# Detection of Pneumonia from Chest X-Rays using Deep Learning Models (CNNs)

by

Pavithra Ramasubramanian Ramachandran

This thesis has been submitted in partial fulfillment for the degree of Master of Science in Artificial Intelligence

in the
Faculty of Engineering and Science
Department of Computer Science

May 2020

# Declaration of Authorship

This report, Detection of Pneumonia from Chest X-Rays using Deep Learning Models (CNNs), is submitted in partial fulfillment of the requirements of Master of Science in Artificial Intelligence at Cork Institute of Technology. I, Pavithra Ramasubramanian Ramachandran, declare that this thesis titled, Detection of Pneumonia from Chest X-Rays using Deep Learning Models (CNNs)and the work represents substantially the result of my own work except where explicitly indicated in the text. This report may be freely copied and distributed provided the source is explicitly acknowledged. I confirm that:

- This work was done wholly or mainly while in candidature Master of Science in Artificial Intelligence at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

CORK INSTITUTE OF TECHNOLOGY

## *Abstract*

Faculty of Engineering and Science
Department of Computer Science

Master of Science

by Pavithra Ramasubramanian Ramachandran

The primary aim of this project is to detect pneumonia by providing chest X-rays as input to a deep learning model that uses Classification Convolution Neural Networks to classify these X-ray images into 'Normal' or 'Pneumonia' affected. Pneumonia could become fatal if unattended for more than 5-7 days. According to UNICEF/WHO, this fatal disease is the leading cause of death in children below the age of 5[1]. To address this issue, a computer-aided method using Convolutional Neural Networks to detect pneumonia-affected lungs would enable efficient and quicker treatment of the condition. Many attempts have been made at using CNNs to address this problem, however, a maximum of only 85% accuracy has been achieved till date in this aspect. In this project, the Xception and the VGG-16 models are fine-tuned to a dataset of chest X-rays images in order to predict whether the image is normal or pneumonia affected [2]. Additional variations like ensembling, varying model parameters,varying input dataset, etc. are used to enhance the performance of the model and make it a robust one that can handle any similar data.

# *Acknowledgements*

I hereby thank the Almighty Lord for having given me the strength and direction to pursue my project with complete dedication and enthusiasm.

I also owe it to the Department of Computer Science, the Faculty and all my fellow classmates for being a part of this wonderful journey throughout my thesis.

Finally, I am extremely grateful to Professor Triona McSweeney for having contributed in abundance to my thesis. If not for her guidance and push, I may not have been able to pursue my thesis with commitment and zeal.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **CAD** | **C**omputer **A**ided **D**etection |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **RGB** | **R**ed **G**reen **B**lue |
| **SIFT** | **S**cale - **I**nvariant **F**eature **T**ransformation |
| **SURF** | **S**peed **U**p **R**obust **F**eature |
| **DoG** | **D**ifference **o**f **G**uassians |
| **D** | **D**imensional |
| **R-CNN** | **R**egion - **C**onvolutional **N**eural **N**etwork |
| **VGG-16** | **V**isual **G**eometry **G**roup - 16 |
| **COCO** | **C**ommon **O**bjects in **CO**ntext |
| **OpenCV** | **Open** Source **C**omputer **V**ision Library |
| **PIL** | **P**ython **I**maging **L**ibrary |
| **SGD** | **S**tochiastic **G**radient **D**escent |
| **GPU** | **G**raphical **P**rocessing **U**nit |

*Dedicated to all the health workers working hard to save lives on the frontline.*

# Chapter 1

# Introduction

Pneumonia is defined as "inflammation of the lung caused by bacteria, in which the air sacs (alveoli) become filled with inflammatory cells and the lungs become solid"[12]. Diagnosis of pneumonia is done primarily using chest X-rays both frontal and lateral, patient history and many other medical symptoms and parameters. Our objective is to detect the presence of the disease in frontal chest X-rays, thereby partially, if not completely indicate to medical specialists that the chest X-ray is of considerable concern.

## 1.1 Motivation

I personally feel the need to improve healthcare systems and disease diagnosis systems because while I was researching rigorously about the thesis topics that I could choose to do my Master's thesis, I was somehow already sure that it should be in the healthcare domain. It just felt more contributive to the society we live in. But when it came to narrowing down to one health issue, the impact that this fatal disease 'pneumonia' has had on children and infants struck a deep chord in me. Being a computer science engineer and well-trained in Artificial Intelligence systems, I believed the best way to contribute to the field of healthcare would be to technologically provide solutions to what the system lacks at.

Pneumonia could be very fatal if gone unnoticed or undetected. It is the reason for highest deaths amongst children below the age of 5 [1]. Although radiologists and doctors can predict pneumonia cases with much better precision, the fact that there is a dearth for qualified, experienced and good radiologists/doctors to detect the disease aptly compels a CAD(Computer Aided Detection) system to identify diseases quickly and accurately[13]. In this regard, there have been many image reading, image processing systems that detect and identify objects. These algorithms perform fairly well in object localization and identification. However, recent advancements have proven that deep learning has gone a long way in identifying not only objects in images, but also minor changes in the same images,etc.

Identifying differences in images of Chest X-rays can be major challenge as most images would look very similar. However, deep neural networks(CNNs) have the ability to identify/detect the presence of some unusual traces or patches on the X-ray image and thus tag the image as pneumonia affected using supervised learning method. The idea is to build Convolutional Neural Network models, tune their hyper-parameters and analyze their performances. The approach is going to be experimental in terms of model selection and hyper-parameter tuning, however, the base ideology of implementation using CNNs remains the same.

The CNN model would be trained with a set of frontal chest X-rays which are labelled with the class they belong to - 'Pneumonia' or 'Normal'. The input to the model would be these images along with their labels. Since our problem is a binary classification problem, we would have one output from the model, either 'Pneumonia' or 'Normal' for every test image.

Although several experiments have been conducted and many research papers have been published with regards to this problem, since CNN models are not easily reproducible, chances are that we may not be able to achieve state-of-the-art performances even if we strictly follow the same methodologies. However, by tweaking parameters made available in the deep learning module by Keras package, it is possible to reproduce state-of-the-art results.

This project is not a novel idea to address the problem. It is an earnest attempt to not only improve the overall performance of the model used in the reference paper [2], but also to make the model more generalized to be able to handle any such similar data. This is achieved by introducing a different modelling technique where individual models are ensembled to form a more complex deep learning model. Several experiments also will be conducted to test the adaptability of the developed models to new data and the robustness of the models to handle unseen data.

## 1.2 Contribution

Although the state-of-the-art promises many methodologies to accurately and efficiently detect a disease affected chest X-ray using deep learning models, there is always scope for improvements in the approach, in the performance of the deep learning models and in generalizing models to be able to perform well for unseen data. From this project, the contributions are:

- Identification of best model for detection of pneumonia from images of chest X-rays.
- Best Possible generalized model for identification of any disease from images of X-rays
- Performance Evaluation of model experimented with the dataset.

## 1.3 Structure of This Document

This document is structured as follows: Chapter 2 presents state-of-the-art as well as background information on methodologies and approaches used in detection of pneumonia from images of chest X-rays using deep learning models; Chapter 3 presents the proposed work towards predicting presence of pneumonia in chest X-rays, the algorithms to be employed and the process used for training and tuning the models; Section 4 details the process taken, the tools used, with special emphasis on the parameters used; Section 5 discusses the results and summarizes the output of the project; Section 6 closes the document with a summary.

# Chapter 2

# Literature Review

## 2.1 Image Processing

Humans can read and perceive the colours, shapes and objects present in images by virtue of a sensory organ called eyes which captures all the data about images.This data is then sent to the brain as pulses via the optical nerve, where it is then processed and a perception about the shape, colour and object is made. With computers, this is not the case. Computers operate on numbers and signals - 0s and 1s. Images are also read as numbers by computers[14].

The smallest unit of an image is known as pixel and digital images are stored in a computer's memory as a combination of pixels. A pixel could be a 1 dimensional value or a 3 dimensional value based on whether the image is grayscale or colour. A grayscale image would have 1 dimensional pixels whereas a colour image would have 3 dimensional pixels, one for each of the primary colours "RGB"(red, green, blue). The pixel values range from 0 to 255 based on the intensity of the colour and these values are read as binary values by the computer[14]. Figure 2.1 illustrates how an image is perceived by a computer.

## 2.2 Traditional Object Detection Approaches

With computers being able to store and display images using pixels and signal pulses, the next step was to make sense of out the image data. Identifying objects in images became a challenging task as it meant lending a brain to the computer. Object detection is a technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class in digital images and videos.

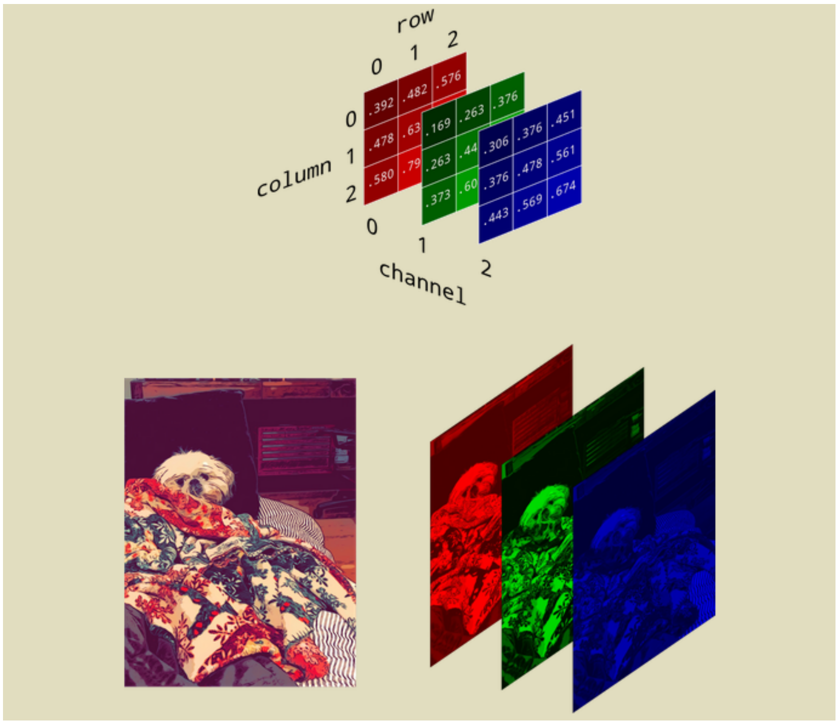


FIGURE 2.1: An image as perceived by a computer[3]

Several algorithms such as Scale-Invariant Feature Transform(SIFT), Speed up Robust Feature(SURF), Hough transforms, Geometric hashing enabled the detection of objects in images through computer vision[15].

The SIFT algorithm enabled extraction of feature points from images, that is, important points that distinguish a part of an image from some other part. This was achieved by obtaining the scale space representations of the image, computing the difference of their Guassians(DoG), and then identifying key-points by ensuring that they have no neighbouring pixels that are higher than them in both scale and space dimensions[16]. The SURF algorithm is a scale and rotation- invariant interest point detector and descriptor which uses a 'Fast Hessian' detector to identify key-points and Haar-wavelet responses within the neighbourhood of the key-point as the descriptor of the key-point[17]. Similarly, the Hough Transforms algorithm and several other algorithms enabled computers to identify objects in an image using feature extraction methodologies. The main drawback of this category of methods is that handcrafted feature set cannot efficiently represent all images of interest and therefore cannot provide sufficient detection performance[18]. This is when the shift to machine learning techniques began to take place with the Haar cascade method being one of the most prominent methods.

The Viola-Jones Haar-cascade classifier is an effective object detection (most commonly used for face detection) approach which uses a cascade function that is trained from a lot of images both positive and negative. Based on the feature training, the cascade function is then used to detect the objects in the other images. This method contributed three main aspects to the field of computer vision in terms of object detection[4].

- A new representation of images called 'integral images' which enabled faster computation for feature identification.
- construction of a classifier that selects a small number of important features from the image using the AdaBoost algorithm as this again reduces computation time and focuses only on critical features.
- combination of several complex classifiers in a cascade structure which again increases the computational speed and also focuses attention only on important features (typically the object in the image), thereby allowing further complex processing only on those important features[4].

The Haar cascade algorithm uses rectangular filters as features as shown in the top row of figure 2.2 . These features are used to identify objects in an image. An image is processed using several features and at each stage, many of these features get rejected if the image does not contain pixels similar to the feature. An illustration of this scenario is depicted in figure 2.2, describing the process involved in face detection. It is a well-known fact that the eye area on a face tends to be darker than the skin on the upper-cheeks. Similarly, the eyes tends to be darker than the bridge of the nose. Therefore, on applying Haar-cascade algorithm on a grayscale, scale-representation of an image consisting of a face, the cascading classifiers filter these rectangular features until the end to finally predict the presence of a face in the image[4].

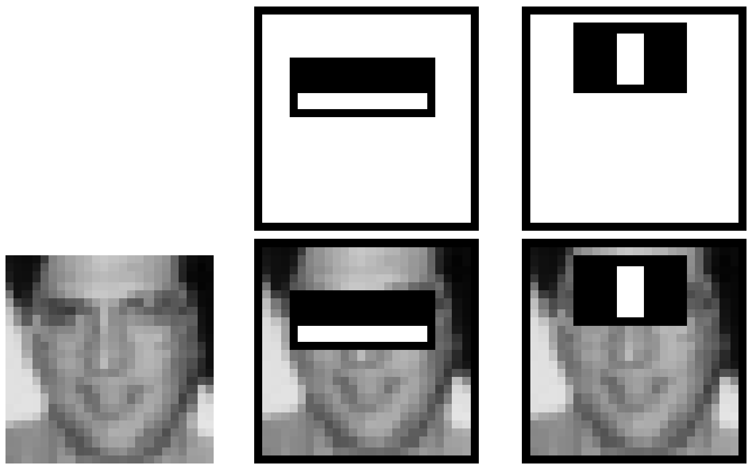


Figure 2.2: Detection of facial features using feature filters in Haar-cascade algorithm[4]

## 2.3 Deep Neural Network based Image Recognition

Within the last two decades, a major transition from traditional computer vision techniques to machine learning methods of image recognition has taken place. Several advancements in the field of machine learning to identify features in images, detect objects in images, etc. led to new beginnings in the field of computer vision. The core idea used in image recognition using machine learning was based on techniques invented by pioneers in the field of Artificial Intelligence.

Artifical Intelligence is a field of study in Computer Science which aimed at developing intelligent systems that could behave like human beings. In simple words, it involved lending a human brain like processing unit to computer systems which can think and perceive things in the environment on their own. Just like the human brain operates on millions of neurons connected to each other sending signal pulses to communicate, in 1980, Dr. Kunihiko Fukushima introduced the concept of ”neocognitron” which are neuron like structures that were developed using mathematical operations like logistic regression, gradient descent, etc. These ”neocognitrons” eventually took the name ’perceptrons’ after a couple of enhancements and were eventually put to use in classification and regression data models[19].

With constant enhancements to the perceptron, and continuous advancements in the field, the neural network model was born. This model is a combination of several neurons which are connected to each other in layers as shown in figure 2.3. These neural network models take any data - text, numerical, image as 1D tensors as input into their network, process them to predict what they are through the output layers. However, to process images, a more complex and dimensionally robust neural network was built. These networks are known as Convolutional neural networks.

A convolutional neural network is a network of neurons which are connected to each other in layers. It is a simple feed-forward neural network in which each of the neurons perform a very simple task of identifying features like lines, curves, blobs in an image. Convolutions operate over 3D tensors called feature maps. The convolution operation analyzes a small portion or window of the input image and applies a transformation (using a filter) to this window. The window slides slowly along the original feature map and the filter is applied continuously. This produces an output feature map as shown in figure 2.4 which is fed as input to the next layers of convolutional neural network. A similar convolutional operation takes place in the next layer with the feature map from the previous layer as the input. The number of layers in the outputted feature map corresponds to the number of filters applied to the original feature map.Each of these filter maps encode specific features of the image data( horizontal line, vertical lines,

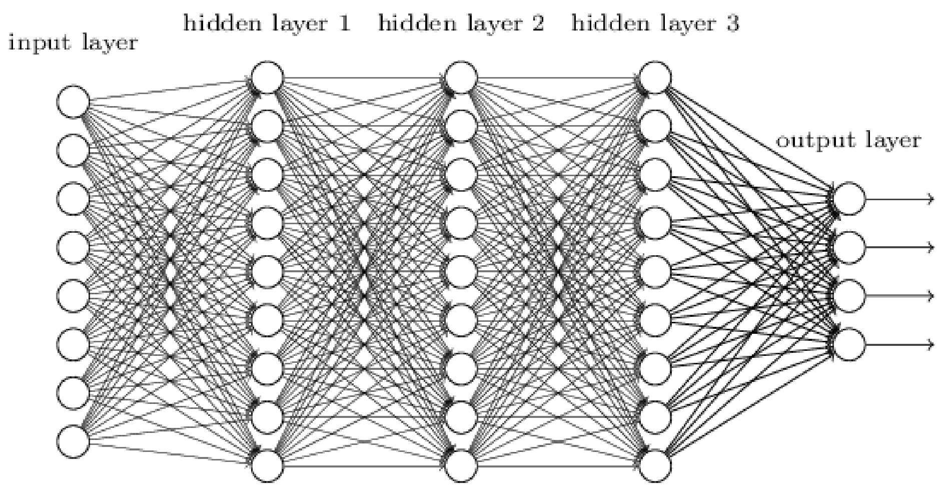


FIGURE 2.3: Neural Network - a combination of neurons connected in layers[5]

slanting lines, curves, etc.). This entire convolutional operation is implemented in Keras ( a python package) using the Convolution layer function[20].

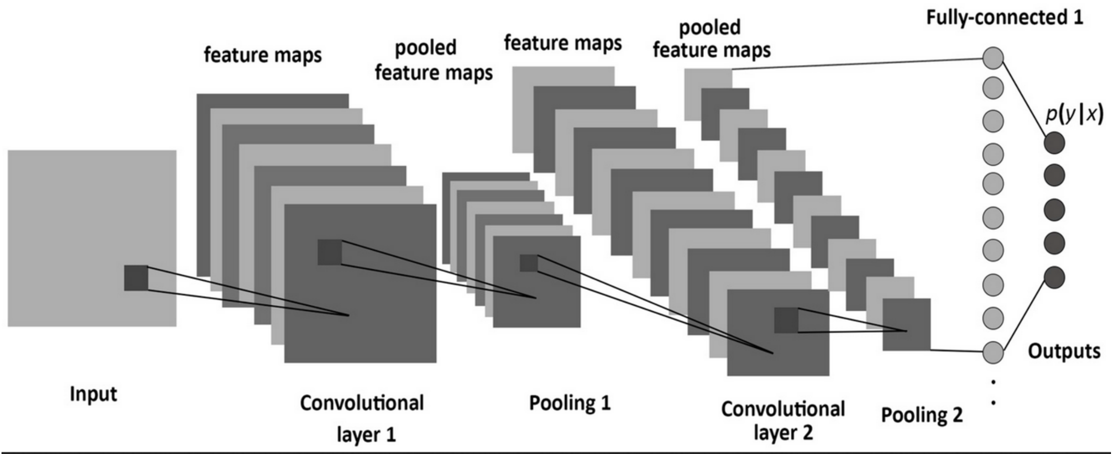


FIGURE 2.4: Convolutional Neural Network with feature maps and pooled layers[5]

The early layers in a convolutional neural network typically identify all the small features in a image and the last few layers combine these small features to identify bigger features in the image and then the classification layer(last layer) classifies the image. Overall, the convolutional neural network accepts image data as input, initializes the parameters of its neurons to random weights only in the beginning, processes the images(identification and classification of images features data) and then outputs the class that the image belongs to. This is called a feed- forward loop. Once this is done, based on the divergence in the output with respect to the true class label, the loss function is calculated, that is, the extent by which the model falls short of predicting the correct class. Based on the loss, the parameters(weights) of the neurons in the network are tweaked. This is known as back-propagation. This process keeps continuing until the loss is minimum and accuracy is maximum and is known as training the model. A complete cycle of the feed-forward loop and the back-propagation step is called an iteration or an 'epoch'[6].

Once the model is trained, a set of unseen data is passed through the feed-forward loop to check the performance of the model. This is known as testing. The model can also be validated during the training process and this information also can be used to tweak parameters to ensure that the model is trained to perform well even with unseen data(test data).

A convolutional neural network can be developed using several combinations of these layers provided by the Keras package - convolutional layer, pooling layer, fully connected layers, activation functions, optimizers, loss functions, etc.

The pooling layers enable pooling of a set of pixels, either by selecting the highest pixel or the average of the pixels of a window filter of some specific size. This feature in neural networks enables in improving the computation time and the also focuses on identifying the important features in the images. The optimizers enable the calculation of the loss and updation of the parameters(weights) of the neurons after every epoch and the activation functions are an integral part of every neuron as they determine the final output of the neurons(binary or a number)[6]. The visualization of the complete process of training of a neural network is depicted in figure 2.5.

Sometimes, image datasets tend to be very complex, large or noisy. In such situations experimentations on building a deep convolutional neural network model is a very time consuming process and may or may not produce the best results possible. To handle these situations, transfer learning was introduced where models can be created using pre-trained models such as Retinanet, mask R-CNN, ChestXNet, AlexNet, Xception, Inception, VGG16, etc. provided by Keras. These pre-trained models are trained on the 'Imagenet', the 'Microsoft COCO'[18] datasets, or other datasets and have been tuned to obtained the best possible weights for the neurons in the models.

Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones[10]. As the models are trained on some dataset, they can be used to classify similiar datasets of images. This can be done through several approaches of transfer learning - feature extraction, fine tuning and pre-trained models. The images of the dataset can be sent as input to pre-trained models and outputs classifying the images can be obtained by just manipulating the number of classes in the pre-trained models. The pre-trained model would predict the classes of the images in the new dataset using the pre-trained dataset's weights. This would work well only if the dataset being tested is very similar to the pre-trained dataset. The feature extraction method of transfer learning involves extraction of features from a dataset

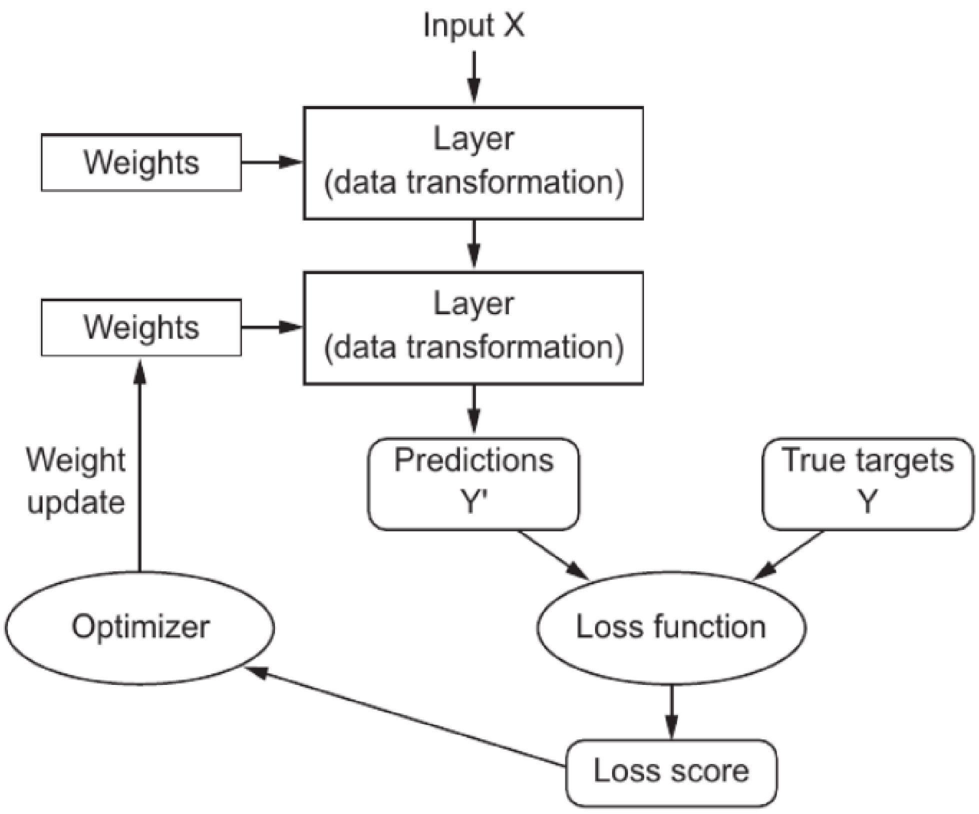
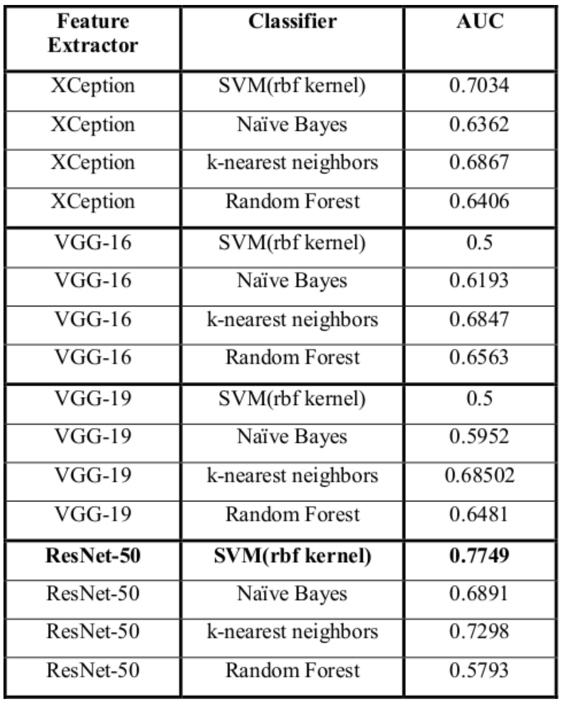


FIGURE 2.5: Training of a Convolution Neural Network[6]

using the weights of a pre-trained CNN model and then usage of a simple machine learning classifier to predict the classes of the images[6]. An experimental approach of training a combination of several pre-trained models - Xception, VGG-16, VGG-19 and ResNet-50 as feature extractors and machine learning classifiers - SVM, Naive Bayes, KNN, Random forest was conducted on chest X-ray dataset to detect pneumonia and the highest test accuracy of 77.49% was achieved by the ResNet-50 + SVM combination[7]. The results of the experiments conducted are as shown in figure 2.6

The fine-tuning approach occurs in two phases where in the first phase a pre-trained model is chosen and amputed of its classification layer. Then fully connected and classification layers are attached to this pre-trained network and the complete model is trained on the dataset with the pre-trained model's layers frozen so that the imagenet weights are not tampered. Furthering into phase two, a couple of layers from the pre-trained network are unfreezed and the model is again trained with a very low learning rate. In phase two, early layers remain frozen while the later layers are unfrozen because the early layers of the pre-trained model are already trained to identify small features in



| Feature Extractor | Classifier | AUC |
|---|---|---|
| XCeption | SVM(rbf kernel) | 0.7034 |
| XCeption | Naïve Bayes | 0.6362 |
| XCeption | k-nearest neighbors | 0.6867 |
| XCeption | Random Forest | 0.6406 |
| VGG-16 | SVM(rbf kernel) | 0.5 |
| VGG-16 | Naïve Bayes | 0.6193 |
| VGG-16 | k-nearest neighbors | 0.6847 |
| VGG-16 | Random Forest | 0.6563 |
| VGG-19 | SVM(rbf kernel) | 0.5 |
| VGG-19 | Naïve Bayes | 0.5952 |
| VGG-19 | k-nearest neighbors | 0.68502 |
| VGG-19 | Random Forest | 0.6481 |
| **ResNet-50** | **SVM(rbf kernel)** | **0.7749** |
| ResNet-50 | Naïve Bayes | 0.6891 |
| ResNet-50 | k-nearest neighbors | 0.7298 |
| ResNet-50 | Random Forest | 0.5793 |

Figure 2.6: Feature Extraction Results[7]

images whereas the later layers in the model are trained as they need to accomodate bigger features in the new chest X-ray dataset[21].

In order to enforce transfer learning techniques in image classification applications, it is important to choose the apt pre-trained models based on the dataset and computation capacity of the hardware machine. Some of the earliest CNN models such as AlexNet, Residual Network and mask R-CNN gave accuracies of 72%, 85% and 78% respectively on chest X-ray dataset to detect pneumonia[11][13].

The VGG-16 model is a 16 layered CNN model pre-trained on the 'ImageNet' dataset, with 2 blocks of 2 convolutional layers followed by max pooling layers and then 3 blocks of 3 convolutional layers followed by max pooling layers. These layers are followed by 3 fully connected layers out of which the last layer is the output layer[2]. Figure 2.7 represents the architectural structure of the VGG-16 model.

The Xception model is a linear stack of depthwise separable convolutional layers with 36 convolutional layers forming the feature extraction base of the network. The 36 convolutional layers are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules. All separable convolutional layers are followed by batch normalization layers which normalize the data flow between the layers to enable faster computation. The Xception model is also pre-trained using the 'ImageNet' datset. In effect, the paper hypothesises that the mapping of cross-channels correlations and spatial correlations in the feature maps of the model can be entirely

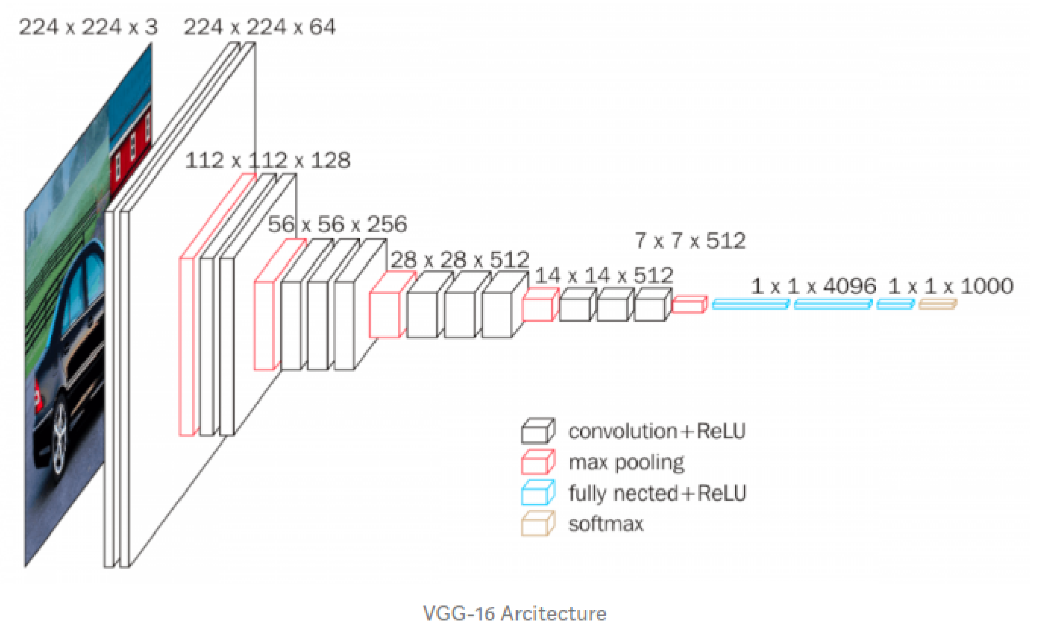


FIGURE 2.7: Architecture of the VGG-16 Model[8]

decoupled. This means that correlations between the pixels in the 3 channels for a colour image and the correlations between the a pixel and its spatial neighbourhood are decoupled in this model[9]. Figure 2.8 depicts the architecture of the Xception pre-trained model. The input data is fed to the Entry flow block, the through the middle flow which is repeated 8 times and then through the exit flow.

Another important aspect to deal with while using convolutional neural networks for image recognition and classification is the pre-processing of the input image data. It is very essential to ensure that the image data that is entering the CNN model for training do not differ in size or scale as it may have an adverse impact during testing and predictions. This can be done using OpenCV or PIL packages provided by Python. However, the ImageDataGenerator class in Keras enables this feature to pre-process input data before being fed to the network. Images are resized and rescaled to the same dimensions.

Additionally, but more importantly, the ImageDataGenerator class from Keras package solves another major issue - overfitting. Overfitting is a phenomenon that occurs when the model is so accustomed to the training data that it cannot perform well with unseen data(test data). Usually, overfitting occurs when the amount of data available to train the model is insufficient. The model tends to learn all the features available in the small dataset avaiable and may perform poorly on new data. The overfitting phenomenon can be detected by plotting a graph of the validation and training data accuracy/loss across the time of training(epoch) as depicted in figure 2.9. As seen in the figure, a very huge

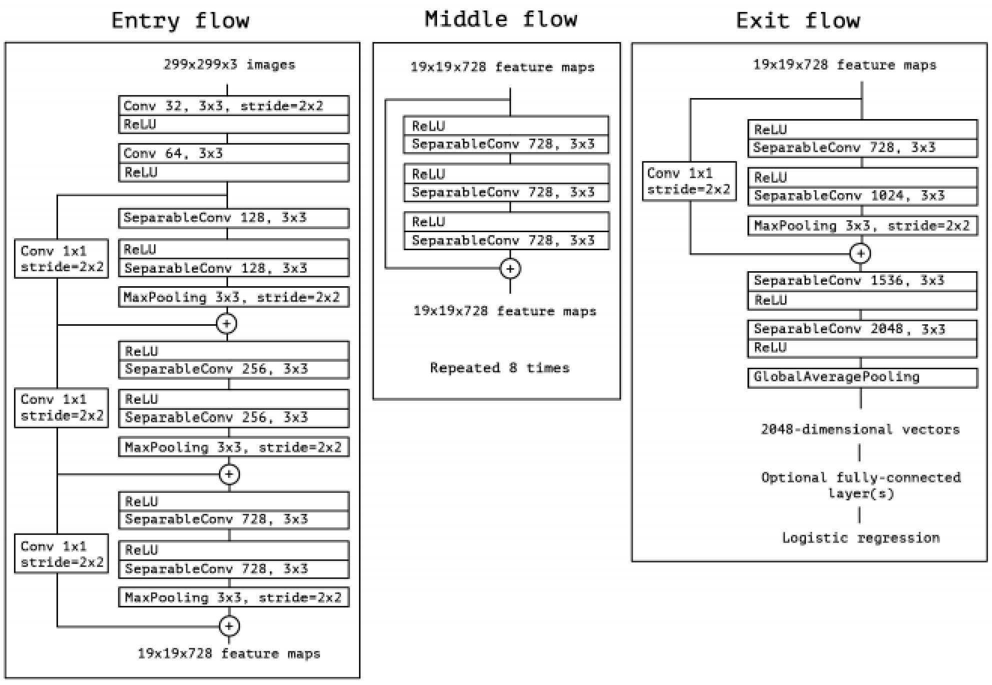


FIGURE 2.8: Architecture of the Xception Model[9]

gap eventually forms between the validation and train curves with increasing epochs. This indicates that the model is very accustomed(overfitted) to the training data that it is unable to perform well on the validation data[10].

The ImageDataGenarator class enables augmentation of images to the dataset, that is, it generates new images from existing input training images using data augmentation techniques. These techniques include cropping, rotating, brightening, zooming, blurring, shifting, flipping, etc. of an image in the training data. This leads to an increase in the number of images in the training data, thereby increasing the size of the training data and also introducing more variations to an image enabling the model to be able to identify any such variations from the unseen(test) data relatively easily. This indirectly helps reduce overfitting of the model too. Figure 2.10 is an excellent illustration of data augmentation of the image of a skin lesion. On training the skin lesions dataset without augmentation, an accuracy score of 78% was achieved whereas after data augmentation, an accuracy score of 81% was achieved[10].

With gradual advancements in deep neural network models, the Vgg-16 and Xception models emerged to be much very efficient in predicting image data. The VGG-16 and Xception models pre-trained on the 'ImageNet' dataset were fine-tuned with chest X-ray dataset from ZhangLab[22] to predict the presence of pneumonia. It was observed that the Vgg-16 model and the Xception model performed fairly well. The VGG-16 model predicted the test data of 624 images with a test accuracy of 87% and the Xception

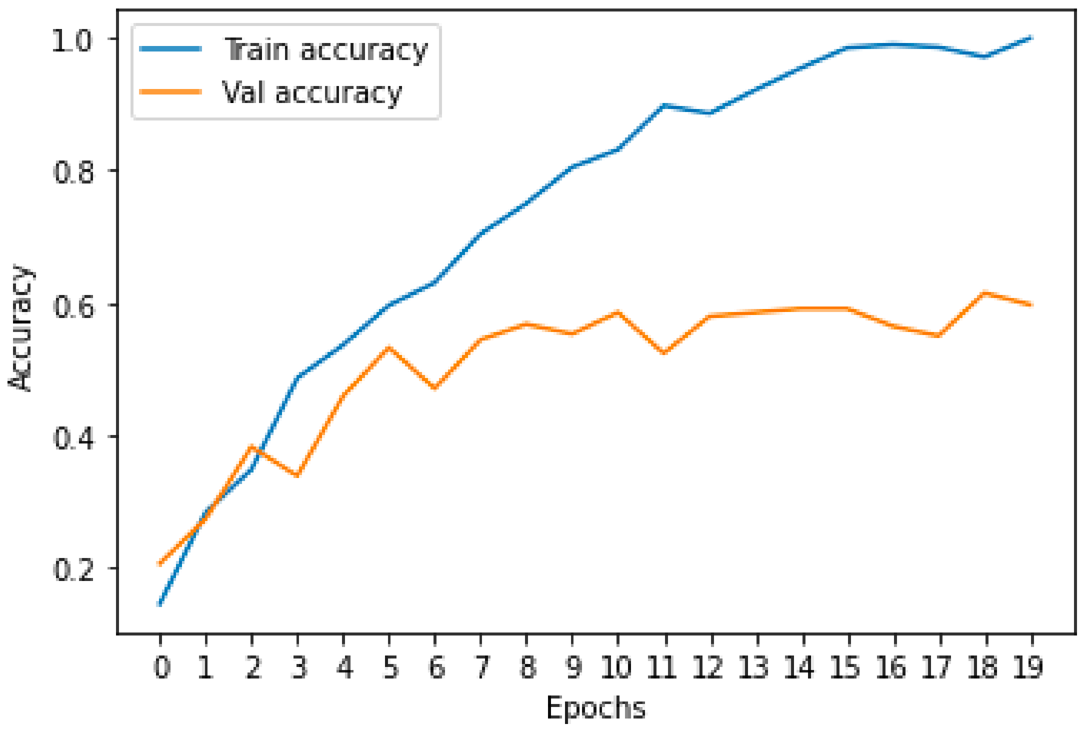


FIGURE 2.9: Overfitting Phenomenon

model with a test accuracy of 82%. The state-of-the-art is evolving with more advanced techniques to get better prediction results. However, the VGG-16 and the Xception CNN models are two of the best performing models in the Deep learning world[2].

### 2.3.1 Challenges

Numerous challenges were faced during training and prediction of pneumonia in chest X-rays using CNNs. Some of the noticeable challenges were:

1. Lack of data to train the model: Although millions of chest X-rays are generated over time, since they are personal medical data of patients, it is hard to access this data. Therefore, in most cases, chest X-rays data will be minimal, thereby affecting training of models. To address this issue, transfer learning and augmentation of images is used. these techniques increase the size of the data and also train the model to work in real-time[13].

2. Lack of information for exact prediction of pneumonia disease: Presence of pneumonia cannot be predicted just by scanning through chest X-rays. Much more information is required for this purpose such as patient history, lateral chest X-ray images, other symptoms, blood pressure,etc. There is no database yet that has a combination of all the data[13].

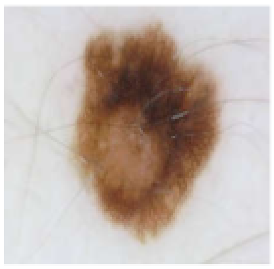


Fig. 3. Original skin lesion

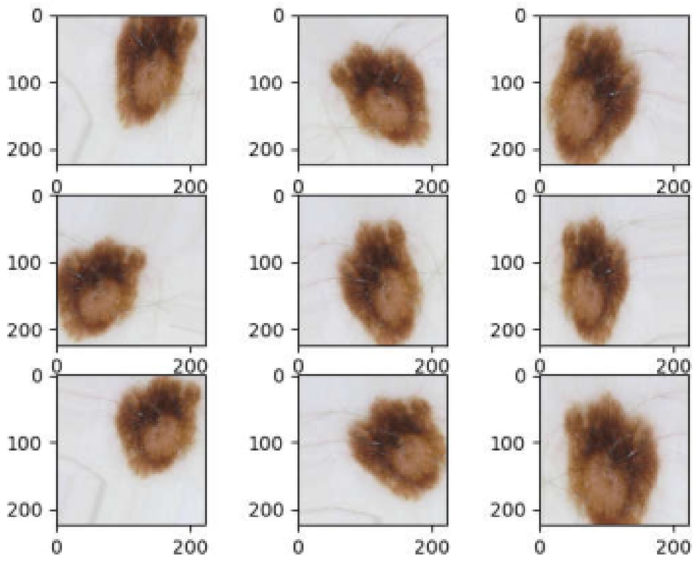


FIGURE 2.10: Image of a skin lesion augmented using ImageDataGenerator class[10]

3. Overfitting due to deep neural network layers: Since most models consist of numerous network layers, there is very high probability of overfitting during training. This scenario was avoided using data augmentation, batch normalization and max pooling layers between convolutional and activation layers[2].

4. Erroneous Data: Chest X-ray images may not be of the best quality. These images may contain noise, medical instruments such as pacemakers, maybe tilted, multi-dimensional, of varied sizes, may have patches depicting other lung diseases, etc. To ensure model's performance, these data images need to be pre-processed before training[2].

# Chapter 3

# Design

## 3.1 Problem Definition

In the medical industry, doctors and radiologists are trained to read X-ray images of the human body. With a keen eye to details, good observational skills and consistent practice, identifying some abnormality in an X-ray becomes a cake walk to these doctors and radiologists. However, with rising population, increasing number of diseases, there is always a need for more doctors and radiologists who have expertise in detecting abnormalities. In order to cater to this need, many Computer Aided Diagnosis(CAD) systems were and are still being developed using deep learning models.

The current state-of-the-art in deep neural networks has massively conquered and is capable of predicting abnormalities in X-rays with  80% accuracy which is a giant leap from where it all started. However, medical industry is such that a small mistake could cost many lives. Therefore, accuracy in prediction is one main factor that needs atmost attention. To improve a deep learning model's accuracy, there are several methods including data augmentation, transfer learning, ensemble modelling, etc. In this project all three techniques are implemented using Tensorflow and Keras.

## 3.2 Objectives

The primary task of the neural network deep learning model that would be developed would be to accurately and efficiently distinguish between pneumonia affected chest X-rays images and normal chest X-ray images. In this process of prediction, it is essential to ensure that the model is able to

- Identify the best model for detection of pneumonia from images of chest X-rays.
- Best Possible generalized model for identification of disease from images of X-rays
- Performance Evaluation of model experimented with the dataset.

## 3.3 Requirements

### 3.3.1 Functional Requirements

Functional requirements of a problem can be defined as the description of the functionality of the problem solution, that is, what does the solution do to solve the problem[23].

The functional requirements of this project are as follows:

- The deep learning model should be able to classify X-ray images of the human lungs into two classes - normal and pneumonia.
- The model should be able to pre-process (rescale and resize) the images that are input to the model before classifying them.
- The model should be able to output the accuracy score of a test dataset, that is, number of images that were classified correctly out of the test dataset.

### 3.3.2 Non - Functional Requirements

Non- Functional requirements can be defined as how the how of the solution. The constraints under which the solution should work[23].

The non - functional requirements of this project are as follows:

- The deep learning model should be able give an accuracy score on test data(unseen) of atleast 80%.

### 3.3.3 Software Requirements

The implementation of this project requires the following:

1. Python version 3.0+ installed
2. Tensorflow version 1.0+ installed

3. Keras package version 2.0+ installed
4. GPU 1.0 - NVIDIA Tesla K80

## 3.4 Design

This project aims to use deep convolutional neural networks to classify a dataset of frontal chest X-ray images into normal and pneumonia. Initially, the models in paper [2] are replicated to achieve results at-least close to that of the paper. Post this, an ensemble of the two models is developed to evaluate and check if the ensemble model performs better than the base models.

### 3.4.1 Model Checkpointing

Training deep learning models is a time consuming process and this is because models are allowed to train over several iterations where in each iteration the model's error in predicting values are documented as loss and based on this loss factor the model updates the weights of its neurons in the following iteration. It may so happen that the model may perform its best, way ahead of the specified epochs and then may begin to overfit on the data. In order to obtain this model weights of the epoch in which the model performs the best, model-checkpointing is used. Model- checkpointing enables capturing and saving the weights of the neurons of a model when a certain parameter is set to minimum or maximum. For instance, the model performs the best either when the validation loss is minimum or the validation accuracy is maximum. This model which is saved can later be loaded for testing on the test data[24].

### 3.4.2 Model Architecture - Base

Two pre-trained models - VGG16 and Xception are used in this project. These models are fine- tuned to accomodate and predict the classes that images belong to. Data augmentation is also implemented in order to curb overfitting and also increase the number of images in the train dataset.

#### 3.4.2.1 VGG16 Model

The VGG16 pre-trained model is used to build the model to classify X-ray images into normal and pneumonia. The last classification layers of VGG16 model is removed and

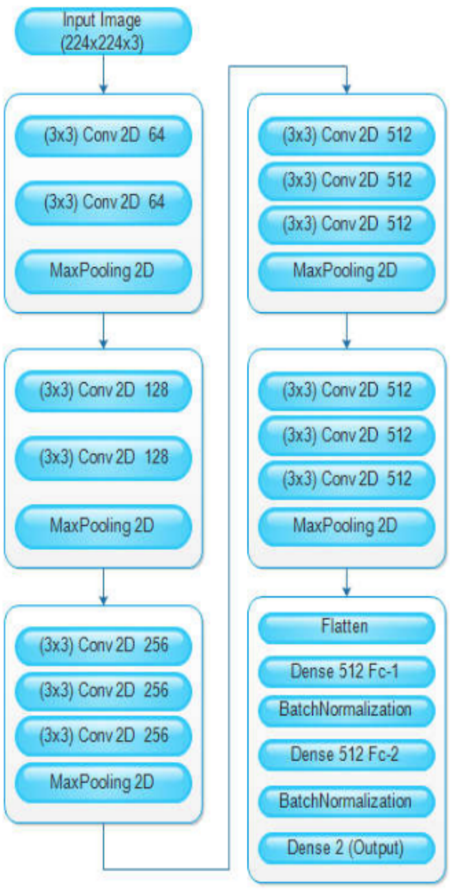


FIGURE 3.1: VGG16 - Finetuned architecture[2]

additional layers have been added. Figure 3.1 depicts the architecture of the VGG-16 model used in the project. The additional layers are as follows

- A Flatten layer to convert the 3D feature map to a 1D vector.
- A fully connected Dense layer wit 512 neurons.
- A batch normlization layer to normalize the weights that passed to and from neurons in the network.
- Another fully connected layer with 512 neurons.
- Another batch normalization layer
- A final Dense layer with softmax actiavtion with 2 neurons (one for normal and one for pneumonia).

During fine-tuning process, the weights of the early 8 layers were kept frozen.

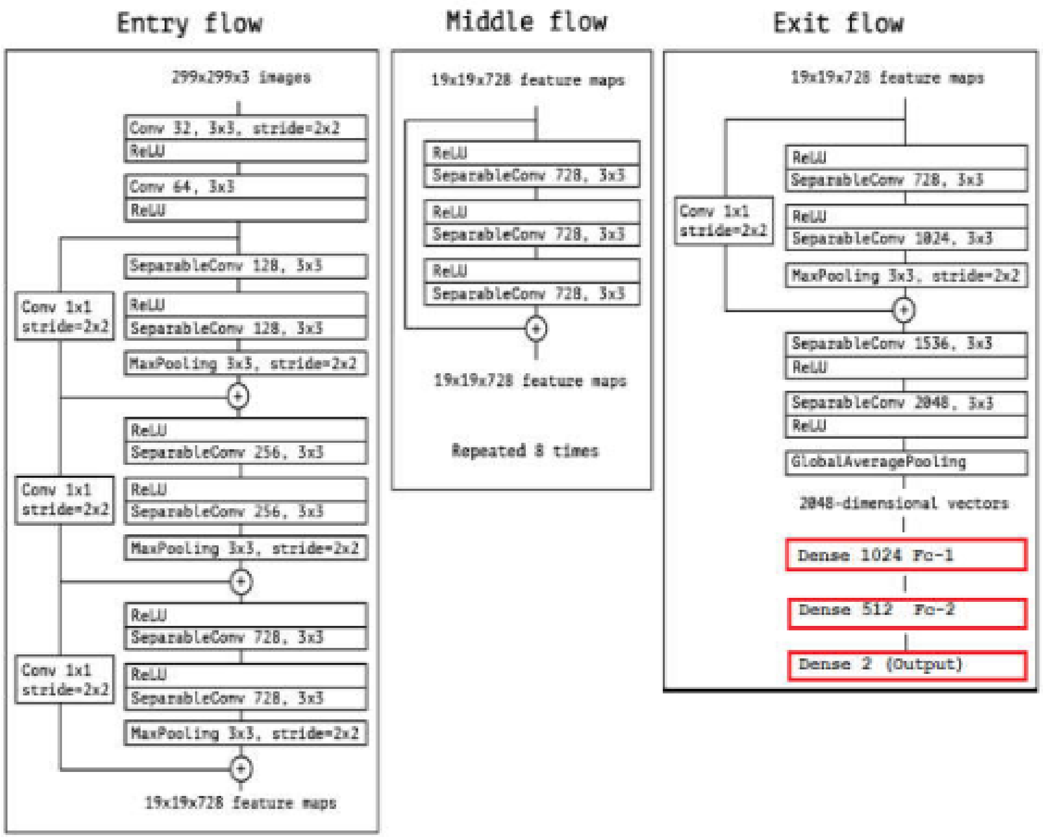


FIGURE 3.2: Xception - Fine tuned architecture[9]

#### 3.4.2.2 Xception Model

The Xception pre-trained model is used to build the model to classify X-ray images into normal and pneumonia. The last classification layers Xception model is removed and additional layers have been added. Figure 3.2 depicts the architecture of the VGG-16 model used in the project.

- A Flatten layer to convert the 3D feature map to a 1D vector.
- A fully connected Dense layer wit 1024 neurons.
- Another fully connected layer with 512 neurons.
- A final Dense layer with softmax actiavtion with 2 neurons (one for normal and one for pneumonia).

During fine-tuning process, the weights of the early 10 layers were kept frozen.

### 3.4.3 Model Architecture - Improved

In the base architecture, two models are implemented and tested for their performances individually. An improved architecture would be to combine these two models and create an ensemble model. This ensemble model would give better performance as it combines the positives of both the models and the models tend to cancel out each other's negative aspects.

While network ensembling for classification and segmentation tasks can be formulated using the weighted voting scheme, ensembling of object detection predictions is not straightforward. Perhaps an object detector produces many overlapping predictions with different confidence levels that can be merged with other object detector predictions[13]. An ensemble is typically created by indvidually training the two pre-trained networks and then averaging out their softmax probabilities. The class which has the highest average probability is chosen to be the class label for the particular image. Figure 3.3 represents the architecture of the Ensemble model.

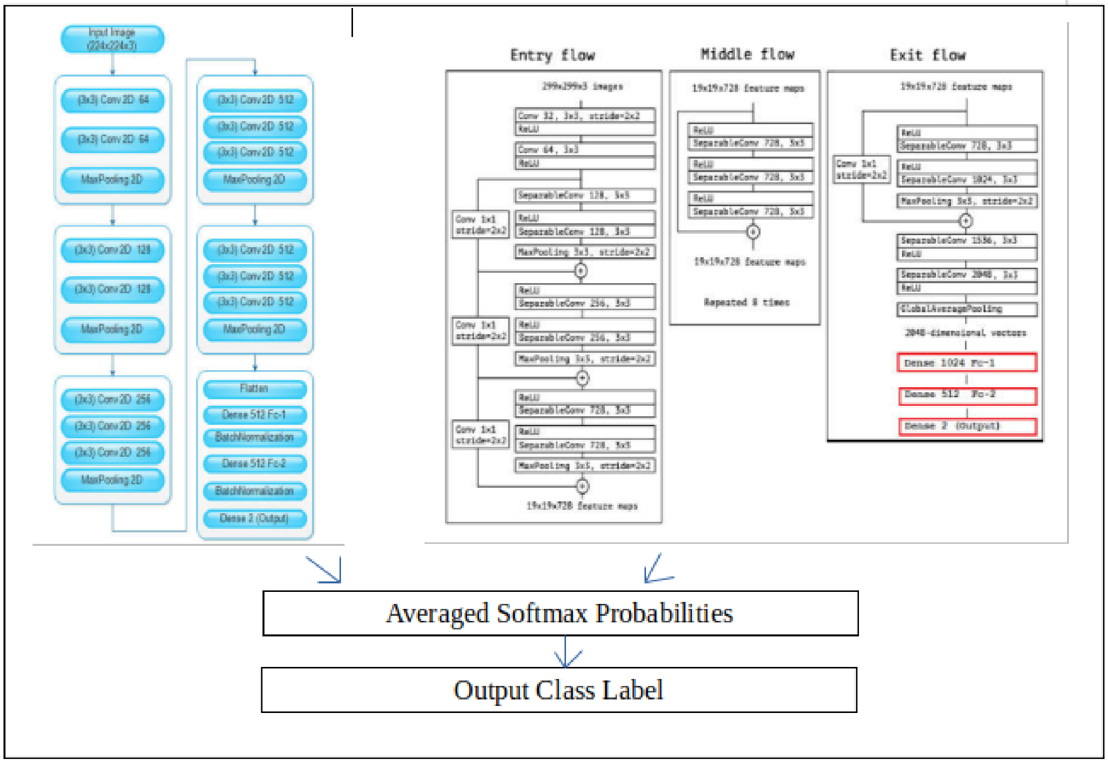


FIGURE 3.3: Ensemble architecture[2]

This ensembling can be coded in two ways:

- Low level programming :This can be done by using predict() on every model to predict the probabilities of every class for all the test samples. Then using np.mean(), the averages can be computed and the using np.argmax() function, the class with the highest average probability can be determined.

- Using Keras package :In this methodology, we use nested modelling to create our ensemble model. We define an input layer with the shape of the images. Then, create a nested model of the base learners with the specified input layer. The output layers will be the average of the layers of these base learner models. Using the defined input and output, an ensemble model can be created, compiled and tested.

# Chapter 4

# Implementation

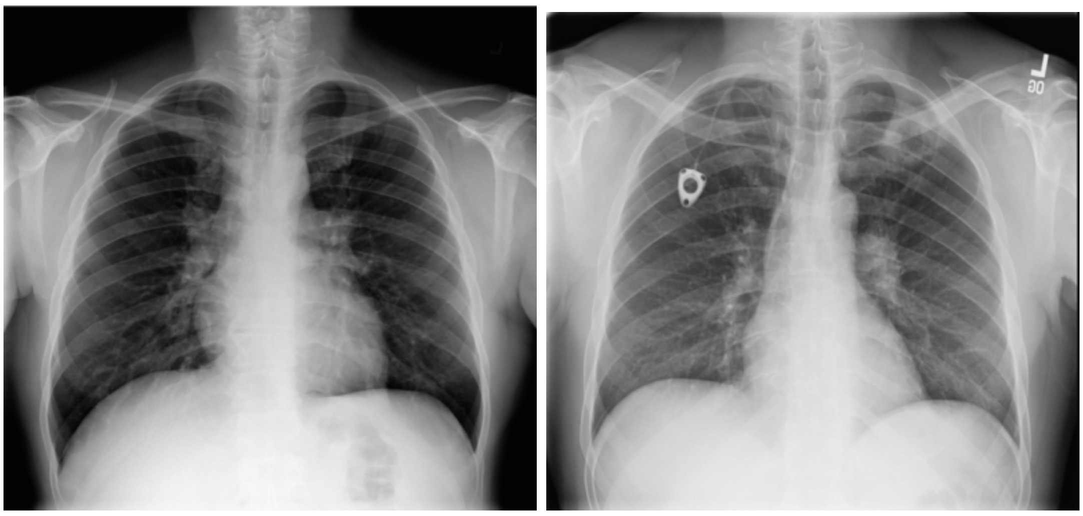
## 4.1 Data

In this project two datasets are used:

- A primary dataset from ZhangLab[22]
- A secondary dataset from Kaggle[25]

Figure 4.1 depict sample images of the chest X-rays which are normal and are affected by pneumonia.

Table 4.1 details the split of images into train, validation and test sets for the primary and secondary datasets.



(a) Normal (b) Pneumonia

FIGURE 4.1: Chest X-ray images of a normal and pneumonia affected patient[11]

| Dataset | Train | Validation | Test | Total |
|---|---|---|---|---|
| Primary | 5232 | 624 | 624 | 5856 |
| Secondary | 1863 | 251 | 293 | 2407 |

Table 4.1: Dataset Split : Train, Validation and Test

## 4.2 Implementation of Base Model

The base model includes two models - fine-tuned VGG-16 and fine-tuned Xception models. The parameter and steps involved in the implementation of these models are as follows. The implementation of the base models was done using the primary dataset.

### 4.2.1 Data Pre-processing and Augmentation

The data that is divided into train, validation and test sets are pre-processed before they are sent for further processing. The pre-processing techniques applied to the train, validation and test set are:

- Resizing images to a uniform size of (224, 224, 3) depicting the height, width and depth of the images.
- Setting the class mode for these images to 'categorical'. It could also be set to 'binary'.But since the final classification layer in the CNN model is a 'softmax' layer, the class model is set to 'categorical'.
- Since the depth of the image is 3, the color_mode is set to 'rgb'.
- Data is sent to the CNN model in batches of 32 images each while training, validating and test. This improves the computational time of the model.

While only pre-processing techniques are applied on the validation and test data, the train set is not only pre-processed but also several augmentation techniques are applied to it. The reason why the test and validation data are not augmented is because this data is used only to validate or test the model and therefore score the model with respect to its performance accuracy. The augmentation techniques applied on the training data are as follows:

- Rescale factor - 1./255. This rescales all the pixel values of the images between 1 and 255, thus standardizing the images inputted to the network.
- Shear range and zoom ranges are fixed to 0.2. This enables augmentation of data and therefore contributes to an increased number of images for training.

- Rotation range is fixed to 20. This generates variations of an images with rotations upto 20 degrees.
- Horizontal flip - Setting this parameter to True allows the ImageDataGenerator to generate horizontal flips of the image.

The python code for implementation of data pre-processing and augmentation is in Appendix A ( A.1).

### 4.2.2 VGG-16 Model

The VGG-16 model is a pre-trained model provided by Keras trained on the ImageNet dataset which has 1000 object classes. It has 2 blocks of 2 convolutional layers followed by a max pooling layer and 3 blocks of convolutional layers followed by max-pooling layer and fully connected layers before the classification layer. In this project, the VGG-16 model has been fine-tuned with the chest X-rays dataset in two phases.

Steps in Phase 1 of training:

- The pre-trained VGG-16 model is downloaded with its ImageNet weights and the classification layer is removed from the model by setting the include_top to 'False'.
- The model is made incapable of training itself to update the weights of its neurons by setting the trainable parameter to 'False'.
- A sequential model is created and the non-trainable VGG-16 model is added as its first layer.
- Since the VGG-16 model's classification layer is removed, its output would be a 3D feature map. This in turn would be the output of the sequential model. In order to add fully connected layers to the model, this 3D feature map should be flattened to a 1D tensor using the Flatten layer from Keras.
- Two blocks of a fully connected layer with 512 neurons and 'relu' activation followed by a Batch normalization layer are added to the sequential model using the Dense and Batch normalization layers respectively from Keras.
- A final classification layer with 2 neurons and 'softmax' activation function are added to the sequential model using Dense layer from Keras. The classification layer is the output layer which outputs whether an image is classified as 'Normal' or Pneumonia'. The 2 neurons represent the 2 classes respectively and the probability that an image belongs to a particular class would be the output of the neuron representing that class.

- The sequential model is then compiled with loss - 'categorical_crossentropy', optimizer - SGD with a learning rate of 0.0001 and metrics - accuracy.
- The train data is then sent as input to the model using the fit() function. The model is trained with the chest X-ray image data for 25 epochs and validated with the validation data.

The python code for implementation of phase 1 training of VGG-16 model is present in Appendix A ( A.2).

Steps in Phase 2 for training:

- Now that the layers of the sequential model are trained except the VGG-16 layers, the real fine-tuning begins. The initial 10 layers of the model are kept frozen. The VGG-16 model layers until the 'block3_conv2' layer are still kept frozen, that is, non-trainable, whereas the layers in the VGG-16 model from the 'block3_conv2' layer are made trainable by setting their trainable parameters to 'True'. This is done so as to train the weights of the sequential model of which the VGG-16 pre-trained model is a part of. The layers in the VGG-16 model which are set to be trained fine-tune their own weights in accordance with the features of the chest X-ray dataset thereby adapting themselves to the current dataset that they need to predict for.
- The sequential model is again compiled with loss - 'categorical_crossentropy', optimizer - SGD with a learning rate of 1e-10 and metrics - accuracy. The reason why the learning rate is a very negligible number in phase 2 is because the purpose of this phase is to fine tune the pre-trained VGG-16 model and usage of a bigger value for learning rate might deprive the performance of the model by manipulating the weights of the pre-trained layers drastically.
- The train data is then again sent as input to the model using the fit() function. The model is trained with the chest X-ray image data for 50 epochs and validated with the validation data. Model checkpointing is also enabled in this phase as the intention is to obtain the best performing model.

The python code for implementation of phase 2 training of VGG-16 model is present in Appendix A ( A.3). Once the training process is completed, the model is loaded with the best weights recorded using model checkpointing and evaluated on the test data to obtain the performance metrics. The code for this implementation is present in Appendix A ( A.4).

### 4.2.3 Xception Model

The Xception model is a pre-trained Convolutional Neural Network Model that was trained on ImageNet dataset which consists of images of 1000 classes of objects. It has 36 Convolutional layers and can be visualized as blocks of convolutional layers, pooling layers, regularization layers, activation layers, etc. In this project, the Xception model has been fine-tuned with the chest X-rays dataset in two phases.

Steps in Phase 1 for training:

- The pre-trained Xception model is downloaded with it ImageNet weights and the classification layer is removed from the model by setting the include_top to 'False'.
- The model is made incapable of training itself to update the weights of its neurons by setting the trainable parameter to 'False'.
- A sequential model is created and the non-trainable Xception model is added as its first layer.
- Since the Xception model's classification layer is removed, its output would be a 3D feature map. This in turn would be the output of the sequential model. In order to add fully connected layers to the model, this 3D feature map should be flattened to a 1D tensor using the Flatten layer from Keras.
- Two fully connected layers are then added to the sequential model with 1024 and 512 neurons each and 'relu' activations using the Dense layer from Keras.
- A final classification layer with 2 neurons and 'softmax' activation function are added to the sequential model using Dense layer from Keras. The classification layer is the output layer which outputs whether an image is classified as 'Normal' or Pneumonia'. The 2 neurons represent the 2 classes respectively and the probability that an image belongs to a particular class would be the output of the neuron representing that class.
- The sequential model is then compiled with loss - 'categorical_crossentropy', optimizer - SGD with a learning rate of 0.0001 and metrics - accuracy.
- The train data is then sent as input to the model using the fit() function. The model is trained with the chest X-ray image data for 25 epochs and validated with the validation data.

The python code for implementation of phase 1 training of Xception model is present in Appendix A ( A.5). Steps in Phase 2 for training:

- Now that the layers of the sequential model are trained except the Xception layers, the real fine-tuning begins. The initial 10 layers of the model are kept frozen. The Xception model layers until the 'block2_sepconv2' layer are still kept frozen, that is, non-trainable, whereas the layers in the Xception model from the 'block2_sepconv2' layer are made trainable by setting their trainable parameters to 'True'. This is done so as to train the weights of the sequential model of which the Xception pre-trained model is a part of. The layers in the Xception model which are set to be trained fine-tune their own weights in accordance with the features of the chest X-ray dataset thereby adapting themselves to the current dataset that they need to predict for.

- The sequential model is again compiled with loss - 'categorical_crossentropy', optimizer - SGD with a learning rate of 1e-20 and metrics - accuracy. The reason why the learning rate is a very negligible number in phase 2 is because the purpose of this phase is to fine tune the pre-trained Xception model and usage of a bigger value for learning rate might deprive the performance of the model by manipulating the weights of the pre-trained layers drastically.

- The train data is then again sent as input to the model using the fit() function. The model is trained with the chest X-ray image data for 50 epochs and validated with the validation data. Model checkpointing is also enabled in this phase as the intention is to obtain the best performing model.

The python code for implementation of phase 2 training of Xception model is present in Appendix A ( A.6).

Once the training process is completed, the model is loaded with the best weights recorded using model checkpointing and evaluated on the test data to obtain the performance metrics.The code for this implementation is present in Appendix A ( A.7).

## 4.3 Implementation of Improved Model

The improved architecture of the model is an ensemble of both the models - Xception model and VGG-16 model. An ensemble of the two models is developed by training both the models on the same training data and determining the class of the images with the highest average of their output probabilities. The primary dataset is used to develop this ensemble of the two models and the process is implemented in the following manner:

- The Xception and VGG-16 models are loaded with their best performing weights.

- The input shape of the ensemble model is defined as the image size (224,224,3).
- Nested models of the Xception and VGG-16 fine-tuned models are created by sending to them, the input shape of the ensemble model to obtain the output layers.
- The output layer of the ensemble model is defined as the average of the output layers of the 2 nested models.
- The ensemble model is defined with its input and output layers, compiled with loss - 'categorical_crossentropy' and optimizer - SGD (learning rate = 0.0001) and evaluated on test data.

The python implementation of the Ensemble model is present in Appendix A ( A.8). The model is then evaluated for the test data. The code for this implementation is present in Appendix A ( A.9).

## 4.4 Experiments

Three experiments were conducted as a part of the project.

### 4.4.1 Experiment 1

The first experiment was conducted to determine the performance of the model on new data(secondary dataset). The aim was to identify if the model was generalized enough to train and test on similar data. For this purpose, a new set of data was obtained from Kaggle platform.

The Xception fine-tuned model, VGG-16 fine-tuned model and the ensemble models were trained, validated and tested with the new data.

### 4.4.2 Experiment 2

In order to further check the adaptability and flexibility of the models, a data exchanged testing was conducted, that is, the model trained on the primary dataset was tested with the test set from secondary dataset and vice versa. This experiment further tested the capabilities of the models and also determined the reusability of the model architectures.

### 4.4.3 Experiment 3

Fine-tuning process involves fine-tuning the weights of a pre-trained model to enhance its capabilities in identifying and classifying new data that is partially similar to the ImageNet dataset. In this process, the initial layers of the pre-trained model are usually kept frozen in phase 2 of training, that is, the weights of the layers are maintained in a non-trainable state while the last layers before the classification layer are open to fine-tuning of weights in order to capture the bigger features of the new dataset. This experiment aims to identify a pattern, if any, while changing the number of layers that remain frozen during phase 2 of training of the Xception fine-tuned model. The Xception model is tested with 10 layers, 50 layers and 80 layers frozen in phase 2 of the training considering every layer in the summary of Xception layers as a separate layer. The secondary dataset is used to conduct this experiment as it is a relatively smaller dataset. Therefore, the training time will be lesser.

The links to the datasets, and Python code files can be found in Appendix B.

# Chapter 5

# Testing and Evaluation

## 5.1 Model Architecture - Base

### 5.1.1 VGG-16 Model

The VGG-16 model produced excellent results with the primary dataset. Table 5.1 details the results obtained.

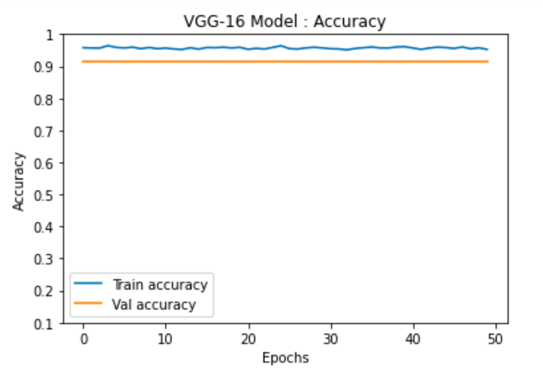
| Parameters | Train | Validation | Test |
|---|---|---|---|
| Loss | 0.1426 | 0.2303 | 0.2315 |
| Accuracy | 95.8% | 92.78% | 92.78% |

Table 5.1: Base Model - VGG-16 Model Results on Primary Dataset

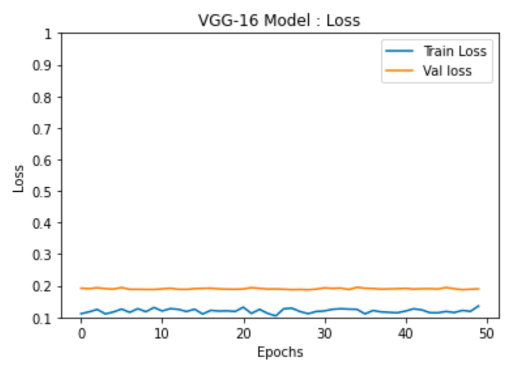
From table 5.1, it can be observed that the model has an accuracy of 92.78% with error of just 0.2 on the test data indicating a 5% leap in performance in comparison to its performance in the paper(87%)[2].

The VGG-16 model was trained with the train data for 50 epochs in the phase 2 of training. The behaviour of the model with respect to loss and accuracy across the 50 epochs is represented in figure 5.1.

From the above graphs, it can be observed that the two lines representing the performance of the training data(blue) and the performance about the validation data(orange) go hand in hand without diverging away from each other. This indicates that the model is not overfitting with the train data. It will be able to perform well on any kind of unseen data.



(a) VGG-16 : Accuracy



(b) VGG-16 : Loss

FIGURE 5.1: Base Model - VGG-16 Model's Accuracy and Loss on primary dataset

### 5.1.2 Xception Model

The performance of the Xception model on the primary dataset are tabulated in table 5.2

| Parameters | Train | Validation | Test |
|---|---|---|---|
| Loss | 0.1669 | 0.6760 | 0.6723 |
| Accuracy | 94.02% | 73.08% | 73.08% |

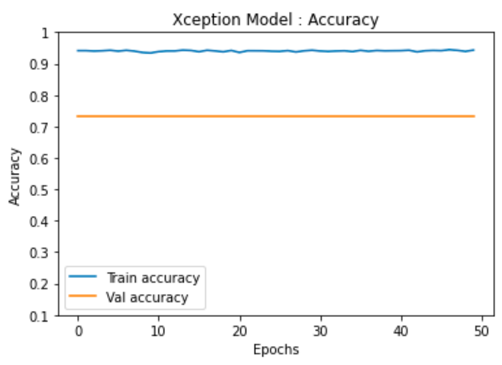
TABLE 5.2: Base Model - Xception Model Results on Primary Dataset

From table 5.2, it can be observed that the results for the Xception model on the primary data are not as impressive as that of the VGG-16 model. The model has performed fairly well with an accuracy score of 73.08% on test data but is almost a 9% dip in accuracy with respect to what was recorded by the paper(82%)[2]. This may be due to the lack data for training a huge model like the Xception model with 121 layers which in turn is leading to overfitting of the model.

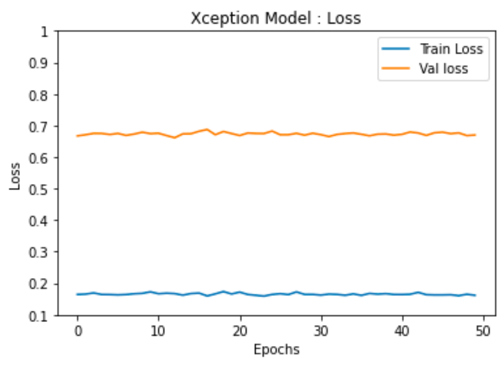
In phase 2 of the training process, the Xception model was trained for 50 epochs on the train data. The behaviour of the model across these 50 epochs is represented in figure 5.2

It can be observed from figure 5.2 that the two lines representing the train(blue) and validation(orange) performance are far apart from each other, indicating that the performance of the validation set is not at par with the performance of the training data and therefore may not be good even on the test data.

A common observation in the graphs from figure 5.1 and 5.2 is that both the train and validation curves start from the same values of loss and accuracy only to further without much fluctuation until the end of the 50th epoch which is a little unusual. This is because the graphs depict only the training process of the second phase of fine tuning. A dip in



(a) Xception : Accuracy



(b) Xception : Loss

FIGURE 5.2: Base Model - Xception Model's Accuracy and Loss on primary dataset

the loss and a rise in the accuracy values can be spotted in phase1. However, since phase 1 training does not involve alteration of the weights of the pre-trained models(VGG-16 and Xception), it seems to be that the weights of the pre-trained models were minimally fine-tuned in phase 2(reason why there is no dip in loss or rise in accuracy), just enough to accommodate the chest X-rays dataset. Had the learning rates or number or neurons or any other parameters in the model been a little different, this would not have been the case. The models would have performed with lesser accuracy after phase 2 of training.

## 5.2 Model Architecture - Improved

The improved model architecture comprises of the ensembled outputs from both the VGG-16 and Xception base models. The results are tabulated in table 5.3

| Parameters | Validation | Test |
|---|---|---|
| Loss | 0.2792 | 0.2761 |
| Accuracy | 92.94% | 92.94% |

TABLE 5.3: Ensemble Model Results on primary dataset

It can be observed from table 5.3 that the shortcomings of the performance of the Xception model are cancelled out by the the VGG-16 model, therefore producing a very good accuracy score of 92.94% which is 0.2% higher than the performance of the VGG-16 model individually and a huge 20% higher than the performance of the Xception model.

From the result recorded by the ensemble model, it is obvious that ensembles outperform the base models that they are constituted of.

## 5.3 Experiments

### 5.3.1 Experiment 1

The first experiment aimed at measuring the performance of the model on new data which is similar to the primary dataset. The secondary dataset comprising of frontal chest X-rays with and without pneumonia, obtained from Kaggle is used for conducting this experiment. All the three models , the VGG-16 model, Xception model and Ensemble models were trained and test on this dataset.

#### 5.3.1.1 VGG-16 Model

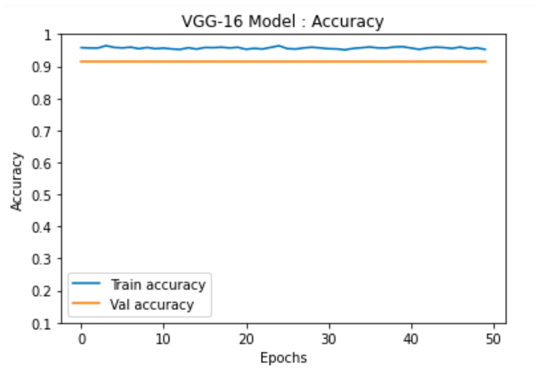
The performance of the VGG-16 base model on the secondary data is tabulated in table 5.4.

| Parameters | Train | Validation | Test |
|---|---|---|---|
| Loss | 0.1127 | 0.1893 | 0.2535 |
| Accuracy | 95.92% | 91.63% | 89.79% |

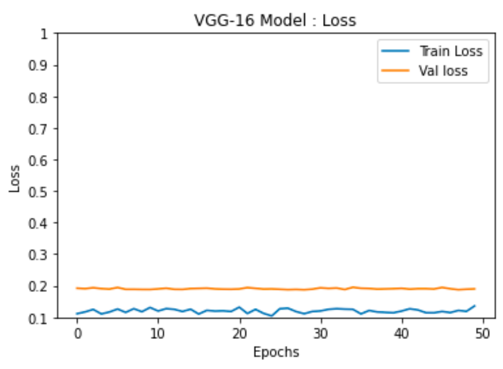
TABLE 5.4: Base Model - VGG-16 Model Results on Secondary Dataset

From table 5.4, it can be observed that the VGG-16 model has achieved an accuracy score of 89.79% on the test data which is 2% lesser than its performance on the primary dataset. Although there is a dip in the accuracy score, it is not of major concern as the model was not just able to accommodate new data but also perform well on the data as it is without any changes in the model architecture or parameters.

The graphs in figure 5.3 depict the performance of the VGG-16 model on the secondary data during the phase 2 of training across 50 epochs.



(a) VGG-16 : Accuracy



(b) VGG-16 : Loss

FIGURE 5.3: Base Model - VGG-16 Model's Accuracy and Loss on Secondary dataset

It can be observed from the graphs in figure 5.3 that the model has performed fairly well on the dataset without any major distortions in the validation curves. Overall, the VGG-16 model is robust enough to handle any such similar dataset.

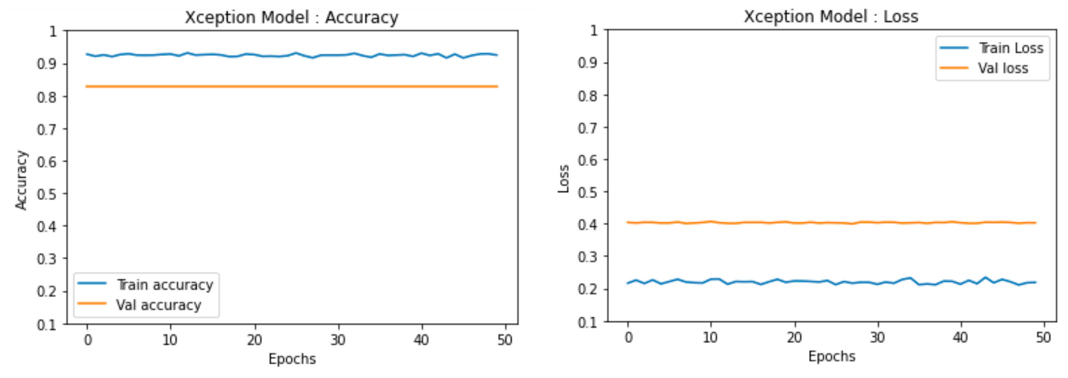
#### 5.3.1.2 Xception Model

The performance of the Xception model on the secondary dataset was much lesser than its performance on the primary dataset. Table 5.5 details the performance of the Xception model on the secondary dataset.

| Parameters | Train | Validation | Test |
|---|---|---|---|
| Loss | 0.2214 | 0.3341 | 0.5998 |
| Accuracy | 92.46% | 86.45% | 68.6% |

Table 5.5: Base Model - Xception Model Results on Secondary Dataset

It is evident from table 5.5 that the Xception model has performed relatively poorly with an accuracy score of 68.6% when compared to it performance on the primary dataset(73.08%). This again may be due to the lack of a huge dataset for a model with 121 layers leading to overfitting.



(a) Xception : Accuracy (b) Xception : Loss

Figure 5.4: Base Model - Xception Model's Accuracy and Loss on Secondary dataset

Figure 5.4 depicts the performance of the Xception model across 50 epochs of phase 2 training.

#### 5.3.1.3 Ensemble Model

The performance of the ensemble model on the secondary data was appreciable.

Table 5.6 details the performance of the ensemble model on the secondary dataset.The ensemble model performed with an accuracy of 86.68% on the test data. Although the

| Parameters | Validation | Test |
|---|---|---|
| Loss | 0.2155 | 0.3393 |
| Accuracy | 92.03% | 86.68% |

Table 5.6: Ensemble Model Results on secondary dataset

VGG-16 model scored a better accuracy(89.79%) with this data, the ensemble model made up for the poor performance of the Xception model(68.6%) on the secondary data.

### 5.3.2 Experiment 2

This experiment aims at identifying the adaptability of the trained models to be tested with different datasets. The 3 models - VGG-16, Xception and Ensemble models are trained with the primary dataset and tested with the train, validation and test data from the secondary dataset and vice versa.

To obtain the train data from the second dataset for the purpose of testing, a new instance of the ImageDataGenerator class needs to be created with only two pre-processing techniques - rescaling and resizing. Data augmentation techniques should not be initiated for this instance of the train data as it makes no sense to have augmented images in the test data.

1. Models trained on the Primary Dataset and tested with train, validation and test data from the Secondary Dataset.

| Models | Train Data | Validation Data | Test Data |
|---|---|---|---|
| VGG-16 | 0.2883 | 0.0.1710 | 0.1734 |
| Xception | 0.2833 | 0.4724 | 0.4792 |
| Ensemble | 0.2152 | 0.2457 | 0.2446 |

Table 5.7: Loss :Model trained on Primary Dataset, Tested on Secondary Dataset

| Models | Train Data | Validation Data | Test Data |
|---|---|---|---|
| VGG-16 | 90% | 93.42% | 93.42% |
| Xception | 87.84% | 77.56% | 77.56% |
| Ensemble | 93.4% | 92.94% | 92.94% |

Table 5.8: Accuracy :Model trained on Primary Dataset, Tested on Secondary Dataset

Tables 5.7 and 5.8 detail the performances of the models on the secondary dataset that they have not been trained on. The fact that the VGG-16 model scored more than 90% on the train, validation and test data of the secondary dataset in addition to the fact that the train data that was used for testing consisted of more than

1800 images, proves that the model is robust to any kind of data. Similarly, the Xception model exhibited a test accuracy of 80% on the secondary dataset proves the adaptability of the model to any similar data. Finally, the performance of the ensemble model has been exemplary with the train, validation and test data of the secondary dataset with accuracy scores of 93%.

2. Models trained on Secondary Dataset and tested with train, validation and test data from Primary Dataset.

| Models | Train Data | Validation Data | Test Data |
|---|---|---|---|
| VGG-16 | 0.2331 | 0.0.2125 | 0.3893 |
| Xception | 0.4370 | 0.39990 | 0.9244 |
| Ensemble | 0.2150 | 0.2131 | 0.4609 |

TABLE 5.9: Loss :Model trained on Secondary Dataset, Tested on Primary Dataset

| Models | Train Data | Validation Data | Test Data |
|---|---|---|---|
| VGG-16 | 94.31% | 92.43% | 88.05% |
| Xception | 78.42% | 82.47% | 63.82% |
| Ensemble | 96.72% | 95.61% | 84.64% |

TABLE 5.10: Accuracy :Model trained on Secondary Dataset, Tested on Primary Dataset

Tables 5.9 and 5.10 detail the loss and accuracy values of the models when tested on the train, validation and test set of the primary dataset that the models were not trained on. The VGG-16 model has performed outstandingly on the train and validation test data with accuracy scores greater than 95%. The Xception model has performed well again on the train and validation data with accurcay scores 80%. The ensemble model again outperfomed the base models with 95% accuracy with the train and validation test data.

Overall, the models seem to be robust and adaptable to similar datasets and therfore can be leveraged for similar causes like COVID-19 detection, cancer detection, etc..

### 5.3.3 Experiment 3

Since the Xception model performed the worst amongst the 3 models, an experiment was conducted which involved freezing the weights of more number of layers in the Xception model during Phase 2 of training process on the secondary dataset. This experiment would determine the impact of fine-tuning lesser number of layers of the Xception model on the accuracy scores. Secondary dataset was chosen as it is a relatively smaller one and would consume less time for training.

| Data | 10-layers | 50-layers | 80-layers |
|---|---|---|---|
| Validation | 86.45% | 84.8% | 84.8% |
| Test | 68.6% | 68.2% | 68.25% |

TABLE 5.11: Xception Model : Accuracy based on Frozen layers in Phase 2 of Fine-Tuning

Table 5.11 details the performance of the Xception model if 10, 50 and 80 layers are frozen. The accuracy seems to dip in the case of the validation data and stagnate in the case of test data. However, this is not a very reliable trend as a previous run of the same experiment produced very different results. This behaviour of the model can be attributed to several factors such as random initialization of weights, learning rate, etc.

## 5.4 Comparative Analysis of Performances of the Three CNN Models

Figure 5.5 represents a comparison between the test accuracies of the three models on the two datasets. Out of all the models, the ensemble model has performed the best on the primary dataset with an accuracy score of 92.94% on the test data. Overall, the VGG-16 model has performed very well on both the datasets and has contributed majorly to the performance of the ensemble model.

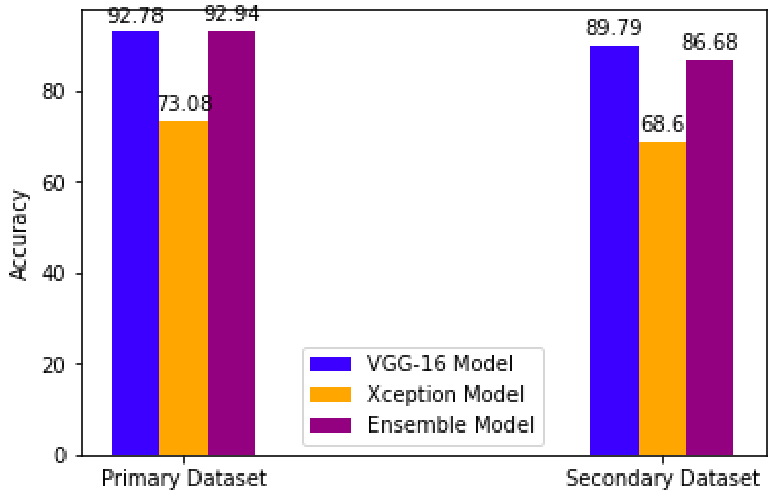


FIGURE 5.5: Comparison of Test Accuracy of Models on the two datasets

# Chapter 6

# Discussion and Conclusions

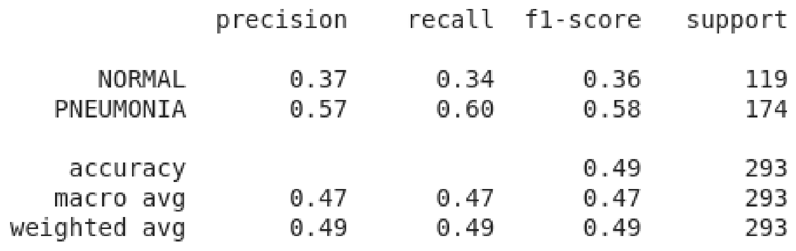
## 6.1 Discussion

Several chronic diseases have been affecting humans ever since industrialization and lifestyle changes have taken place in the past centuries. Lung diseases have been a major part of these diseases. Usage of tobacco, air pollution, constant exposure to hazardous air emissions in industrial units, etc. are the most common factors which lead to chronic lung diseases in humans. Most times lung diseases lead to death because they go unnoticed or untreated, especially in children below the age of 5. An evidence of this is the coronavirus pandemic which is also a type of pneumonia which has affected people of all age groups worldwide. Many children and old people have succumbed to the pandemic and have lost their lives. During such times of health crisis, technology proves to be a helping hand. What started as detection of objects from images has now evolved to detection of abnormalities from radiological reports. Reading an X-ray report is not a layman thing. It requires study in the field of radiology and medical sciences.

Deep Learning and AI systems have evolved to read images of X-ray reports to identify and predict abnormalities in the lungs, bones, brain, etc. Convolutional Neural networks are architectured in such a way that they receive images as inputs, process these images, that is, use filters to extract features from these images and reconstruct the image by stitching the features back to then predict the images. Several techniques such as data augmentation, batch normalization, transfer learning, etc. have evolved to amplify the performance of the CNN models. These enhancements have in turn helped the CNN models to identify minor indistinguishable features from chest X-ray images.

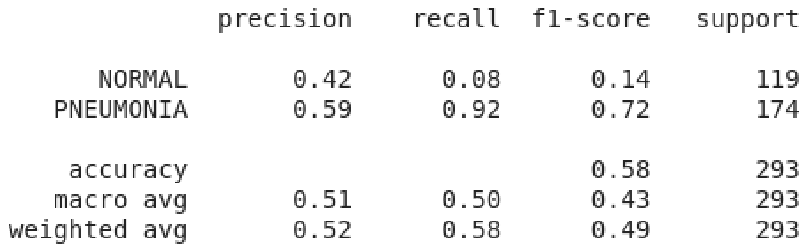
Application of CNN models to predict abnormalities in chest X-rays is a major risk with respect to the field of medicine and healthcare as a wrong prediction could cost a life.

Hence, it is essential to ensure that no pneumonia affected patient is wrongly identified to be normal. To ensure this aspect, it is important to obtain a high recall score from the CNN models. In this project, the main objective of the models was to produce very good accuracy scores. Therefore, the focus was only on accuracy scores. However, recall values of models also can be taken into vital consideration before determining the performance of a model.



```
              precision    recall  f1-score   support

      NORMAL       0.37      0.34      0.36       119
   PNEUMONIA       0.57      0.60      0.58       174

    accuracy                           0.49       293
   macro avg       0.47      0.47      0.47       293
weighted avg       0.49      0.49      0.49       293
```

(a) VGG-16 Model



```
              precision    recall  f1-score   support

      NORMAL       0.42      0.08      0.14       119
   PNEUMONIA       0.59      0.92      0.72       174

    accuracy                           0.58       293
   macro avg       0.51      0.50      0.43       293
weighted avg       0.52      0.58      0.49       293
```

(b) Xception

FIGURE 6.1: Performance Report of VGG-16 and Xception model on secondary data

Figure 6.1 details the performance report of the VGG-16 model on the secondary data. From the report,it can be observed that the recall score of the VGG-16 model is 60% whereas the recall score of the Xception model is 92%. This very factor cancels out all the negatives of the Xception model(accuracy score, performance). However, this scenario applies only to this domain of problems where the prediction of a patient suffering from pneumonia if wrongly detected to be normal could cost a precious life. Therefore, metrics should be chosen for any problem based on the domain. Accuracy score, if soaring high(90% or more) definitely would ensure that all metrics perform well.

Dealing with medical datasets is a challenge as the selection of models, metrics and CNN techniques would play a vital role in saving lives of people.

## 6.2 Conclusion

- In conclusion, the ensemble model outperforms all the other models with an accuracy score of 92.94% on the test data of the primary dataset. The VGG-16 model also performs fairly well with both the datasets. However, the performance of the Xception model is very mediocre when compared to the VGG-16 and ensemble models. The ensemble model also proved to be very robust in performing on new unseen data(Experiment 1).

- Deep learning CNN models are not easily reproducible. Since the weights of the neurons of CNN models get initialized randomly, everytime it is trained, results tend to vary for the same parameters. Therefore, model checkpointing is a very important feature that needs to be implemented everytime a deep learning model is being trained.

- There can neither be a specific model architecture for a particular type of problem nor a certain set of values for parameters to train a model. Every model architecture, parameter setting is data specific. However, there are certain aspects that have been observed after immense research in parameter selection. Parameters like optimizers need to be chosen based on the data and the type of problem, loss functions should be chosen based on the problem domain, learning rate should be chosen based on the data size, size of the model and numbers of training iterations.

- Ensemble models produce high performance on data when compared to individual models. However, ensembling models is a very computationaly expensive process as it involves training of all individual models before ensembling their output layers for prediction.

- The state of the art in the field of Deep learning has proven to be acceptable for commercial uses. However, rigorous implementation of deep learning modules in industries is a far thing as every step in deep learning prediction models come with a risk. Not all models that are built perform robustly. They fall short in some aspect in dynamic environments with dynamic data inputs. Major fields like the healthcare industries, chemical industries, etc. do encourage projects which do not come with high precision. Therefore, CNNs still have a wide scope for improvements.

## 6.3 Future Work

Although the project meets the current state of the art in terms of results and digs a little into ensemble modelling which is still a development in progress, there is tremendous scope for further improvements in this project.

- Creating an ensemble model is not as easy and simple as selecting two CNN models and obtaining their collective output. Selection of models to develop an ensemble model is in itself a research area. Several factors such as computational time, relevance to problem, ensembling techniques such as majority voting, averaging, ability of inidividual models in contributing to a wholesomely correct output, etc. matter significantly in selecting models for ensembled computation. This area can be researched further and implemented in essential areas such as the medical industry which require most accurate results.

- Although deep learning models tend to operate in a predictable manner with respect to certain parameters, sometimes they tend to deviate from usual trends indicating scope for further research. Parameters such as optimizers, loss functions, number of layers, number of neurons, normalizations, regularizations, etc. impact the performance of a CNN model. However, there is no specific trend in the impact of changing these parameters. A CNN works differently for different datasets and for different parameters. The key is to find the optimum number of layers, neurons, normalizations, regularizations, etc. to obtain the best result possible. In this regard, there is immense scope for experimentation and research.

- Datasets play a vital role in the performance of CNN models. Training and testing a model on various sizes and types of similar data definitely enhances the performance of a model. Also, training models with larger dataset comprising of varied orientations and aspects of data makes the model more adaptable and robust in identifying and predicting unseen test data. Obtaining large real-time datasets from hospitals and radiologists would be a challenging task but would definitely enhance the performance of the CNN models, in turn support the healthcare industry.

- Developing CNN models are extremely expensive in terms of time consumed for the training process. This is because during the training process, several millions of parameters need to be computed in every iteration and it is a very operationally expensive process. Running these training processes on absolutely fast GPUs can allow additional training, thereby improving the model's performance considerably.

- In order to reduce the computational time of model training, the parameters to be computed are reduced by pooling layers. Pooling layers typically reduce the size of the input image or feature map by either selecting the maximum pixel from a filter window of pixels or by averaging the pixels in the filter window across the entire image or feature map. This leads to loss of feature data which in turn affects the ability of the model to perform well. This drawback in CNN architecture led to the development of Capsule networks which uses dynamic feature routing mechanism using spatial orientation of features to forward only relevant features to the their relevant bigger features. This methodology is very new to the world of deep learning and using these capsule network architecture on the chest X-ray data may produce exciting results[26].

# Bibliography

[1] ”pneumonia”. [Online]. Available: ”https://www.who.int/news-room/fact-sheets/detail/pneumonia”

[2] E. AYAN and H. M. ÜNVER, “Diagnosis of pneumonia from chest x-ray images using deep learning,” 2019.

[3] ”Brandon”. (2019, November) ”how to convert an rgb image to grayscale”. [Online]. Available: ”https://brohrer.github.io/convert_rgb_to_grayscale.html”

[4] *Rapid Object Detection using a Boosted Cascade of Simple Features*, 2001.

[5] (2018) How to teach a computer to see with convolutional neural networks. [Online]. Available: ”https://towardsdatascience.com/how-to-teach-a-computer-to-see-with-convolutional-neural-networks-96c120827cd1”

[6] Ted$_Scully$, “$Lecturenotesindeeplearning, msc.artificialintelligence, cit,''$ 2020.

[7] L. A. R. N. Dimpy Varshni, Kartik Tharkal and A. Mittal, “”pneumonia detection using cnn based feature extraction”.” IEEE, 2019.

[8] (2019) Cnn architecture series — vgg-16 with implementation(part i). [Online]. Available: ”https://medium.com/datadriveninvestor/cnn-architecture-series-vgg-16-with-implementation-part-i-bca79e7db415”

[9] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807.

[10] E. AYAN and H. M. ÜNVER, “”data augmentation importance for classification of skin lesions via deep learning”.” IEEE, 2018.

[11] W. O’Quinn, R. J. Haddad, and D. L. Moore, “Pneumonia radiograph diagnosis utilizing deep learning network,” in *2019 IEEE 2nd International Conference on Electronic Information and Communication Technology (ICEICT)*, 2019, pp. 763–767.

[12] Pneumonia. [Online]. Available: ”https://www.physio-pedia.com/Pneumonia”

[13] D. J. A. M. Abdullah Faqih Al Mubarok and A. H. Thias, “Pneumonia detection with deep convolutional architecture,” 2019.

[14] S. Pokhrel”. (2019) ”how does computer understand images?”. [Online]. Available: ”https://towardsdatascience.com/how-does-computer-understand-images-c1566d4537bf”

[15] “”deep learning vs. traditional computer vision”.” [Online]. Available: https://arxiv.org/pdf/1910.13796.pdf

[16] “Image identification using sift algorithm: Performance analysis against different image deformations,” 2018.

[17] *SURF: Speeded Up Robust Features*, ser. Lecture Notes in Computer Science, vol. 3951. Springer, Berlin, Heidelberg, 2006.

[18] *Deep neural network ensemble for pneumonia localization from a large-scale chest x-ray database*, vol. 78. Elsevier, September 2019.

[19] “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.” *Biol. Cybernetics*, 1980.

[20] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[21] T. Alshalali and D. Josyula, “Fine-tuning of pre-trained deep learning models with extreme learning machine,” in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2018, pp. 469–473.

[22] “Large dataset of labeled optical coherence tomography (oct) and chest x-ray images,” 2018.

[23] Functional and non- funtional requirements. [Online]. Available: ”https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/”

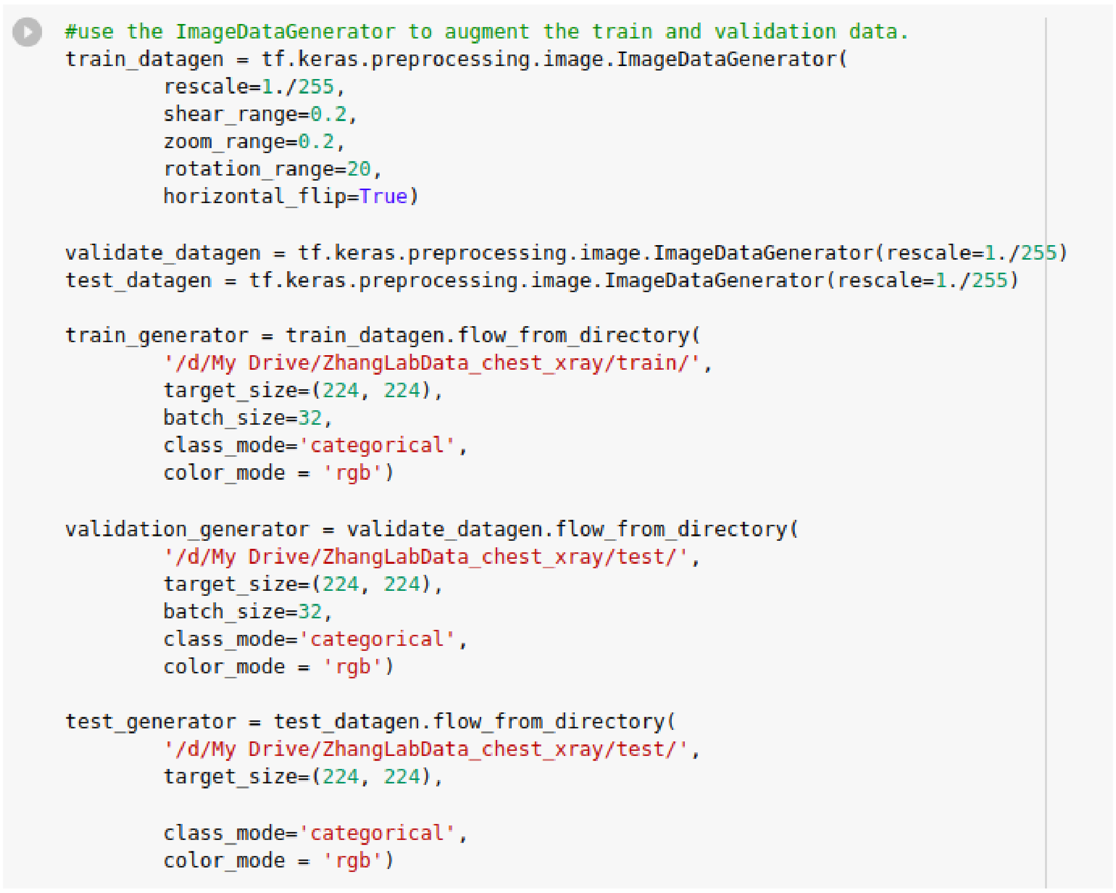
[24] K. G. Shin, T. Lin, and Y. Lee, “Optimal checkpointing of real-time tasks,” *IEEE Transactions on Computers*, vol. C-36, no. 11, pp. 1328–1341, 1987.

[25] Chest x-ray images (dataset). [Online]. Available: ”https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia”

[26] P. Afshar, A. Mohammadi, and K. N. Plataniotis, “Brain tumor type classification via capsule networks,” in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 3129–3133.

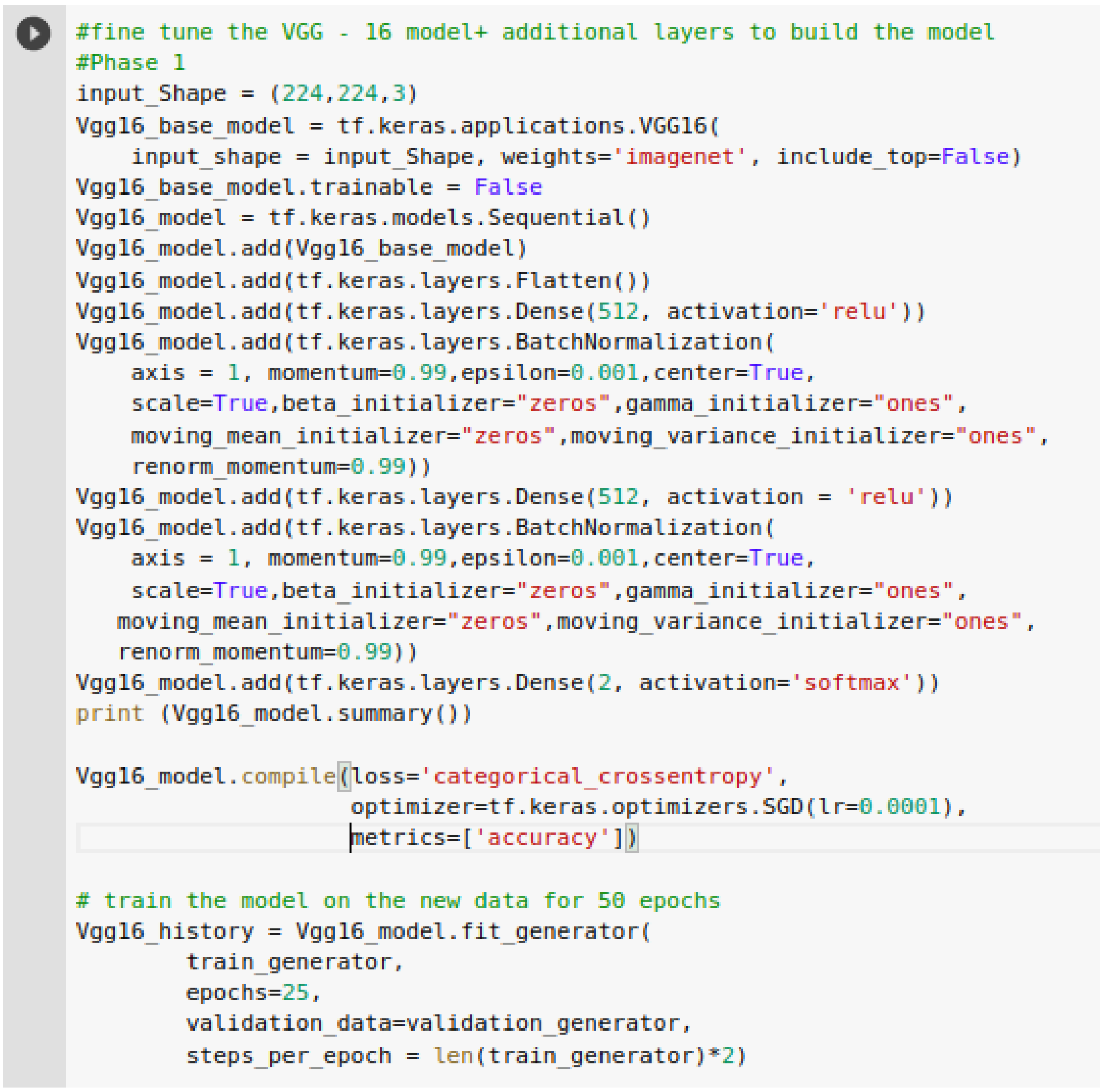
# Appendix A

# Code Snippets



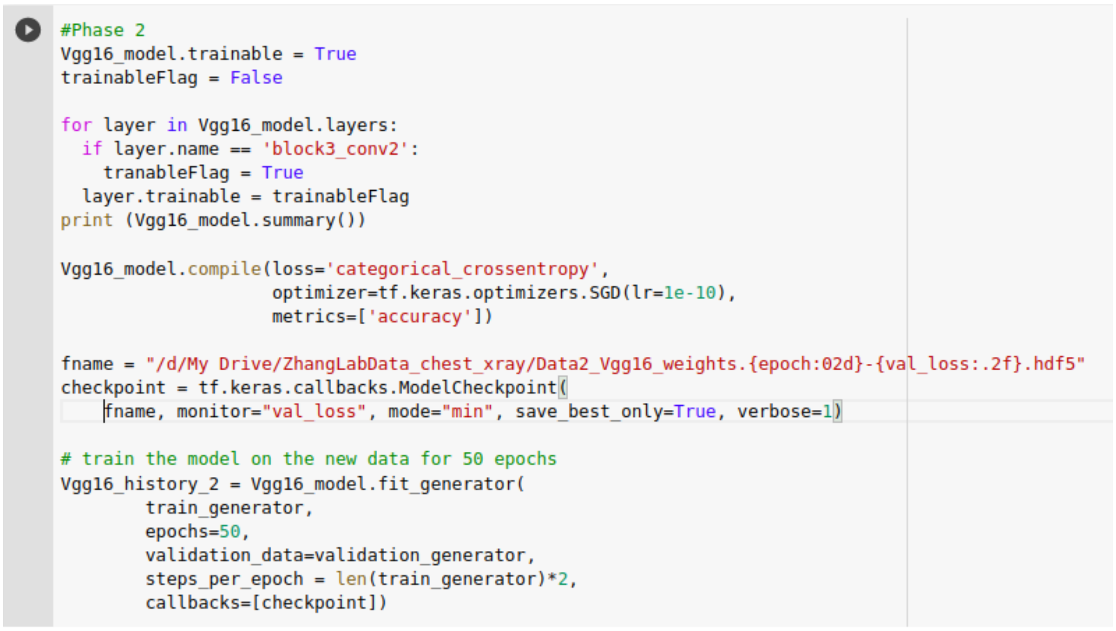
```
#use the ImageDataGenerator to augment the train and validation data.
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=20,
        horizontal_flip=True)

validate_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        '/d/My Drive/ZhangLabData_chest_xray/train/',
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical',
        color_mode = 'rgb')

validation_generator = validate_datagen.flow_from_directory(
        '/d/My Drive/ZhangLabData_chest_xray/test/',
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical',
        color_mode = 'rgb')

test_generator = test_datagen.flow_from_directory(
        '/d/My Drive/ZhangLabData_chest_xray/test/',
        target_size=(224, 224),

        class_mode='categorical',
        color_mode = 'rgb')
```

Figure A.1: Data Augmentation Code



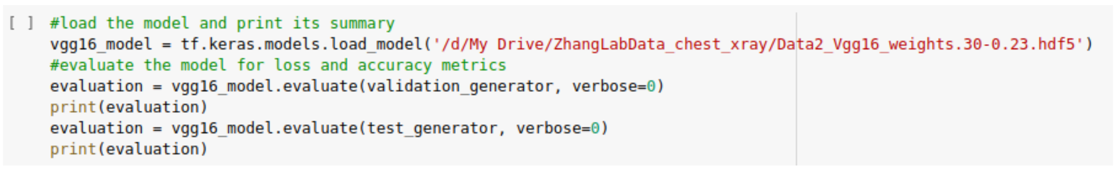
```
#fine tune the VGG - 16 model+ additional layers to build the model
#Phase 1
input_Shape = (224,224,3)
Vgg16_base_model = tf.keras.applications.VGG16(
    input_shape = input_Shape, weights='imagenet', include_top=False)
Vgg16_base_model.trainable = False
Vgg16_model = tf.keras.models.Sequential()
Vgg16_model.add(Vgg16_base_model)
Vgg16_model.add(tf.keras.layers.Flatten())
Vgg16_model.add(tf.keras.layers.Dense(512, activation='relu'))
Vgg16_model.add(tf.keras.layers.BatchNormalization(
    axis = 1, momentum=0.99,epsilon=0.001,center=True,
    scale=True,beta_initializer="zeros",gamma_initializer="ones",
    moving_mean_initializer="zeros",moving_variance_initializer="ones",
    renorm_momentum=0.99))
Vgg16_model.add(tf.keras.layers.Dense(512, activation = 'relu'))
Vgg16_model.add(tf.keras.layers.BatchNormalization(
    axis = 1, momentum=0.99,epsilon=0.001,center=True,
    scale=True,beta_initializer="zeros",gamma_initializer="ones",
   moving_mean_initializer="zeros",moving_variance_initializer="ones",
   renorm_momentum=0.99))
Vgg16_model.add(tf.keras.layers.Dense(2, activation='softmax'))
print (Vgg16_model.summary())

Vgg16_model.compile(loss='categorical_crossentropy',
                    optimizer=tf.keras.optimizers.SGD(lr=0.0001),
                    metrics=['accuracy'])

# train the model on the new data for 50 epochs
Vgg16_history = Vgg16_model.fit_generator(
        train_generator,
        epochs=25,
        validation_data=validation_generator,
        steps_per_epoch = len(train_generator)*2)
```

Figure A.2: VGG-16 Model: Phase 1 Code



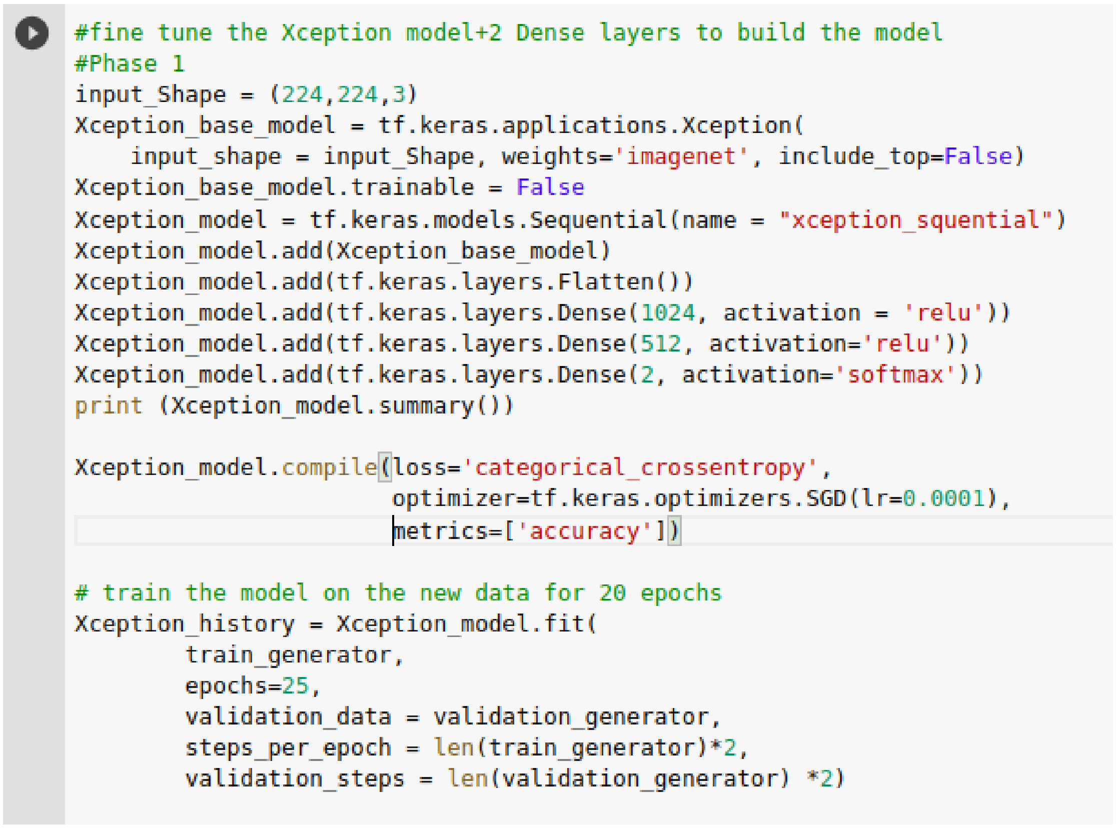
```
#Phase 2
Vgg16_model.trainable = True
trainableFlag = False

for layer in Vgg16_model.layers:
  if layer.name == 'block3_conv2':
    tranableFlag = True
  layer.trainable = trainableFlag
print (Vgg16_model.summary())

Vgg16_model.compile(loss='categorical_crossentropy',
                    optimizer=tf.keras.optimizers.SGD(lr=1e-10),
                    metrics=['accuracy'])

fname = "/d/My Drive/ZhangLabData_chest_xray/Data2_Vgg16_weights.{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    fname, monitor="val_loss", mode="min", save_best_only=True, verbose=1)

# train the model on the new data for 50 epochs
Vgg16_history_2 = Vgg16_model.fit_generator(
        train_generator,
        epochs=50,
        validation_data=validation_generator,
        steps_per_epoch = len(train_generator)*2,
        callbacks=[checkpoint])
```

FIGURE A.3: VGG-16 Model: Phase 2 Code



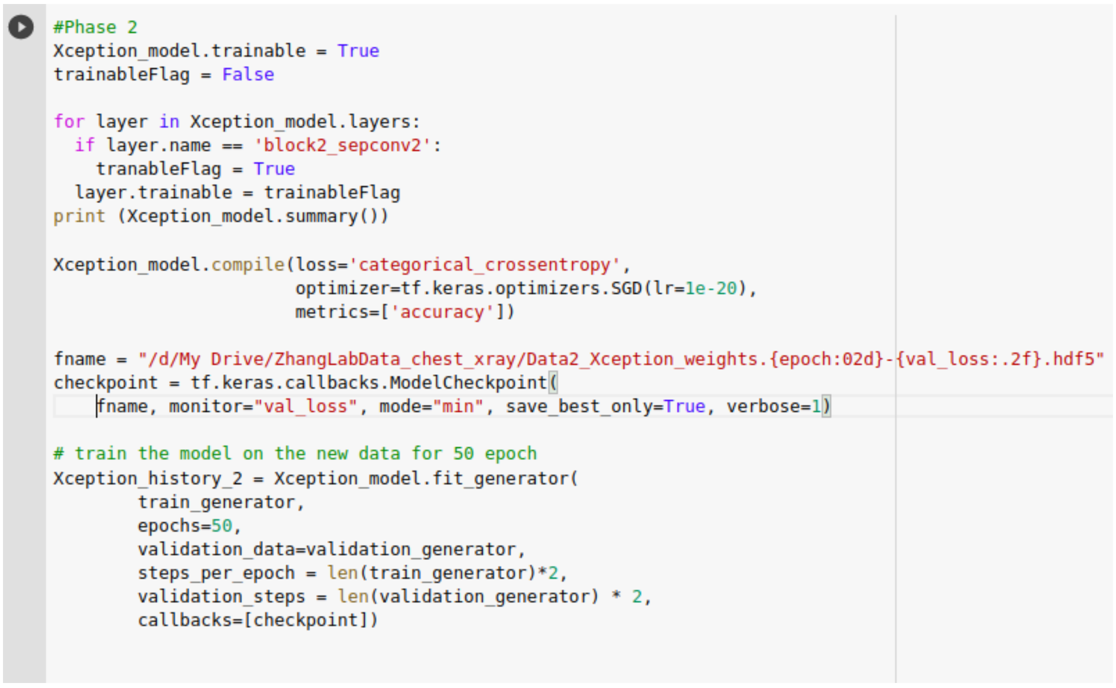
```
#load the model and print its summary
vgg16_model = tf.keras.models.load_model('/d/My Drive/ZhangLabData_chest_xray/Data2_Vgg16_weights.30-0.23.hdf5')
#evaluate the model for loss and accuracy metrics
evaluation = vgg16_model.evaluate(validation_generator, verbose=0)
print(evaluation)
evaluation = vgg16_model.evaluate(test_generator, verbose=0)
print(evaluation)
```

FIGURE A.4: VGG-16 Model: Evaluation



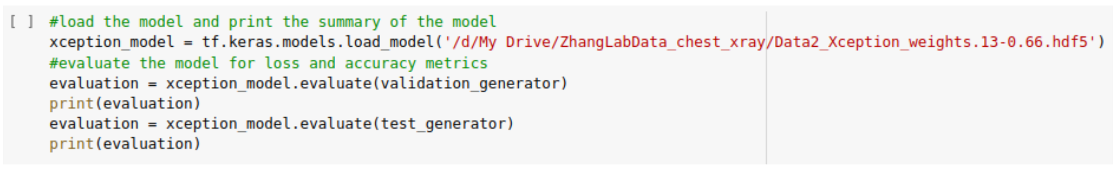
```python
#fine tune the Xception model+2 Dense layers to build the model
#Phase 1
input_Shape = (224,224,3)
Xception_base_model = tf.keras.applications.Xception(
    input_shape = input_Shape, weights='imagenet', include_top=False)
Xception_base_model.trainable = False
Xception_model = tf.keras.models.Sequential(name = "xception_squential")
Xception_model.add(Xception_base_model)
Xception_model.add(tf.keras.layers.Flatten())
Xception_model.add(tf.keras.layers.Dense(1024, activation = 'relu'))
Xception_model.add(tf.keras.layers.Dense(512, activation='relu'))
Xception_model.add(tf.keras.layers.Dense(2, activation='softmax'))
print (Xception_model.summary())

Xception_model.compile(loss='categorical_crossentropy',
                       optimizer=tf.keras.optimizers.SGD(lr=0.0001),
                       metrics=['accuracy'])

# train the model on the new data for 20 epochs
Xception_history = Xception_model.fit(
        train_generator,
        epochs=25,
        validation_data = validation_generator,
        steps_per_epoch = len(train_generator)*2,
        validation_steps = len(validation_generator) *2)
```

FIGURE A.5: Xception Model: Phase 1 Code



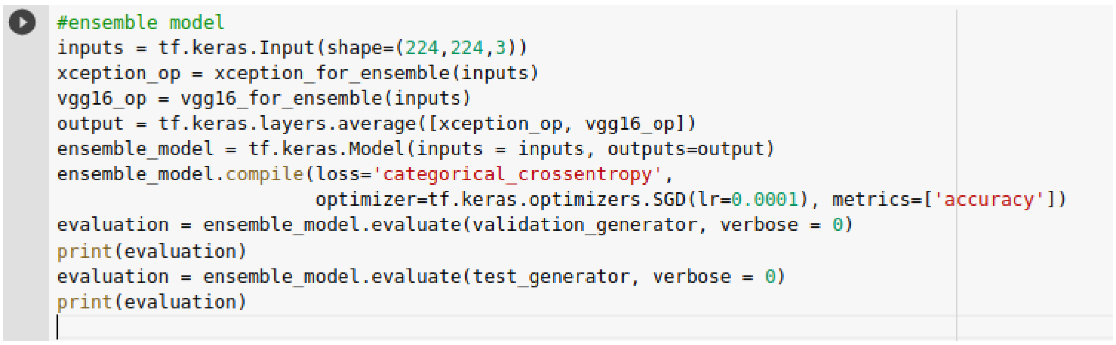
```python
#Phase 2
Xception_model.trainable = True
trainableFlag = False

for layer in Xception_model.layers:
  if layer.name == 'block2_sepconv2':
    tranableFlag = True
  layer.trainable = trainableFlag
print (Xception_model.summary())

Xception_model.compile(loss='categorical_crossentropy',
                       optimizer=tf.keras.optimizers.SGD(lr=1e-20),
                       metrics=['accuracy'])

fname = "/d/My Drive/ZhangLabData_chest_xray/Data2_Xception_weights.{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(
  fname, monitor="val_loss", mode="min", save_best_only=True, verbose=1)

# train the model on the new data for 50 epoch
Xception_history_2 = Xception_model.fit_generator(
        train_generator,
        epochs=50,
        validation_data=validation_generator,
        steps_per_epoch = len(train_generator)*2,
        validation_steps = len(validation_generator) * 2,
        callbacks=[checkpoint])
```

FIGURE A.6: Xception Model: Phase 2 Code



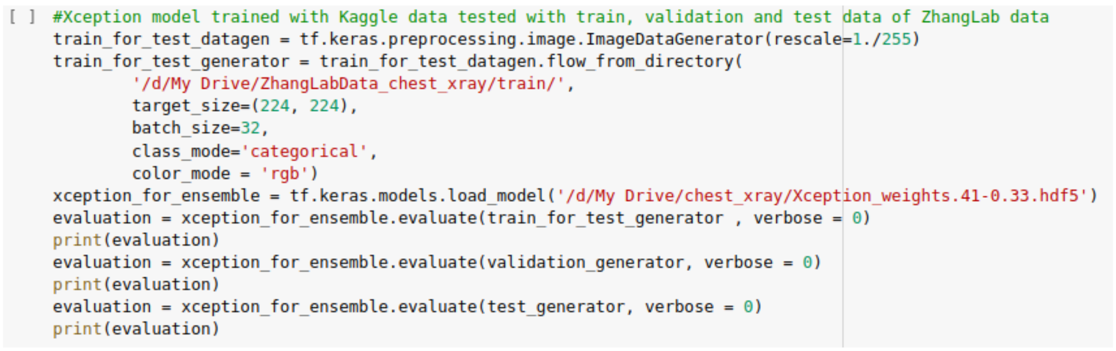
```
[ ] #load the model and print the summary of the model
    xception_model = tf.keras.models.load_model('/d/My Drive/ZhangLabData_chest_xray/Data2_Xception_weights.13-0.66.hdf5')
    #evaluate the model for loss and accuracy metrics
    evaluation = xception_model.evaluate(validation_generator)
    print(evaluation)
    evaluation = xception_model.evaluate(test_generator)
    print(evaluation)
```

Figure A.7: Xception Model: Evaluation



```
#ensemble model
inputs = tf.keras.Input(shape=(224,224,3))
xception_op = xception_for_ensemble(inputs)
vgg16_op = vgg16_for_ensemble(inputs)
output = tf.keras.layers.average([xception_op, vgg16_op])
ensemble_model = tf.keras.Model(inputs = inputs, outputs=output)
ensemble_model.compile(loss='categorical_crossentropy',
                       optimizer=tf.keras.optimizers.SGD(lr=0.0001), metrics=['accuracy'])
evaluation = ensemble_model.evaluate(validation_generator, verbose = 0)
print(evaluation)
evaluation = ensemble_model.evaluate(test_generator, verbose = 0)
print(evaluation)
```

Figure A.8: Ensemble Model Code



```
[ ] #Xception model trained with Kaggle data tested with train, validation and test data of ZhangLab data
    train_for_test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
    train_for_test_generator = train_for_test_datagen.flow_from_directory(
            '/d/My Drive/ZhangLabData_chest_xray/train/',
            target_size=(224, 224),
            batch_size=32,
            class_mode='categorical',
            color_mode = 'rgb')
    xception_for_ensemble = tf.keras.models.load_model('/d/My Drive/chest_xray/Xception_weights.41-0.33.hdf5')
    evaluation = xception_for_ensemble.evaluate(train_for_test_generator , verbose = 0)
    print(evaluation)
    evaluation = xception_for_ensemble.evaluate(validation_generator, verbose = 0)
    print(evaluation)
    evaluation = xception_for_ensemble.evaluate(test_generator, verbose = 0)
    print(evaluation)
```

Figure A.9: Ensemble Model Evaluation

# Appendix B

# Code and Data Links

- Code file 1 : AI_Reseach_project_ZhangLabData.ipynb
  This file consists of the Python code of the base models and ensemble model on primary dataset and a part of Experiment 2.
  Link : `https://colab.research.google.com/drive/1YJyWgTd8n0fxNZBB80WctwmInH2DJ5cv?usp=sharing`

- Code file 2 : AI_Reaserch_Project_KaggleData.ipynb
  This file consists of the Python code of the base models and ensemble model on secondary dataset(Experiment 1) and a part of Experiment 2.
  Link : `https://colab.research.google.com/drive/1jAWUXMQjgcPaAL5-L69uYM5DuKoD_ijE?usp=sharing`

- Code file 3 : AI_Research_Project_Frozen layers.ipynb
  This file consists of the Python code of the Xception model on secondary dataset(Experiment 3)
  Link : `https://colab.research.google.com/drive/1LNr8OBTpLPsIn8QFIxBe1T967--VB6F1?usp=sharing`

- Primary Data : ZhangLabData
  Link : `https://drive.google.com/open?id=1ZQtrR2tU6ho2T82m6s8Z0I0QjnJHMV0A`

- Secondary Data : Kaggle Data
  Link : `https://drive.google.com/open?id=1HX-CGldE4d11_7yCwZE3TaDOrMhdg15Y`