COVID-19 AND PNEUMONIA DETECTION IN X-RAY IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES

OF

NEAR EAST UNIVERSITY

By

ABDULLAHI ISMA'IL

In Partial Fulfillment of the Requirements for the Degree of Master of Science

In

Computer Engineering

NICOSIA, 2020

COVID-19 AND PNEUMONIA DETECTION IN X-RAY IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES

OF

NEAR EAST UNIVERSITY

By

ABDULLAHI ISMA'IL

In Partial Fulfillment of the Requirements for the Degree of Master of Science

In

Computer Engineering

NICOSIA, 2020

Abdullahi ISMA’IL: COVID-19 AND PNEUMONIA DETECTION IN X-RAY IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

Approval of Director of Graduate School of Applied Sciences

[illegible]

Prof. Dr. Nadire ÇAVUŞ

We certify that this thesis is satisfactory for the award of the degree of Master of Science in Computer Engineering

Examining Committee in Charge:

Prof. Dr. Rahib Abiyev [illegible]	Committee Chairman, Supervisor, Department of Computer Engineering, NEU
Assoc. Prof. Dr. Kamil Dimililer [illegible]	Department of Automotive Engineering, NEU
Assist. Prof. Dr. Mohammad K.S.Ma’aita [illegible]	Department of Management and Information System, NEU

I hereby declare that all the information, results, figures, and diagrams in this research work that is not originally my work is fully cited and referenced according to the academic rules and ethics of research. I have also declared that I have given credits to those that I use anything from their work in this thesis. Otherwise, any information, result, figures, and diagrams are originals.

Name, Last Name:	Abdullahi Isma'il
Signature:	[illegible]
Date:	12/01/2021

ACKNOWLEDGEMENTS

Alhamdulillah! I will like to express my deepest appreciation and gratitude to my supervisor who is also my adviser prof. Dr. Rahib Abiyev for his support and guidance throughout the research. I have to say a big thank you to my parents for their prayers, financial support, and everything. I will not forget with my friends that care so much for me especially Saminu Abdullahi who is not tired of always asking me how my thesis is going; he is such a darling friend that I am deeply grateful to have.

I dedicate this thesis research to all COVID-19 patients around the world.

ABSTRACT

COVID-19 and non-COVID-19 viral pneumonia are diseases that affect the human lungs. World health organization (W.H.O) announced coronavirus as a pandemic in 2020; the virus started from china and propagated to other countries of the world. Early diagnosis of the patients containing the virus helps in saving the patient and preventing further spread of the virus. The convolutional neural network (CNN) model is proposed in this research work to help in the early diagnosis of the virus using chest X-Ray images, as it is one of the fastest and low-cost ways of diagnosing the disease. Two convolutional neural networks (CNN) models were trained with two different datasets, the first model was trained for binary classification with one of the datasets that only have pneumonia case and normal chest X-Ray images, where the second model makes use of the knowledge learned by the first model using transfer learning and trained for 3 class classifications on COVID-19, pneumonia, and normal cases chest X-Ray images which is the second dataset. The model gives promising results of Accuracy, Recall, precision, and F1_score of 98.3, 97.9, 98.3, and 98.0 respectively on test data.

Keywords: CNN; deep learning; transfer learning; COVID-19, pneumonia; chest X-Ray images; diagnosis.

ÖZET

COVID-19 ve COVID-19 olmayan viral pnömoni, insan akciğerlerini etkileyen hastalıklardır. COVID-19, 2020 yılında dünya sağlık örgütü tarafından salgın ilan edildi, koronavirüs Çin'de başladı ve dünyanın diğer ülkelerine yayıldı. Virüs içeren hastaların erken teşhisi, hastayı kurtarmaya ve virüsün daha fazla yayılmasını önlemeye yardımcı olur. Konvolüsyonel sinir ağı (CNN) modeli, bu araştırmada, hastalığı teşhis etmenin en hızlı ve düşük maliyetli yollarından biri olduğu için göğüs röntgeni görüntülerini kullanarak virüsün erken teşhisine yardımcı olmak için önerilmiştir. İki evrişimli sinir ağı (CNN) modeli, iki farklı veri kümesiyle eğitildi, ilk model, yalnızca pnömoni vakası ve normal göğüs röntgeni görüntülerine sahip veri kümelerinden biri ile ikili sınıflandırma için eğitildi; ikinci model, bilgiyi kullanır. transfer öğrenmeyi kullanan ilk model tarafından öğrenildi ve ikinci veri seti olan COVID-19, pnömoni ve normal vakalarda göğüs röntgeni görüntüleri üzerinde 3 sınıf sınıflandırması için eğitildi. Model, test verilerinde sırasıyla 98.3, 97.9, 98.3 ve 98.0 olan Doğruluk, Geri Çağırma, hassasiyet ve F1_score'unun umut verici sonuçlarını verir.

Anahtar Kelimeler: CNN; derin öğrenme; transfer öğrenimi; COVID-19, pnömoni; göğüs röntgeni görüntüleri; Teşhis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI:	Artificial Intelligence
AUC:	Area Under Curve
CNN:	Convolutional Neural Network
CPU:	Central Processing Unit
CT:	Computer Tomography
CXR:	Chest X-Ray
DL:	Deep Learning
GPU:	Graphical Processing Unit
Grad_CAM:	Gradient Class Activation Map
ML:	Machine Learning
MLP:	Multilayered Perceptron
ReLU:	Rectified Linear Unit
RGB:	Red, Green, Blue
TL:	Transfer Learning
TPU:	Tensor Processing Unit

CHAPTER 1
INTRODUCTION

COVID-19 is a respiratory infection that affects the human lungs, which is now declared as a pandemic that is affecting the globe. As of today October 2, 2020, there are over 34 million total cases, 23.9 million recoveries, and 1.02 million deaths reported to the world health organization ("WHO Coronavirus Disease (COVID-19) Dashboard | WHO Coronavirus Disease (COVID-19) Dashboard," n.d.). The initial case of COVID-19 was in December 2019 at Wuhan, Hubei province, china(Q. Li et al., 2020), from where it started propagating to other countries of the world. As the COVID-19 virus is transmittable, early detection is very important for both the patients and the peoples around them, the patient will get proper care and the people will be protected. The best way to fight against the COVID-19 pandemic is the early diagnosis of the patient that contains the virus as early as possible and given special care and treatments. Reverse transcription-polymerase chain reaction (RT-PCR) is commonly used in the diagnosing test of COVID-19, this method has low sensitivity in the early stage of the virus, and hence, it may lead to further transmissions(L. Li et al., 2020). This test kit is expensive and scares, therefore, for early diagnosis chest X-Ray images and computer tomography (CT) scans are the best option to uses in diagnosing any patient that shows symptoms of pneumonia.

Non-COVID-19 Pneumonia is also one of the most leading diseases that cause death among young children and old peoples. According to the center for disease control and prevention (CDC) over 1 million adult pneumonia patients are hospitalized and almost 50,000 patients die every year from this disease in the USA alone ("Pneumonia | Home | CDC," n.d.). As stated by W.H.O that, chest X-rays are the best available way in diagnosing pneumonia disease (World Health Organization, 2001). Pneumonia is a respiratory infection that affects the lungs; it can be caused by bacteria, viruses, or fungi. Diagnosing pneumonia is considered a tedious task, even by the expert radiologist, because its symptoms appeared to be similar to other pathologies that affect the lungs. In this thesis, the study will use a CNN model to diagnose the presence of these viral diseases on CXR images. In recent years deep learning models prove to be a promising method in the field of medicine for pathologies diagnosis, those pathologies are not just long pathology which is the focus of this study but it gives a very promising result in diagnosing breast cancer, and some other skin diseases, with this, we make use of it in this study in diagnosing COVID-19 and pneumonia. As the diagnosis of those diseases appears to be a tedious task, even among the expert radiologist, this study is aiming to help the radiologist to diagnose pneumonia and COVID-19 from CXR scans easily within a short time.

This thesis aims to design an intelligent system that will detect COVID-19 and pneumonia diseases with high accuracy. In the thesis, Convolutional Neural Network (CNN) models were developed to help in detecting COVID-19 and pneumonia cases in X-Ray images to help for early diagnosis to prevent its transmission to other peoples. Two different datasets were used in this research, one contains only pneumonia scans and normal scans chest X-Ray, and the other one contains COVID-19 CXR scans, pneumonia, and normal chest X-Ray scans. Two CNN models were developed; the first one was trained on pneumonia and normal cases chest X-Ray images, where the second model make use of the knowledge that was learned from the first model and trained on COVID-19, pneumonia, and normal cases data. Transfer learning approach was utilized to transfer the weight/knowledge of the first model to the second model for COVID-19, pneumonia, and normal class classification.

This study focused on the way to diagnose the COVID-19 virus and pneumonia from chest X-ray scans with a CNN model with the help of the transfer learning method. The transfer learning method is employed in this thesis research to archive a high performance in training the network with a small number of images and archive a promising result, unlike the traditional way which is known as the data consuming. The study aims at developing a system that will help the radiologist in detecting COVID-19 patients and pneumonia patients using CXR images easily and within a short time. And it also aims to help those that have no access to a radiologist for early diagnosis so that proper action can be taken before the situation becomes worst and puts the patient in danger.

This study is important to both the expert radiologist and those that have no access to a radiologist. It is important to expert radiologists because the system is designed to help them diagnose COVID-19 and pneumonia more accurately in a short period. And those that have no access to a radiologist, it helps them to be diagnosed early so that proper action can be taken before the infection becomes chronic. This study contributes to the field of medicine by helping them to diagnose the presence of COVID-19 and pneumonia in a short period, unlike the traditional way in which it takes series of examination processes before confirming the cases. This study also contributes to the artificial intelligence field by showing that the deep learning model is capable to diagnose pathologies of a human body.

This study is limited to the only use of the chest X-ray image, unlike the professional radiologist that also used the physical symptoms of a patient together with the CXR scans to diagnose the presence of COVID-19 or pneumonia patients. The delimitation part of this research is the practical aspect in terms of the availability of high-performance processing hardwire for the fast and easy development of the algorithm that is used to diagnose COVID-19 and pneumonia. The use of a small portion of data in the system development is also delimitation, though the study is aiming to find a way to develop the system which traditionally is a large data consuming using only small data.

1.1 Thesis Overview

The thesis remaining parts is structured and organized as follows:

- Chapter 2 reviews the literature of the research study, where some research papers are reviewed to solidify the understanding and application of the research problem.

- Chapter 3 described the thesis methodology, including how the dataset is collected, the preprocessing techniques used, and the explanation of convolutional neural network parts like convolutional layers, pooling layers, activation functions, loss functions, and optimization algorithms. And also explain some important evaluation matrices used to evaluate the experimental result of the thesis.

- Chapter 4 shows the experimental results of the thesis research and provided the evaluation matrices results from the classification task performed.

- Chapter 5 concludes the thesis research by providing a summary of the overall thesis methodology and the experimental results, and also a recommendation that will improve the performance of the system.

CHAPTER 2
LITERATURE REVIEW

2.1. Review on Deep Learning in Diagnosing of Pneumonia

Different algorithms based on deep learning models have been developed for diagnosing Coronavirus and viral pneumonia diseases in CXR scan images and computer tomography (CT) scans images.

(Q. Li et al., 2020) used CheXNet, DenseNet, VGG19, MobileNet, InceptionV3, ResNet18, ResNet101, and squeezeNet architecture to trained on the 3 class classification using transfer learning and the use of data augmentation techniques on the dataset that they introduced to the public which consisting of 423 COVID-19 cases, 1485 viral pneumonia cases, and 1575 normal cases CXR scan images, and their networks attained a recall, precision, F1 score, specificity, and accuracy of 97.94, 97.95, 97.94, 98.80, and 97.94 respectively.

(L. Li et al., 2020) presents a network architecture called CovXNet to diagnosed COVID-19, viral pneumonia, and bacterial pneumonia, their dataset consists of 1,583 normal X-Ray images, 1493 non-COVID-19 pneumonia CXR images, 2,980 bacterial pneumonia X-Ray images, and 305 COVID-19 CXR scan images cases from different patients, their model performance has the accuracy, recall, precision, F1 score, AUC scores, and specificity of 90.2, 89.9, 90.8, 90.4, 91.1, and 89.1 respectively.

(Gunraj, Wang, & Wong, 2020) proposed a COVID-Net network to diagnose the presence of COVID-19 and pneumonia diseases, they introduced a new dataset of 13,975 CXR scans images from 13,870 patients, and their model attained performance of 93.3 accuracies, the paper only presents an accuracy as the performance matric for 3 class classification.

In (Han et al., 2020), an attention-based deep 3D multiple instance learning (AD3D-MIL) approach for screening of COVID-19 pneumonia from other forms of viral pneumonia, the researchers used a dataset of computer tomography (CT) scans that includes 230 scans of COVID-19 from 79 patients, 100 scans of pneumonia from 100 patients, and 130 CT scans from 130 peoples that do not have pneumonia. They reported that their algorithm attained an overall accuracy of 97.9.

(Rajaraman et al., 2020) proposed iterative pruned deep learning model ensemble to detect Coronavirus in CXR scan images. They trained two models in their research; the first one was trained to classify normal and abnormal chest X-Rays, the second model was trained to classified COVID-19 and pneumonia cases by using the training weights of the first model with help of the transfer learning method. They used the ensemble method to improve the prediction performance of their model. They achieved an accuracy and AUC score of 99.01 and 0.9972 in detecting COVID-19 cases respectively.

(Hammoudi et al., 2020) presented tailored models for early-stage detection of COVID-19 pulmonary symptoms. Their models were trained with a dataset that has bacterial and viral pneumonia and normal chest X-Ray images, in their research they assume that, if their model predicts a chest X-Ray image as viral pneumonia then there will be a high chance that it is COVID-19, they explained that, they do not include any COVID-19 case chest X-Ray image in their model training phase, because they believed that, the number of COVID-19 chest X-Ray images that are currently available at the time of their research is not enough to train deep learning models. Hence, they take their assumption of if their model prediction shows a high probability of viral pneumonia then it is likely to be a COVID-19 case. Their model archived the best performance using tailored DenseNet169 architecture with an accuracy of 95.72.

(Ko et al., 2020) presents a 2D deep learning called first-track COVID-19 classification network (FCONet) to diagnosed COVID-19 on a single chest computer tomography scan. They used a transfer learning approach to training the FCONet model. Computer tomography (CT) scans of 3,993 with pneumonia, normal and COVID-19 were used for training and testing the FCONet models. In all the pre-trained FCONet models, ResNet50 FCONet has a high-performance result of 99.8, 100, and 99.87 sensitivity, specificity, and Accuracy respectively on the test dataset.

(Rajpurkar et al., 2017) developed an algorithm that detects pneumonia using a convolutional neural network, where they used 121-layers convolutional neural network on a dataset that contains 100,000 frontal views of chest X-ray images. Their algorithm was compared with an expert Radiologist from the Stanford university Radiology department, their model was found to do better than the expert Radiologist that has many years of experience in the field. They extended their algorithm to detect 14 different pathologists in the chest and archive state of the art result in all the 14 different diseases that affect human respiration. They used the F1 score to compare their algorithm and the 4 radiologists and have 0.387 and 0.435 F1 average scores respectively. Even though they found out that their algorithm has some limitations compared to radiologists, like access to patient medical history which was not used to both radiologists and the algorithm, they were only provided with a frontal view of chest X-ray.

In (Ayan & Ünver, 2019) developed a computer-aided diagnosis system, to detect the presence of pneumonia in a chest X-ray image, they used transfer learning in two well-known algorithms that were trained in an image net dataset which is a large dataset of different images, they used pre-train Xception Network and VGG16 Network and archive an accuracy of 0.82% and 0.87% respectively. They used a dataset of only 5,856 frontal X-ray images and archived their result. They compared the performance of the two networks and found out that the Xception network performs well in diagnosing pneumonia cases, and the VGG16 network performs well in diagnosing normal cases. This shows that each of the networks has it is own detection capability.

In (Abiyev, Khaleel, & Ma'aitah, 2018) proposed an algorithm that detects chest pathologies including pneumonia using CXR scan images. The authors employed back propagation neural network (BFNN), competitive neural network (CpNN), and CNN in the detection of diseases. They train both the BFNN and CpNN from the same dataset of

1000 CXR scan images and train the CNN from 120,120 CXR scan images, and compared the pathology recognition rate of the algorithm and concluded that CNN outperforms both the BFNN and the CpNN. The algorithm accuracy of BFNN, CpNN, and CNN is 80.04%, 89.57%, and 92.4% respectively.

Er et al. (2009) presented an artificial immune system and three different neural networks which are multilayers, probabilistic, and learning vector quantization networks to diagnose chronic obstructive pulmonary and pneumonia. They used a dataset of 201 X-ray images containing 38 features (laboratory examinations) and achieved a classification accuracy of multilayers, probabilistic, learning vector quantization, and artificial immune system as 93.08%, 93.92%, 92.65%, and 94% respectively.

Antin et al. (2017) proposed a machine learning logistic regression model and deep learning convolutional neural network for pneumonia diagnosis, 112,120 chest X-ray scans images from 30,805 different patients that are available from Kaggle are used. Their model classified the presence of pneumonia or not in the scanned image. The archived the performance 0.6037 and 0.609 AUC for logistic regression and convolutional neural network respectively.

Jaiswal et al. (2019) used mask recurrent convolutional neural network (Mask RNN) to identify symptoms of pneumonia in image data; their network was pre-trained on COCO weights to extract important image features. Public available chest radiographs dataset from RSNA which is a subset of the original 112,000 chest X-ray datasets are used to train their mask RSNN; they also used data augmentation in the training processes for generalization in identifying the presence of the virus.

Rosenberg et al. (2019) employed an artificial swarm intelligent (ASI) system, that uses eight radiologists connected by the swarming algorithm to increase the accuracy of pneumonia diagnosis. Their studies revealed that the ASI system outperforms the diagnosis of the individual radiologist and the state of the art deep learning algorithm

(CheXNet) when compared in respect to binary classification, mean absolute error, and ROC analysis. They disclosed that previous studies on the CheXNet have higher AUROC with 0.7680 than their ASI system with AUROC of 0.7080. Their studies show that swarm-based technologies are very promising in diagnosis.

Saraiva et al. (2019) developed a CNN model and multilayer perceptron (MLP) to diagnose pneumonia presence in an X-ray image. A Dataset of 5863 X-ray images was used in the studies, the two networks were compared in terms of their accuracy to diagnose pneumonia. 94.40% and 92.16% accuracy of convolutional neural network CNN and multilayer perceptron respectively were archived.

From the analysis of different research works, it was shown that the different models based on deep learning have been designed for improving the accuracy of diagnostic systems. However, the designed models are designated for special cases. In this thesis, we are developing a unique approach for diagnosing pneumonia diseases using a deep learning model. The designed system will detect COVID-19 and pneumonia diseases with high accuracy.

CHAPTER 3
METHODOLOGY

3.1 Design

The design of this study is a software implementation of an algorithm that can diagnose COVID-19 and pneumonia cases from chest X-ray images. Deep learning algorithm specifically convolutional neural network (CNN) is used in the implementation together with the Tensor processing unit (TPU) as the hardware that processes the algorithm. Transfer learning is used for feature extraction to save more training time, reduce overfitting, and improve the accuracy of the algorithm. There are four main stages for the model implementation which are:

1) Input stage
2) Pre-processing stage
3) Training stage
4) Output stage

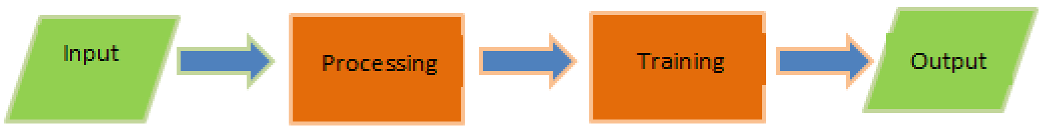


Figure 3.1: block diagram of the design stages

1) ***Input stage:*** The input of the model is a red green blue (RGB) chest X-ray images in PNG format. This is the first stage of the algorithm. After the X-ray images are feed to the model it will then start the next stage which is a pre-processing stage.
2) ***Pre-processing stage:*** This is the second stage of the algorithm, where the input chest X-ray images are check if it is in RGB form or not if it is not then it will be converted to RGB form, then the X-ray images will be resized to 180 by 180, then it will be normalized by dividing each input image pixels by 255 which will make the pixels to range from 0 - 1, not 0 – 225 which is the original range of image pixels. The

preprocessing stage helps the algorithm to easily learn and extract important image features in a short time.

3) ***Training stage:*** Training is the third stage of the model. In this stage the pre-processed chest X-ray images are feed to the input layer of the convolutional neural network (CNN), and then to the hidden layers where features learn and extracted, then the output of the last hidden layer will pass the image to the output layer.
4) ***Output stage:*** This stage is the last and final stage of the model. The final decision of the model is decided in this layer (output layer) of the network by producing three outputs classes which are COVID-19, pneumonia, and normal. This decision comes from the last layer of the convolutional neural network which is in probability form. When a COVID-19 class has a higher probability value then the decision of the algorithm is COVID-19, or if the pneumonia class has a higher probability value then the decision of the algorithm is pneumonia otherwise it is a normal class.

3.2 System Architecture

The system architecture is summarized in fig.4.2 which includes the following steps below.

1. A pre-trained model (base model) which was trained with a dataset of only pneumonia and normal X-Ray images was used as the base model of the research model architecture using transfer learning techniques.
2. The research dataset which includes COVID-19, pneumonia, and normal X-Ray images are preprocessed and prepared for the training phase.
3. The proposed model architecture was trained on the preprocess chest X-ray images of COVID-19, pneumonia, and normal cases.
4. The model output is a decision making part of the model architecture of either the X-Ray image is of COVID-19, pneumonia, or normal cases.

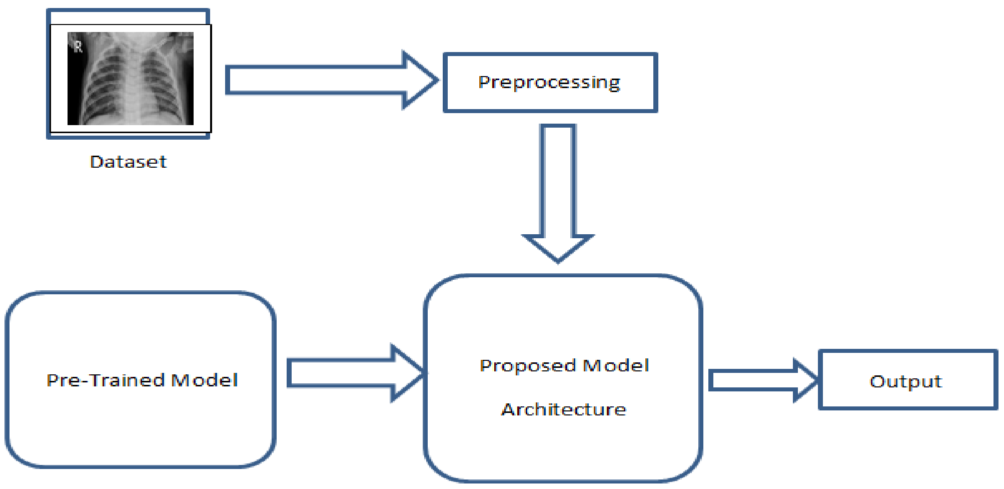


Figure 3.2: System Architecture

3.3 Dataset

The research study uses two datasets, All the datasets that were used in this research study are collected (downloaded) from the Kaggle dataset repository through the Google newly lunch 25 million free datasets early this year. (Kermany et al., 2018) is the First dataset contains 5.863 X-ray images in JPEG format, anterior-posterior chest X-ray images of pediatric patients of age ranging from one to five years old are selected from Guangzhou women and children's medical center, The chest X-ray images were screen to remove all X-Ray images that is unreadable or have low-quality scans? Two expert radiologists check and evaluate the diagnosis (Label). The third expert radiologist also rechecks the validity of the diagnosis to avoid errors. The dataset is categorized into three parts, training, validation, and testing part, each part is sub-categorized into two classes Normal and pneumonia class. The dataset contains bacterial and viral pneumonia which are considered and label as pneumonia class. 4,185 chest X-ray images were used as training data, 1,047 chest X-ray images for validation, 624 images for testing where 390 are pneumonia cases and 234 are Normal cases. All the chest X-ray images from these three parts are both from pneumonia and Normal classes. The second dataset contained COVID-19, pneumonia, and normal cases chest X-Ray images, this dataset was created by a team of researchers from Qatar University, the University of Dhaka Bangladesh with their collaborators from Pakistan and Malaysia in collaboration with medical doctors (Q. Li et al., 2020). The dataset contains 219 COVID-19 chest X-Ray images, 1341 normal chest X-Ray images, and 1345 viral pneumonia chest X-Ray images cases. Below is the sample of chest X-ray images from COVID19, pneumonia, and Normal class.

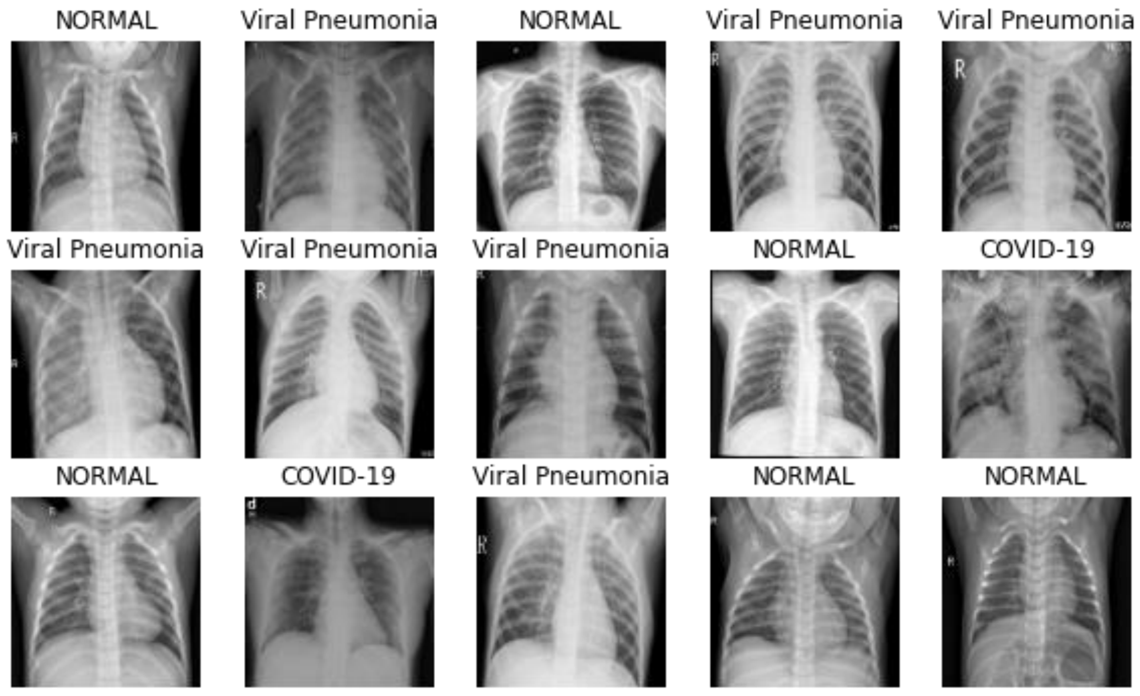
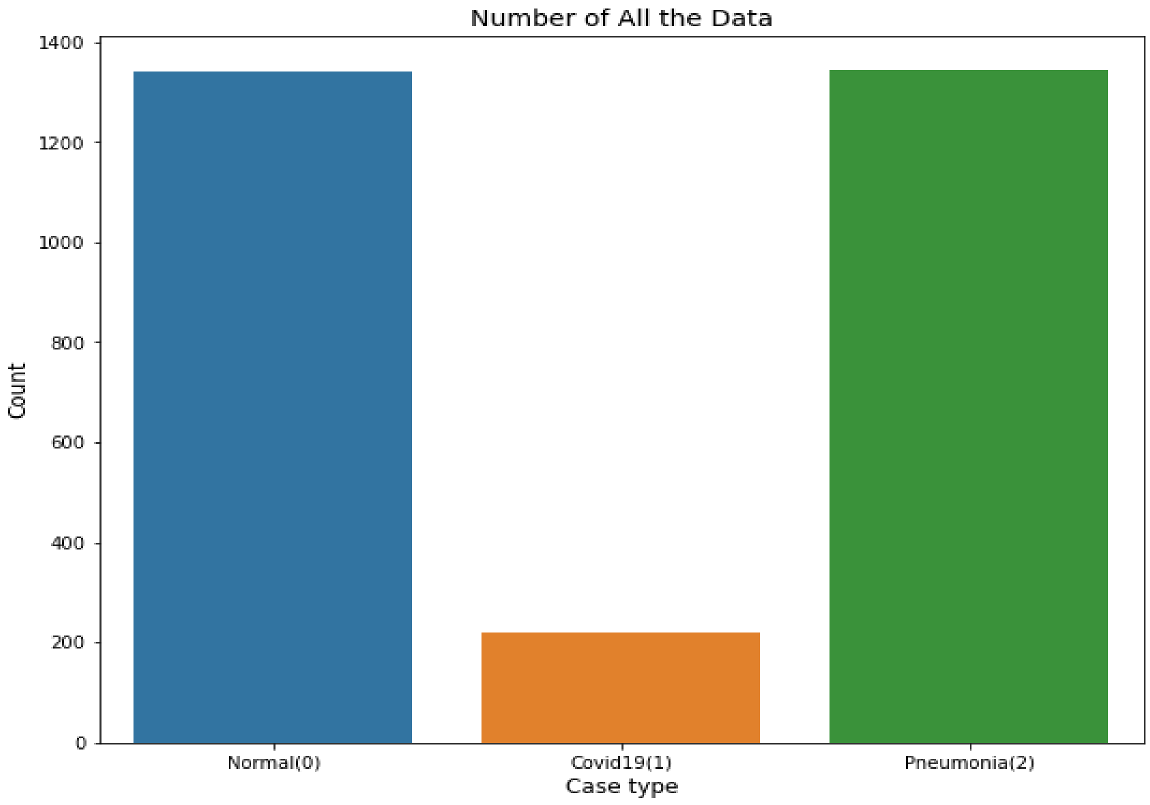
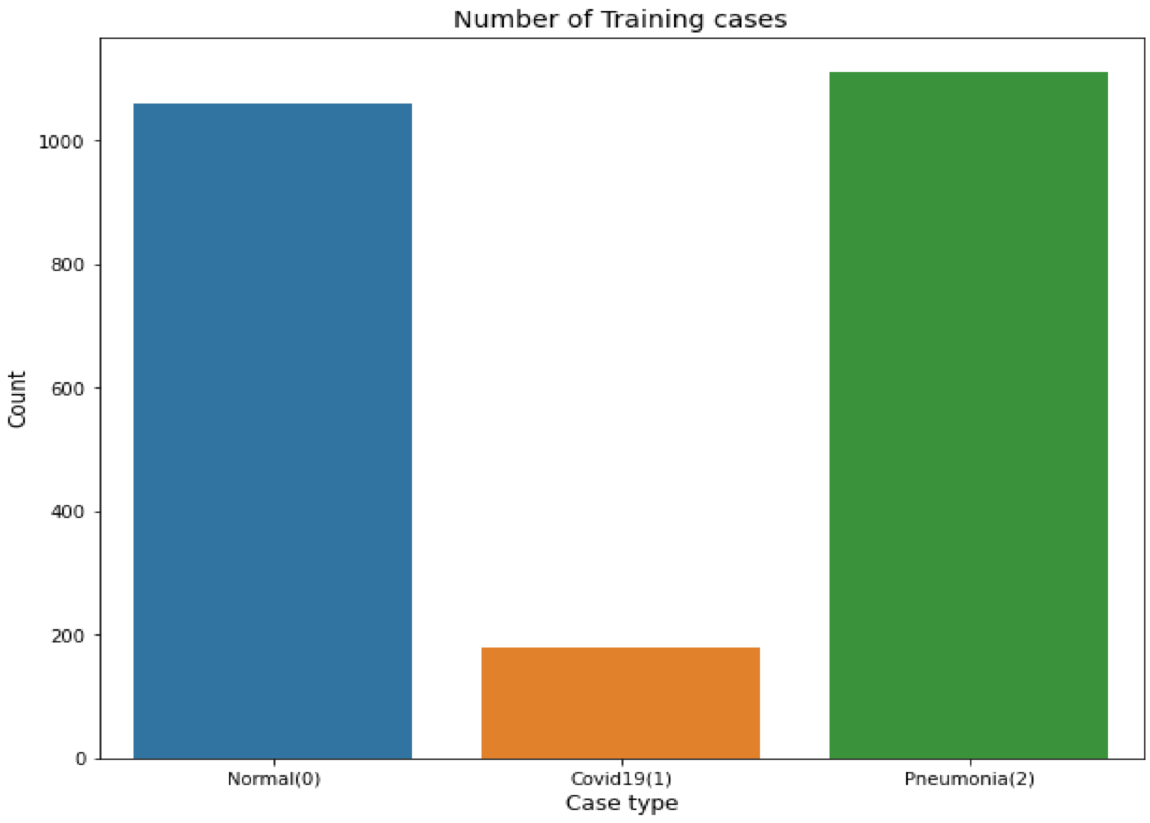
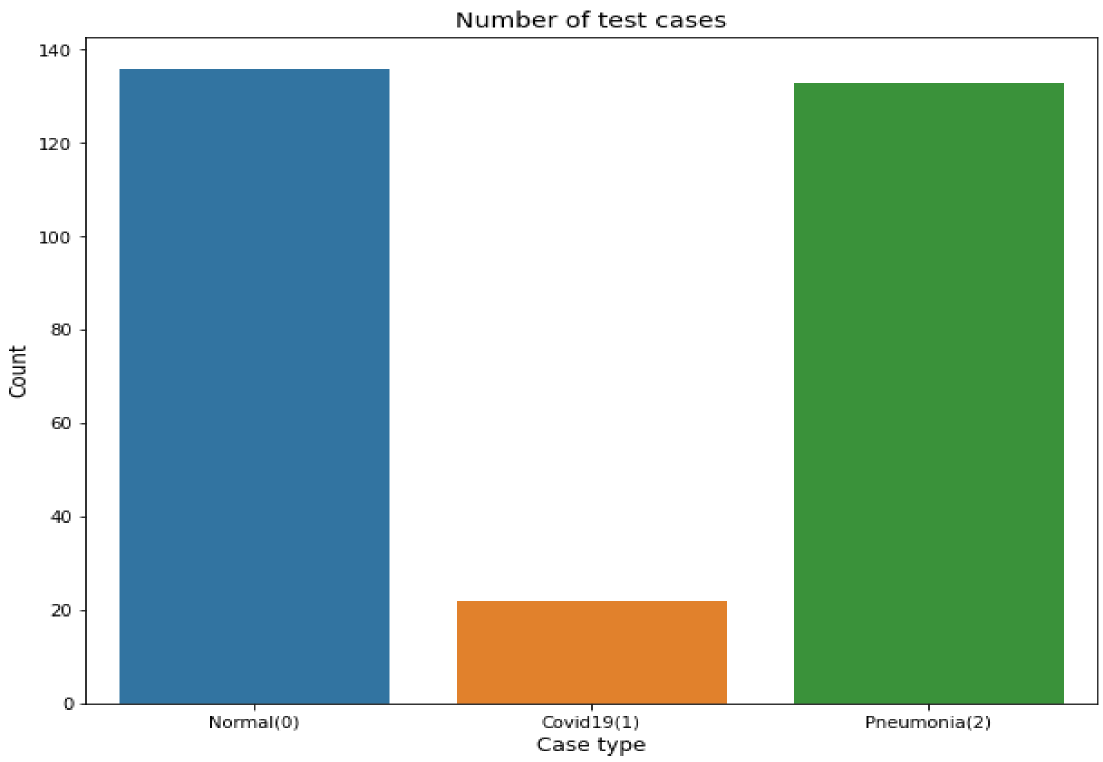


Figure 3.3: COVID-19/Pneumonia/Normal chest X-ray images

The figure below shows COVID-19/Pneumonia/Normal class distribution in the training, validation, and testing part of the dataset in form of a bar chart.


Number of All the Data
Count
1400
1200
1000
800
600
400
200
0
Normal(0)
Covid19(1)
Pneumonia(2)
Case type


Number of Training cases
Count
1000
800
600
400
200
0
Normal(0)
Covid19(1)
Pneumonia(2)
Case type



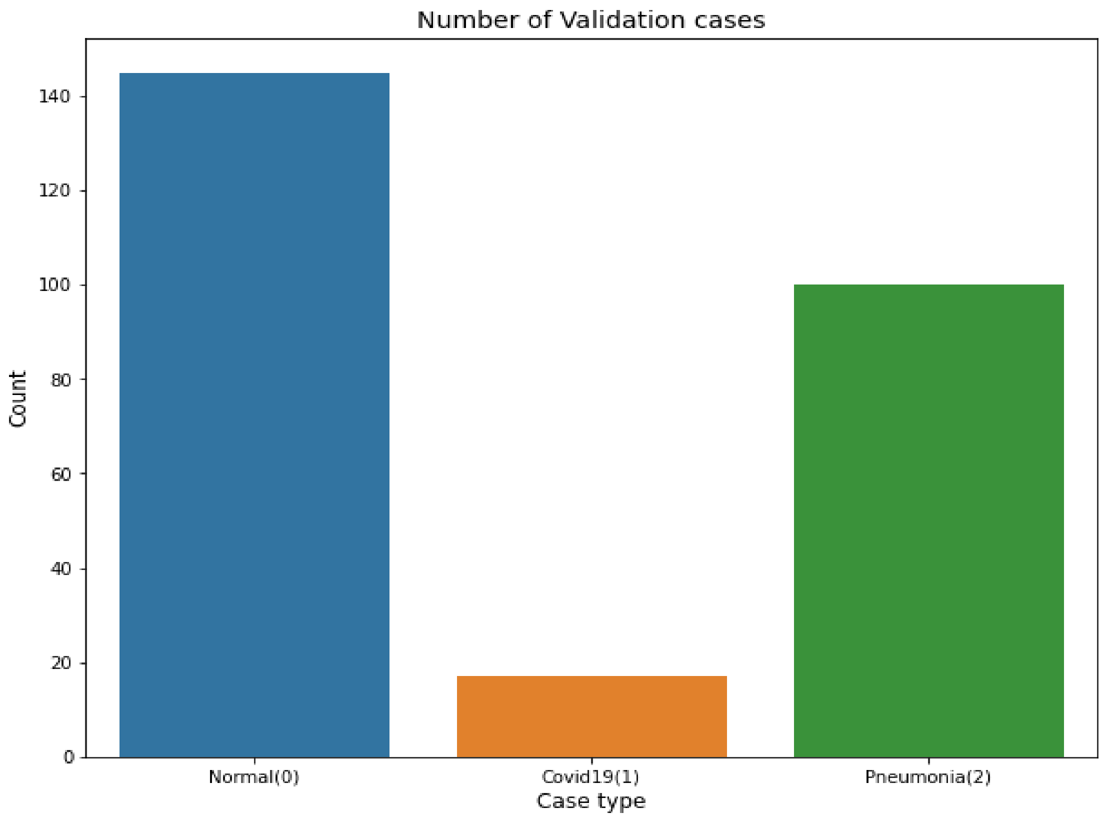


Figure 3.4: top first all data, top second training data, top third test data, top forth validation data

3.4 Preprocessing

After the dataset is collected, the next step is to pre-process the data to march the expectation of our model. First, the chest X-ray images are check to see whether there are in RGB (Red Green Blue) format or not, if it is not it will be converted to RGB format, then the X-ray images will be categorized (label) into 0 for Normal cases and 1 for COVID-19 cases, and 2 for Pneumonia cases. The images will then be resized to 180 by 180, followed by normalizing the resized image by dividing each pixel of the image (X) by 225, this will make the image pixels to range from 0 to 1, instead of 0 to 225 which is the standard intensity level of digital images. The pre-processing steps are useful since it helps the algorithm to learn and extract features from image easily, and it also helps in reducing the training time of the algorithm. The normalization is mathematically given below.

$$\mathrm{X} = \frac{X}{255} \tag{3.1}$$

3.5 Convolutional Neural Network (CNN)

A convolutional neural network is a kind of neural network just like the traditional artificial neural network, that consist of fully connected layers, and activation functions like sigmoid and ReLu with an addition of two important layers: Convolutional layers and pooling layers as shown in fig.8 below. A convolutional neural network was evolved from the visual context studies neocognitron in 1980 by k. Fukushima (Fukushima, 1980). In 1998 Yann Lecun, Leon Bottou, and Yoshua Bengio archived a very important milestone on a convolutional neural network by introducing a nowadays well-known architecture called LeeNet-5 (LeCun, Bottou, Bengio, & Haffner, 1998), which is widely used in the handwritten recognition task. As mention convolutional neural network has two main building block layers: convolutional layers and pooling layers, we will dive deep into these two layers to understand how they work.

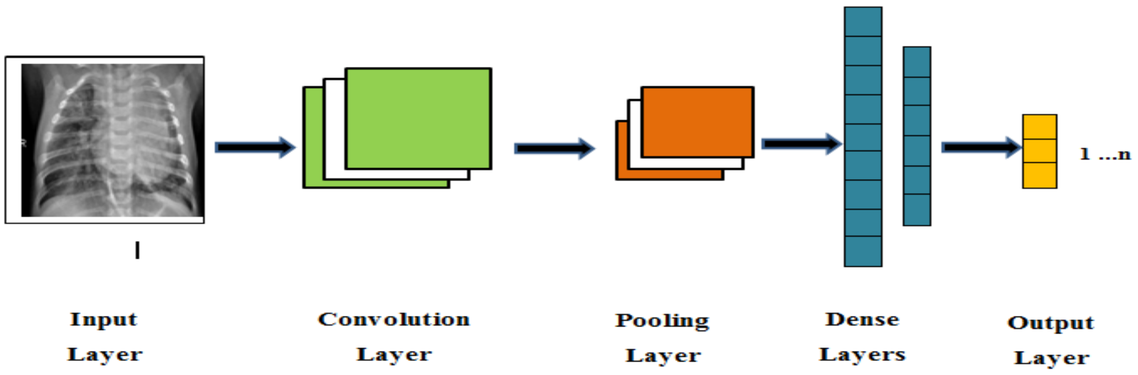


Figure 3.5: Convolutional Neural network structure/architecture

3.5.1 Convolutional Layer

The convolutional layer is one of the important building blocks of a convolutional neural network. The layer is used to learn and extract features from an image, a convolutional layer is where the input image pixels values which have some weight and height are multiplied or convolute with convolutional filters or kernels. The convolutional result output dimension will have fewer dimensions than before the convolution operation. Filters, stride, and padding types are the hyperparameters of convolutional layers that have to be set. Fig.8 shows a sample example of convolution operation on a 7x7x1 input image, convolute with a 3x3x1 filter with the stride of 2 and valid padding type (with padding). Fig. 9 shows a sample example of a convolution operation of a 7x7x1 input image, convolute with a 3x3x1 filter with a stride of 2 and the same padding type (zero paddings).

The convolution operation performs by the convolutional layer can be represented by the mathematical formula below.

$$\sum_{i=0}^{i=n} X * \omega + b \qquad (3.3)$$

Where;

X is the input

ω is the filter and

b is the bias

1

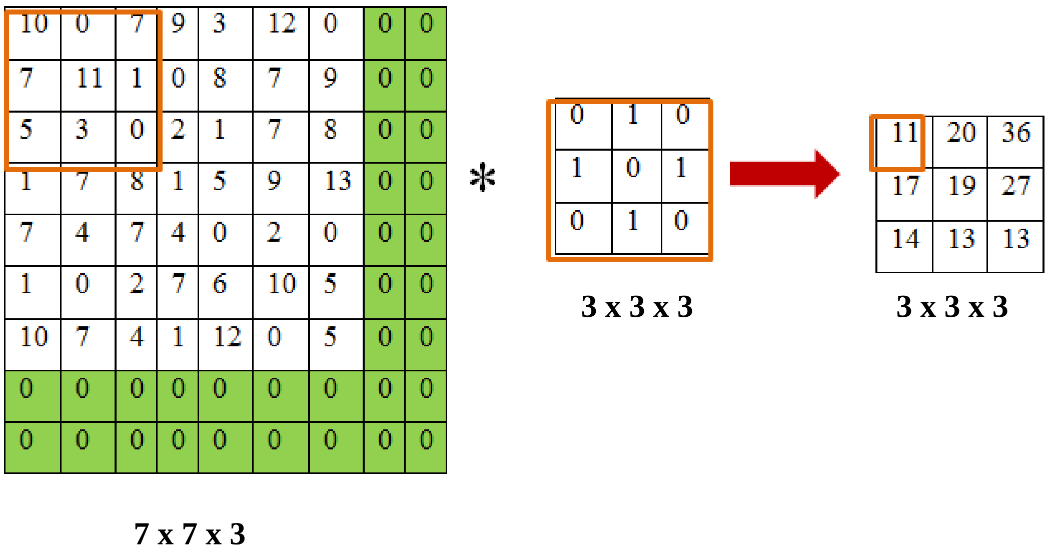


Figure 3.6: Convolutional operation with the same padding

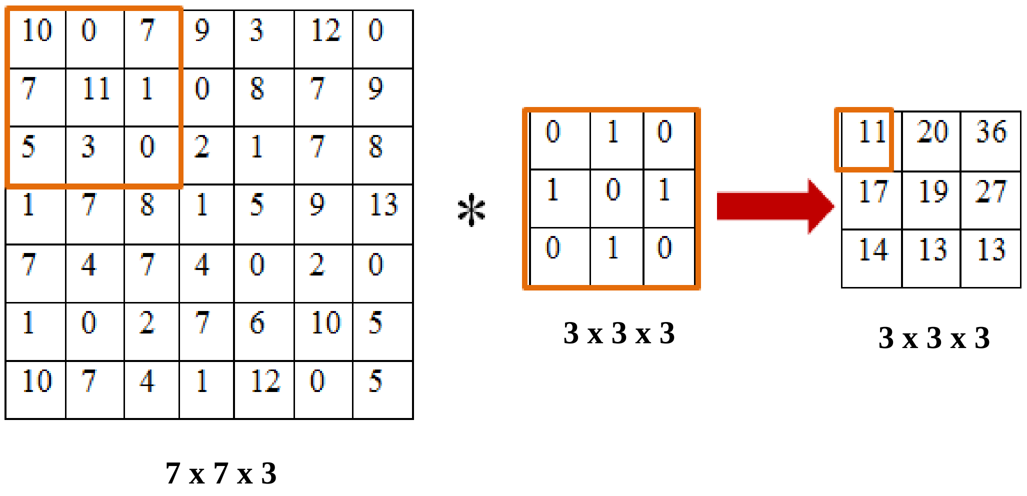


Figure 3.7: Convolutional operation with valid padding

The output dimension of the convolution of input image pixels with a convolution filter or kennel can be calculated using the below mathematical formula.

$$\lfloor \frac{n + 2p - f}{s} + 1 \rfloor \qquad (3.4)$$

Where;

n is the image dimension

p is the padding type, 0 for no padding and 1 for padding

f is the filter or kennel size and

s is the stride value.

3.5.2 Pooling Layer

The pooling layer is a layer that comes after the convolutional layer, where its main purpose is to shrink or reduce the input image size. The number of parameters, memory usage, and computational power will be reduced by pooling layers, and it also helps in reducing model overfitting risk. As in convolutional layers, pooling layers also have some hyper-parameters that must be set. The size of the pooling layer, the padding, and the stride are the hyper-parameters in each pooling layer that must be set. Pooling layers are not like convolutional layers that will do some convolutional operation; rather it just uses aggregate functions to aggregate the input. The commonly used aggregation functions are max aggregation, min aggregation, and average aggregation also known as max-pooling, min-pooling, and average-pooling respectively. Fig. 10, Fig. 11, and Fig.12 show max-pooling, min-pooling, and average pooling with 4 x 4 pooling kennel, a stride of 2, and no padding for all the three pooling types.

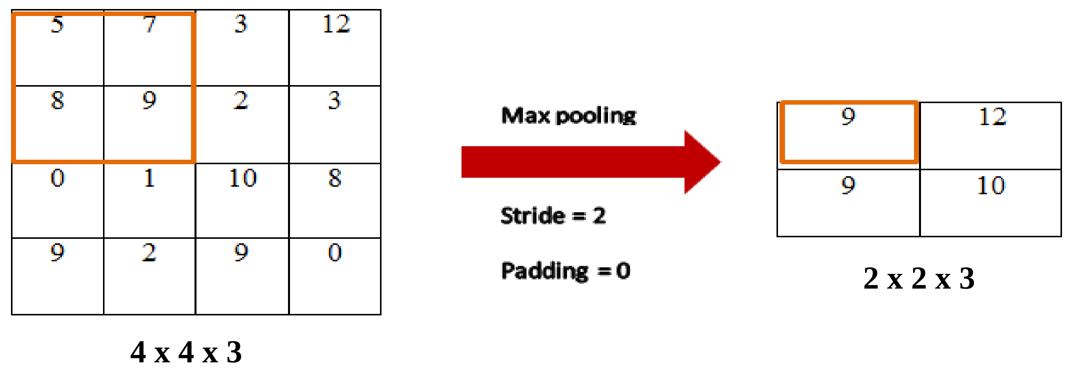


Figure 3.8: Max pooling operation

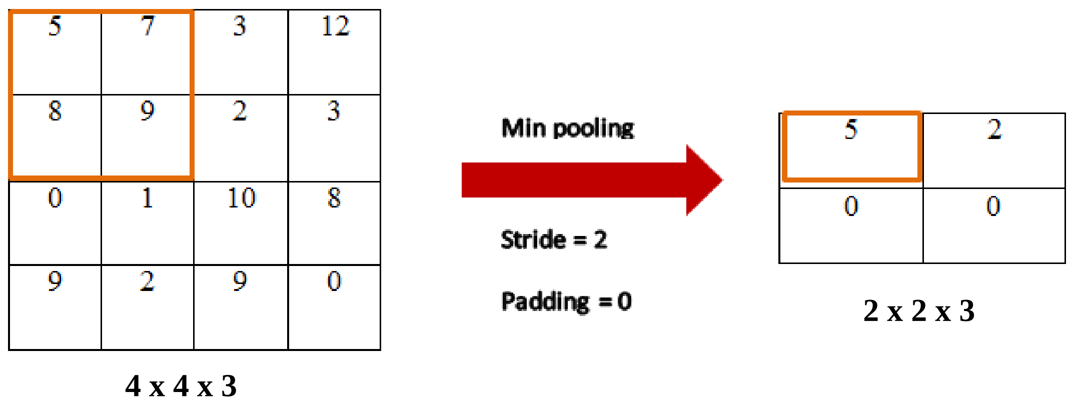


Figure 3.9: Min pooling operation

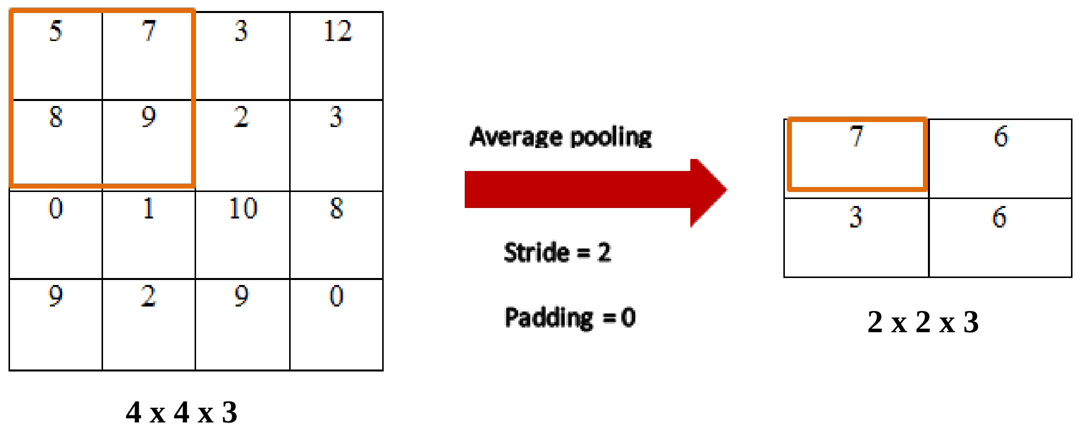


Figure 3.10: Average pooling operation

As we can see in the pooling layer operation from the above figures, it is obvious that pooling layers are a powerful layer which with only 2 x 2 kennels and stride of 2, shrinks the pooling layer output by half.

3.5.3 Activation Functions

Activation is a function that decides on either an output of a neural network layer will be passing to the next layer or not. The activation function is added at the end of each layer of a neural network (NN). There are linear and nonlinear activation functions, which are used to decide on the output of each neural network layer. Nonlinear activation functions are the functions that are mostly used in neural networks and deep learning algorithms. In this research, the five most widely used activation functions will be highlighted. Which are:

1. Sigmoid activation function
2. Hyperbolic Tangent (tanh) activation function
3. Rectified Linear Unit (ReLu) activation function
4. Softmax activation function, and
5. Exponential Linear Unit (Elu) activation function

1) *Sigmoid Activation Function*

A sigmoid activation function is a probabilistic decision-making approach activation function. Its values range from 0 to 1 as shown graphically in fig.13, this activation function can be used to predict or decide on the output result of neural network layers, it can be used in both regression and classification problems, but is mostly used on algorithms that it prediction result is needed in probability form. Below is the mathematical equation for the sigmoid activation function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

Where;

x is the input tensor.

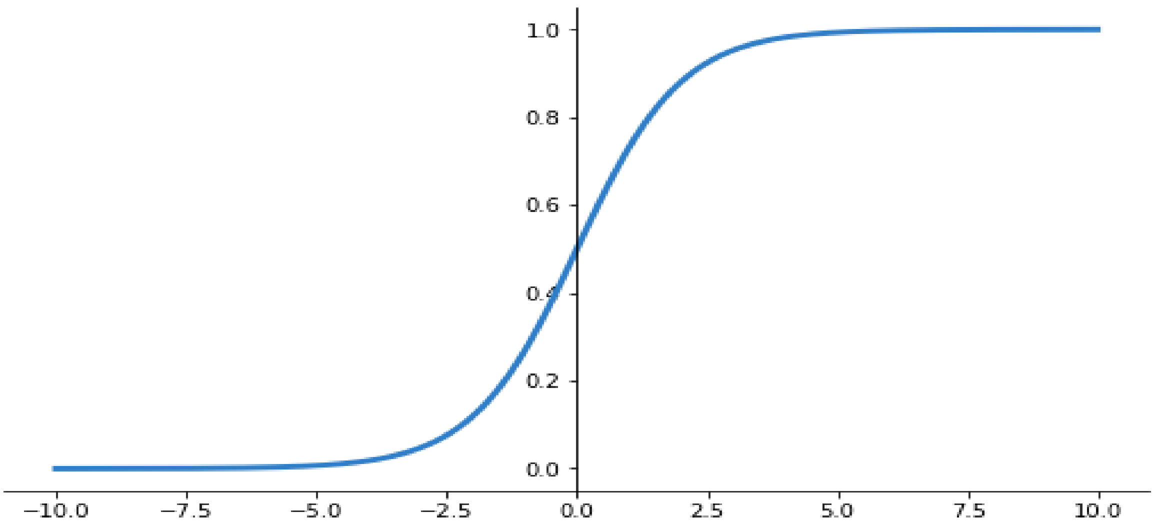


Figure 3.11: sigmoid activation function

2) Hyperbolic Tangent (Tanh) Activation Function

Hyperbolic tangent activation function (Tanh) is another activation function that is used to makes decisions on regression problems and classification problems. This activation function ranges from -1 to 1 as shown graphically in fig.14, and it maps all negative inputs into negative values. Hyperbolic tangent activation function (Tanh) is the exponential of the input tensors minus the exponential of the negative input tensors divided by the sum of the exponential of input tensors and exponential of the negative input tensors, as mathematically shown in the below equation.

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.6}$$

Where;

x is the input tensor

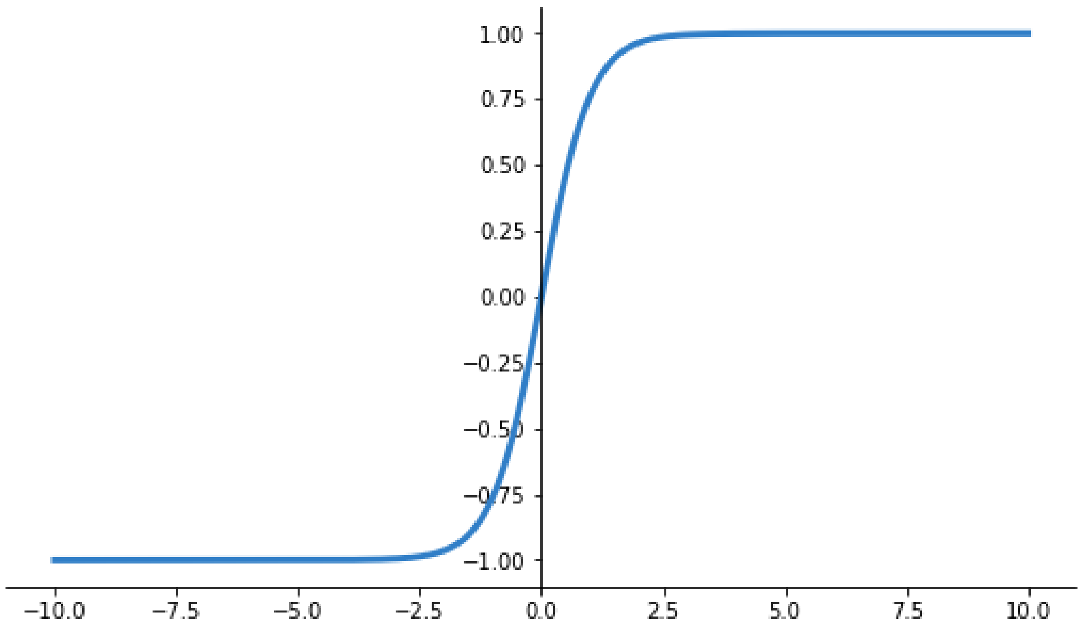


Figure 3.12: Hyperbolic Tangent (Tanh) Activation function

3) Rectified Linear Unit (ReLu) Activation Function

The Rectified Linear Unit activation function is a decision function that is mostly used in today's research. This activation function ranges from 0 to infinity as shown graphically in fig.15, if there is any negative values tensor it will be converted to zero by this activation function. The rectified linear unit activation function is the maximum value between the input tensors and zeroes as shown mathematically in the equation below.

$$f(x) = \max(x, 0) \tag{3.7}$$

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.8}$$

Where;

x is the input tensor.

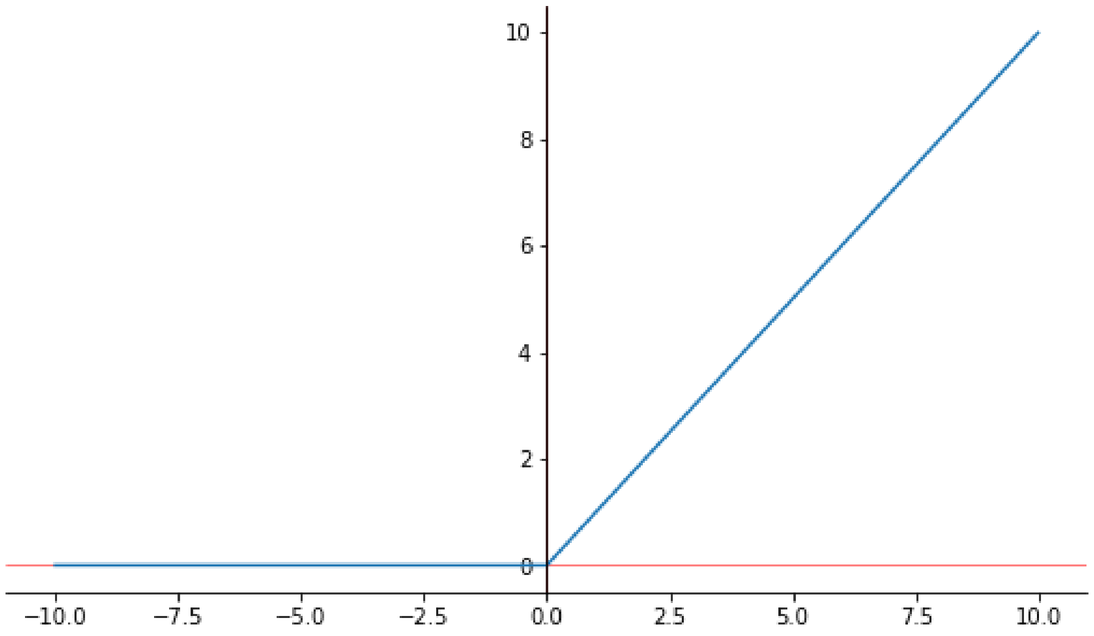


Figure 3.13: Rectified Linear Unit Activation Function

4) Softmax Activation Function

A softmax activation function is a function that is used to make decisions, mostly it is used in the last layer of a neural network, this activation function gives a value to the input tensors in relation to their weight, and if all the values are added, they will sum up to one. The softmax activation function is generally used in binary classification and multi-class classification problems. The mathematical equation for the softmax activation function is given below.

$$f(x) = \frac{e^{x}}{\sum_{j=1}^{n} e^{xj}} \tag{3.9}$$

Where;

x is the input tensor

n is the number of the input tensors

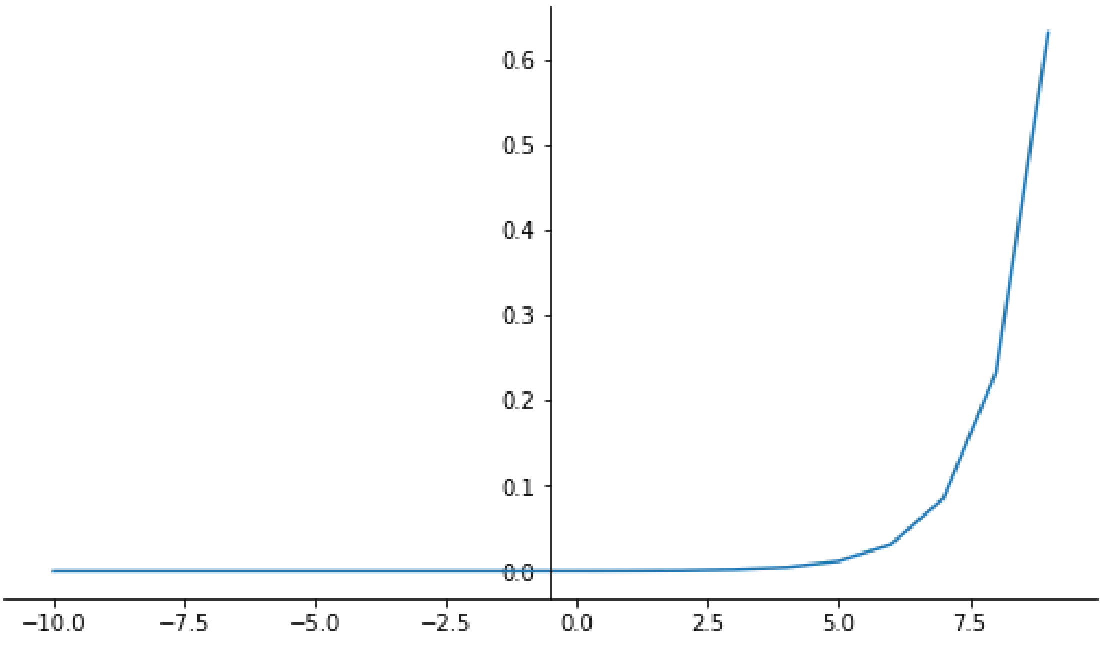


Figure 3.14: Softmax activation function

5) Exponential Linear Unit (Elu) Activation Function

The exponential linear unit (Elu) activation function is a new decision function introduced in 2016 by Djork-Arne clevert, Thomas unterthiner, and sepp Hochreiter. In their research paper, they stated and proved that their activation function gives higher classification accuracy and less time for learning in deep neural networks compare to other activation functions like Relu (Clevert, Unterthiner, & Hochreiter, 2016). Unlike the Relu activation function, elu has negative values as shown in fig.17. Below is the mathematical equation for the exponential linear unit activation function.

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases} \tag{3.10}$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ f(x) + \alpha, & x \leq 0 \end{cases} \tag{3.11}$$

Where;

x is the input tensor

α is the hyperparameter that controls the values to which elu saturate negative inputs

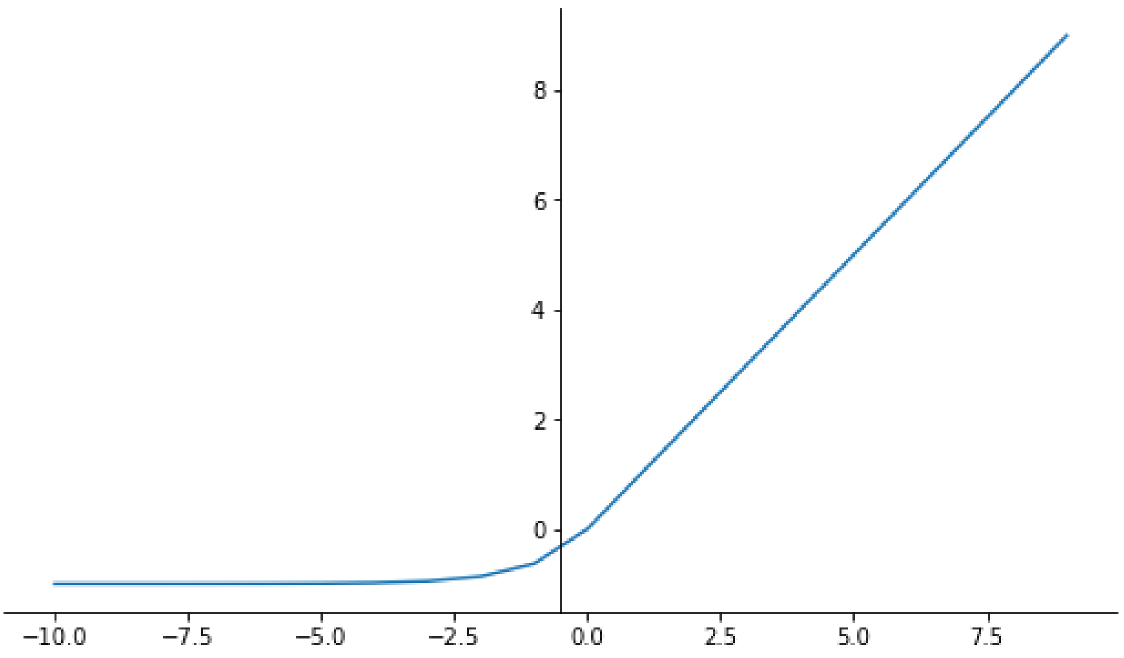


Figure 3.15: Exponential linear unit activation function

3.5.4 Loss Functions

The loss function is a function that is used to estimate the error or loss of a model. The model has to know how well or bad it perform to optimize the loss while training. This is done by knowing the loos or error it makes, which is provided by the loss function. The model will optimize the loos by updating the weights of the algorithm to attend a good performance using the result return by the loss function. The loss function is computed from the predicted labels provided by the model and the true labels. There are many loss functions that can be used depending on the problem at hand, such as regression and classification problems. In this thesis research, the five most widely used regression, classification problems loss functions will be highlighted, which are:

1. Root Mean Square Error (RMSE)
2. Mean Absolute Error (MAE)
3. Binary Cross-Entropy
4. Categorical Cross-Entropy, and
5. Sparse Categorical Cross-Entropy

Root mean square error and mean absolute error are loss function that is used to estimate losses in regression related problems, while binary cross-entropy, categorical cross-entropy, and sparse categorical cross-entropy are loss functions that are used to measure the degree of losses or error in classification related problems. Table.1 below summarizes the uses, functions, and formulas of the above-listed loss functions.

Table 3.1: Loss Functions

Loss Function	Label Class	Label Type	Formula
Root Mean Square Error(RMSE)	_	continuous	$\sqrt{\sum_{i=0}^{n} \frac{(f(x)-y)^2}{n}}$
Mean Absolute Error(MAE)	_	continuous	$\frac{1}{n}\sum_{i=0}^{n} \lvert f(x)-y \rvert$
Binary Cross-Entropy	0 or 1	binary	$\frac{-1}{n}\sum_{i=0}^{n} y.\log(f(x) + (1-y).\log(1-f(x)1))$
Categorical Cross Entropy	>=2	One_hot encoded	$-\sum_{i=0}^{n} y\ log(f(x))$
Sparse Categorical Cross Entropy	>=2	integer	$-\sum_{i=0}^{n} y\ log(f(x))$

Wherefrom the above table;

f(x) is the predicted labels

y is the true labels, and

n is the data point number.

3.5.5 Optimizers

An optimizer is an algorithm or process/method that uses the loss function to update the weight of a neural network by reducing the loos in the model, to attend the most accurate performance possible. In this thesis research, I am going to highlights some of the most widely used, and the fastest optimization algorithms used to train a neural network, which is the following:

1. Momentum optimizer
2. Nesterov Accelerated Gradient optimizer
3. AdaGrad optimizer
4. RMSProp optimizer
5. Adam optimizer

1) Momentum Optimizer

Momentum optimizer was proposed by Boris polyak in 1964 (Polyak, 1964), where it aims at considering the previous gradients while computing the weight to soften the convergence and reduced the variance of the model. When compared to the stochastic gradient descent algorithm SGD it has a soften convergence and lower variance. Momentum optimizer converges fast by going directly to the global minimum without going through an irrelevant direction toward the global minimum. Each iteration this optimizer sums up the local gradient to its parameter called momentum represented by m. the weight is updated by subtracting the momentum from it. The momentum algorithm is shown in the below equation.

1. $m \leftarrow \beta m + \alpha \nabla j(\theta)$ (3.14)

2. $\theta \leftarrow \theta - m$ (3.15)

Where;

m is the momentum

β is the decay rate

j(θ) is the cost function

α is the learning rate

θ is the weight parameter

2) Nesterov Accelerated Gradient Optimizer

Nesterov accelerated gradient optimizer or Nesterov momentum optimizer was proposed in 1983 by Yurii Nesterov. This optimizer is almost the same as the momentum optimizer with some small changes. Nesterov momentum optimizer is using position slightly ahead of local minimum to measure the gradient of the cost function, and this position is of course toward the direction of the momentum. The optimizer also updated the weight by subtracting the momentum from the weight as shown in the equation below.

1. $m \leftarrow \beta m + \alpha \nabla j(\theta + \beta m)$ (3.16)

2. $\theta \leftarrow \theta - m$ (3.17)

Where;

m is the momentum

β is the decay rate

α is the learning rate

θ is the weight parameter

3) AdaGrad Optimizer

AdaGrad optimizer is using a second-order optimization algorithm, the optimizer is proposed by Duchi, Hazan, and Singer in 2011 (Duchi, Bartlett, & Wainwright, 2012). The learning rate in this optimizer is not manually set or constant as in the other optimization algorithms, the learning rate is set according to the updated frequency of the parameter while training. The equation for the AdaGrad algorithm is shown below.

1. $s \leftarrow s + \nabla j(\theta) \otimes \nabla j(\theta)$ (3.18)

2. $\theta \leftarrow \theta - \alpha \nabla j(\theta) \oslash \sqrt{s + \epsilon}$ (3.19)

Where;

s is the square of the gradient vector

j(θ) is the cost function

α is the learning rate

θ is the weight parameter

$\otimes$ is the element-wise multiplication symbol

$\oslash$ is the element-wise division symbol

4) RMSProp Optimizer

RMSProp optimizer was proposed by Hinton in 2012 (Lyon & Lyon, 2017), where its optimization performance is better than that of AdaGrad optimizer, in AdaGrad optimization technique it goes down to the gradient too fast and may end up never converge to the global minimum, RMSProp solve this problem by considering the most recent gradient in the iteration cycle, this is done by using the exponential decay from the first step. This optimizer is the choice of many researchers before the introduction of the Adam optimizer. Below is the equation showing the RMSProp optimization algorithm.

1. $s \leftarrow \beta s + (1-\beta)\, \nabla j(\theta) \otimes \nabla j(\theta)$ (3.20)

2. $\theta \leftarrow \theta - \alpha \nabla j(\theta) \oslash \sqrt{s+\epsilon}$ (3.21)

Where;

s is the square of the gradient vector

j(θ) is the cost function

α is the learning rate

β is the decay rate

θ is the weight parameter

$\otimes$ is the element-wise multiplication symbol

$\oslash$ is the element-wise division symbol

5) Adam Optimizer

Adam optimizer is proposed by P. Kingma and Lee Ba in 2014(Kingma & Ba, 2015). Adam stands for an adaptive moment estimator. This optimizer encapsulated the idea of momentum optimizer and RMSProp optimizer by monitoring both the exponential decay average of the previous gradient and the exponential decaying average of the previous square gradient respectively. Adam optimizer is the best among the other optimizers because the algorithm is too fast and also converges rapidly. The equation below shows the Adam optimization algorithm.

1. $s \leftarrow \beta_1 s + (1-\beta_1)\, \nabla j(\theta) \otimes \nabla j(\theta)$ (3.22)

2. $m \leftarrow \beta_2 m + (1-\beta_2)\nabla j(\theta)$ (3.23)

3. $m \leftarrow \frac{m}{1-{\beta_2}^T}$ (3.24)

4. $s \leftarrow \frac{s}{1-{\beta_1}^T}$ (3.25)

5. $\theta \leftarrow \theta - \alpha m \oslash \sqrt{s+\epsilon}$ (3.26)

Where;

m is the momentum

s is the square of the gradient vector

j(θ) is the cost function

α is the learning rate

θ is the weight parameter

$\otimes$ is the element-wise multiplication symbol

$\oslash$ is the element-wise division symbol

β_1 is the momentum decay hyperparameter

β_2 is the momentum decay hyperparameter

ϵ is the smoothing term

T is the iteration number starting from 1

3.6. Transfer learning

Transfer learning is the method of reusing an already train machine learning model which is train with a large amount of data in a particular task and re-use that model to train a new classifier of a similar or different task by turning the model hyperparameters or freezing all or some of the layers of the already trained model as shown in fig.21. Transfer learning helps us to train a convolutional neural network with a small size of dataset. In transfer learning, the new model that is built with a pre-train model does not have to train with a large dataset to perform well. Since all the base features are already learned in the pre-train model, the training time is not as much as to train from scratch that is to train without using the transfer learning method, the memory, and computational resources will also be reduced compared to training from scratch. Many pre-train algorithms are trained with large datasets like the ImageNet dataset (Russakovsky et al., 2015), which have over 15 million images from around 22,000 categories. Those pre-train algorithms are once that is used mostly as a base in the transfer learning method.

Transfer learning is a process of reusing an already trained model that is trained for a specific tasking to a new task either the tasks are similar or not. The transfer learning method is usually and most widely used method in computer vision-related tasks. Transfer learning help in the following ways:

I. It reduced the training time of a model.
II. It reduced the computational cost.
III. It prevents or reduced model over-fitting
IV. It allows the training of large CNN with a small amount of data.
V. It also increases/bust the performance of a model.

Xception Network, VGG16 Network, ResNet50 Network, Inception V3 Network, DenseNet121 Network are the popular and most used pre-train algorithms which are the state-of-the-art algorithms that are trained with ImageNet dataset.

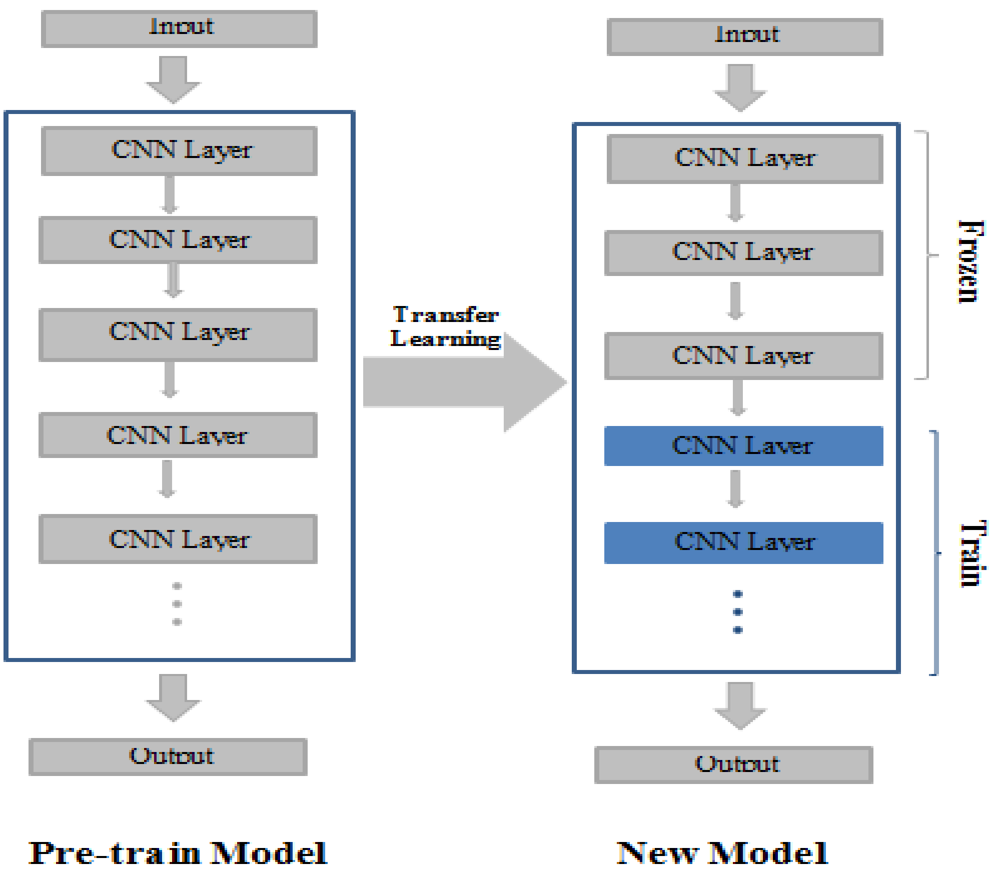


Figure 3. 216: transfer learning structure

3.7 Evaluation Matrices

Evaluation matrices are matrices that are used to measure the performance of machine learning or deep learning models. There exist different types of evaluation matrices that can be used to evaluate models. Different evaluation matrices are uses to evaluate the performance of different models regarding the problem at hand. Some evaluation matrices are best in measuring the performance of regression models, while some are best for the classification models. As mentioned, there exist many types of evaluation matrices, wherein this thesis research, I am going to highlight the most widely used evaluation matrices in today's research community, which are as follows:

1. Confusion matrix
2. Accuracy
3. Recall
4. Precision
5. F1 score, and
6. Area Under the Rock Curve (AUC)

3.7.1 Confusion Matrix

Confusion matrix is a matric that is used to measure the performance of classification algorithms. The classification problems may be a binary or multi-class classification. Confusion matric provides the exact number of true positive class, false-positive class, true negative class, and false-negative class by comparing the actual classes from the original data with a predicted label from the classification algorithm. Below is the sample confusion matric of a binary classification problem.

		Predicted labels	
		P	N
True labels	P	*TP*	*FN*
	N	*FP*	*TN*

Figure 3.17: Confusion matric

Where;

P is the positive class

N is negative class

TP is the true positive class

TN is a true negative class

FP is the false positive class

FN is false negative class

3.7.2 Accuracy Matric

Accuracy is an evaluation metric that is used in measuring the performance of classification algorithms or regression algorithms. Accuracy will be problematic or misleading performance matric when use to evaluate a model that is trained on unbalanced data. For this evaluation matric to provide a good and reliable performance measure, the data to be used in training the model must be balanced. Accuracy is computed by summing up the true positive and true negative class divided by the summation of true positive, true negative, false positive, and false negative classes as shown in the formula below.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.27}$$

3.7.3 Recall Matric

A recall is another evaluation metric that is used to measure the performance of the classifier. A recall is a correctly classified class from the classification model. A recall is computed by dividing the true positive class with the summation of the true positive class and false-negative class as shown in the formula below.

$$Recall = \frac{TP}{TP+FN} \tag{3.28}$$

3.7.4 Precision Matric

Precision is also an evaluation matric usually used together with recall evaluation matric to measure the performance of classification algorithms. Precision is a positive prediction result that corresponds to the correctly classified class. Precision is computed by dividing the true positive class with the summation of true positive and false positive class as shown in the formula below.

$$Precision = \frac{TP}{TP+FP} \tag{3.29}$$

3.7.5 Precision/Recall Tradeoff

This is a tradeoff between the recall and the precision of classifiers, where a classifier will have to give off one to gain the other. Here giving off one does not mean to not use it but to have less performance than the other. For instance, let’s say you have a problem with diagnosing a patient with pneumonia cases; here you will want your classifier to classify all the patients with pneumonia cases than to classified patients with pneumonia cases as a patient with no pneumonia. In this particular case, you will have to trade off the classifier precision to have more recall since the problem at hand needs more recall than the precision. The same goes for the precision also; thresholds have to be set between the recall and the precision defending on the problem that is to be solved. Below shows a plot of a tradeoff between precision and recall at a certain threshold.

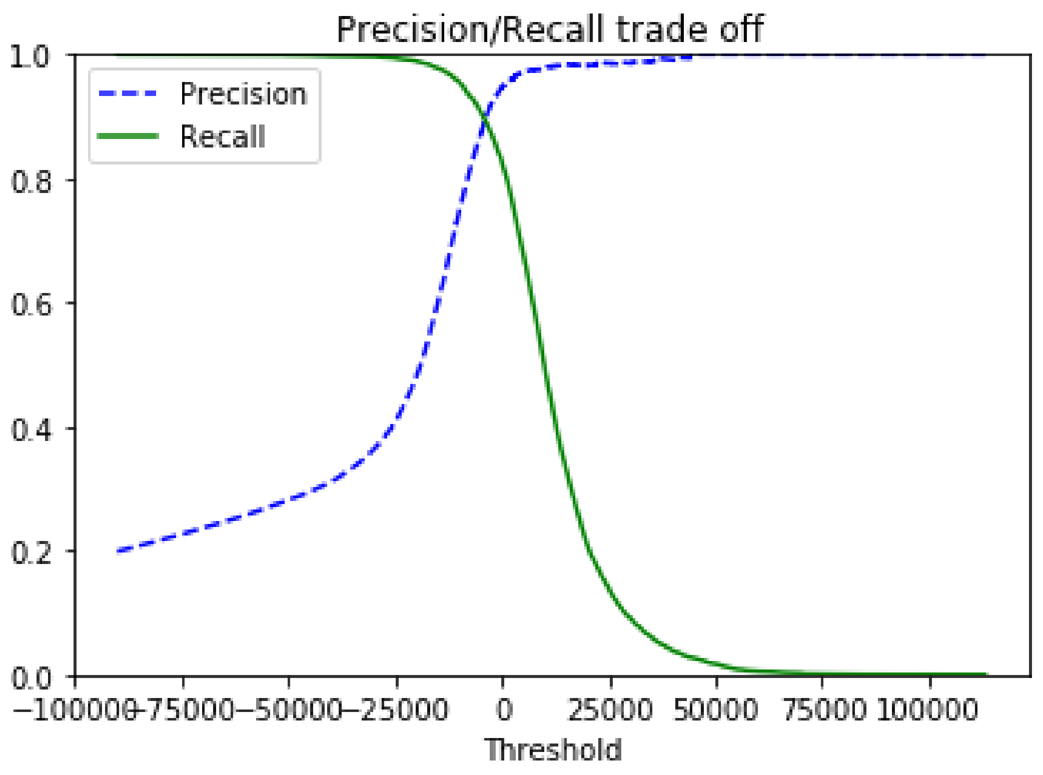


Figure 3.18: precision-recall tradeoff

3.7.6 F1 Score Matric

F1 score evaluation matric is used to measure the performance of a classifier by combining the recall and precision evaluation matrices as one single performance measure evaluation matric. F1 score evaluation matric is computed by dividing the twice of precision multiply by recall with a summation of recall and precision evaluation matric result. Below shows how the F1 score evaluation matric is computed.

$$F1\,Score \;=\; \frac{2\times precision\times recall}{precision+recall} \tag{3.30}$$

3.7.7 Area Under the ROC Curve (AUC)

Ares under the ROC curve is a performance evaluation matric that is used to measure the aggregate performance of all the possible thresholds of the classifier. AUC of greater than 80% is generally considered to be an excellent performance by the classifier. AUC is a plot of the true positive rate versus the true negative rate from the classifier. Below shows a sample AUC plot.

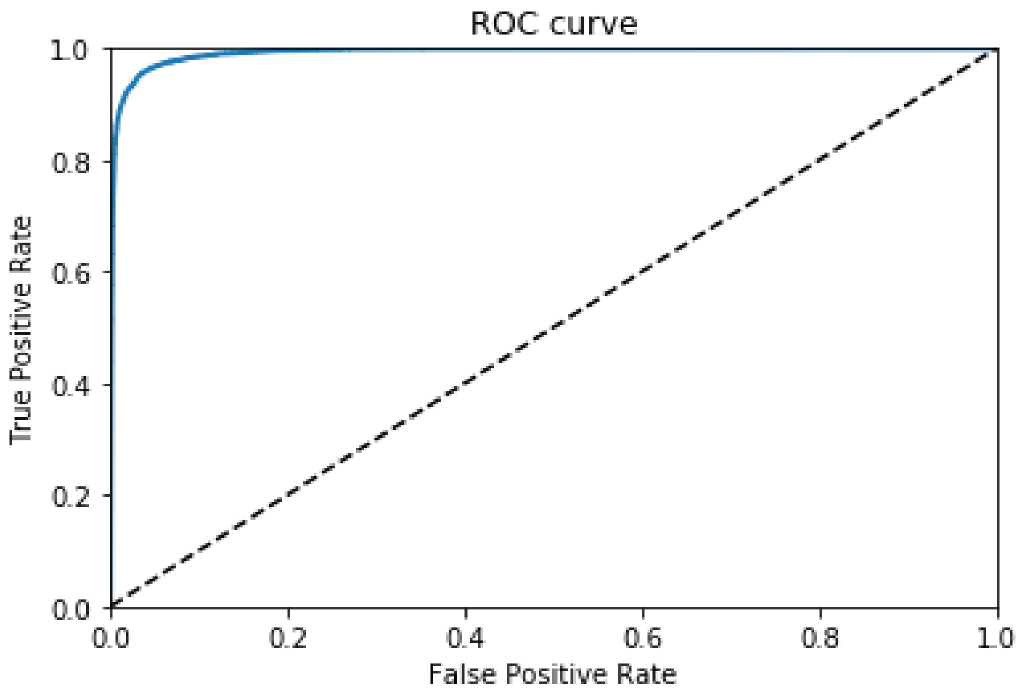


Figure 3.19: Area under the curve plot

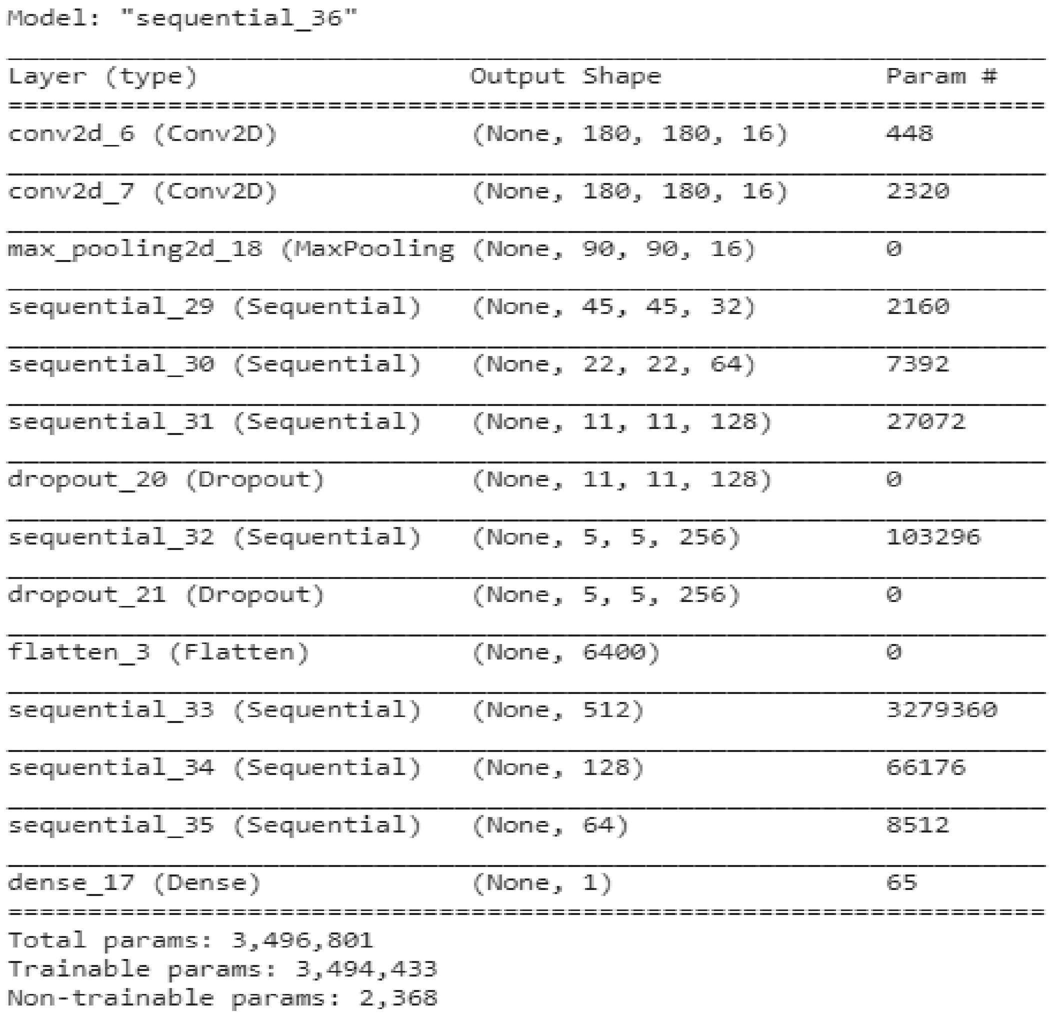
CHAPTER 4
IMPLEMENTATION RESULTS

4.1 Results

The main aim of this research is to develop a system that will correctly diagnose the presence of COVID-19 and viral pneumonia in X-Ray images. The transfer learning approach was used in developing the system, two models were trained in two different datasets, the first model was trained on a dataset that contain only normal and pneumonia cases, the second model used the first model as the base model so as to say it uses the knowledge learned by the first model to train on the dataset that contains COVID-19, pneumonia, and normal cases. The training and testing of the model were performed using Keras and TensorFlow framework in a python programming language with tensor processing unit (TPU) as an accelerator in Kaggle kennel.

4.2 First Model Results

Figure 4.1 below shows the configuration of the first model that used the first dataset to learn the features and detect the presence of Pneumonia cases in X-Ray images. The first model takes an X-Ray image of size 180 x 180 x 3 as an input, and output probabilistic results from the last layer of the network which used sigmoid activation function, the result of this layer is pneumonia if the probabilistic result is greater than 0.5, otherwise, the X-Ray image is normal.



```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 180, 180, 16)      448
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 180, 180, 16)      2320
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 90, 90, 16)        0
_________________________________________________________________
sequential_29 (Sequential)   (None, 45, 45, 32)        2160
_________________________________________________________________
sequential_30 (Sequential)   (None, 22, 22, 64)        7392
_________________________________________________________________
sequential_31 (Sequential)   (None, 11, 11, 128)       27072
_________________________________________________________________
dropout_20 (Dropout)         (None, 11, 11, 128)       0
_________________________________________________________________
sequential_32 (Sequential)   (None, 5, 5, 256)         103296
_________________________________________________________________
dropout_21 (Dropout)         (None, 5, 5, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6400)              0
_________________________________________________________________
sequential_33 (Sequential)   (None, 512)               3279360
_________________________________________________________________
sequential_34 (Sequential)   (None, 128)               66176
_________________________________________________________________
sequential_35 (Sequential)   (None, 64)                8512
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 65
=================================================================
Total params: 3,496,801
Trainable params: 3,494,433
Non-trainable params: 2,368
```

Figure 4.1: First model summaries

Table 4. 1: First Dataset distribution

Cases	Training data	Validation data	Test data
Normal	1,067	282	234
Pneumonia	3,118	765	390
Total	4,185	1,067	624

Table 4.1 above shows the distribution of the first dataset which includes pneumonia and normal chest X-Ray images, this dataset is used to train the first model.

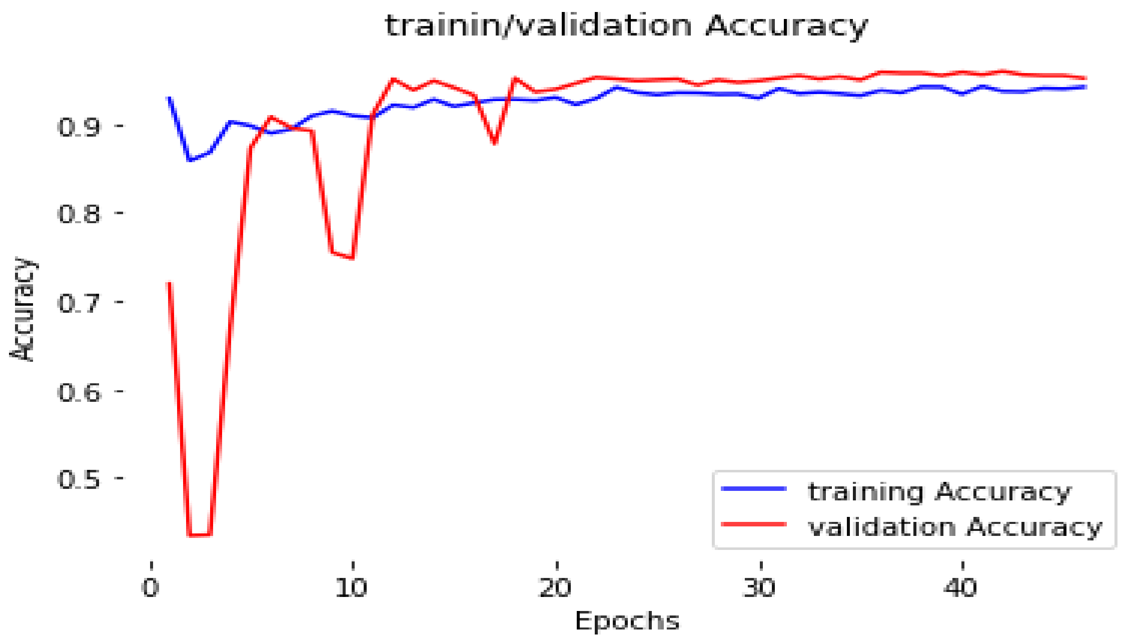


Figure 4. 2: First model accuracy per epoch

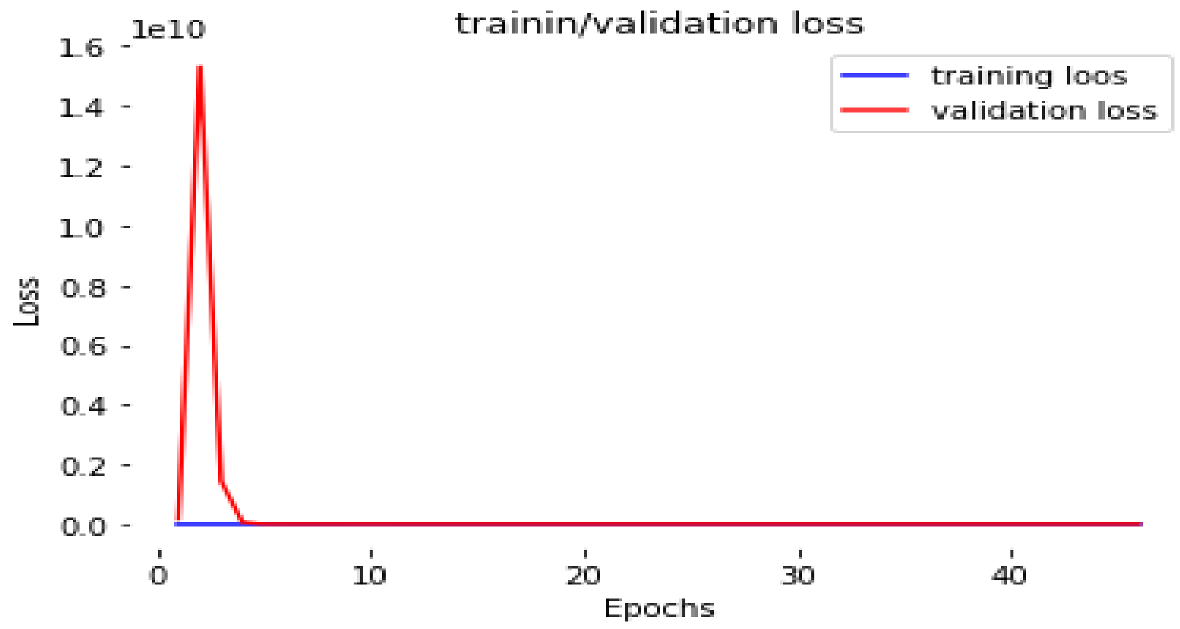


Figure 4.3: First model loos per epoch

The above figure 4.2-4.3 are the training accuracy and loos respectively for the first model, the model is trained to the extends that it learn the important basic X-Ray image features and can differentiate pneumonia case from normal with good accuracy, though the model plot above shows a sign of model underfitting, and this is because the first dataset used in the model training is small, so this issue can be solved by adding the training data, and further tuning the hyperparameters of the model.

Table 4. 2: First model test results

	Epoch	**Recall**	**Precision**	**F1_Score**	**Accuracy**	**Loos**
First Model	46	0.96	0.82	0.88	0.85	0.44

Table 4.2 above shows the evaluation results of the first model that was trained with the first dataset. The model attends a recall, precision, F1_scores, and Accuracy of 0.96, 0.82, 0.88, and 0.85 respectively. The performance of this model can be improved by turning the model hyperparameters, the addition of more training data, and using data augmentation techniques. Though with this result we are able to get what we need in order to help us in building the main model that classified COVID-19 and Pneumonia cases from chest X-Ray images.

4.3 Second Model Results

Figure 4.4 shows the overall architecture of the proposed model. The architecture received an X-Ray images data of size 180 x 180 x 3, it has 2 convolutional layers followed by one max-pooling layer, then 3 sequential layers followed by a dropout layer, then another sequential layer followed by dropout layer, flatten layer flowed by 3 sequential layers, a dense layer followed by an output layer which performed classification. All the layers use the “RELU” activation function, “SAME” padding type, and filter of size 3 for all the convolutional layers. Adam optimizer with default parameters and categorical cross-entropy loss is used in the model training.

Figure 4.5 – 4.6 presents the training and validation plots of the model accuracy and loss against each epoch. As shown on the plot, the model has achieved high performance with only 20 epochs, from the plot it is clearly seen that both the accuracy and the model loss looks great, and this is because of the use of transfer learning which helps in reducing the training time, and also in attaining the high performance with a small number of training data and number of epochs.

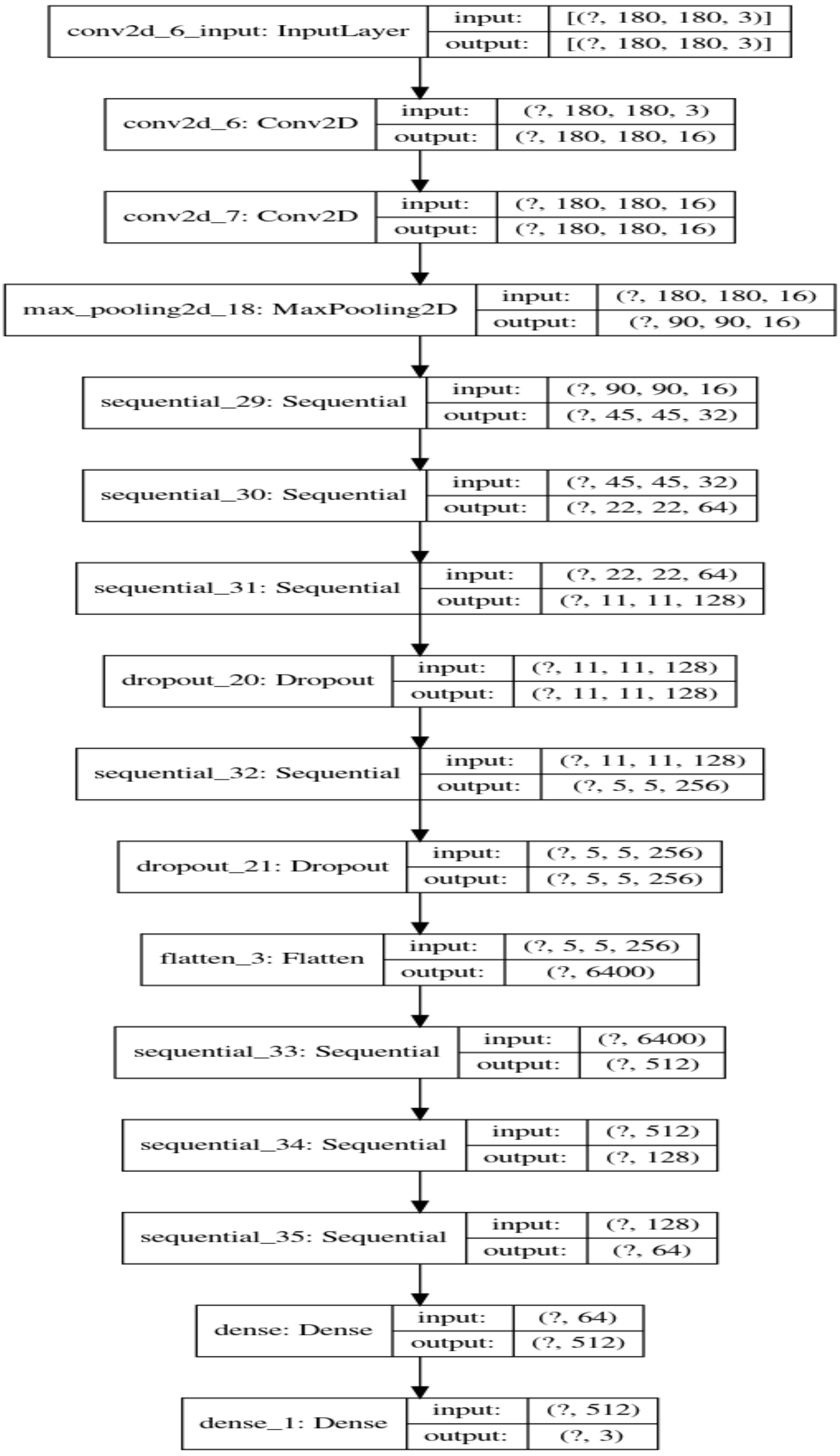


Figure 4. 4: The architecture of the proposed model

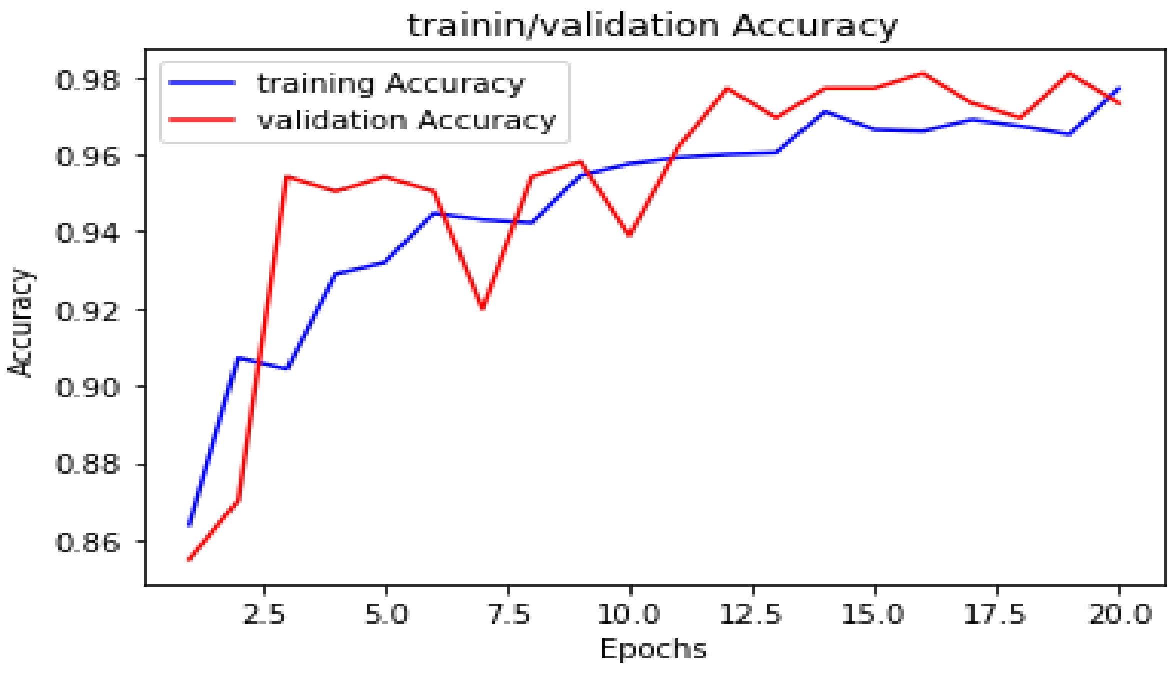


Figure 4.5: Second Model Accuracy per Epoch

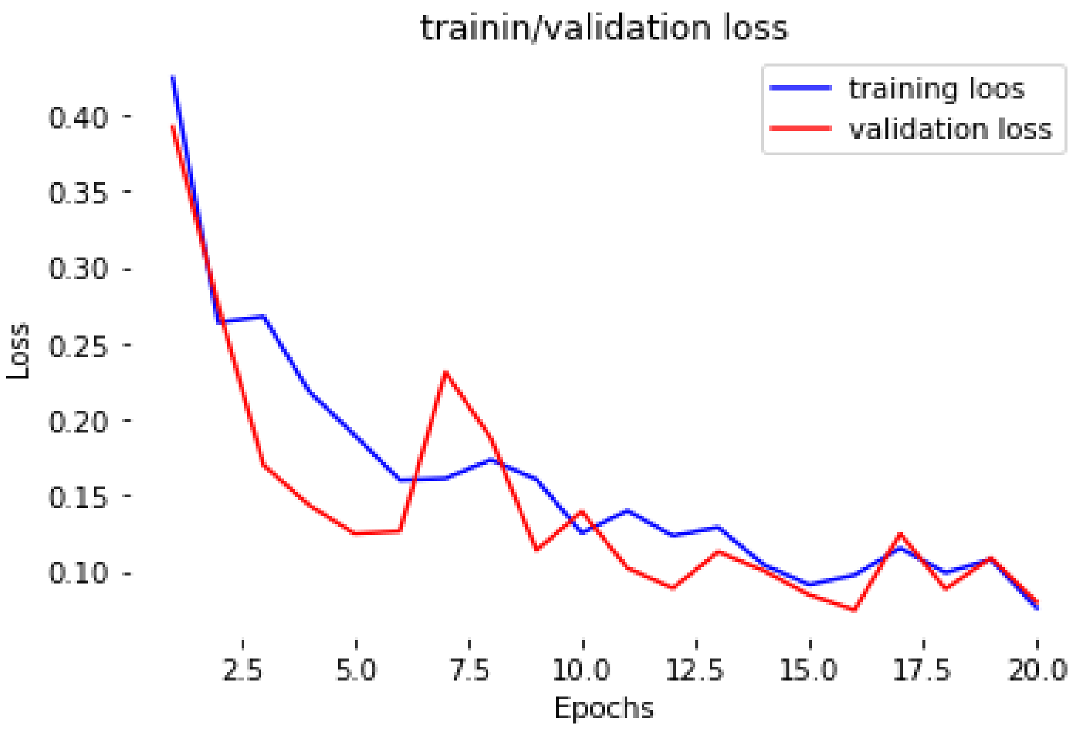
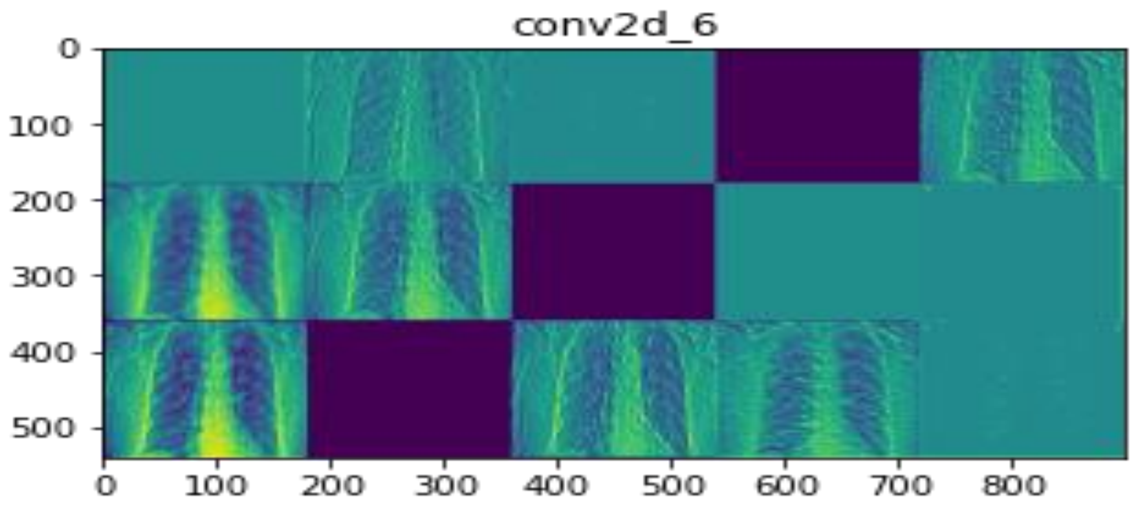


Figure 4.6: Second Model Loos per Epoch

Figure 4.7 shows the activation map of the convolutional layers of the network. The activation maps show what a network learned at a particular layer. For instance, the Conv2d_6 layer shows the high-level features learn by the network, and the sequential_30 layer shows low-level features which are more specific features of the classification classes in the data. The deeper the network the more and more specific features to be learned for each class by the network.

Figure 4.8 shows the Grad_CAM of COVID-19, and viral Pneumonia cases. The Grad_CAM helps us to visualize where exactly the model looks to perform the prediction on each and every X-Ray image. The first 10 X-Ray images are for COVID-19 cases, and the last 10 X-Ray images are for viral pneumonia cases. In all the X-Ray images, the part that is rainbow/yellowish in color is the most important part of the image used by the model in making decisions, while the parts with purple color are less important to the model for the making decision. Only 20 X-Ray images are shown in figure 5.4, more X-Ray images can be shown for a clear understanding of each case predicted by the model.



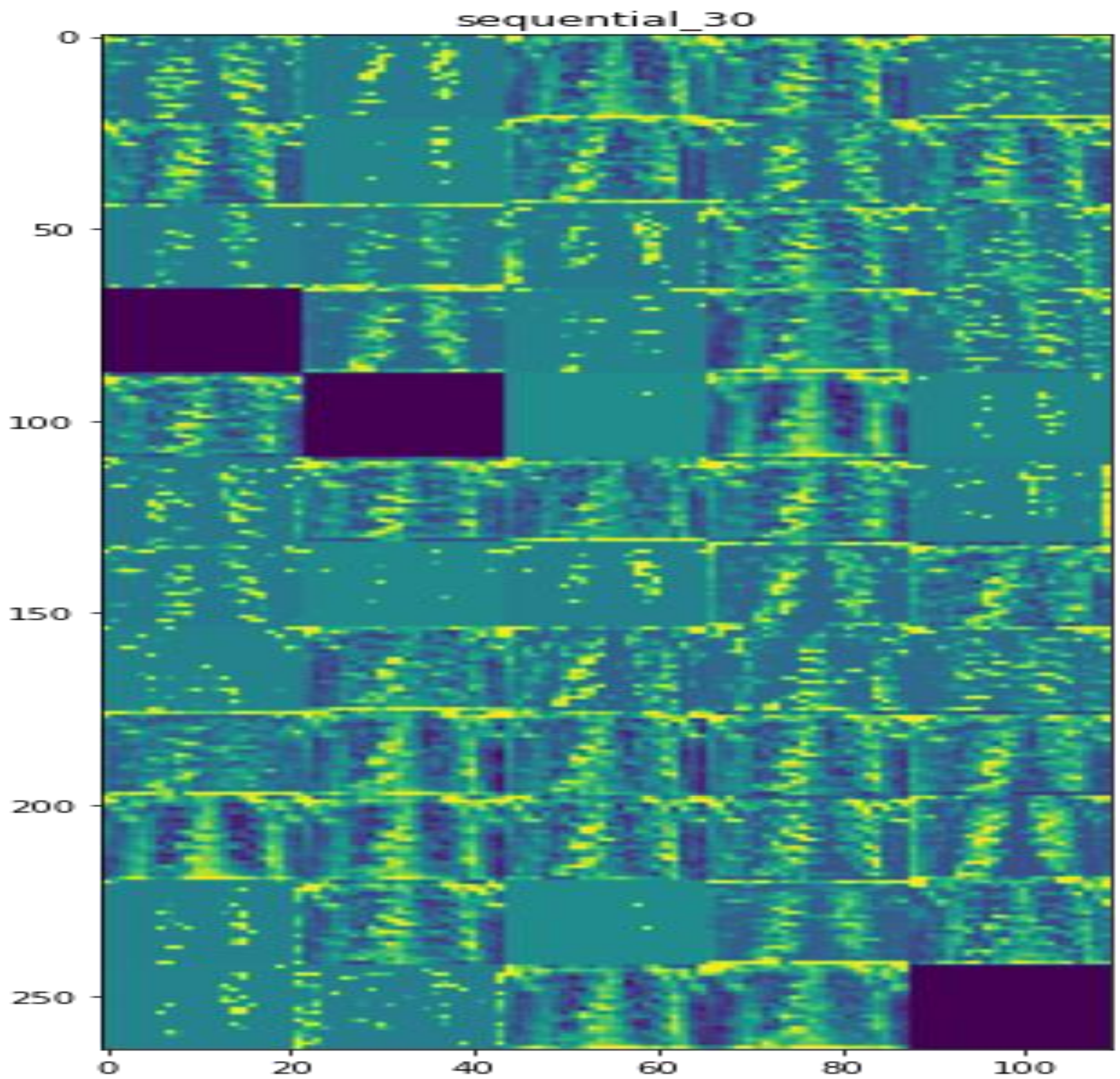


Figure 4. 7: shows the activation map in some convolutional layers of the network

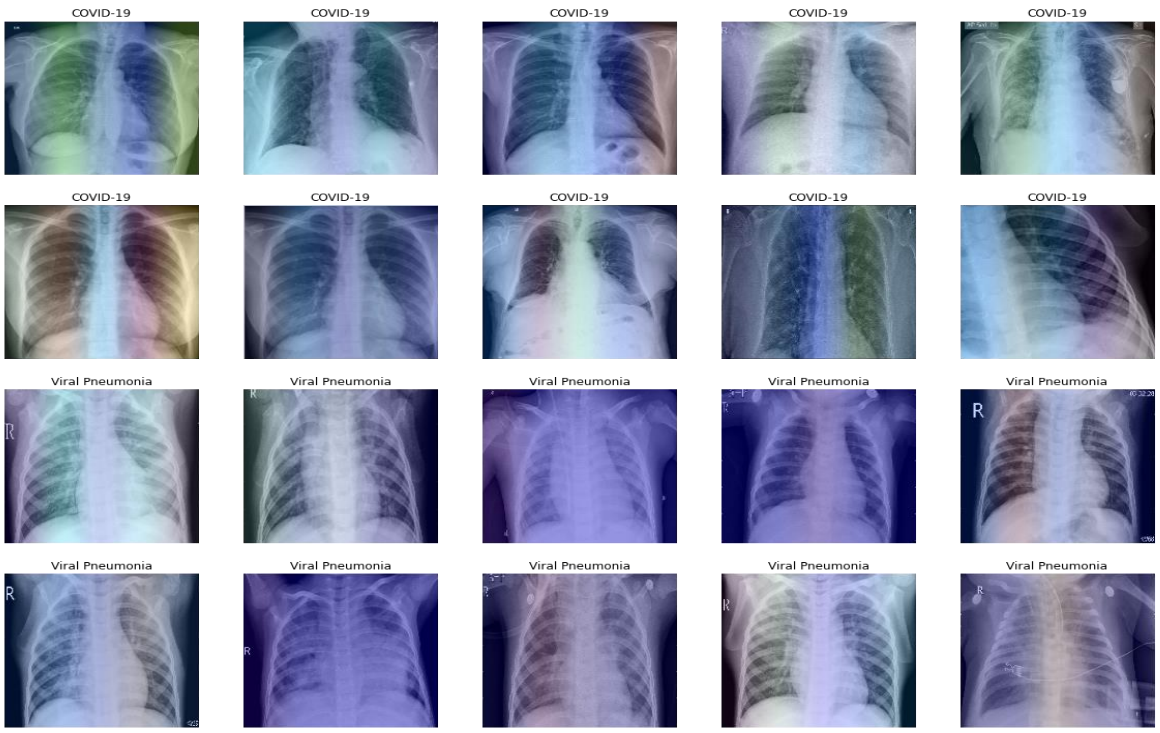


Figure 4. 8: shows the Grad_CAM of COVID-19, and viral Pneumonia cases

Table 4.3: Dataset splitting for training, validation, and testing

Cases	Training sets	Validation set	Testing set	Total
Normal	1,082	118	141	1,341
COVID-19	186	12	21	219
Pneumonia	1,084	132	129	1,345
Total	2,352	262	291	2,905

Table 4.4 shows the result of the model on test data. The result below is the model evaluation on test data, the table provided the recall, precision, F1 score, and accuracy which are 0.979, 0.982, 0.980, 0.982 respectively archived by the model, which are remarkable performance.

Table 4.4: Second Model test results

	Epoch	**Recall**	**Precision**	**F1 Score**	**Accuracy**	**Loss**
Second Model	20	0.979	0.982	0.980	0.982	0.045

5.4 Comparison

Table 4.5: Comparative performance on test data for this research, and other state-of-the-art approaches

Authors (Year)	**Recall (%)**	**Precision (%)**	**Specificity (%)**	**F1 Score (%)**	**Accuracy (%)**
Chowdhury et al., (2020)	97.94	97.95	98.80	97.94	97.94
Cavallo et al., (2020)	-	-	90	-	90.8
Lin & Lee, (2020)	92.6	89.7	-	-	93.1
Rajpal, et al., (2020)	-	-	-	95	94.4

Echtioui,et al., (2020)	88.33	91	-	89.66	91.34
Pham, (2020)	-	95	98	96	95
Neural et al., (2020)	-	-	100	-	96.9
Alotaibi, (2020)	97.42	97.42	-	97.23	98.3
This research	**97.94**	**98.27**	**-**	**98.0**	**98.28**

CHAPTER 5
CONCLUSION

To conclude, the research work was aimed at developing a system that will help radiologists in diagnosing COVID-19 and pneumonia cases easily in this trying time of the pandemic. Therefore, CNN algorithms were used as shown in the methodology section for the detection of these viruses as early as possible. Due to the lack of COVID-19 data, we have shown how the transfer learning approach was used to bridge the gap of these issues. As explained, two CNN algorithms were trained with two unique datasets; the first model was trained for binary classification (pneumonia/normal) on the first dataset that contains only pneumonia cases and normal CXR images. While the second model with the help of the transfer learning method uses the first model as the base model and trained on the second dataset that contained COVID-19, pneumonia, and normal cases images for three classes classification (COVID-19, pneumonia, and normal). The implementation result of the model that diagnose COVID-19 and pneumonia achieved the performance of 98.3, 97.9, 98.3, and 98.0 Accuracy, Recall, precision, and F1_scores respectively, hence, the proposed model proved to be efficient in diagnosing COVID-19 and pneumonia cases. A convolutional neural network (CNN) is known as a black box, hence, class activation map of some convolutional layers were shown to help understand what the model learn at a particular layer. Grad_CAM is also shown to help us know where exactly the model is looking at on image data to perform the classification task. At the end of the research, some of the other author’s results were compared with this research work result, and this work archived a high performance than the others.

REFERENCES

Abiyev, R. H., Khaleel, M., & Ma'aitah, S. (2018). *Deep Convolutional Neural Networks for Chest Diseases Detection*. https://doi.org/10.1155/2018/4168538

Alotaibi, A. (2020). Transfer Learning for Detecting Covid-19 Cases Using Chest X-Ray Images. *International Journal of Machine Learning and Networked Collaborative Engineering*, *4*(1), 21–29. https://doi.org/10.30991/ijmlnce.2020v04i01.003

Antin, B., Kravitz, J., & Martayan, E. (2017). *Detecting Pneumonia in Chest X-Rays with Supervised Learning*. 1–5.

Ayan, E., & Ünver, H. M. (2019). Diagnosis of pneumonia from chest X-ray images using deep learning. *2019 Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science, EBBT 2019*, 1–5. https://doi.org/10.1109/EBBT.2019.8741582

Cavallo, A. U., Troisi, J., Forcina, M., Mari, P., Forte, V., Sperandio, M., … Geraci, F. (2020). *Texture Analysis in the Evaluation of Covid-19 Pneumonia in Chest X-Ray Images: a Proof of Concept Study*. 1–12. https://doi.org/10.21203/rs.3.rs-37657/v1

Chowdhury, M. E. H., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M. A., Mahbub, Z. Bin, … Islam, M. T. (2020). Can AI Help in Screening Viral and COVID-19 Pneumonia? *IEEE Access*, *8*, 132665–132676. https://doi.org/10.1109/ACCESS.2020.3010287

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 1–14.

Duchi, J. C., Bartlett, P. L., & Wainwright, M. J. (2012). Randomized smoothing for (parallel) stochastic optimization. *Proceedings of the IEEE Conference on Decision and Control*, *12*, 5442–5444. https://doi.org/10.1109/CDC.2012.6426698

Echtioui, A., Zouch, W., Ghorbel, M., Mhiri, C., & Hamam, H. (2020). Detection Methods of COVID-19. *SLAS Technology*. https://doi.org/10.1177/2472630320962002

Er, O., Sertkaya, C., Temurtas, F., & Tanrikulu, A. C. (2009). A comparative study on chronic obstructive pulmonary and pneumonia diseases diagnosis using neural networks and artificial immune system. *Journal of Medical Systems*, *33*(6), 485–492.

https://doi.org/10.1007/s10916-008-9209-x

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

Gunraj, H., Wang, L., & Wong, A. (2020). *COVIDNet-CT: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest CT Images*. 1–12. Retrieved from http://arxiv.org/abs/2009.05383

Hammoudi, K., Benhabiles, H., Melkemi, M., Dornaika, F., Arganda-Carreras, I., Collard, D., & Scherpereel, A. (2020). *Deep Learning on Chest X-ray Images to Detect and Evaluate Pneumonia Cases at the Era of COVID-19*. 1–6. Retrieved from http://arxiv.org/abs/2004.03399

Han, Z., Wei, B., Hong, Y., Li, T., Cong, J., Zhu, X., … Zhang, W. (2020). Accurate Screening of COVID-19 Using Attention-Based Deep 3D Multiple Instance Learning. *IEEE Transactions on Medical Imaging*, *39*(8), 2584–2594. https://doi.org/10.1109/TMI.2020.2996256

Jaiswal, A. K., Tiwari, P., Kumar, S., Gupta, D., Khanna, A., & Rodrigues, J. J. P. C. (2019). Identifying pneumonia in chest X-rays: A deep learning approach. *Measurement: Journal of the International Measurement Confederation*, *145*, 511–518. https://doi.org/10.1016/j.measurement.2019.05.076

Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C. S., Liang, H., Baxter, S. L., … Zhang, K. (2018). Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. *Cell*, *172*(5), 1122-1131.e9. https://doi.org/10.1016/j.cell.2018.02.010

Khademi, G., & Simon, D. (2019). Convolutional neural networks for environmentally aware locomotion mode recognition of lower-limb amputees. *ASME 2019 Dynamic Systems and Control Conference, DSCC 2019*, *1*(June). https://doi.org/10.1115/DSCC2019-9180

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Ko, H., Chung, H., Kang, W. S., Kim, K. W., Shin, Y., Kang, S. J., … Lee, J. (2020). COVID-19 pneumonia diagnosis using a simple 2d deep learning framework with a single chest CT image: Model development and validation. *Journal of Medical Internet Research*, *22*(6), 1–13. https://doi.org/10.2196/19569

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323. https://doi.org/10.1109/5.726791

Li, L., Qin, L., Xu, Z., Yin, Y., Wang, X., Kong, B., … Xia, J. (2020). Using Artificial Intelligence to Detect COVID-19 and Community-acquired Pneumonia Based on Pulmonary CT: Evaluation of the Diagnostic Accuracy. *Radiology*, *296*(2), E65–E71. https://doi.org/10.1148/radiol.2020200905

Li, Q., Guan, X., Wu, P., Wang, X., Zhou, L., Tong, Y., … Feng, Z. (2020). Early transmission dynamics in Wuhan, China, of novel coronavirus-infected pneumonia. *New England Journal of Medicine*, *382*(13), 1199–1207. https://doi.org/10.1056/NEJMoa2001316

Lin, T. C., & Lee, H. C. (2020). Covid-19 chest radiography images analysis based on integration of image preprocess, guided grad-CAM, machine learning and risk management. *ACM International Conference Proceeding Series*, 281–288. https://doi.org/10.1145/3418094.3418096

Lyon, R. F., & Lyon, R. F. (2017). Neural Networks for Machine Learning. *Human and Machine Hearing*, 419–440. https://doi.org/10.1017/9781139051699.031

Neural, A. C., & Cnn, N. (2020). *COVID-19 DISEASE DIAGNOSIS FROM RADIOLOGY DATA WITH DEEP LEARNING*.

Pham, T. D. (2021). Classification of COVID - 19 chest X - rays with deep learning : new models or fine tuning ? *Health Information Science and Systems*. https://doi.org/10.1007/s13755-020-00135-3

Pneumonia | Home | CDC. (n.d.).

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

Rajaraman, S., Siegelman, J., Alderson, P. O., Folio, L. S., Folio, L. R., & Antani, S. K. (2020). Iteratively Pruned Deep Learning Ensembles for COVID-19 Detection in Chest X-Rays. *IEEE Access*, *8*, 115041–115050. https://doi.org/10.1109/ACCESS.2020.3003810

Rajpal, S., Rajpal, A., Lakhyani, N., & Kumar, N. (2020). COV-ELM classifier: An Extreme Learning Machine based identification of COVID-19 using Chest X-Ray Images. *ArXiv*, *2019*.

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., … Ng, A. Y. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*. Retrieved from https://stanfordmlgroup.

Rosenberg, L., Lungren, M., Halabi, S., Willcox, G., Baltaxe, D., & Lyons, M. (2019). Artificial Swarm Intelligence employed to Amplify Diagnostic Accuracy in Radiology. *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2018*, (Ml), 1186–1191. https://doi.org/10.1109/IEMCON.2018.8614883

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Saraiva, A. A., Santos, D. B. S., Costa, N. C., Sousa, J. V. M., Fonseca Ferreira, N. M., Valente, A., & Soares, S. (2019). Models of learning to classify X-ray images for the detection of pneumonia using neural networks. *BIOIMAGING 2019 - 6th International Conference on Bioimaging, Proceedings; Part of 12th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2019*, (Biostec), 76–83. https://doi.org/10.5220/0007346600760083

Taylor, L., & Nitschke, G. (2017). *Improving Deep Learning using Generic Data Augmentation*. Retrieved from http://arxiv.org/abs/1708.06020

WHO Coronavirus Disease (COVID-19) Dashboard | WHO Coronavirus Disease (COVID-19) Dashboard. (n.d.).

World Health Organization. (2001). *Standardization of interpretation of chest radiographs for the diagnosis of pneumonia in children / World Health Organization Pneumonia*

Vaccine Trial Investigators' Group. Retrieved from http://www.who.int/iris/handle/10665/66956

APPENDICES

APPENDIX 1

FIRST MODEL CODE

```
import re
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from kaggle_datasets import KaggleDatasets
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)

print(tf.__version__)



AUTOTUNE = tf.data.experimental.AUTOTUNE
GCS_PATH = KaggleDatasets().get_gcs_path()
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
IMAGE_SIZE = [180, 180]
EPOCHS = 25



filenames = tf.io.gfile.glob(str(GCS_PATH + '/chest_xray/trai
n/*/*'))
filenames.extend(tf.io.gfile.glob(str(GCS_PATH + '/chest_xray
/val/*/*')))


train_filenames, val_filenames = train_test_split(filenames,
test_size=0.2)
```

```
COUNT_NORMAL = len([filename for filename in train_filenames
if "NORMAL" in filename])
print("Normal images count in training set: " + str(COUNT_NOR
MAL))

COUNT_PNEUMONIA = len([filename for filename in train_filenam
es if "PNEUMONIA" in filename])
print("Pneumonia images count in training set: " + str(COUNT_
PNEUMONIA))

train_list_data = tf.data.Dataset.from_tensor_slices(train_fi
lenames)
val_list_data = tf.data.Dataset.from_tensor_slices(val_filena
mes)


for f in train_list_data.take(5):
    print(f.numpy())


TRAIN_IMG_COUNT = tf.data.experimental.cardinality(train_list
_data).numpy()
print("Training images count: " + str(TRAIN_IMG_COUNT))

VAL_IMG_COUNT = tf.data.experimental.cardinality(val_list_dat
a).numpy()
print("Validating images count: " + str(VAL_IMG_COUNT))


CLASS_NAMES = np.array([str(tf.strings.split(item, os.path.se
p)[-1].numpy())[2:-1]
                        for item in tf.io.gfile.glob(str(GCS_
PATH + "/chest_xray/train/*"))])
CLASS_NAMES


def get_label(file_path):
    # convert the path to a list of path components
    parts = tf.strings.split(file_path, os.path.sep)
    # The second to last is the class-directory
    return parts[-2] == "PNEUMONIA"


def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_jpeg(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,
1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, IMAGE_SIZE)


def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label


train_data= train_list_data.map(process_path, num_parallel_ca
lls=AUTOTUNE)
val_data= val_list_data.map(process_path, num_parallel_calls=
AUTOTUNE)


for image, label in train_ds.take(1):
    print("Image shape: ", image.numpy().shape)
    print("Label: ", label.numpy())

testl_list_data = tf.data.Dataset.list_files(str(GCS_PATH + '
/chest_xray/test/*/*'))
TEST_IMAGE_COUNT = tf.data.experimental.cardinality(testl_lis
t_data).numpy()

test_data= testl_list_data.map(process_path, num_parallel_cal
ls=AUTOTUNE)
test_data= test_ds.batch(BATCH_SIZE)

TEST_IMAGE_COUNT


def prepare_for_training(ds, cache=True, shuffle_buffer_size=
1000):
    # This is a small dataset, only load it once, and keep it
 in memory.
    # use `.cache(filename)` to cache preprocessing work for
datasets that don't
    # fit in memory.
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()

    ds = ds.shuffle(buffer_size=shuffle_buffer_size)

    # Repeat forever
    ds = ds.repeat()

    ds = ds.batch(BATCH_SIZE)

    # `prefetch` lets the dataset fetch batches in the backgr
ound while the model
    # is training.
    ds = ds.prefetch(buffer_size=AUTOTUNE)

    return ds


train_data= prepare_for_training(train_ds)
val_data= prepare_for_training(val_ds)


image_batch, label_batch = next(iter(train_ds))


def show_batch(image_batch, label_batch):
    plt.figure(figsize=(10,10))
    for n in range(25):
        ax = plt.subplot(5,5,n+1)
        plt.imshow(image_batch[n])
        if label_batch[n]:
            plt.title("PNEUMONIA")
        else:
            plt.title("NORMAL")
        plt.axis("off")

show_batch(image_batch.numpy(), label_batch.numpy())


def conv_block(filters):
    block = tf.keras.Sequential([
        tf.keras.layers.SeparableConv2D(filters, 3, activatio
n='relu', padding='same'),
        tf.keras.layers.SeparableConv2D(filters, 3, activatio
n='relu', padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D()
    ]
    )

    return block


def dense_block(units, dropout_rate):
    block = tf.keras.Sequential([
        tf.keras.layers.Dense(units, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(dropout_rate)
    ])

    return block


def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3
)),

        tf.keras.layers.Conv2D(16, 3, activation='relu', padd
ing='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padd
ing='same'),
        tf.keras.layers.MaxPool2D(),

        conv_block(32),
        conv_block(64),

        conv_block(128),
        tf.keras.layers.Dropout(0.2),

        conv_block(256),
        tf.keras.layers.Dropout(0.2),

#          ########## added layer
#          conv_block(512),
#          tf.keras.layers.Dropout(0.2),

        tf.keras.layers.Flatten(),
#          dense_block(1024, 0.8), ######## added layer
```

```
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),

        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    return model


initial_bias = np.log([COUNT_PNEUMONIA/COUNT_NORMAL])
initial_bias

weight_for_0 = (1 / COUNT_NORMAL)*(TRAIN_IMG_COUNT)/2.0
weight_for_1 = (1 / COUNT_PNEUMONIA)*(TRAIN_IMG_COUNT)/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))


with strategy.scope():
    model = build_model()

    METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=METRICS
    )


history = model.fit(
    train_ds,
    steps_per_epoch=TRAIN_IMG_COUNT // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=val_ds,
    validation_steps=VAL_IMG_COUNT // BATCH_SIZE,
    class_weight=class_weight,)
```

```
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("PneomoNet
Best.h5",
                                                           save_best
_only=True)



early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience
=10,
                                                            restore_
best_weights=True)


def exponential_decay(lr0, s):
    def exponential_decay_fn(epoch):
        return lr0 * 0.1 **(epoch / s)
    return exponential_decay_fn


exponential_decay_fn = exponential_decay(0.01, 20)

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(expon
ential_decay_fn)


history = model.fit(
    train_ds,
    steps_per_epoch=TRAIN_IMG_COUNT // BATCH_SIZE,
    epochs=100,
    validation_data=val_ds,
    validation_steps=VAL_IMG_COUNT // BATCH_SIZE,
    class_weight=class_weight,
    callbacks=[checkpoint_cb, early_stopping_cb, lr_scheduler
]
)


fig, ax = plt.subplots(1, 4, figsize=(20, 3))
ax = ax.ravel()


for i, met in enumerate(['precision', 'recall', 'accuracy', '
loss']):
    ax[i].plot(history.history[met])
    ax[i].plot(history.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)
    ax[i].legend(['train', 'val'])


loss, acc, prec, rec = model.evaluate(test_ds)

model.save('PneomoNetBest.h5')
```

APPENDIX 2

SECOND MODEL CODE

```
import re
import os
import random
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from kaggle_datasets import KaggleDatasets
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.layers import Dropout
from keras.applications.vgg16 import VGG16
import matplotlib.cm as cm
from sklearn.model_selection import train_test_split


try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)

print(tf.__version__)


AUTOTUNE = tf.data.experimental.AUTOTUNE
GCS_PATH = KaggleDatasets().get_gcs_path("covid19-
radiography-database")
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
IMAGE_SIZE = [180, 180]


filenames = tf.io.gfile.glob(str(GCS_PATH + '/COVID-
19 Radiography Database/COVID-19/*'))
filenames.extend(tf.io.gfile.glob(str(GCS_PATH + '/COVID-
19 Radiography Database/NORMAL/*')))
filenames.extend(tf.io.gfile.glob(str(GCS_PATH + '/COVID-
19 Radiography Database/Viral Pneumonia/*')))
```

```
random.seed(1337)
tf.random.set_seed(1337)
random.shuffle(filenames)


COUNT_NORMAL = len([filename for filename in filenames if "NO
RMAL" in filename])
print("Normal images count : " + str(COUNT_NORMAL))

COUNT_COVID = len([filename for filename in filenames if "/CO
VID-19/" in filename])
print("COVID-19 images count : " + str(COUNT_COVID))

COUNT_PNEUMONIA = len([filename for filename in filenames if
"Viral" in filename])
print("Pneumonia images count : " + str(COUNT_PNEUMONIA))


import seaborn as sns
# intialise data of lists.
data = {'Cases':['0', '1', '2'],
        'Cases_count':[COUNT_NORMAL, COUNT_COVID, COUNT_PNEUM
ONIA]
       }

# Create DataFrame
df = pd.DataFrame(data)

# Get the counts for each class in the data

plt.figure(figsize=(10,8))
sns.barplot(x=df.index, y= df['Cases_count'].values)
plt.title('Number of All the Data', fontsize=14)
plt.xlabel('Case type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(df.index)), ['Normal(0)', 'Covid19(1)',
'Pneumonia(2)'])
plt.show()

# Print the output.
print(df)


train_filenames, test_filenames = train_test_split(filenames,
 test_size=0.1)
```

```
train_filenames, val_filenames = train_test_split(train_filen
ames, test_size=0.1)


COUNT_NORMAL_TRAINING = len([filename for filename in train_f
ilenames if "NORMAL" in filename])
print("Normal images count in training set: " + str(COUNT_NOR
MAL_TRAINING))

COUNT_COVID_TRAINING = len([filename for filename in train_fi
lenames if "/COVID-19/" in filename])
print("COVID-
19 images count in training set: " + str(COUNT_COVID_TRAINING
))

COUNT_PNEUMONIA_TRAINING = len([filename for filename in trai
n_filenames if "Viral" in filename])
print("Pneumonia images count in training set: " + str(COUNT_
PNEUMONIA_TRAINING))


import seaborn as sns
# intialise data of lists.
data = {'Cases':['0', '1', '2'],
        'Cases_count':[COUNT_NORMAL_TRAINING, COUNT_COVID_TRA
INING, COUNT_PNEUMONIA_TRAINING]}

# Create DataFrame
df = pd.DataFrame(data)

# Get the counts for each class in training data

plt.figure(figsize=(10,8))
sns.barplot(x=df.index, y= df['Cases_count'].values)
plt.title('Number of Training cases', fontsize=14)
plt.xlabel('Case type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(df.index)), ['Normal(0)', 'Covid19(1)',
'Pneumonia(2)'])
plt.show()

# Print the output.
print(df)
```

```
COUNT_NORMAL_VALODATION = len([filename for filename in val_f
ilenames if "NORMAL" in filename])
print("Normal images count in validation set: " + str(COUNT_N
ORMAL_VALODATION))

COUNT_COVID_VALODATION = len([filename for filename in val_fi
lenames if "/COVID-19/" in filename])
print("COVID-
19 images count in validation set: " + str(COUNT_COVID_VALODA
TION))

COUNT_PNEUMONIA_VALODATION = len([filename for filename in va
l_filenames if "Viral" in filename])
print("Pneumonia images count in validation set: " + str(COUN
T_PNEUMONIA_VALODATION))

import seaborn as sns
# intialise data of lists.
data = {'Cases':['0', '1', '2'],
        'Cases_count':[COUNT_NORMAL_VALODATION, COUNT_COVID_V
ALODATION, COUNT_PNEUMONIA_VALODATION]}

# Create DataFrame
df = pd.DataFrame(data)

# Get the counts for each class in validation data

plt.figure(figsize=(10,8))
sns.barplot(x=df.index, y= df['Cases_count'].values)
plt.title('Number of Validation cases', fontsize=14)
plt.xlabel('Case type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(df.index)), ['Normal(0)', 'Covid19(1)',
'Pneumonia(2)'])
plt.show()

# Print the output.
print(df)


COUNT_NORMAL_TEST = len([filename for filename in test_filena
mes if "NORMAL" in filename])
print("Normal images count in test set: " + str(COUNT_NORMAL_
TEST))
```

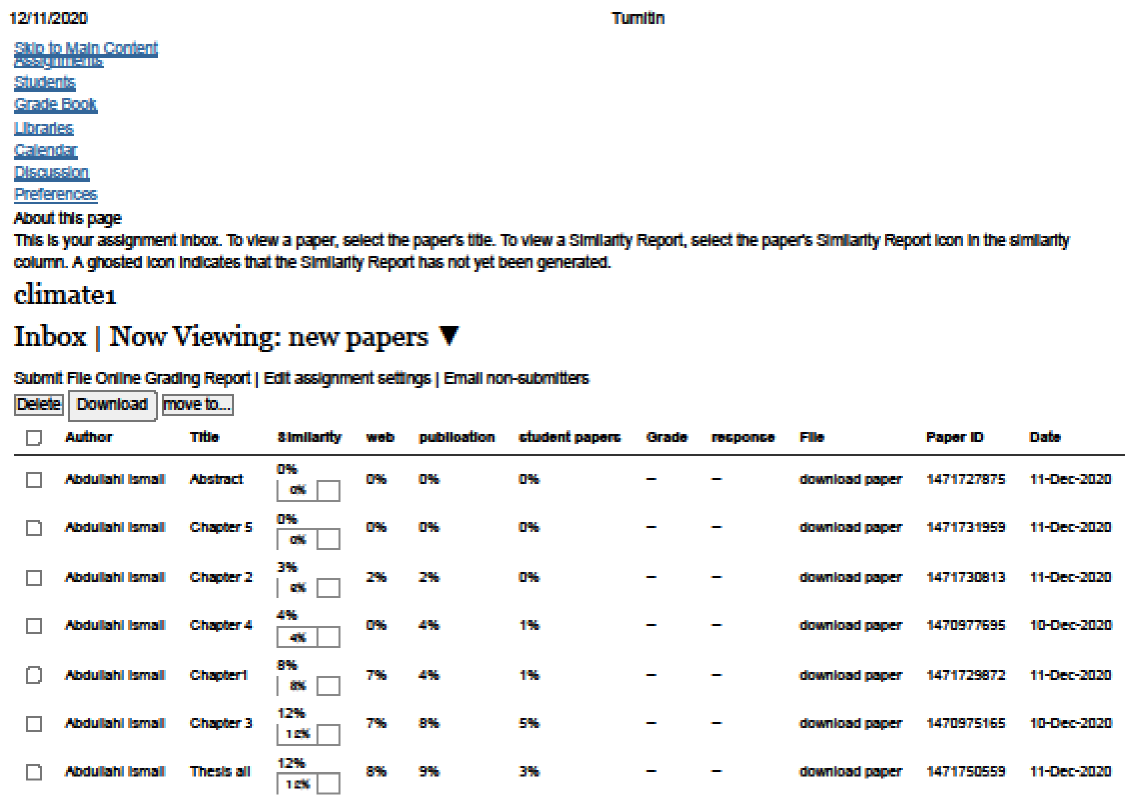
```
COUNT_COVID_TEST = len([filename for filename in test_filenam
es if "/COVID-19/" in filename])
print("COVID-
19 images count in test set: " + str(COUNT_COVID_TEST))

COUNT_PNEUMONIA_TEST = len([filename for filename in test_fil
enames if "Viral" in filename])
print("Pneumonia images count in test set: " + str(COUNT_PNEU
MONIA_TEST))

import seaborn as sns
# intialise data of lists.
data = {'Cases':['0', '1', '2'],
        'Cases_count':[COUNT_NORMAL_TEST, COUNT_COVID_TEST, C
OUNT_PNEUMONIA_TEST]}

# Create DataFrame
df = pd.DataFrame(data)

# Get the counts for each class in test data

plt.figure(figsize=(10,8))
sns.barplot(x=df.index, y= df['Cases_count'].values)
plt.title('Number of test cases', fontsize=14)
plt.xlabel('Case type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(df.index)), ['Normal(0)', 'Covid19(1)',
'Pneumonia(2)'])
plt.show()

# Print the output.
print(df)


train_list_data = tf.data.Dataset.from_tensor_slices(train_fi
lenames)
val_list_data = tf.data.Dataset.from_tensor_slices(val_filena
mes)
testl_list_data = tf.data.Dataset.from_tensor_slices(test_fil
enames)


TRAIN_IMG_COUNT = tf.data.experimental.cardinality(train_list
_data).numpy()
print("Training images count: " + str(TRAIN_IMG_COUNT))
```

```
VAL_IMG_COUNT = tf.data.experimental.cardinality(val_list_dat
a).numpy()
print("Validating images count: " + str(VAL_IMG_COUNT))

Test_IMG_COUNT = tf.data.experimental.cardinality(testl_list_
data).numpy()
print("Testing images count: " + str(Test_IMG_COUNT))

CLASSES = ['NORMAL', 'COVID-19', 'Viral Pneumonia']


def get_label(file_path):
    # convert the path to a list of path components
    parts = tf.strings.split(file_path, os.path.sep)
    # The second to last is the class-directory
    return parts[-2] == CLASSES


def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,
1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, IMAGE_SIZE)



def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label


train_data = train_list_data.map(process_path, num_parallel_c
alls=AUTOTUNE)
val_data = val_list_data.map(process_path, num_parallel_calls
=AUTOTUNE)
test_data = testl_list_data.map(process_path, num_parallel_ca
lls=AUTOTUNE)


def prepare_for_training(ds, cache=True):
    # This is a small dataset, only load it once, and keep it
 in memory.
    # use `.cache(filename)` to cache preprocessing work for
datasets that don't
    # fit in memory.
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()

    ds = ds.shuffle(buffer_size=1000)
    ds = ds.batch(BATCH_SIZE)

    if cache:
        ds = ds.prefetch(buffer_size=AUTOTUNE)

    return ds


train_data = prepare_for_training(train_data)
val_data = prepare_for_training(val_data)
test_data = prepare_for_training(test_data, False)


def show_batch(image_batch, label_batch):
    plt.figure(figsize=(10,10))
    for n in range(15):
        ax = plt.subplot(5,5,n+1)
        plt.imshow(image_batch[n])
        plt.title(CLASSES[np.argmax(label_batch[n])])
        plt.axis("off")


image_batch, label_batch = next(iter(train_data))
show_batch(image_batch.numpy(), label_batch.numpy())

early_stopping_cb = keras.callbacks.EarlyStopping(patience=5,
                                                  restore_bes
t_weights=True)


with strategy.scope():
    PneumoCovidNet = keras.models.load_model("../input/pneumo
netbest-model/PneomoNetBest.h5")
    PneumoCovidNet.pop()
    PneumoCovidNet.add(layers.Dense(512, activation='relu'))
#     PneumoCovidNet.add(layers.Dense(128, activation='relu')
)
#     PneumoCovidNet.add(layers.Dense(64, activation='relu'))
    PneumoCovidNet.add(keras.layers.Dense(3, activation='soft
max'))

    METRICS = [
        'accuracy',
        keras.metrics.Precision(name="precision"),
        keras.metrics.Recall(name="recall")
    ]

    PneumoCovidNet.compile(
        optimizer="adam",
        loss="categorical_crossentropy",
        metrics=METRICS,
    )


history = PneumoCovidNet.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    callbacks=[early_stopping_cb]
)

fig, ax = plt.subplots(2, 2, figsize=(15, 10))
ax = ax.ravel()


for i, met in enumerate(['precision', 'recall', 'accuracy', '
loss']):
    ax[i].plot(history.history[met])
    ax[i].plot(history.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)
    ax[i].legend(['train', 'val'])


PneumoCovidNet.evaluate(test_data, return_dict=True)

# PneumoCovidNet.save('PneumoCovid_Model.h5')
```

```
saved_Model = keras.models.load_model('../input/best-
model/PneumoCovid_Model.h5')
saved_Model2 = keras.models.load_model('../input/best-model-
v2/PneumoCovid_Model V2.h5')

saved_Model.evaluate(test_data, return_dict=True)
saved_Model2.evaluate(test_data, return_dict=True)


def get_intermediate_Activation(model, imge_path):

    img_path = imge_path
    from keras.preprocessing import image
    import numpy as np
    img = image.load_img(img_path, target_size=(180, 180))
    img_tensor = image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.
    import tensorflow as tf
    from tensorflow.keras import models
    layer_outputs = [layer.output for layer in model.layers[:
5]]
    activation_model = models.Model(inputs=model.input, outpu
ts=layer_outputs)
    activations = activation_model.predict(img_tensor)

    layer_names = []
    for layer in model.layers[:5]:
        layer_names.append(layer.name)
    images_per_row = 5
    for layer_name, layer_activation in zip(layer_names, acti
vations):
        n_features = layer_activation.shape[-1]
        size = layer_activation.shape[1]
        n_cols = n_features // images_per_row
        display_grid = np.zeros((size * n_cols, images_per_ro
w * size))
        for col in range(n_cols):
            for row in range(images_per_row):
                channel_image = layer_activation[0,
                                                 :, :,
                                                 col * images_per_row
 + row]
                channel_image -= channel_image.mean()
                channel_image /= channel_image.std()
                channel_image *= 64
                channel_image += 128
                channel_image = np.clip(channel_image, 0, 255
).astype('uint8')
                display_grid[col * size : (col + 1) * size,
                      row * size : (row + 1) * size] = channe
l_image
        scale = 1. / size
        plt.figure(figsize=(scale * display_grid.shape[1],
        scale * display_grid.shape[0]))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis
')

get_intermediate_Activation(saved_Model, '../input/covid19-
radiography-database/COVID-19 Radiography Database/COVID-
19/COVID-19 (1).png')


# ploting the NeumoNet2 architecture
from keras.utils import plot_model
plot_model(saved_Model, to_file='PneumoCovidNet.png', show_sh
apes=True)

# last convolution block of the model
saved_Model.layers[7].layers


def get_img_array(img_path, size=IMAGE_SIZE):
    img = keras.preprocessing.image.load_img(img_path, target
_size=size)
    # `array` is a float32 NumPy array
    array = keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch
"
    # of size (1, 180, 180, 3)
    array = np.expand_dims(array, axis=0) / 255.0
    return array


def make_gradcam_heatmap(img_array, model):
    # First, we create a model that maps the input image to t
he activations
    # of the last conv layer
    last_conv_layer = model.layers[7]
    last_conv_layer_model = keras.Model(model.inputs, last_co
nv_layer.output)

    # Mark the classifying layers
    classifier_layers = model.layers[-6:]

    # Second, we create a model that maps the activations of
the last conv
    # layer to the final class predictions
    classifier_input = keras.Input(shape=last_conv_layer.outp
ut.shape[1:])
    x = classifier_input
    for classifier_layer in classifier_layers:
        x = classifier_layer(x)
    classifier_model = keras.Model(classifier_input, x)

    # Then, we compute the gradient of the top predicted clas
s for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make
 the tape watch it
        last_conv_layer_output = last_conv_layer_model(img_ar
ray)
        tape.watch(last_conv_layer_output)
        # Compute class predictions
        preds = classifier_model(last_conv_layer_output)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]

    # This is the gradient of the top predicted class with re
gard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, last_conv_layer_
output)

    # This is a vector where each entry is the mean intensity
 of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the t
op predicted class
    last_conv_layer_output = last_conv_layer_output.numpy()[0
]
```

```
    pooled_grads = pooled_grads.numpy()
    for i in range(pooled_grads.shape[-1]):
        last_conv_layer_output[:, :, i] *= pooled_grads[i]

    # The channel-wise mean of the resulting feature map
    # is our heatmap of class activation
    heatmap = np.mean(last_conv_layer_output, axis=-1)

    # For visualization purpose, we will also normalize the h
eatmap between 0 & 1
    heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    return heatmap



def superimposed_cam(file_path):
    # Prepare image
    img_array = get_img_array(file_path)

    # Generate class activation heatmap
    heatmap = make_gradcam_heatmap(
        img_array, saved_Model
    )

    # Rescale the original image
    img = img_array * 255
    # We rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # We use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # We use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # We create an image with RGB colorized heatmap
    jet_heatmap = keras.preprocessing.image.array_to_img(jet_
heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape
[0]))
    jet_heatmap = keras.preprocessing.image.img_to_array(jet_
heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * 0.4 + img
    superimposed_img = keras.preprocessing.image.array_to_img
(superimposed_img[0])

    return superimposed_img, CLASSES[np.argmax(saved_Model.pr
edict(img_array))]


covid_filenames = tf.io.gfile.glob('../input/covid19-
radiography-database/COVID-19 Radiography Database/COVID-
19/*')
pneumonia_filenames = tf.io.gfile.glob('../input/covid19-
radiography-database/COVID-
19 Radiography Database/Viral Pneumonia/*')


plt.figure(figsize=(20,20))
for n in range(10):
    ax = plt.subplot(5,5,n+1)
    img, pred = superimposed_cam(covid_filenames[n])
    plt.imshow(img)
    plt.title(pred)
    plt.axis("off")
for n in range(10, 20):
    ax = plt.subplot(5,5,n+1)
    img, pred = superimposed_cam(pneumonia_filenames[n])
    plt.imshow(img)
    plt.title(pred)
    plt.axis("off")
```

APPENDIX 3



12/11/2020 Turnitin

Skip to Main Content
Assignments
Students
Grade Book
Libraries
Calendar
Discussion
Preferences

About this page
This is your assignment inbox. To view a paper, select the paper's title. To view a Similarity Report, select the paper's Similarity Report icon in the similarity column. A ghosted icon indicates that the Similarity Report has not yet been generated.

climate1

Inbox | Now Viewing: new papers ▼

Submit File Online Grading Report | Edit assignment settings | Email non-submitters

Delete Download move to...

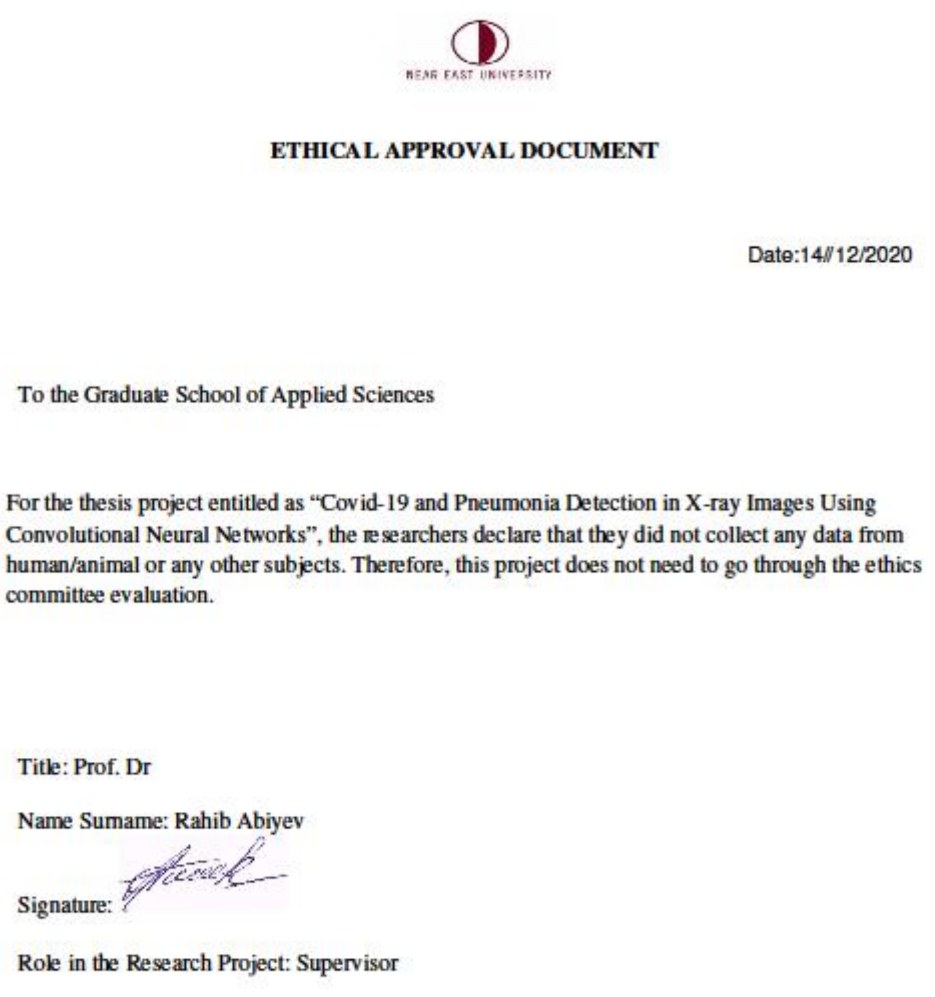
☐	Author	Title	Similarity	web	publication	student papers	Grade	response	File	Paper ID	Date
☐	Abdullahi Ismail	Abstract	0% 0%	0%	0%	0%	–	–	download paper	1471727875	11-Dec-2020
☐	Abdullahi Ismail	Chapter 5	0% 0%	0%	0%	0%	–	–	download paper	1471731959	11-Dec-2020
☐	Abdullahi Ismail	Chapter 2	3% 3%	2%	2%	0%	–	–	download paper	1471730813	11-Dec-2020
☐	Abdullahi Ismail	Chapter 4	4% 4%	0%	4%	1%	–	–	download paper	1470977695	10-Dec-2020
☐	Abdullahi Ismail	Chapter1	8% 8%	7%	4%	1%	–	–	download paper	1471729872	11-Dec-2020
☐	Abdullahi Ismail	Chapter 3	12% 12%	7%	8%	5%	–	–	download paper	1470975165	10-Dec-2020
☐	Abdullahi Ismail	Thesis all	12% 12%	8%	9%	3%	–	–	download paper	1471750559	11-Dec-2020

Prof.Dr.Rahib Abiyev
Chair of Computer Engineering
Department
11.12.20

[illegible]

https://www.turnitin.com/t_inbox.asp?aid=55438470&lang=en_us&session-id=656de803b4bc49f680e33605484f0f30 1/1

APPENDIX 1: Ethical Approval letter



NEAR EAST UNIVERSITY

ETHICAL APPROVAL DOCUMENT

Date:14//12/2020

To the Graduate School of Applied Sciences

For the thesis project entitled as "Covid-19 and Pneumonia Detection in X-ray Images Using Convolutional Neural Networks", the researchers declare that they did not collect any data from human/animal or any other subjects. Therefore, this project does not need to go through the ethics committee evaluation.

Title: Prof. Dr

Name Surname: Rahib Abiyev

Signature: [illegible]

Role in the Research Project: Supervisor