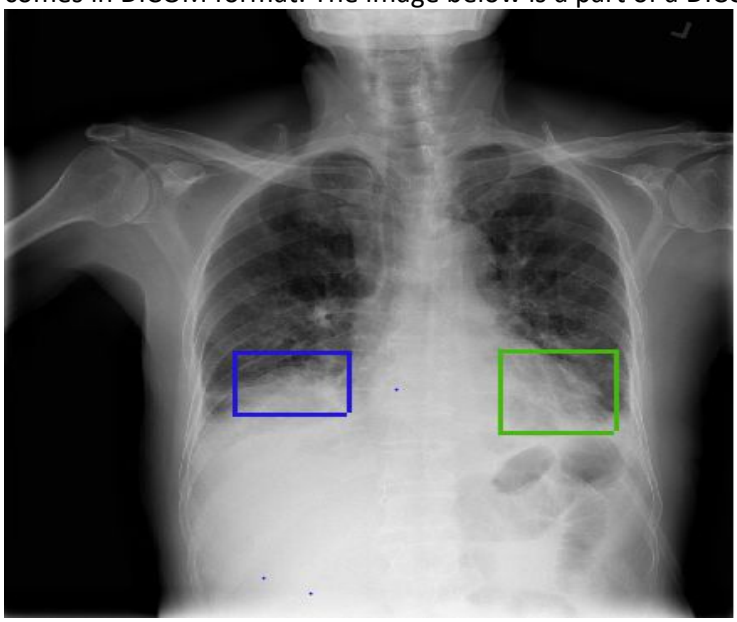

Figure 5 Pneumonia DICOM (Radiological Society of North America, 2018)

In Figure 5, we can see a sample picture of a patient with pneumonia. Knowing how to read CXR helps in understanding the data better. Let us break down the process of X-Ray imaging, X-Ray passes through the body and reaches the detector on the other side. During its journey it encounters matter with different densities, dense materials like bones and tissues absorb the radiation and appear white on CXR, other materials like air do not really absorb X-Ray, thus it reaches the target.

Figure 6 Non-pneumonia DICOM (Radiological Society of North America, 2018)

Briefly speaking:
Air is indicated by the black colour, bones are the white colour, tissues and fluids are grey. Now, we can analyse the pictures above, as we can see the sample patient 1 has pneumonia which is labelled by the bounding boxes, and sample patient 2 has clear and healthy lungs, as illustrated in Figure 6.

In addition to pneumonia, similar opaque regions can be produced by other dense objects, like lung cancer or fluid like water in the lungs. Considering this, there are have 3 classes in the dataset: Lung Opacity, Not Normal and Normal. The target variable can have 2 values, 0 and 1. So it is a binary classification task. However, the positions of bounding boxes must be predicted as well. (Radiological Society of North America, 2018)

4.3 **Machine Learning**

For a long time, people have been solving many problems using conventional programming. Engineers have written a set of commands that a computer needed to execute. Although it is very helpful and proven to be effective to execute repetitive and mundane tasks, it fails to implement tasks that are natural to us, humans. Tasks like vision, understanding natural language or learning are extremely difficult for a computer to master.

For these kinds of problems, new sets of algorithms and techniques are needed. Techniques that will enable computers to mimic and replicate learning. These algorithms are called **Machine Learning algorithms**.

Let us explain why machine learning is needed. Have you ever wondered how YouTube finds videos that you might like or how Facebook recognizes your face on someone else's picture? Machine learning is an integral part of all modern technologies. It can be used to detect cancer or predict the trend of Bitcoin for next several months.

In order to understand the underlying technology better, the most common example will be considered, prediction of iris flower species. First step in machine learning process is data collection. Luckily, **scikit-learn** library already has the necessary dataset pre-packaged. This dataset has 3 classes of iris flower in it, in order to predict the class of the flower based on the given features like sepal length and sepal width. It is practically impossible to accomplish this task with conventional programming, because specifying the code for each case would be necessary. Instead, this data is fed to a machine learning algorithm and it learns the underlying patterns between the features and the target variable. This technology is believed to be the next electricity.

4.3.1 Data Transformations

Machine learning models are based on certain statistical assumptions. One of those assumptions is **normality**. Normality basically implies that the distribution of data points should look like Gaussian distribution. This is very important since several statistical tests rely on this (t-statistics, etc.). Next important assumption is **homoscedasticity**, which means that a set of random variables or a vector have the same finite variance, as shown in Figure 7.

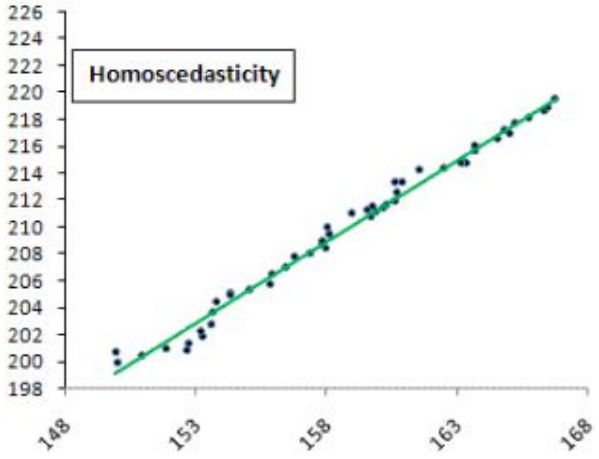

Figure 7 Homoscedasticity (Fox Jr. & Weisberg, 2011)

Linearity is the next assumption, there must be a check for it. As the name suggests, linearity means that the variables exhibit linear pattern on the scatter plot.

In order to meet these requirements, some data transformations must be introduced. For instance, if the distribution shows some skewness, the log transform works quite well.

4.3.2 Model Selection

Now, several models and techniques to choose the right model will be discussed. Machine learning tasks can be broadly classified into three categories: supervised learning, unsupervised learning and reinforcement learning. **Supervised** is when the given data has labels. For instance, in cat pictures there will be a label that it is a cat. **Unsupervised** problems on the other hand have the data without labels. In the case of **Reinforcement learning**, artificial agents are taught to operate in an environment by taking a set of actions and analyzing rewards. We had labelled data, so we dealt with supervised learning. Supervised learning tasks can be split into two types of problems: classification and regression. In classification, some categorical variable like a cat or a dog is predicted, while in regression, the value is predicted, such as the price of a bitcoin. So, in our case we dealt with a supervised classification task.

There are numerous models to choose from, I will briefly explain most common ones. First, let us explain the most basic classification model – **logistic regression**. Logistic regression is a linear model that is used to separate datapoints into two or more groups by applying the sigmoid function and specific loss function, illustrated in Figure 8.

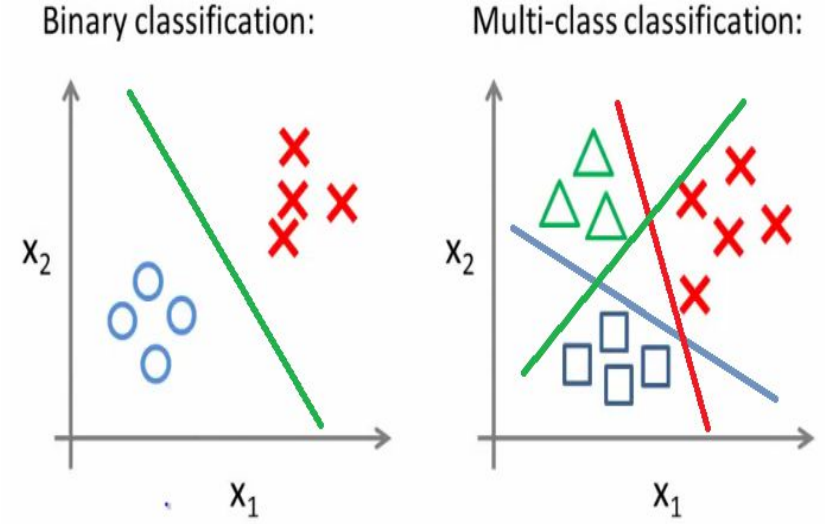

Figure 8 Logistic Regression (Cournapeau, 2007)

The next widely used linear model is **Support Vector Machine** (SVM), illustrated in Figure 9. SVM works by construction one or multiple hyper-planes in a high dimensional space which can be used for classification or regression. The hyper-plane is constructed by optimizing so-called functional margin to decrease generalization error.

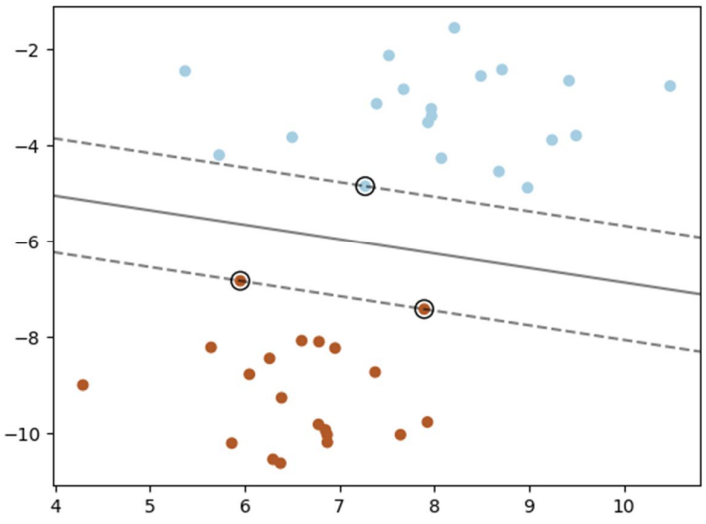

Figure 9 Support Vector Machine (Cournapeau, 2007)

The next algorithm, depicted in Figure 10, is a bit different from the others, k-Nearest Neighbours (KNN) does not construct an internal model or a representation based on data. Instead, it stores the training data and

the classification is performed by a simple majority vote. The data point is assigned to the class it is closest to.

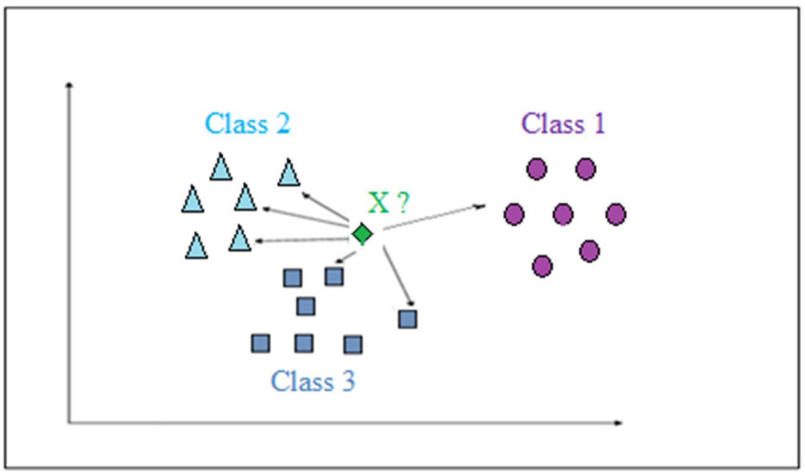

Figure 10 k-Nearest Neighbours (Cournapeau, 2007)

The algorithm shown in Figure 11 is known as **Decision Tree**, which serves as a basis for other very popular techniques, so it is crucial to understand the concept of it. The idea behind decision trees is actually very simple, in order to predict a target value, simple decision rules from the data are learnt. It is helpful to think of it as splitting the plot where the classes need to be separated.

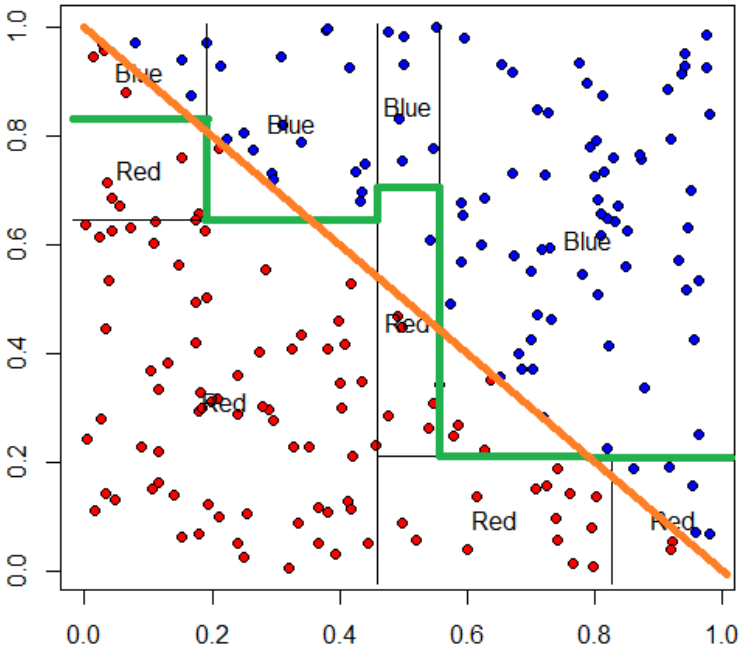

Figure 11 Decision Trees (Cournapeau, 2007)

Sometimes it is better to use simple linear model instead of a decision tree. However, decision tree is used to create powerful ensembles which will be discussed later.

The last model that performs better than any of the above on large amounts of data is the neural network. (Cournapeau, 2007) . It will be explained in a separate section.

4.3.3 Training

When training a model is mentioned, it is always referring to tuning special kinds of parameters called the hyperparameters of each model. There are hyperparameters that are specific to a model like number of hidden layers in a neural net or number of trees in random forest. In addition to that there are hyperparameters that are general for all models like number of epochs, learning rate and an optimizer.

It is important to find an optimal set of hyperparameters in order to get an accurate prediction. Hyperparameter space is enormous, it is wonderful that we have helpful tools and techniques that simplify the searching process. There are 3 common ways to choose the right hyperparameters: Grid Search, Random Search and Bayesian Optimization.

First technique is **grid search**, which is, just brute forcing the parameters. What kind of hyperparameters could be good for a model are defined. Then the algorithm tries the hyperparameters out. This technique is very computationally expensive. The next algorithm is a bit better, **random search** tries random hyperparameters in a range of values that we introduce. This technique does not guarantee that the best result will be achieved, as grid search, but it runs a lot faster. These techniques are not bad, but can we do better? Is it possible to have a model learn the necessary hyperparameters? Yes, thanks to **Bayesian optimization**. This technique is based on Bayes Theorem, which is a way to determine conditional probability. It demonstrates us how to update our prediction given new evidence, it is mathematically expressed in Figure 12.

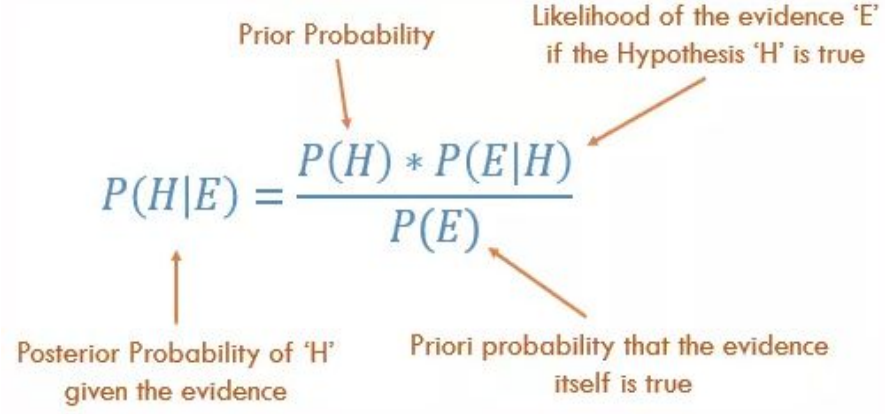

Figure 12 Bayes Theorem (Miller & Childers, 2012)

A helpful way to think about the Bayes theorem is visualized in Figure 13.

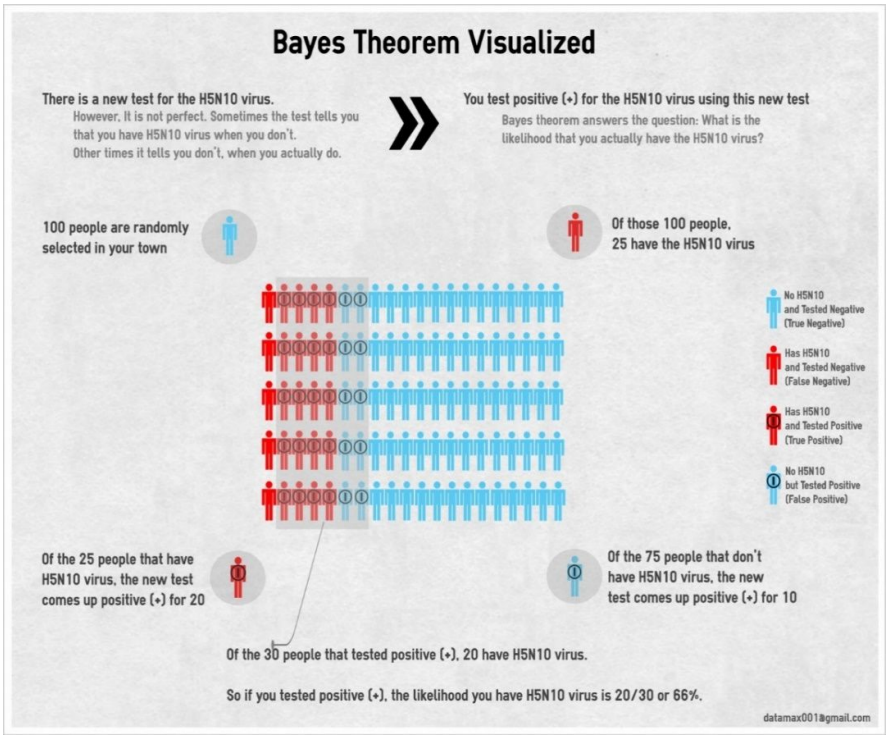

Figure 13 Bayes Theorem Visualized (DataMax, 2014)

Bayesian optimization, illustrated in Figure 14, is built using special method called a gaussian process. Gaussian process is a way of guessing a function based on its inputs and outputs. In addition to that, GP gives you a probability distribution over a set of outputs.

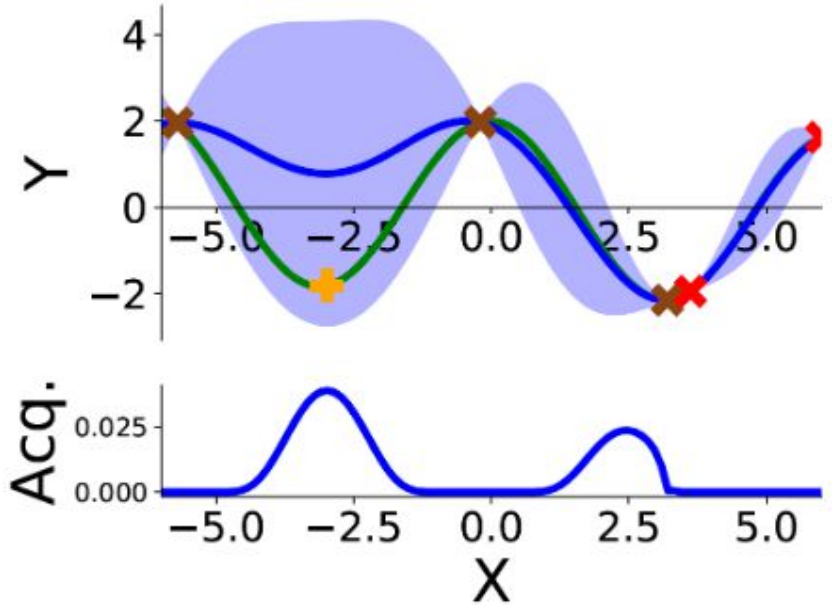

Figure 14 Bayesian Optimization (Pohang University of Science and Technology, 2017)

Bayesian optimization uses previously evaluated points to compute a posterior expectation of the loss, then samples a loss at a new point that maximizes some utility of the expectation (i.e. best regions to sample from). These steps are repeated until convergence.

4.3.4 Testing

Learning the parameters on some dataset then testing the model on the same dataset is a very big mistake. A model that was trained on that dataset would just repeat the labels it has already seen before but will fail to make an accurate prediction on something new. This problem of failing to predict on yet unseen data is known as overfitting. To overcome it, usually the dataset is split into two groups, one for training and one for testing purposes. During testing some estimator like SVM, there is still a risk of overfitting, because an estimator can tweak its parameters until optimal result is achieved. To avoid this, the training set is split again, holding out part of the training set as validation set. This approach usually performs fine when there is a relatively large dataset. However, on small datasets, the training set would also be very small, which would affect the final predictive capabilities.

In this case, another approach called cross-validation (CV) is favored. In its basic form called k-fold CV, a part of the dataset is still held as a test set, but the rest is split into k smaller sets, and following steps are repeated for each of the k folds.

- A model is trained on the k-1 folds.
- A model is validated on the remaining fold.

The performance will then be the average of the values computed in the process. (Cournapeau, 2007)

4.3.5 Ensembling

Multiple dumb trees learn to correct each other. This phrase can almost perfectly explain the essence of ensembling, combining several models that perform a little bit better than coin tossing, increases the accuracy.

It is even better if unstable models that will be dramatically affected by a small change in input are used. That is why decision trees and regression models are often used compared to k-NN and Bayesian models.

There are 3 classic ensembling methods being discussed here: **Stacking**, **Bagging** and **Boosting**.

In case of stacking, multiple different models are trained on the same training data. Then, the output of those models is fed to another model that makes a prediction. It is important to use different models, because, same models trained on the same data is pointless. It is up a developer to choose the models. Regression is frequently used as the final stacking model.

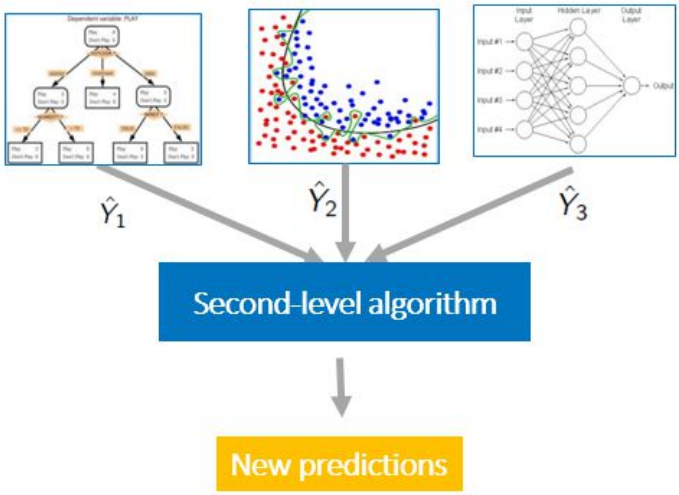

Figure 15 Stacking (Güneş, 2017)

The next algorithm that is frequently used is bagging. The idea behind bagging is very simple. Several sets from the dataset are subsampled and feed them to the several estimators and average their prediction. The most popular bagging algorithm is random forest. Random forest is just a collection of multiple decision trees. When you start your camera and see that a face has been detected. Chances are that random forest is used, since neural nets would have been slow. Bagging's ability to parallelize gives it a huge advantage in most use cases.

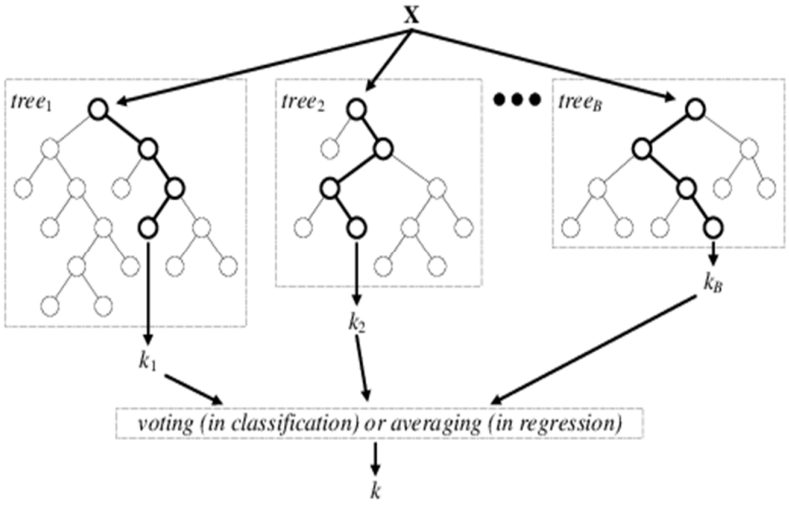

Figure 16 Random Forest (Verikas, Vaiciukynas, Parker, Gelzinis, & Olsson, 2016)

Final technique to be discussed is boosting. Boosting is currently dominating the Kaggle arena, due to its surprising effectiveness. As in any other ensemble method, several models and subsamples from the dataset are used. However, models are ensembled sequentially and subsampling is not random. Instead, the focus shifts on the cases where the previous models made a mistake. Thus, the models are taught to correct each other. The most popular way of boosting is gradient

boosting of decision trees (GBDT). One disadvantage of GBDT is poor scalability due to its sequential nature. For classical machine learning tasks, GBDT is known to be a go-to model to try first, since it is very accurate. There are a lot of popular implementations of this algorithm like: XGBoost, LightGBM and CatBoost.

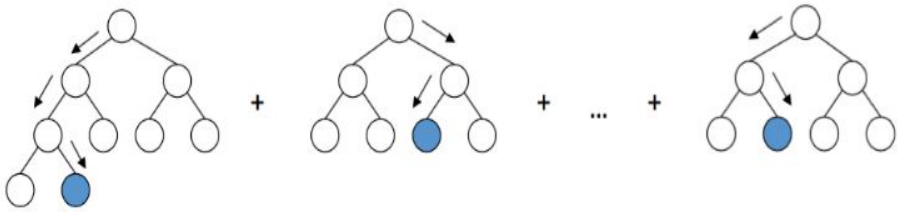

Figure 17 Gradient Boosting (Rogozhnik, 2016)

Gradient boosting basically builds an ensemble of trees and sums up their predictions. For instance, consider some function f(x) to model. And current ensemble is: $E = d_1(x) + d_2(x)$. The next tree tries to minimize the residual (the difference between E and f). Ideally, the result would be: $E + d_3(x) = f(x)$, but this never happens, and adding trees to the ensemble continues. (Cournapeau, 2007)

4.4 **Neural Networks**

The first thing that comes to mind when a person hears the word "neural" is the complex information processing that goes on in brains. It just so happens that Artificial Neural Networks (further referred to as ANNs) are inspired by the way our nervous system works. A very common example used to explain ANNs in most books and instructional materials, is the digit recognition system. Digit recognition is an unconscious task for people, yet it is a very complex task that engineers have managed to perfect only in recent years. Digit recognition systems are so reliable that banks and postal services use them.

The way the neurons, connections between neurons and the nervous system as a unit works is rather complex and not fully understood as of today. The first artificial neuron, illustrated in Figure 18, was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. (Stregiou & Siganos, 1996)

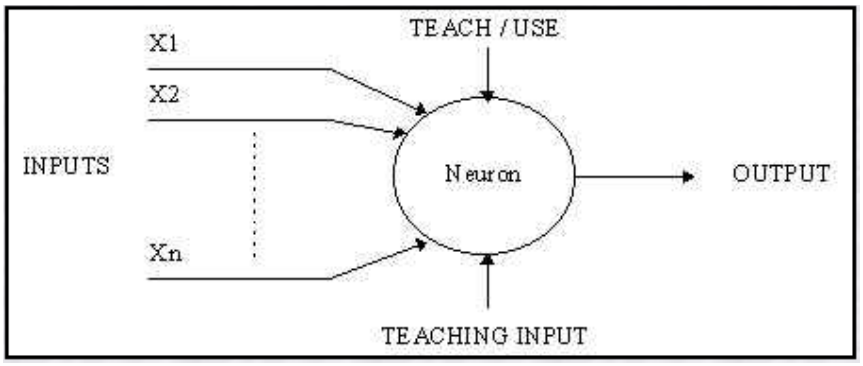

Figure 18 Artificial Neuron (Aleksander & Morton, 1996)

The above diagram represents the way a simple neuron works. It has several inputs and gives a single output. The neuron has two states or modes. The 'TEACH' mode is used to train the neuron to give an output if a specific combination of input values is fed in. This is known as a neuron "firing" if the right combination is in the input. In the 'USE' mode, the neuron expects the taught output and fires. If the combination is different from the expected values, a specific set of firing rules is used to determine if the neuron fires or not. (Aleksander & Morton, 1996)

An advanced neuron model has weighted inputs and the neuron fires if the threshold values are exceeded. Two very common architectures are the feed-forward and feedback network. A few decades of further development, and the concept of a perceptron came to notion. This is a modified version of the neuron with weighted inputs, with additional capabilities of extracting specific features from input data.

The issue with binary outputs is that small changes in weights and inputs can make the neural network behaviour hard to control. The solution for this is to use the Sigmoid function. This function lets the neurons map the output to the values between 0 and 1, for example 0.32.
And the output is defined as σ(w·x+b), where σ is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The whole equation for the sigmoid output, including weights and bias is:

$$\sigma(z) = \frac{1}{1 + \exp(-\Sigma_j w_j x_j - b)}$$

Combining multiple neurons in layers and connecting layers with each other gives us artificial neural networks. Following up this brief intro, two very important concepts in neural networks will be explored.

4.4.1 Backpropagation and SGD

The artificial neural network in Figure 19 is one that is used to recognize digits from the MNIST dataset. This dataset contains 60,000 images for training purposes and 10,000 for the testing phase. Each image is a greyscale and with a size of 28x28 pixels. Observing from the right to the left, there is the output layer with 10 output neurons for each digit (0…9). Then, there is the hidden layer, that is neither an input or output layer. Each neuron in the hidden layer fires when it detects a specific pattern of pixels. The input layer consists of 784 neurons. (Nielsen, 2015)

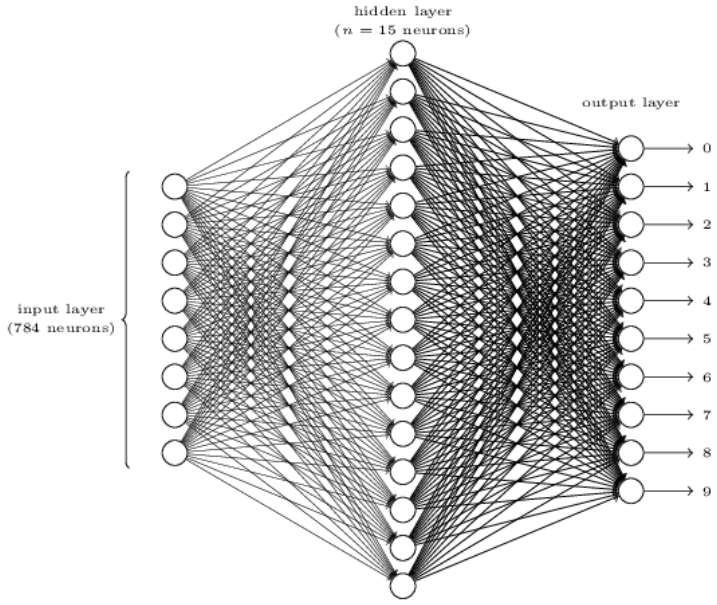

Figure 19 Neural Network (Nielsen, 2015)

Let's say that our goal is to recognize the digits individually in an image consisting a sequence of numbers. The way this problem is tackled is by using a segmentation method and then classifying the digits with the neural network above. The next step is to define a loss function that will let us know how well our algorithm is performing the task of finding the right weights and biases that will approximate the desired output for a given input. It is sometimes referred to as the *cost* or *objective* function.

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

The goal here is to make this loss function, C (w, b), as small as possible using the gradient descent algorithm. Considering the large amount of inputs for the network mentioned above, randomizing the weight of the input neurons will give us a completely random output. Consider the graph in Figure 20 to easier understand how the gradient descent will help.

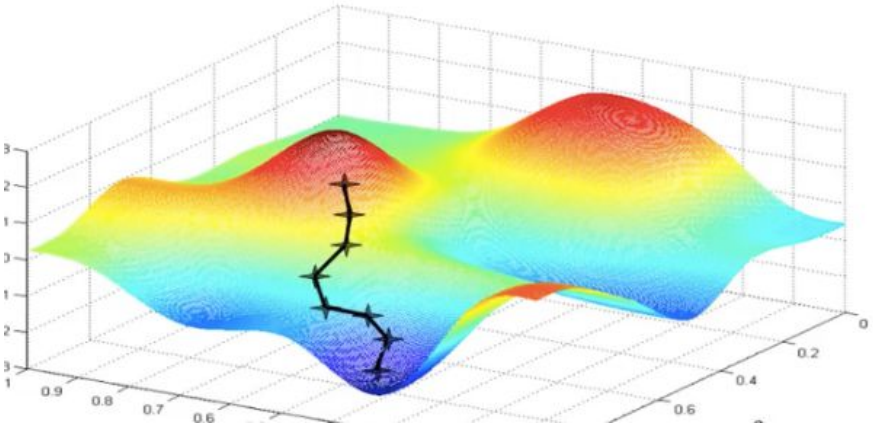

Figure 20 Gradient Descent (Ng, 2018)

A loss function is graphed above with respect to the inputs and outputs of the network. To decrease the loss function, it is vital to determine which way the weight value must descent as quickly as possible to decrease the loss. From multi-variable calculus, the “gradient” of a function gives the steepest increase. Thus, the negative of the gradient will let the function reach a local or global minimum as the highlighted pathway shows. Now to clarify, the graph above is the representation of a simple function. The digit recognizing example has thousands of weights and visualizing it, is hard to do. However, the principle remains the same, the average loss function of the network gets reduced and the system classifies the digits better and better. (Goodfellow, Bengio, & Courville, 2016)

It must be established that the gradient descent values for each weight are stored as a vector, and the magnitude of each value determines how important to the system for recognizing a digit. The sign determines if the value of the weight should decrease or increase.

$(W^{\cdots\rightarrow})$ = [w_0
w_1
w_2
$w_{\ldots}$
$w_{13,001}$
$w_{13,002}$]

$-\nabla C(W^{\cdots\rightarrow})$ = [0.32 w_0 must increase
0.02 w_1 must increase
-0.42 w_2 must decrease
…
-1.2 $w_{13,001}$ must decrease
2.04 $w_{13,002}$ must increase]

Now the back-propagation algorithm produces the changes desired from a single training example to the weights and biases of the network. For example, the network is trained for detecting the digit ‘4’ from a training image, random weights and biases are initially assigned to each neuron. Thus, using back propagation the desired change for the weight and bias of each neuron on the preceding layer according to our output layer is found. And then to totally complete the back-propagation process, the same calculations through each layer from output to input are made. And to complete the gradient descent, this same procedure for each training example is required.

This would be rather slow in application. That is where the stochastic gradient descent comes in. This is the most computationally cheap way of getting the gradient descent required. It works by sub dividing the training set on random batches, computing the descent gradient step for all the data according to a single batch. Using this technique makes the training much faster and retains the needed effectivity for the digit recognition system. (Nielsen, 2015)

4.4.2 Convolutional Neural Networks

The difference between Convolutional Neural Networks (referred to as CNN onwards) and Neural Networks is that the hidden layer in the latter is replaced by three-dimensional layers, as depicted in Figure 21. These dimensions are: width, height and depth. There is also a change in the connection of neurons of a layer to the next layer. Originally, each neuron is connected all the neurons of the next layer, in CNNs the neurons are connected to only a few of the neurons from the next layer. The output of the network is organized in a vector containing probability results on the depth dimension.

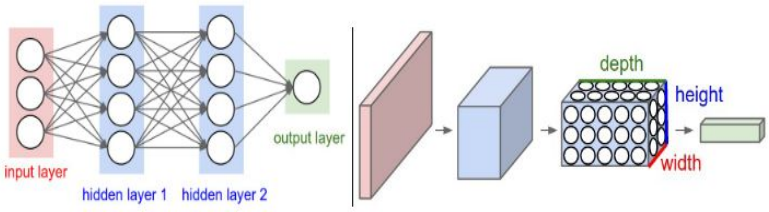

Figure 21 ANNs vs. CNNs (Karpathy, 2018)

The term convolution means combining two mathematical functions to produce a third function. Together with the pooling and fully-connected layers, the convolutional layer is a building block of a CNN.
The key point in a convolution layer, is the filter that will be used over each raw pixel of the input layer. It will also perform matrix multiplication and summation. This will all be stored in a feature map. This procedure is commonly known as feature extraction. (Karpathy, 2018)

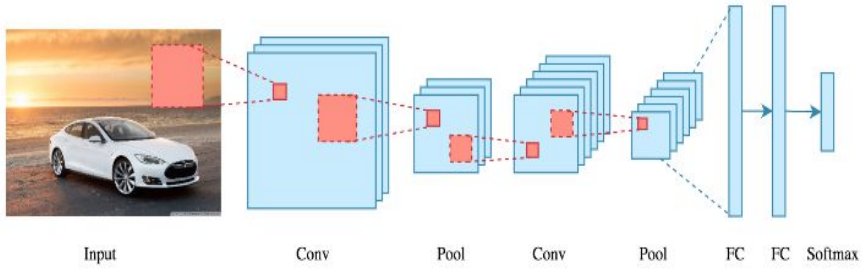

Figure 22 CNN architecture (Karpathy, 2018)

The results from the convolution operation illustrated in Figure 22 are passed through an activation function that is usually ReLU, this makes sure the network achieves its full potential. The ReLU activation function is graphed in Figure 23.

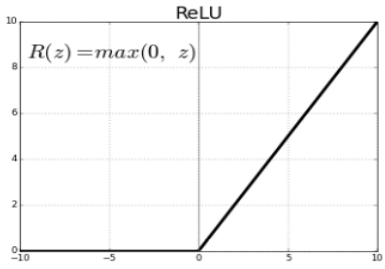

Figure 23 ReLU activation function (Sarkar, 2018)

Two more terms that must be discussed are stride and padding. Striding is the movement of the filter across the input pixels. The padding value is set at 1 by default. Increasing the padding results in a smaller feature map. If the user wants to maintain the same dimensions of the input, for the feature map, padding can be added to the input. It works by adding zeroes around the input pixels.
The pooling layer comes is applied after the convolution layer. Its function is to reduce the width and height dimensions in order to cut down on training time and reduce overfitting. A very common pooling layer is the max pooling. The aim here is to take only the maximum values from the input. This results in a layer with a reduced width and height as visualized in Figure 24.

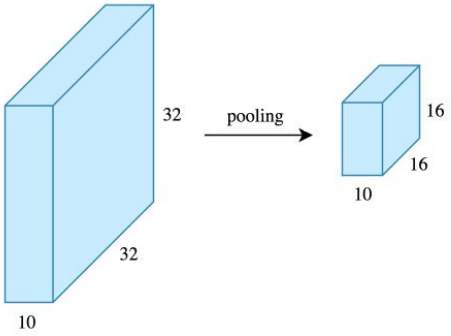

Figure 24 Max Pooling (Karpathy, 2018)

A quick recap for this part, convolutional neural networks are mainly based on feature extraction and are combined with fully connected layers for classification of desired objects or shapes. Through pooling and filtering the most import features are extracted, and the excessive data is cut out. To follow up, consider the python implementation of a CNN in Keras, in Figure 25.

```
1   model = Sequential()
2   model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
3                    input_shape=(150, 150, 3)))
4   model.add(MaxPooling2D((2, 2), name='maxpool_1'))
5   model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
6   model.add(MaxPooling2D((2, 2), name='maxpool_2'))
7   model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
8   model.add(MaxPooling2D((2, 2), name='maxpool_3'))
9   model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
10  model.add(MaxPooling2D((2, 2), name='maxpool_4'))
11  model.add(Flatten())
12  model.add(Dropout(0.5))
13  model.add(Dense(512, activation='relu', name='dense_1'))
14  model.add(Dense(128, activation='relu', name='dense_2'))
15  model.add(Dense(1, activation='sigmoid', name='output'))
```

Figure 25 Classifier Model

On the code above, the only two concepts that have not been discussed are the Dropout and Flatten functions in line 12 and 11. Flattening is done after the convolution and pooling layers to make the output ready for the fully connected layers of classification.

The dropout is a technique used on most state-of-the-art deep learning applications nowadays. The idea here is to drop random neurons across the layers in order to increase the adaptability of the network. The dropping out of neurons happens during training and it is temporary. Using dropout usually increases the performance by about 2% and reduces overfitting.

Another extra step to increase the performance of models is to augment the data. This extra step can increase the amount of training data considerably. The idea is to flip, crop, zoom in and rotate each image and use the result as a new image on the training set.

4.4.3 Object Detection

Although, it is very helpful to know what class image belongs to, sometimes users also need to know the exact position of an object on a picture. The problem of localizing an object in an image is known as object detection.

There are numerous reasons to develop methods for detection: localizing brain tumor on MRI images or face detection for cameras. The first object detection algorithm was introduced in 2001 by Paul Viola and Michael Jones. The algorithm was based on the problem of face detection, but it can be trained on variety of objects. The algorithm is able to detect faces in real-time on a webcam. Now, this algorithm is implemented in the image processing library OpenCV.

After that in 2005, Dalal and Triggs came up with the new procedure for object detection that they named Histograms of Oriented Gradients (HOG). This algorithm calculates respective gradient for each pixel in an image, then creates 16 by 16 representation based on those gradients. Face can then be detected using that representation, most similar part of the image will be detected as a face.

In 2012, deep learning entered the arena of object classification and became the leading algorithms among all. One of the ways to perform object detection using CNNs is to run a kernel across all parts of an image then try to classify each part. Obviously, this approach is very ineffective, since it would take a lot of time to perform this operation.

There are better techniques and algorithms like Regional and Single shot models, that will be discussed further on.

4.4.4 Regional Models

As it has already been pointed out, it is not efficient to use a sliding window approach. Instead, something called **selective search** can be

used. Selective search is basically a grouping method, single pixels are initiated in their own group, then two groups that have similar textures are merged. This process is repeated until everything is combined. At the end selective search generates so-called **regions of interest (ROI)**.

Regional Convolutional Neural Nets (R-CNN) take advantage of selective search technique to generate about 2000 regions of interest. The regions are then warped into individual images and fed into a CNN. Once the features are extracted, the region can be classified using a classifier like SVM and a bounding box can be refined using the regression technique. While better than HOG, this technique is still very slow as it processes all ROIs separately. Something faster is required. The downside of R-CNN is that for each region proposal, the network extracts features separately, it takes a lot of time. (Girshick, Donahue, Darrell, & Malik, 2013)

In Fast R-CNN, instead of extracting features one by one, it is done only once for the whole image. After that, the region is proposed using selective search and applied to the extracted feature map. Then, the final patches are passed through ROI pooling to resize them. Finally, the classifier and regressor are applied as usual. (Girshick, Fast R-CNN, 2015)

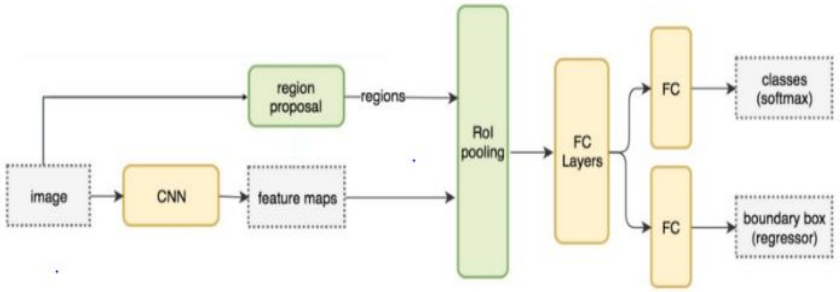

Figure 26 Fast RCNN (Hui, 2018)

This is 150 times faster than regular R-CNN, when it comes to inferencing and 10 times faster in training.

Fast R-CNN is good, but can it get better? There is a certain way to achieve the desired objective using Faster R-CNN. Faster R-CNN replaces the regular region proposal technique by a deep network and the feature maps are used to generate ROI. The region proposal network achieves high accuracy and speed compared to selective search, making it much more efficient. (Shaoqing , He, Girshick, & Jian , 2015)

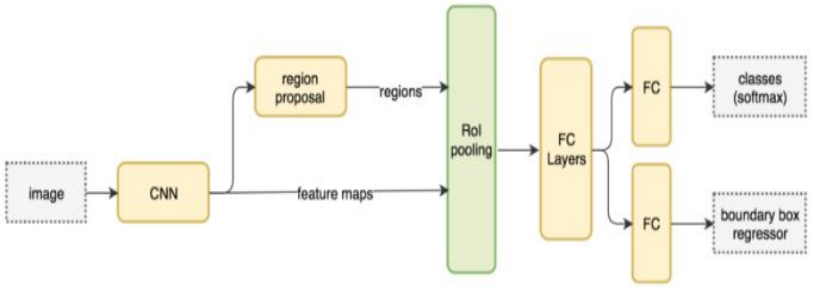

Figure 27 Faster RCNN (Hui, 2018)

4.4.5 One Shot Models

Faster R-CNN is highly accurate, but inefficient for detecting objects in real-time. For those tasks, another algorithm that will speed up the process, must be developed. Luckily, there is something exactly for those purposes. One of the algorithms is Single Shot Detector (SSD), SSD uses VGG19 architecture as a feature extractor. In addition to that, custom convolutional layers are added, and a convolutional filter to make predictions as shown in Figure 28.

Figure 28 SSD

One disadvantage of convolutional layers is that they reduce the dimensions or down-sample the input, which leads to inability to detect small objects. This problem can be solved by adding a filter and making predictions after each convolutional filter. In addition to that it is helpful to use high resolution images for more accurate predictions. (Wei, et al., 2015)

The algorithm depicted in Figure 29 concentrates on the speed of detection is known as YOLO. It uses darknet as a backbone feature extractor follow by 3 convolutional layers, then the output of that layer is concatenated with the output of the darknet and passed to another convolutional layer that is used for prediction.

Figure 29 YOLO

There are many improvements implemented in new versions of YOLO: YOLOv3, YOLO9000, etc.

The new backbone, darknet-53, is used in YOLOv3. In addition to that it uses a technique called feature pyramids to detect smaller objects more accurately. (Redmon, Divvala, Girshick, & Farhadi, 2015)

Let us give a brief overview of what feature pyramid networks (FPN) are. Basic intuition behind the FPN architecture is shown in Figure 30.

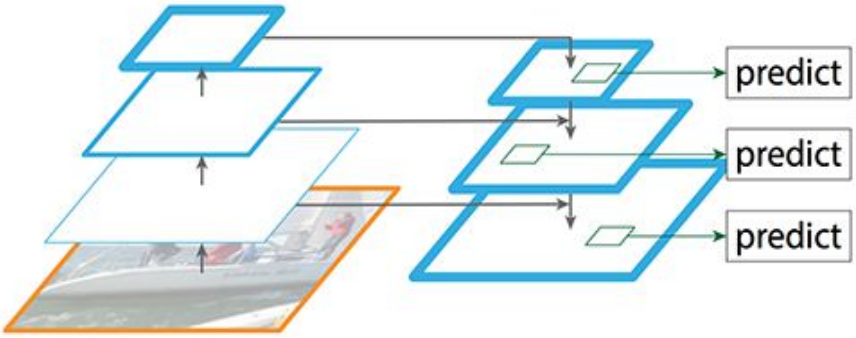

Figure 30 Feature Pyramid Network (Hui, 2018)

As you can see, the FPN includes the top-down pathway in addition to regular CNN architecture, which helps to improve the accuracy of localization. FPN is not a detector by itself, it acts a feature detector. So, it needs to be used with other detectors, such as Faster R-CNN. However, one model that incorporates the FPN technique performed as fast as SSDs and more accurate than Faster R-CNN. This network is known as RetinaNet. (Lin, et al., 2016)

5 IMPLEMENTATION

As mentioned in earlier chapters, our approach was to run the data through a classifier and then through a detector which would make pneumonia regional predictions for the patient. Splitting these two tasks simplified the project pipeline and reduced the computational time required otherwise. Further on, consider the classifier code.

5.1 Classifier

The data was arranged into folders according to the class of each patient. The amount of data we had for each class was as follows:

Pneumonia patient files (Target 1) – 5659 files
Normal patient files (Target 0) – 4830 files
No Pneumonia/Not Normal patient files (Target 0) – 11500 files

In total, there were 21,989 files for the training directories. The rest of the data was split equally between the testing and validation folders. In figure 31, there is a snapshot of the code for the file paths (full code included in the link Appendix 1):

```
import os

train_dir_normal = "../content/drive/My Drive/models/research/object_detection/images/train/Target_0_PNGs/*.png"
train_dir_not_normal = "../content/drive/My Drive/models/research/object_detection/images/train/Target_0NN_PNGs/*.png"
train_dir_pneumonia = "../content/drive/My Drive/models/research/object_detection/images/train/Target_1_PNGs/*.png"
val_dir_normal = "../content/drive/My Drive/models/research/object_detection/images/train/VAL_norm/*.png"
val_dir_pneumonia = "../content/drive/My Drive/models/research/object_detection/images/train/VAL_pneu/*.png"
test_dir_normal = "../content/drive/My Drive/models/research/object_detection/images/train/new NORMAL/*.png"
test_dir_pneumonia = "../content/drive/My Drive/models/research/object_detection/images/train/new PNEUMONIA/*.png"
%matplotlib inline
```

Figure 31 File paths

The paths are long because our data was stored in GoogleDrive. In figure 32, there are a few snippets from the pre-processing functions:

```
def shuffle_in_unison(a,b):
  assert len(a) == len(b)
  p = np.random.permutation(len(a))
  return a[p], b[p]
```

```
def convert_to_one_hot(vec, num):
    Y = np.eye(num)[vec.reshape(-1)].T
    return Y
```

Figure 32 Classifier functions

These two functions in Figure 32, shuffle and convert the data to One-Hot format. The next part of the code is from the **load_data** function, which resizes the pictures. The complete function can be found in the repository in Appendix 1:

```
for num, file in enumerate(val_files_normal + val_files_pneumonia):
    img = cv2.imread(file, 1)
    img = cv2.resize(img, (200,200))
    val_data.append(img)
    if(num+1 <= 250):
        val_labels.append(0)
    else:
        val_labels.append(1)
```

Figure 33 Pre-processing

Following the definition of the functions, the training model is imported. In our case, we used VGG16 model with pre-trained weights from the ImageNet dataset:

```
base_model = applications.VGG16(include_top = False, weights = 'vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5')
```

```
sgd = optimizers.Adam(lr = 1e-4)
```

```
# now we build a function to train the model and add a bottleneck model to our pre-trained one
def Train_model(X_train, Y_train, X_test, Y_test, optimizer, base_model, batch_size = 64, num_epochs = 10):

    X = base_model.output
    X = Dropout(0.5)(X)
    X = GlobalAveragePooling2D()(X)
    X = Dense(128, activation='relu')(X)
    X = BatchNormalization()(X)
    X = Dense(2, activation='sigmoid')(X)
    model = Model(inputs = base_model.input, outputs = X)

    for layer in base_model.layers:
        layer.trainable = False

    model.compile(optimizer = optimizer,
                  loss = 'categorical_crossentropy',
                  metrics = ['accuracy'])

    model.fit(X_train,Y_train,
              batch_size = batch_size,
              epochs = num_epochs,
              validation_data = (X_test,Y_test))

    return model
```

Figure 34 Model definition

At this point, the arguments were double checked, and we mostly used the default settings from the starter code (Appendix 1). The final stage was to train and save our results.
The training stage is depicted in Figure 35.

```
[ ]  model = Train_model(X_train, Y_train, X_test, Y_test, sgd, base_model, batch_size = 64, num_epochs = 10)
     model.save('model.h5')
```

```
Train on 9543 samples, validate on 500 samples
Epoch 1/10
9543/9543 [==============================] - 90s 9ms/step - loss: 0.5268 - acc: 0.7435 - val_loss: 0.6773 - val_acc: 0.6860
Epoch 2/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3910 - acc: 0.8374 - val_loss: 0.7206 - val_acc: 0.7020
Epoch 3/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3561 - acc: 0.8498 - val_loss: 0.5792 - val_acc: 0.7620
Epoch 4/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3389 - acc: 0.8584 - val_loss: 0.5627 - val_acc: 0.7700
Epoch 5/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3254 - acc: 0.8651 - val_loss: 0.4926 - val_acc: 0.8040
Epoch 6/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3178 - acc: 0.8693 - val_loss: 0.4639 - val_acc: 0.8020
Epoch 7/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3128 - acc: 0.8719 - val_loss: 0.4868 - val_acc: 0.8080
Epoch 8/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3094 - acc: 0.8711 - val_loss: 0.4380 - val_acc: 0.8060
Epoch 9/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3005 - acc: 0.8747 - val_loss: 0.4533 - val_acc: 0.8120
Epoch 10/10
9543/9543 [==============================] - 79s 8ms/step - loss: 0.3048 - acc: 0.8748 - val_loss: 0.4932 - val_acc: 0.7980
```

```
[ ]  s_model = model.save('model.h5')
```

Figure 35 Training stage

In this instance we achieved an accuracy of approximately 80% in classification. This training was done without using the No Pneumonia/Not Normal class. We trained several models with our data and the results for each model are displayed in Figure 36:

Model/Method	Validation Loss	Validation Accuracy
VGG16	0.445	0.806
VGG19	0.5478	0.758
InceptionV3	0.556	0.673

Figure 36 Model Results

The top performer at 80.5% accuracy is the VGG16 model, followed up by VGG19 and Inception. Furthermore, the 'ensembling' technique came into use. The code in the Figure 37 implements the technique itself and creates a confusion matrix.

```
[25] #loading pre-trained models

     vgg19 = load_model('VGG19.h5')

     vgg16 = load_model('VGG16.h5')

     inception = load_model('inception.h5')
```

```
[39] # summing all the predictions from each model
     output = vgg16.predict(X_test) + vgg19.predict(X_test) + inception.predict(X_test)
```

```
[49] ensemble_pred = np.exp(output/3)-1
```

```
#confusion matrix
ensemble_pred = np.argmax(ensemble_pred, axis = 1)
y_true = np.argmax(Y_test, axis = 1)
ens_conf_mat = confusion_matrix(y_true, ensemble_pred)
ens_conf_mat
```

```
array([[200,  50],
       [ 40, 210]])
```

Figure 37 Ensembling code

The confusion matrix, seen in Figure 38, visualizes the accuracy of our models over our test data. A total prediction rate of 82% was achieved through ensembling. The values in the black boxes indicate the misclassified data points.

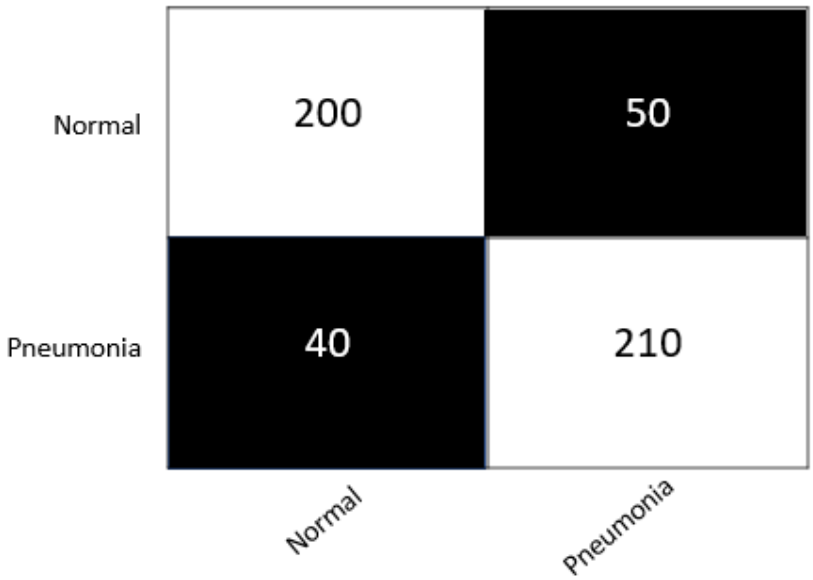

Figure 38 Confusion Matrix

5.2 **Detector**

The next part in our project was building a detector. In order to prototype fast, we chose to implement a YOLO detector.

First, we will need to clone the YOLO repository and to build it. This could be done using the script illustrated in Figure 39.

```
!git clone https://github.com/pjreddie/darknet.git

# Build gpu version darknet
!cd darknet && sed '1 s/^.*$/GPU=1/; 2 s/^.*$/CUDNN=1/' -i Makefile

# -j <The # of cpu cores to use>. Change 999 to fit your environment.
!cd darknet && make -j 999 -s
!cp darknet/darknet darknet_gpu
```

Figure 39 YOLO setup

Since the provided data format was not supported by YOLO, we pre-processed it (Details are in project's GitHub repository).

After pre-processing was done, we needed to prepare various configuration files for our project. These files were: data paths and names of classes, a pre-trained model and a training file. All the scripts were provided in the repository. We used this detector and tested it on Kaggle dataset and got the following result in the leaderboard, seen in Figure 40.

Figure 40 Ranking in Kaggle

This result corresponded to the 213th position among 1499 contestants.

6 **RESULTS AND CONCLUSIONS**

This project gave us a lot of new concepts to digest and a lot of experimenting with code. Overall, it was a really good experience and helped us improve fast over a short period of time. Adequate results were achieved through experimenting on different models and machine learning techniques. Throughout the project, we overcame several obstacles. Most noticeably, hardware and software limitations.
To sum it up, it would be fair to say that deep learning technologies have opened paths for engineers and scientists to attempt to solve global problems in areas such as healthcare and medicine. This was a challenging and exciting problem and we will certainly experiment more and try new ideas in order to get better performance. Likewise, the accessibility of machine learning materials and tools is undoubtably helpful for students to enter the field.

6.1 **Possible Improvements**

During this project, we documented possible improvements for future attempts at this problem. This was done due to the time restrictions of the project.

There are several ways we could have reached a better performance. To name a few:

- Use different augmentation parameters.
- Train for more epochs.
- Optimize the hyperparameters.
- Use RetinaNet
- Ensemble different algorithms in various ways.
- Segment the images.
- Get more Data.
- Use cloud-based services

In order to try out so many ideas, we would need better hardware. This project was implemented using a single Nvidia Tesla K80 GPU, which was provided by the Colaboratory environment. AWS or Google-Cloud Platform would be good alternatives due to the increase in computational power and time. As for the model training, exploiting techniques such as reducing the learning rate on the plateau would have benefited our performance. Another alternative to our method, would be using pre-built tools that provide state-of-the-art object detection algorithms, such as TensorFlow Object Detection API or Detectron.

References

Aleksander, I., & Morton, H. (1996). *An introduction to neural computing 2nd edition.*

America, R. S. (2018, August 27). Retrieved from RSNA Pneumonia Detection Challenge: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge

American Cancer Society. (2014, June 24). Retrieved from https://www.cancer.org/cancer/small-cell-lung-cancer/if-you-have-small-cell-lung-cancer-sclc.html

American Thoracic Society. (2015). Retrieved from https://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf

Colaboratory. (2017). Retrieved from https://colab.research.google.com

Cournapeau, D. (2007, June). Retrieved from Scikit-learn: Machine Learning in Python: https://scikit-learn.org/stable/

DataMax. (2014, March 24). *Bayes Theorem Visualization.* Retrieved from https://visual.ly/community/infographic/education/bayes-theorem

Fox Jr., J., & Weisberg, H. S. (2011). *An R Companion to Applied Regression.* SAGE Publications.

Girshick, R. (2015, April 30). *Fast R-CNN*. Retrieved from https://arxiv.org/pdf/1504.08083.pdf

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013, November 11). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Retrieved from https://arxiv.org/pdf/1311.2524.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning).* Massachusetts Institute of Technology.

Güneş, F. (2017, May 18). Retrieved from https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

Hui, J. (2018, March 28). *What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?* Retrieved from Medium: https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

Karpathy, A. (2018). *CS231n: Convolutional Neural Networks for Visual Recognition*. Retrieved from http://cs231n.stanford.edu/

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2016, December 9). *Feature Pyramid Networks for Object Detection*. Retrieved from https://arxiv.org/pdf/1612.03144.pdf

Miller, S., & Childers, D. (2012). *Probability and Random Processes: With Applications to Signal Processing and Communications.* Elsevier Inc.

National Electrical Manufacturers Association . (1993). Retrieved from Digital Imaging and Communications in Medicine: https://www.dicomstandard.org/

Ng, A. (2018). *Machine Learning Course*. Retrieved from https://www.coursera.org/learn/machine-learning

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Retrieved from http://neuralnetworksanddeeplearning.com/

Pohang University of Science and Technology. (2017). *Research.* Retrieved from Machine Learning Group: http://mlg.postech.ac.kr/research/

Radiological Society of North America. (2018, August 27). Retrieved from RSNA Pneumonia Detection Challenge: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015, June 8). *You Only Look Once: Unified, Real-Time Object Detection*. Retrieved from https://arxiv.org/pdf/1506.02640.pdf

Rogozhnik, A. (2016, June 24). *Gradient Boosting explained (demonstration)*. Retrieved from http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Sarkar, K. (2018, May 31). *ReLU : Not a Differentiable Function: Why used in Gradient Based Optimization? and Other Generalizations of ReLU.* Retrieved from Medium: https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec

Shaoqing , R., He, K., Girshick, R., & Jian , S. (2015, June 4). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Retrieved from https://arxiv.org/pdf/1506.01497.pdf

Stregiou, C., & Siganos, D. (1996). *Neural Networks*. Retrieved from http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

Verikas, A., Vaiciukynas, E., Parker, J. M., Gelzinis, A., & Olsson, C. (2016, April). *Patterns during Golf Swing: Activation Sequence Profiling and Prediction of Shot Effectiveness.* Retrieved from https://www.researchgate.net/publication/301638643_Electromyographic_Patterns_during_Golf_Swing_Activation_Sequence_Profiling_and_Prediction_of_Shot_Effectiveness

Wei, L., Anguelov, D., Dumitru , E., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. (2015, December 8). *SSD: Single Shot MultiBox Detector*. Retrieved from https://arxiv.org/pdf/1512.02325.pdf

APPENDICES

Appendix 1

GitHUB repository:
All the code, starter code and other files can be found in the following repository:
https://github.com/Floki1337/thesis