Automatic Detection and Multi-Class Classification of COVID-19, Pneumonia, and Tuberculosis Diseases in Chest X-ray Images Using Deep Learning Techniques

by

Mohamed Ali Hashem Metwally

A Report Submitted in Partial Fulfillment of the Requirements for the

Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Automatic Detection and Multi-class
classification of Covid19, Pneumonia, and
Tuberculosis Diseases in Chest X-ray Images
Using Deep Learning Techniques

by

Mohamed Ali Hashem Metwally

Supervisory Committee

Dr. Fayez Gebali, Co-Supervisor

(Department of Electrical and Computer Engineering)

Dr. Haytham El Miligi, Co-Supervisor

(Department of Electrical and Computer Engineering)

ABSTRACT

Recent technology advancements have set the stage for deep learning-based approaches to be applied in nearly every aspect of life. The precision of deep learning techniques makes it feasible to be utilized in the medical field for the automatic detection and classification of various illnesses. The recent coronavirus (COVID-19) epidemic has put enormous strain on the global health system. COVID-19 can be diagnosed via PCR testing and medical imaging. Because COVID-19 is extremely infectious, and PCR tests has some limitations regarding the processing time and highly false positive probability, chest X-ray diagnosis is deemed safe in a variety of conditions as it has less radiation compared to CT-scans, along with being widely available and cost effective.

A deep learning-based approach is suggested in this study to classify COVID-19 infection from other non-COVID-19 infections in the lungs. Multiclassification and detection of COVID-19, Normal, Pneumonia, and Tuberculosis was held using three distinct pre-trained models EfficientNetB0, Xception, and NasNetLarge which were selected based on their variance in size and accuracy on ImageNet. A dataset consists of 7135 X-ray images with four classes of Normal and different lung diseases including Covid-19 was utilized to train and evaluate the deep learning models, and two different training strategies were adopted. Not only three deep learning models were used in this study after tuning the optimum hyperparameters, but also a new classification head was generalized to boost performance and fit the classification problem along with data preprocessing techniques which were employed on the dataset such as pixel normalization and data augmentation to address the limitation of available data. Furthermore, during the model's training phase, a strategy with a class weight balancing technique was used, which significantly enhanced the performance of the deep learning models. Several performance parameters were used to assess the performance of the suggested strategies. NasNetLarge surpassed the other models especially in strategy II where class balancing technique was deployed in training phase with overall accuracy of 91.3%, recall, precision and F1 score of 0.94, 0.91, 0.91 respectively. EfficientNetB0 had also shown a competitive result with an accuracy of 88% since it has lesser execution time and considered more cost effective due its lightweight. This study is anticipated to be useful in enhancing medical practitioners' judgments and accuracy when identifying COVID-19 and other lung illnesses. This discovery will help future researchers reduce analysis duplication and pick the best network for their jobs.

Contents

List of Tables

List of Figures

ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Nueral Network
CNN	Convolutional Neural Network
COVID-19	Corona Virus
CV	Computer Vision
CT Scan	Computed Tomography Scan
CXR	Chest X-Ray
DL	Deep Learning
FE	Feature Extraction
FP	False Positive
FN	False Negative
GGO	Ground-glass opacification
LR	Learning Rate
MERS	Middle East Respiratory Syndrome
ML	Machine Learning
NumPy	Numerical Python
ReLU	Rectified Linear Unit
RSNA	Radiology Society of North America
RT-PCR	Real-time Polymerase Chain Reaction
SARS	Severe Acute Respiratory Syndrome
WHO	World Health Organization
TP	Tuberculous Pneumonia
TP	True Positive
TN	True Negative

Acknowledgment

I am thankful to Almighty Allah, who gave me strength and capability to get things done, as well as my parents and siblings for their unconditional support, wishes, prayers, and encouragement.

I would like to thank my supervisor, Dr. Fayez Gebali, for taking me as a student and all his support, guidance, mentorship, flexibility, patience, and encouragement throughout this program. I would like also to thank my co supervisor, Dr. Haytham El Miligi for his guidance, help and support.

I am also grateful to the University of Victoria for setting such a great environment to learn, enrich with top qualified professors and top-notch facilities.

Chapter 1

1. Introduction

By the end of 2019, many patients with pneumonia from unknown causes were reported in Wuhan, China. The Market of Huanan, which is a local wholesale for seafood and live animals, was the center of the region where these reported patients were working or living [1].

Pneumonia (lung infection) is considered an acute pulmonary disease. Pathogens, physicochemical and pharmacological agents, and immunological injuries can lead to an inflammatory condition. There are significant ways to classify pneumonia, whether it is non-infectious pneumonia or contagious pneumonia. It is classified based on the different pathogens that cause the infection, which leads to pneumonia being classified as bacteria, viruses, myco-plasmas, chlamydial pneumonia, and so on [2].

Acute tuberculous pneumonia (TP) is like typical bacterial pneumonia and can sometimes be misdiagnosed as regular pneumonia, but in matter-of-fact Tuberculosis (TB) is caused by the bacillus Mycobacterium tuberculosis (M. tuberculosis) and it is infectious and considered a communicable disease. M. tuberculosis can conquer several organs of the human body. The most common one is pulmonary tuberculosis (PTB). PTB is considered a chronic consumptive disease and can essentially lead to lung injury and tubercles, but it can also appear as acute pneumonia [3].

An unidentified etiology caused the early stages of the pneumonia that took place and spread in Wuhan. Infected people had severe acute respiratory syndrome (SARS), which led to serious complications in some of them, while others had rapidly increased acute respiratory distress disorder [4].

The World Health Organization (WHO) has received a report on this group of unidentified pneumonia cases in Wuhan, China on December 31, 2019. On the 7th of January, the "Chinese Center for Disease Control and Prevention" performed a throat swap of a patient, and a novel coronavirus (nCoV) was diagnosed and declared. In 2020, the World Health Organization (WHO) declared it a "public health emergency of international concern" and subsequently named it 2019nCoV, and soon after, it classified the novel disease as a pandemic by March 2020 [5].

The coronavirus may cause a wide range of illnesses, from ordinary colds to more serious disorders like Middle East Respiratory Syndrome (MERS) or severe acute respiratory syndrome (SARS). Coronaviruses (CoVs) are a large and diversified group of enclosed positive-sense single-stranded RNA viruses with envelopes. The viruses are highly transmissible and pathogenic, and they can spread by respiratory droplets or among humans in close proximity. It can cause a number of serious health problems, including damage to organs such as the heart and liver, as well as diabetes and pulmonary embolism. Most infected people exhibit symptoms including as coughing, fever, exhaustion, and loss of smell or taste at first. The infection moved to the lower respiratory system, including the lungs, in numerous fatal instances, producing diseases such as severe pneumonia, followed by multi-organ dysfunction syndrome with several secondary infections and shock [6][7].

1.1 Problem Statement

RT-PCR or real-time reverse transcription polymerase chain reaction test can detect COVID-19. The implementation process requires obtaining the DNA of the infected person with the help of the reverse transcription method, which is then subjected to PCR, which strengthens the DNA required for analysis. Thus, it can diagnose coronavirus since this virus just holds the RNA. However, the PCR test has some limitations, like it can take more than a small number of hours to more than 2 days to diagnose the COVID-19 virus. Furthermore, the PCR kit's results are not reliable because there is a high probability of false negatives [8]. Hence, health experts have a challenge in the form of major issues in identifying COVID-19 during its early phase. The researchers are working to find an effective solution for classifying COVID-19 infections. They are selecting CT scans and X-ray imaging assessments for identifying the coronavirus [9]. Chest X-ray tomography is very useful in detecting coronavirus infection through the observation of various distinct features like lung opacity and blurred patterns. Even though CXRs are not the most accurate examination, they are extensively utilized since they are less expensive than CT machines and do not require substantial patient preparation. Additionally, portable X-ray devices that may be relocated to non-critical hospital locations or even specialized facilities to investigate suspected COVID-19 instances are available.

According to the Radiology Society of North America (RSNA), specific manifestations in chest x-ray scans could be noticed include peripheral and bilateral consolidations with "crazy paving" pattern, and peripheral and bilateral ground-glass opacification (GGO) signals [10].

Furthermore, radiological characteristics can be seen in CXRs obtained from patients five days before getting COVID-19. CXR observations include airspace opacities such as consolidations or GGO, which can be unilateral or bilateral. Figure 1(a) depicts a healthy patient, while Figure 1(b) depicts a person infected with COVID-19.

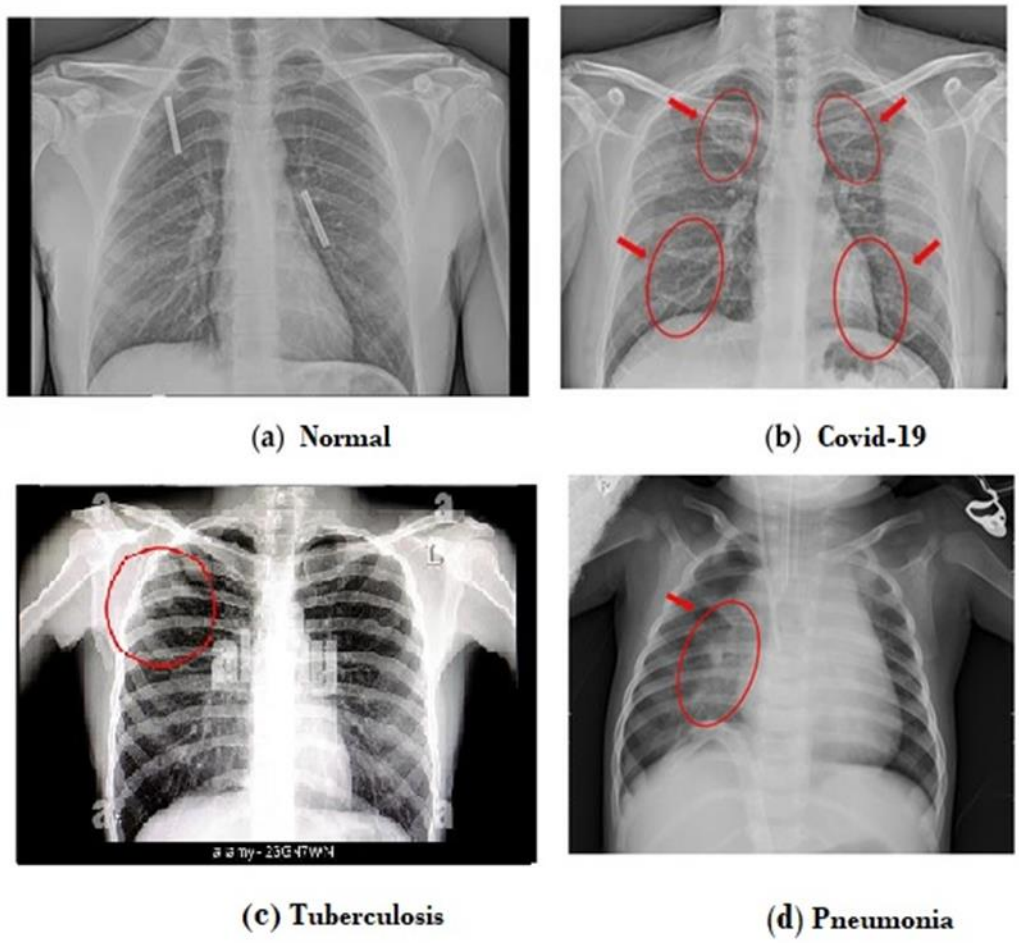


Figure 1. 1: CXR picture examples: (a) depicts a healthy subject with no thoracic anomalies; (b) depicts a COVID-19-infected patient with bilateral opacities; (c) depicts a patient with Tuberculosis; (d) depicts a patient with Pneumonia [11].

Pneumonia, on the other hand, is an infectious condition that also affects the lungs, and it is a leading cause of mortality in children, according to the WHO. A fungus, bacterium, or viral infection can all cause pneumonia.
Pneumonia produces chest discomfort and reduces the sick patient's oxygen intake. Similarly, pneumonia has radiological characteristics such as fluid buildup consolidations [11]. Figure 1 (d) depicts a patient with pneumonia. The right lung contains fluid and consolidations.

Tuberculosis is an infectious illness that causes tissue inflammation, the production of tubercles and other growths, and tissue death [3]. The x-ray of the chest in figure1 (c) indicates advanced pulmonary TB. There are many bright spots (opacities) of varied sizes running together (coalesce). Cavities are indicated by the circle around these bright spots. The left X-ray clearly indicates that the opacities are in the upper portion of the lungs, toward the rear. These remarks are characteristic of chronic pulmonary TB, although it can also be seen in chronic pulmonary histiocytosis and chronic pulmonary coccidioidomycosis.

Manual classification of COVID-19 infected chest X-rays is time-consuming and ineffective. However, the availability of these machines makes them the best choice for COVID-19 detection when there is no testing kits and screening stations. The biggest challenge in X-ray imaging based on the COVID-19 detection approach is the manual examination of each X-ray image and then the extraction of the inferences. It requires a lot of time and the availability of medical professionals. Hence, the computer-aided chest X-ray examination method is necessary for the detection of COVID-19 cases with the help of X-ray images. For this purpose, deep learning methods have proven to be very effective in coming up with very accurate results, further to other advantages like maximum unstructured data utilization, additional cost elimination, feature engineering reduction, and removal of explicit data labeling. Hence, deep learning methods are very effective for us in tracing the required features for classifying images with the help of their autonomous nature. Thus, without a doubt, the deep learning method proves that it is very helpful for analyzing medical images and getting top-notch classification performance through time-consuming simulated tasks [12].

1.2 Motivation

Even after three years of the viral epidemic and about 12 billion doses of vaccination given, the sickness is still destroying human health, life, and the global economy. There have been 599,825,400 confirmed cases of COVID-19, including 6,469,458 deaths reported to WHO as of 24 August 2022[13]**.**

The virus is extremely effective in rapidly mutating and progressively transforming into more lethal versions. Following the significant devastation caused by the Delta variety, a new version known as Omicron was found. The WHO has previously recognized Omicron as a dangerous variation. Omicron is extremely transmissible due to many noteworthy mutations in its spike proteins. Furthermore, there is still a chance of further new mutations in COV-2 in the future, which might lead to a more dangerous variant epidemic.

Inspired by the requirement of fast and correct analysis of radiography images, many researchers have come up with numerous COVID-19 pneumonia detection models which are based on state-of-the-art CNN models that have been proposed, such as Chowdhury et al., Pamuk et al., and Mpesiana et al. [14][15][16] , the results were quite promising, nevertheless, these studies have given us the results with the help of small datasets with only few classes like binary classification (COVID-19 pneumonia and non-COVID-19 cases) or with the help of three class classifications (normal cases, other pneumonia, and COVID-19 pneumonia) problems. In this research paper, we have utilized a larger dataset which consists of X-ray images of bacterial and viral Pneumonia, Normal cases, Tuberculosis, and COVID-19 pneumonia, and three models were trained based on deep learning pretrained models' architecture and transfer learning concepts. Preprocessing techniques and augmentation were performed to increase the dataset and two different training strategies were deployed to evaluate the performance enhancement of the models. The three models were trained and evaluated to help clinicians with automated better screening and detection of COVID-19, Tuberculosis, and bacterial and viral Pneumonia to come up with treatment strategies based on accurately identifying the causes of the infection.

1.3 Agenda

The organization of the project is as follows: Chapter 1 presents the project's problem, associated work and motivation were described, as well as the report's format.

Chapter 2 will go through some of the prior research done on the COVID-19 detection including binary and multi-classification using various Deep Learning methods.

In chapter 3, background was discussed including deep learning, convolution neural network and its structure, python and libraries used in deploying the project experiment

In chapter 4, the proposed framework was illustrated, an explanation of the dataset, data preprocessing steps, data splitting strategies, and DL networks for multi-classification and detecting of COVID-19, Pneumonia, Normal and Tuberculosis diseases.

In chapter 5, details and analysis were discussed involving the evaluation performance and results of the pretrained models on the trained dataset moreover, their performance was weighted after using two training strategies. In chapter 6, conclusion and recommendations for further work were summarized.

Chapter 2

2. Related Work

Zhang et al. [17] came up with 18 layers of residual CNN applied to these X-ray illustrations, and their findings were based on the 3 most important influences. They let the CNN module get the features and then operate on classification. Lastly, they applied this anomaly module for score detection. They experimented on 1531 radiographs, and out of them, there were hundreds of COVID-positive images and 1431 pneumonia-infected images. A sensitivity of 96% was achieved with a specificity of 70.65% in favor of COVID-19 recognition.

Apostolpoulos et al. [18] came up with the CNN model, and they applied it to two very small volumes of data sets. they used two distinct databases. The initial database had 1427 X-ray images, of which 224 were COVID-19 positives, 700 images of bacterial infection, and others of normal patients. The second data set had 1427 Chest X-rays COVID-19 illustrations, the same number of normal cases as the same number of pneumonia infected cases. They applied mobile-NetV2 on the second data set to generate 96.78% accuracy, 96.46% specificity, and 98.66% sensitivity in all CNN models. Then came Tsiknakis et al. [19] with another automated COVID-19 recognition model, which used a deep learning model called INCEPTION V3. They applied this method to 572 cases, which had 122 images of it positive, 150 normal cases, and 150 bacterial and viral infections. They were able to achieve a taxonomy precision of 76%.

Sethy et al. [20] used nine different types of transfer learning approaches on the dataset that had 381 chest X-ray images from which they extracted the features. They made use of SVM for the diagnosis of COVID-19 with the help of these mined features. The RESNET-50 came up as the best one for extracting features from the data set. The SVM and RESNET-50 models ended up with an F1 score and accuracy of 95.34% and 95.33%, respectively. Saha et al. [21] came up with a new method for corona detection based on X-ray images, they called it EMCNET, and it was based on the CNN structure, which is used to extract features from images. On the extraction of features, they applied machine learning ensembling classifiers for COVID-19 infected case classification. This method provided an accuracy of 98.91%.

Mahmud et al. [22] came up with an automatic COVID detection model that uses a deep learning model called COVXNET that automatically extracts the features from X-ray images. It uses depth-wise convolution phenomena to detect it automatically. They conducted experiments on normal X-ray images and pneumonia. Then they applied their proposed deep learning model for the classification of pneumonia pictures and COVID-19 images using X-ray images. This model is comprised of discriminative gradient-based localization and a slack algorithm.

Oh et al. [23] designed a CNN model that adopts a patch-centered structure that applies to a small portion of the data set. The system took the result from choosing the major portion of patches for classifier outcomes. Around 15,043 CXR pictures, which included 8851 normal, 180 COVID positive cases, and 6012 pneumonia cases, participated in the experiment. CNN achieved remarkable results and showed accuracy of 88.9%, recall of 85.9%, and precision of 83.4, 96.4% specificity, and an F1 score of 84.4%.

Xu et al. [24] came up with the COVID-19 detection technique with the help of a deep learning-based model on CT Images. They used two RESNET-based models. They were RESNET18 and a modified version of it with the localization mechanism. They used Noisy-OR Bayesian for final evaluation. This showed an accuracy of 86.7%. Hussain et al. [25] came up with a system known as CORODET, which was based on CNN and used for COVID-19 infection detection. This CNN model had 22 layers and was trained with the help of images from CT scans and chest X-rays. This model can classify non-COVID-19 and COVID-19 cases. It can also classify various classes, which include normal, pneumonia, and COVID-19. This should have excellent classification results.

Khan et al. [26] came up with the COVID-19 classification technique. Their proposed technique makes use of CNN for classification, which is the well-known deep learning model XCEPTION, which was specially modified for this purpose. The authors called the modified model CORONET. A chest X-ray dataset with four classes was used for training, which included fungal pneumonia, viral pneumonia, normal cases, and COVID-19 cases. With the aid of this data set, the model was trained with different combinations of it. It gave 89.6% accuracy. Chowdhury et al. [27] came up with a deep learning technique for the classification of viral pneumonia and COVID-19. They are used for training in this work in different types

of deep learning models. They also made use of the transfer learning approach for training deep learning models; a public data set was used for the training of models. It contains COVID-19 samples, a chest X-ray of normal and healthy people, and typical viral pneumonia. A very high level of classification accuracy was attained by this model.

Ozturk et al. [28] came up with 17-layered dark net models which were used for COVID-19 infection detection. Various sizes of filters have been employed in the darknet CNN layers. This technique classifies binary classes as well as multiple classes. In binary classes, it was classified as "no findings and COVID-19", and in multiple classes, it classified no findings, pneumonia, and COVID-19. In training the models, raw chest X-ray images were used, and the model acquired an accuracy of 98.08% for binary classification, and it attained an accuracy of 87.02% for multiple classes.

Most of the research for COVID-19 detection has been conducted with the help of chest X-rays, which signifies the significance of chest X-rays in chest infection diagnosis and most importantly, for diagnosing COVID-19. The chest X-ray is the most important tool in medical image analysis. Deep learning techniques are much easier than feature extraction techniques based on traditional image processing. Just recently, traditional techniques were surpassed by deep learning techniques [29]. Deep learning techniques require large data. The deep learning models that were trained with the help of limited datasets are not generic, and hence they are not reliable. Through the literature, it has been deciphered that techniques for data augmentation can be applied to resolve minor data set issues [30]. Moreover, the research already available focuses more on the binary classification of COVID-19 [17-22] and very little research has been conducted for multiple classes [23-26]. Overall, there are only a few multiple class research studies which have small datasets and a limited number of classes with classification performance that is not yet adequate and requires improvement.

Chapter 3

3. Background

3.1 Deep Learning

Ever since human-like intelligence has been in the human mind to incorporate it into the machine, they have been fascinated as much as they are when it comes to discovering ancient findings. Since the time of Greek mythology, humans have thought of human-like creatures with the power of speech and learning. As time passed, various technological advances gave humans the hope of fulfilling this ancient dream. Artificial intelligence takes us to the 1950s when humans created algorithms for simulating intelligence. By time the 1990s arrived, we had a different form of artificial intelligence that became popular, known as machine learning. This positively transforms inputs into the output by deciphering various representations between the known inputs and outputs. The growth in machine learning has been revolutionized by advancements in computer hardware and rapid growth in data science, which have made the machine learning subfield feasible. We know this subfield as deep learning, which is deep because it learns data representation with the help of successive layers. The most important implementation of this is a neural network, which is structured with the help of a series of layers.

3.2 Artificial Neural Networks (ANNs)

An ANN happens to be a computer processing system that is based on the human brain, like a biological nerve system. They are generally made of interconnected neurons in the form of computational nodes, which work in a distributed fashion for learning from the input and coming up with final output optimization. Figure 3.1 explains how the ANN's fundamental structure is modeled. The data gets loaded at the input layer, and the input layer distributes it to the hidden layers in multi-dimensional vector form. The hidden layers then assess how the stochastic change benefits within itself or harms the next layers' judgment for the final output. From the alignment of multiple hidden layers and piling them together, we form deep learning.

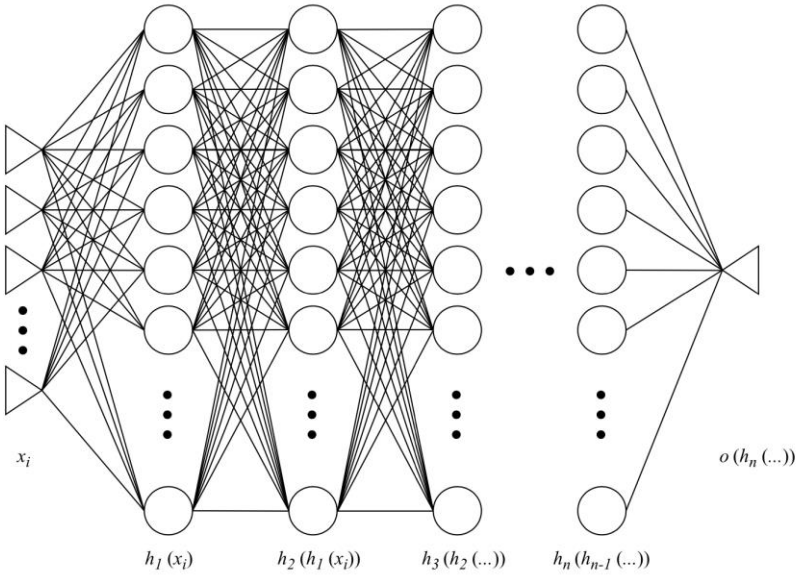


Figure 3. 1: Presents the structure of ANN

Each node a_{nj} in layer n takes in inputs $a_{n-1,i}$ where i = 1, 2, …, N_{n-1} from the previous layer and creates a weighted linear combination of the N_{n-1} inputs

$$a_{nj} = \sum_{i=1}^{N_{n-1}} w_{ij} a_{n-1,i} + b_{nj} \quad (3.1)$$

Where N_{n-1} is the number of nodes at layer (n - 1), and *i, j* give the position of the node. The term b_{nj} is an added bias term. The output is them obtained using a nonlinear function g(.) to produce $z_{nj} = g(a_{nj})$

3.3 Types of Machine Learning

Machine learning happens to be a subset of AI that empowers computers to learn without any programming. The root of it is in computational mathematics, statistics, and data mining. These machine learning algorithms get trained with the help of a data set with which they learn the data and make the prediction finally. Various types of machine learning algorithms have been described below.

3.3.1 Supervised Learning

In supervised learning, the machine learns with the help of pre-labeled inputs, which are the targets. For every training example, we have a set of vectors which are the input values and more or one designated output value. The main purpose of training of this sort is to reduce classification error in the models, with the help of accurate calculation of the output by training of training examples.

3.3.2 Unsupervised Learning

When it comes to unsupervised learning, the training set does not have any labels. And success is generally achieved by deciding whether the network was able to increase or decrease the loss function associated with it. However, it's essential to note that the majority of pattern recognition tasks are image-focused and are classification dependent with the help of supervised learning.

3.3.3 Reinforcement Learning

Reinforcement learning is a machine learning training method that rewards desired behaviors while punishing undesirable ones. A reinforcement learning agent can perceive and interpret its environment, act, and learn through trial and error. In the reinforcement learning model, learning happens with the help of feedback from the result of an event and experiences. This is much like how humans behave and learn from their environment. Aside from supervised learning, reinforcement learning does not need the presentation of labelled input/output pairings or the intentional correction of sub-optimal actions. It is more concerned to strike a balance between exploring new territory and exploiting it (of current knowledge). Some examples of reinforcement learning are Q learning, temporal difference, and deep adversarial networks,

3.4 Convolution Neural Network CNN

Artificial neural networks are performing quite well in various tasks. In applications that involve data in grid form, like images, Convolution Neural Networks have shown very improved and better performance. The convolution neural network makes use of convolution rather than general matrix multiplication in at least one of the layers [31]. Figure 3.2 is a

diagram of linear operation, where we create an output array with the help of multiple matrix multiplications in between the input array subsection and kernel. This process can be visualized by picturization of the kernel which slides over the array which is the input array in such a manner that for image details, it does the scanning of the image pixel group by group. Based on kernel weight, the operation output can give an output with very large values. That allows the model to reject very high-level features. There are hyperparameters here, and the two most important of them are kernel size and the number of filters. Each of the filters isn't a neuron equivalent in the artificial neural network, which is initialized with random weight and the convolution operation does its updating. As discussed, Kernel size is essential as it determines the capability of picking find details versus broad details in images, and the volume of data produced during operation.

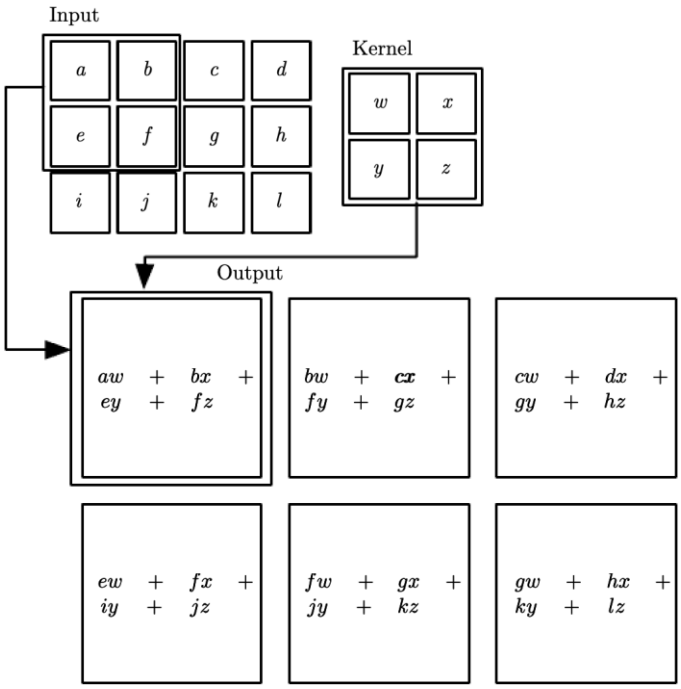


Figure 3. 2: Presents the use of Kernel in CNN

Once the convolution procedure is finished, the output is applied with a nonlinear function and the pooling of results takes place. By pooling, it means a function application to output a statistical summary rather than an input array. An example of this is Max pooling, which makes use of a rectangular array, and average pooling, which make use of the mean. This operation lowers the data size, as well as makes it invariant to minor translations [31]. Thus, the feature detection no longer depends on the location

3.4.1 Basic CNN structure

Convolutional neural networks (**CNN/CONVNET**)

It is a class of deep neural networks commonly used to analyze visual imagery. Convolutional Layer An essential component of CNN architecture, an example of a CNN model is shown below in figure 3.3:

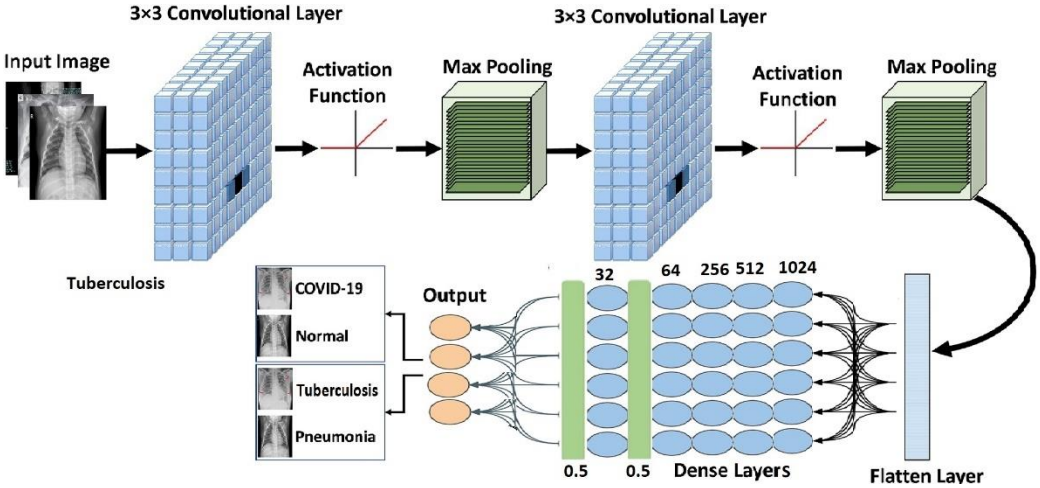


Figure 3. 3: Presents an example of basic CNN architecture

3.4.1.1 Convolutional Layer

Convolution and activation functions, among other linear and nonlinear techniques, are used in the convolution layer to extract features. Or, to put it another way, the objective of this layer is to extract reliable features from input pictures. The output of this layer, known as a feature map, is produced by applying it to its input. All these convolved maps are combined to create the convolution layers' final performance [32].

It's about matrix multiplications, but that is not the case with CONVNET. It uses a special technique called convolution, as stated earlier. To use mathematical terminology, convolution is a process that combines two functions into a single new function that depicts how each function is affected by the other.

Similar to a conventional neural network, the convolutional layer is a linear procedure that includes multiplying a set of weights with the input. The multiplication is carried out between an array of input data and a filter or kernel, The first layer in the above example in fig 3.3 uses 256 filters with a kernel size of (3,3); these filters are smaller than the input data and are multiplied by a dot product with a patch of the input equal to the filter size.

Each filter consists of weights that are trained during training models. Each convolutional layer has a bias that is used to map the feature independently. The number of weights in each convolutional layer is:

$$P = W + B = K^{B} \times C \times N + N \tag{3.2}$$

Where P is the number of weights, W is the number of kernel weights, and B is the number of the bias. K is the kernel size, C is the number of channels, and N is the number of filters.

3.4.1.2 Activation Function

It is a nonlinear mathematical function called the activation function. It generates the input by estimating the nonlinear functions. Rectified Linear Unit (ReLU) is one of the most commonly used Activation functions. It is characterized as: ReLU(x) = max (0, x). Figure 3.4 shows examples of activations functions [33].

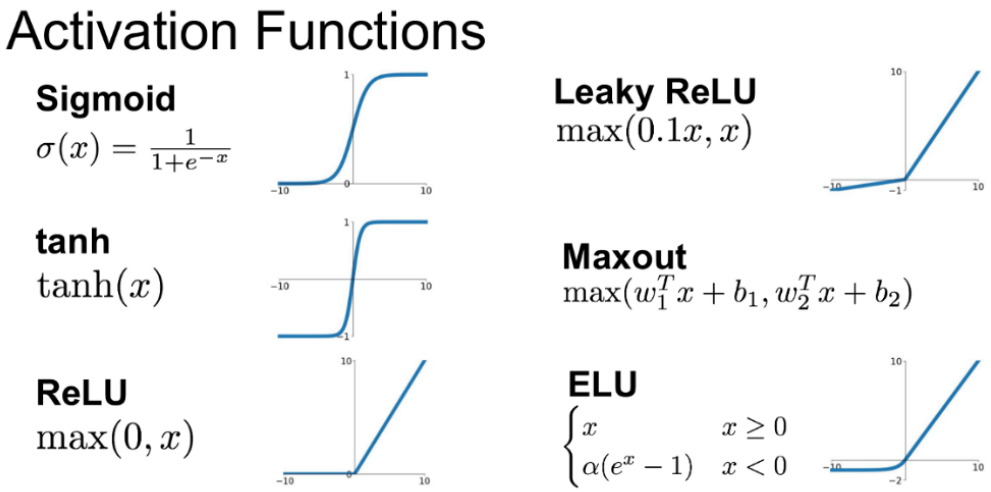


Figure 3. 4: Commonly used activation functions in neural networks.

.

3.4.1.3 ReLU Layer

In order to make the input nonlinear, utilizing activation functions, this layer may receive the convolution layer's output. The convolved feature's noise was eradicated, and its place was taken by the value 0. In the case of the lack of gradient, rectified linear units are the optimum solution. Besides, we apply ReLU activation to avoid the vanishing gradients problem.

3.4.1.4 Pooling operation

A convolved feature's spatial size is reduced by the pooling layer. This is done to reduce the calculations necessary to analyze the data and identify rotation and position invariant dominating features. Pooling is classified into two types: maximum pooling and average pooling. The maximum value from the portion of the picture covered by the kernel is returned by max pooling, whereas the average value is returned by average pooling. Figure 3.5 depicts the results of doing max and average pooling on a picture.

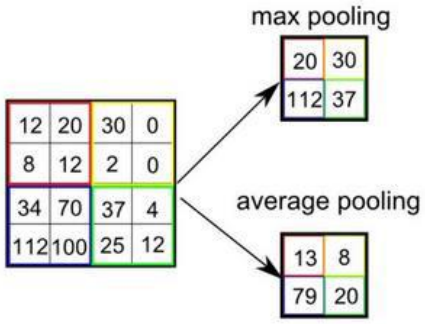


Figure 3. 5: Example of average pooling and max pooling in CNN [34].

3.4.1.5 Max pooling layer

The max pooling layer reduces the number of parameters and computations in the network by progressively shrinking the spatial size of the representation. Each feature map is operated independently by the pooling layer [33]. For each class, it is utilized to calculate its posterior probability, and the class with the highest probability is predicted. In this way, problems with binary and multi-class categorization can be resolved. Figure 3.6 illustrates max pooling.

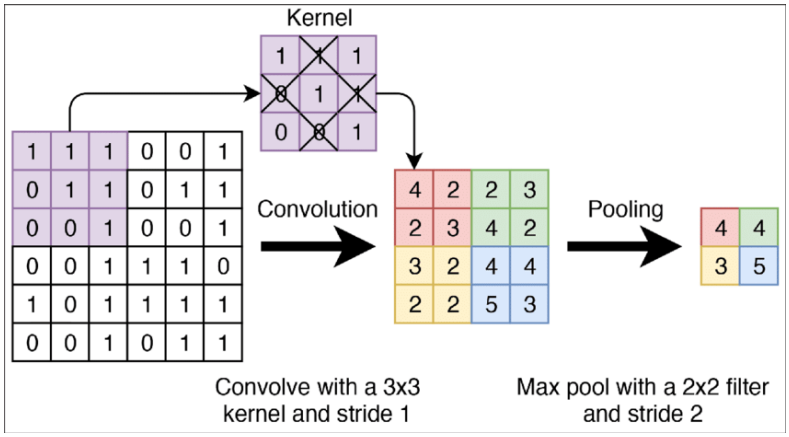


Figure 3. 6: Example of max pooling in CNN [34].

3.4.1.6 Flatten Layer

The 2-Dimensional arrays from the pooled feature maps may be flattened into a single long continuous vector using a technique called flattening. In order to classify an image, a flattened matrix is fed into a fully linked layer as input [34].

3.4.1.7 Dense Layer

Dense layers are added in structing the CNN. the dense layer, which is a basic structure layer made up of just neurons that receive information from those in the prior layer., therefore it is named "dense" or "thick". Based on the output coming out of the convolutional layers, the Dense Layer classifies images [35].

3.4.1.8 Dropout

Dropout is a technique used to avoid overfitting. Overfitting occurs in an ML model when the training accuracy is much larger than the testing or validation accuracy. Dropout is the practice of neglecting neurons during training such that they are not addressed during a certain forward or backward pass, resulting in a reduced network [35]. Figure 3.7 shows an example of how these neurons are picked at random. The dropout rate is the likelihood of training a particular node in a layer, with 1.0 indicating no nodes will dropout and 0.0 indicating that all layer outputs are disregarded.

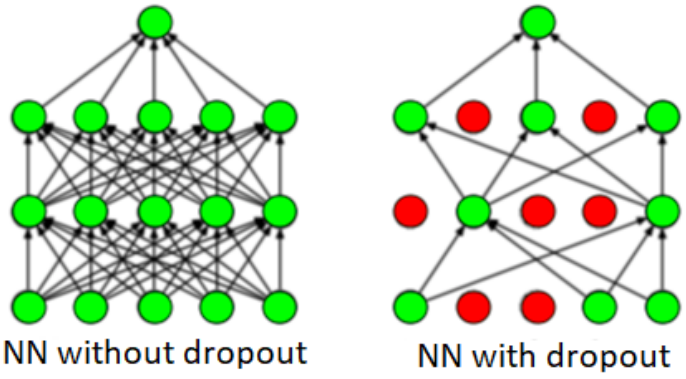


Figure 3. 7: Dropout in a NN

3.4.1.9 SoftMax

In the final layer, SoftMax is utilized with four nodes in a dense layer for four classes. It's a function that converts a vector of K real numbers into vector of K real numbers that ad d up to 1[35]. Regardless of whether the input values are positive, negative, zero, or larger than one, the SoftMax translates them into probabilities between 0 and 1. The SoftMax converts small or negative inputs into small probabilities, and large or positive inputs into high probabilities, yet it will always fall between 0 and 1, equation 3.3 presents SoftMax.

$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad \text{for} \quad i = 1, 2, \ldots, k \tag{3.3}$$

where z_i is the input tensor, z_j is the output tensor and k is The number of classes.

3.5 Python Libraries

Python is among the most developer-friendly free and open-source nature Machine Learning and Deep Learning programming language, and it supports a broad set of libraries:

Tensorflow and Keras: TensorFlow, together with Keras, happens to be a very popular Python library that is considered a deep learning standard. With the improvement in deep learning algorithms, different libraries have been made for implementing these algorithms so bigger datasets are processed effectively. TensorFlow (offered by Google**)** is written with the help of CUDA and C++ for speed, and it boosts performance through the capability of running over the GPU. Keras is a Python open-source neural network library. It was created by a Google developer and shortly supported in TensorFlow's core library, allowing it to be accessible on top of TensorFlow [36]. TensorFlow is an infrastructure layer for differentiable programming deals with tensors, variables, and gradients, whereas Keras is a deep learning user interface that deals with layers, models, optimizers, loss functions, metrics, and more.

NumPy: Numerical Python (NumPy) is an open-source Python library that works with arrays and matrices. In NumPy, an array object is referred to as (nd.array). The CNN inputs are arrays of numbers, and NumPy may be used to transform photos into NumPy arrays so that matrix multiplications and other CNN operations can be performed simply [36].

Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Chapter 4

4. Proposed Framework

Different techniques have been used to develop robust diagnostic models using deep learning methods. Several network designs were tested throughout the selection process, especially the concept of transfer learning, which has been widely used in CNN-based models, including, NasNetLarge, EfficientNet, and Xception.

The proposed framework, shown in Figure 4.1, describes the stages that must be taken, including the X-ray dataset description; preprocessing of the data; selecting pretrained models, i.e., Xception, EfficientNet, and NasNetLarage; training the selected models; and using evaluation metrics to evaluate the performance of the models.

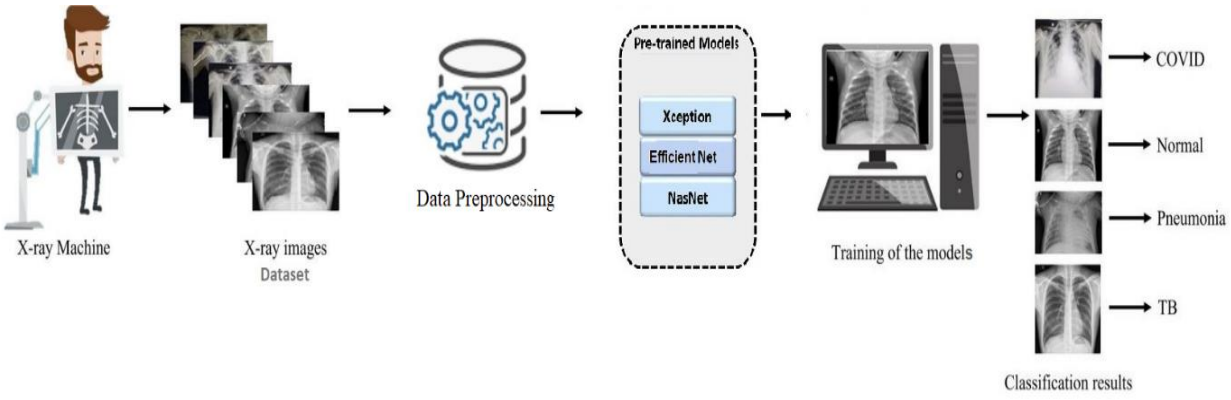


Figure 4. 1: Proposed Framework

4.1 Dataset

The dataset that was used to train and assess the suggested approach is available for free on Kaggle [37]. The dataset under consideration is composed of numerous smaller datasets that fall into one of these datasets, which have undergone several revisions. The dataset utilized in this study has four classifications: COVID-19, Pneumonia, Normal, and Tuberculosis. Given that the employed dataset was formed by integrating different datasets, it is significant to examine its composition in detail. Many diverse sub-datasets contributed to the creation of our dataset classes.

7134 CXR pictures were found in the dataset utilized in this project. The selected dataset from Kaggle was divided by the author into three folders (train, test, and validation) with four

subfolders (Normal/Pneumonia/Covid-19/Tuberculosis) with a total number of 1583 images, 4273 images, 576 images, and 703, respectively for each of the image's class mentioned earlier. Therefore, approximately 22% of the images are Normal, 59% Pneumonia, 8% Covid-19 and 10% Tuberculosis as can be seen in figure 4.2. which indicates a clear class imbalance problem that exists in the dataset.

Medical data often has an imbalance in some classes since the prevalence of a disease can often be small in a population. This imbalance can often hamper the performance of a model. Therefore, some measures such as data augmentation and dataset balancing techniques might be taken to minimize this effect when building and training a model.

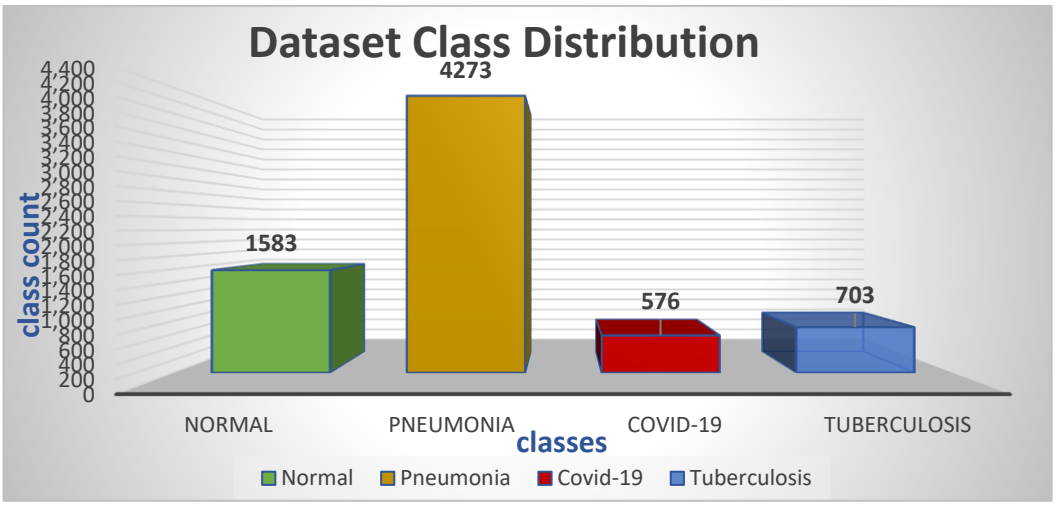


Figure 4. 2: Dataset Class Distribution

The selected dataset was produced using three publicly available databases, which contributed to forming all four classes of the utilized dataset.

Kaggle's "Chest X-Ray Images (Pneumonia)" [38] was used to obtain Normal and Pneumonia images. The Tuberculosis photos were obtained from the "Tuberculosis (TB) Chest X-ray Database" on Kaggle [39]. Finally, the pictures of COVID pneumonia were obtained from the Kaggle: "Covid-Chest Xray-Dataset" [41]. The collected images to form the research's dataset had a variance in their height and width due to the diversity of the sources from which they were collected. Figure 4.3 illustrates several images samples for each class including Normal, COVID-19 impacted chest X-ray images, Pneumonia chest X-ray images, and Tuberculosis Images.

Normal chest X-ray images show a clear lung and there are no irregular "opaque" areas. On the other hand, similar to other pneumonias, COVID-19 pneumonia increases the lung density, which can be seen as lung whiteness on radiographs. The most common changes in the lungs include nodular shadowing, consolidation, and ground glass opacities (GGO). They mainly affect the peripheral and lower parts of the lungs. The peripheral distribution of GGO is where lower lobe predilection is the most common. The images in figure 4.3 demonstrate clearly how hard it is to manually discern between them.

Chest X-ray Pneumonia Images Dataset

There are 5863 CXR images in the Chest X-ray Images (pneumonia) collection. Of those, 2780 are of bacterial pneumonia; the remainder is of normal images and viral pneumonia. The collected CXR scans were performed in Guangzhou, China at the Guangzhou Women and Children's Medical Center. The included images were JPEGs of various resolutions. Building the dataset used in this research required collecting all 2780 bacterial pneumonia photographs as well as the 1493 viral pneumonia photos in addition to the normal scans.

Tuberculosis Chest X-ray Database

The TB chest X-ray database had CXR images of both healthy people and tuberculosis patients. In order to create the dataset, many research teams have worked with medical professionals [39]. 700 tuberculosis images in the scale 512 × 512 Portable Network Graphics (PNG) file format. All 700 TB images were utilized to create the dataset used in this research.

COVID-19 Dataset

Montreal University has made a dataset for COVID-19 and other respiratory illnesses that is regularly updated so that it can be used to help make tools for diagnosing these illnesses.

Images from a variety of respiratory illnesses, including pneumonia, SARS, and MERS, among others, were included in the COVID-19 Image Data Collection, which Cohen et al. [41] presented. Only 576 of the CXRs in this collection were particularly from COVID-19 patients. The majority of them were from adult patients, ranging in age from 12 to 87, and various nations. In the dataset used in this study, only 576 COVID-19 images were taken from Montreal's dataset [41].

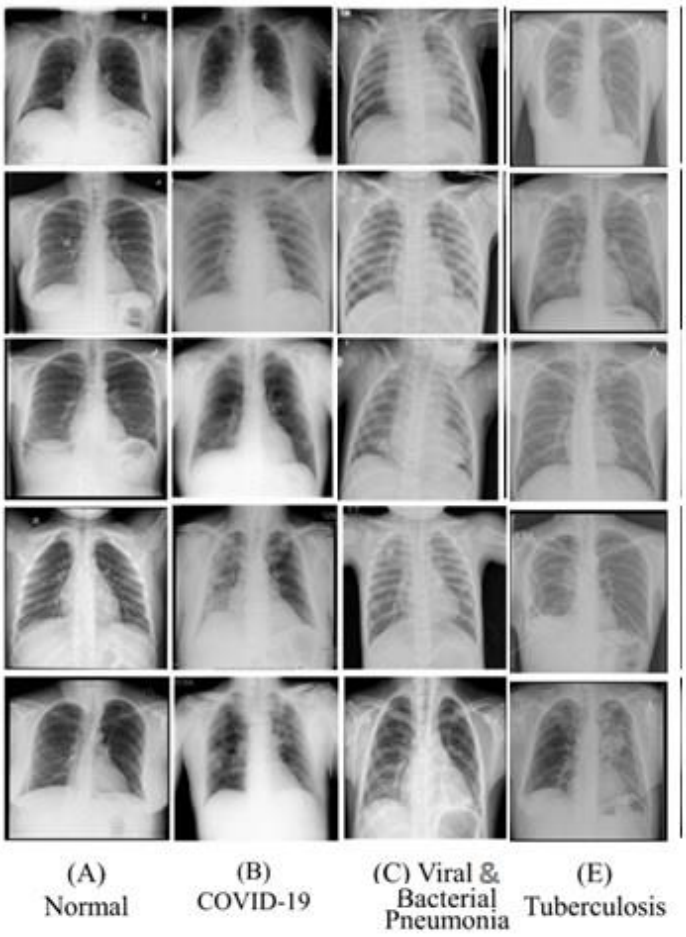


Figure 4. 3: Samples of Dataset Classes Images

4.2 Dataset Preprocessing

4.2.1 Data splitting

The choice of how to use the available data is an important element in machine learning. The most popular methods for dividing data into training, validation, and testing are percentage split and K-fold cross-validation. Data may be quickly and easily divided into training, validation, and testing groups using a percentage split. The two most often used split ratios are 80:10:10 and 70:20:10. The original dataset downloaded from Kaggle didn't have an appropriate split ratio as it was 89.90% for training, 10%, and only 0.05% for validation, which is significantly not proportional, especially for the validation set since the applied ration is quite low and would affect the validation process integrity.

Hence, an adjustment in the split ratio for the original dataset was needed, and the 80:10:10 split ratio set was adopted in this experiment, The dataset was re-split into three sections: training (80%), validation (10%), and testing (10%), with a count of 5708 images for training and 713 images for each of validation and test. The training and validation set of radiographs from normal and multiple chest diseases were used to train the experiment models.

It is essential to avoid using the test set prior to the testing phase with an assurance that it was kept unseen by the model until the test phase is conducted, since not doing so can distort the results and affect model evaluation integrity. Below, table 4.1 shows how the dataset's split ratio was adjusted.

Classes	**Source Dataset**			**Total**	**Modified dataset**			**Total**
	Training	*Validation*	*Test*	*Total*	*Training*	*Validation*	*Test*	*Total*
Normal	1341	8	234	**1583**	1162	211	211	**1584**
Pneumonia	3875	8	390	**4273**	3555	359	359	**4273**
Covid-19	460	10	106	**576**	371	102	102	**576**
Tuberculosis	650	12	41	**703**	621	41	41	**703**
Total	**6326**	**38**	**771**	**7135**	**5709**	**713**	**713**	**7135**

Table 4. 1: Original and modified data split ratios.

4.2.2 Data Normalization

It denotes that each feature has a range of values, with 1 being the greatest value and 0 being the lowest, due to the possibility that variations in scales among the input variables might make the problem being modeled more challenging. As an illustration, a model may learn huge weight values when given big input values (such as a range of hundreds or thousands of units). A model with high weight values is frequently unstable, which means that it may exhibit poor learning performance and sensitivity to input values, leading to a larger generalization error and the model employing the dataset may overfit.

Accordingly, a generally good rule of thumb is that input variables should have low values, preferably in the range of 0–1, or they should be standardized with a mean of 0 and a standard deviation of 1. Some of the pretrained models performs better with images ranging from -1 to 1, hence, Keras' function (preprocess_input) is utilized to adequate the image to the format the model requires, this it normalize the data to keep it either between 0 and 1, or – 1 and 1 as whatever fits with the used model.

4.2.3 Resizing the Images

Resizing the X-Ray pictures is one of the key steps in data preprocessing since various algorithms require different image inputs. It improves the performance of our system by shortening the training period, we employed picture preprocessing techniques including resizing to save processing time and to comply with our deep learning models' recommended image sizes. Thus, all our images were adjusted to either 244x244x3 pixels or 331x331x3 pixels by setting the img_width and img_height to the desired dimensions fit the used models.

4.2.4 Data Augmentation

It is often difficult to obtain datasets for deep learning model training since they are not always easily accessible. For effective training, deep learning models require datasets with a lot of samples that are of high quality [42]. As a result, the data augmentation technique was employed using the Keras ImageDataGenerator class on our dataset to make it more diverse without the need to acquire more data. There are various ways to do augmentation, like flipping horizontally or vertically, rotating, shifting, zooming, cutting, and increasing brightness, as shown in figure 4.4, but only suitable methods were picked and employed with our dataset to solve the data deficiency issue. This implies that each image in the training set is randomly manipulated using the following techniques:

Rotation: Rotating the image clockwise or counterclockwise at an angle of 45 degrees.

Scaling: Randomly sampling the image's frame size between 90% and 110% of its original scale size.

Brightness: Increasing the brightness range between 0.8 to 1.2.

Horizontal Flip: Flipping the picture horizontally with a 0.5 percent probability

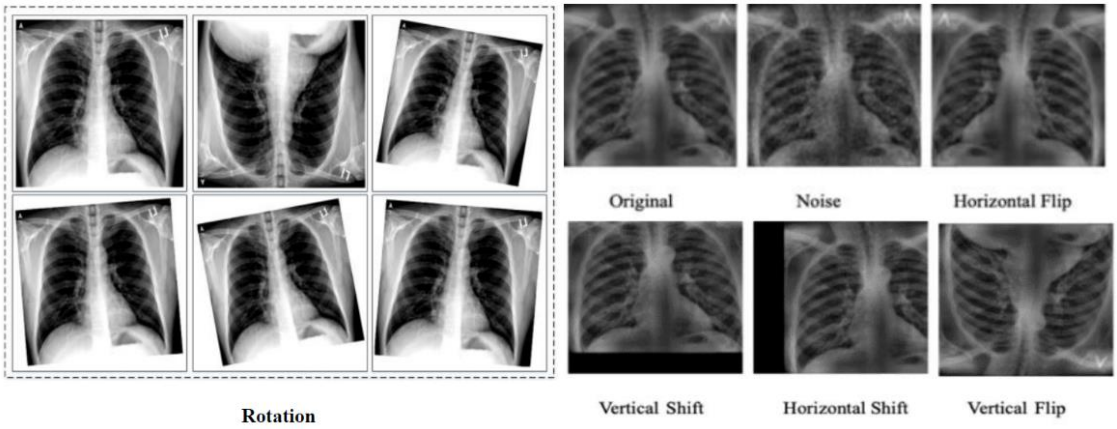


Figure 4. 4: Examples of Data augmented images.

4.3 Deep Learning Models

Convolutional layers are used in deep learning models to extract features from images. These features are also used to classify images by these models. Edges, contours, and other basic features of images are extracted in the first few layers of DL models, but the following layers extract more specific information.

In this work, the feature extraction and classification of chest x-ray images were carried out using three separate pretrained deep learning models: EfficientNetB0, Xception, and NasNetLarge. Theses models were chosen and employed based on some factors that will be explained in the next coming section as well as a brief discussion of the architecture of the models was provided in the following sections.

4.3.1 Selection of the Pretrained Deep Learning Models

TensorFlow and Pytorch have built very accessible libraries of pre-trained models, easily integrable into one's pipelines, allowing the simple leveraging of the transfer learning power. The ultimate question is how to choose between the (very) large catalog of models accessible in open source. One popular method is to follow previous research and literature for a similar problem.

The two main aspects to be considered in choosing deep learning models are often the same in most machine learning tasks; accuracy is one of them (the higher the better) and speed is the other one (the faster the better). The ultimate dream is to have a model that has super fast training with excellent accuracy. But as expected, usually to have better accuracy, a deeper model is needed, which means the model would take more time to train. Thus, the goal is to maximize the trade-off between accuracy and complexity [43]. This trade-off can be observed in the following graph, figure 4.5, and the table in figure 4.6 below.

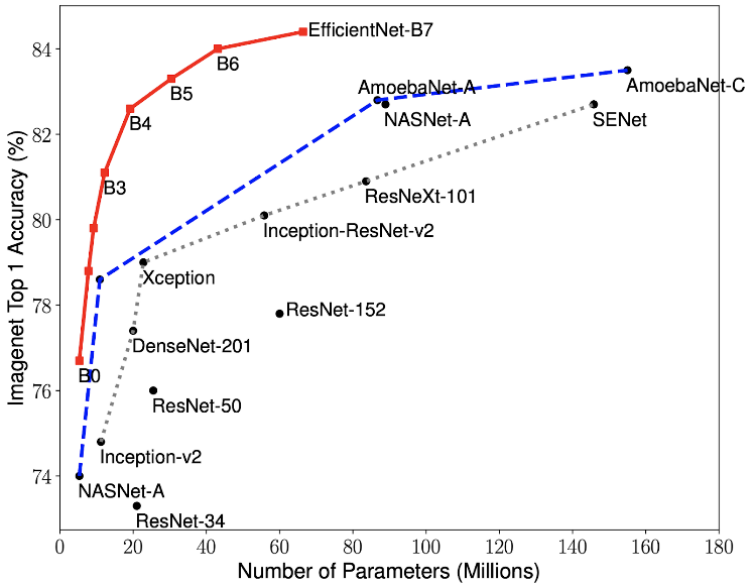


Figure 4. 5: Illustrates the size and parameters of pretrained models [43]

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9

Figure 4. 6: Illustrates the size and parameters of deep learning pretrained models [44].

As observed from the above figures of graph and table that the bigger models in size and parameters are often better in accuracy, but there is always a risk that a more complex model overfits the data because it can give too much importance to subtle details in features. A "good-enough" model that is small and therefore quickly trained is preferred. Nevertheless, if one is aiming for great accuracy with the least interest in quick training, then large models can be targeted.

In accordance, three various models in size and depth were selected and evaluated to determine their performance against the desired dataset, and their performance to the cost of training time was weighted. A relatively small model, which is EfficientNetB0, was chosen, along with a mid-sized model like Xception, and finally, a large model such as NasNetLarge was picked.

4.3.2 Transfer Learning

Deep neural network training frequently necessitates a huge dataset and a lengthy training time. Furthermore, the distribution of training and testing data is assumed to be the same in many deep learning models. Many real-world applications are hampered by the expensive cost of collecting and labeling data. In these cases, retaining and utilizing previously gathered data is essential.

Transfer learning is an approach of machine learning that utilizes previously trained models to be the foundation for a new model or task. Transfer learning entails applying features learned on one problem and leveraging them on a new comparable one. Transfer learning is often employed for tasks where the dataset has limited data to train a full-scale model from scratch. It not only reduces the need for the work required to acquire training data, but it also speeds up the training process.

When applying transfer learning on deep convolutional neural networks, there are two common workflows [44]. The most typical implementation of transfer learning in the context of deep learning is the following workflow, which uses the pretrained model to extract image features for the input of the new classification model while maintaining both its basic architecture and the learned parameters. This workflow freezes layers from a previously trained model to prevent any information they contain from being lost during subsequent training rounds. On top of the frozen layers, some fresh, trainable layers are added. They will learn to turn the old features into predictions on a new dataset and finally train the new layers on the targeted dataset.
Fine-tuning may be considered an optional final step. It entails unfreezing the whole model that was obtained earlier (or a portion of it) and retraining it using the new data at a very slow learning rate. By gradually tweaking the pretrained features to the fresh data, this final optional step has the potential to provide some improvements. Although it has the potential to bring about small advancements, it could also potentially lead to quick overfitting, moreover It should be applied with a very modest learning rate.

The second workflow is called feature extraction. It contains the instantiation of a basic model and load weights of a pretrained model to it, recording the results of one (or more) basic

model layers after running a new dataset through it. The output is then used as the foundation input for a new more compact model. One major benefit of the second workflow is that one can just run the base model once on the whole dataset, as opposed to once for each epoch in the training period. Therefore, it's both much faster and cheaper. However, there is an issue with that second workflow in that it does not permit dynamically altering the input data of the new model during training, which is essential when performing data augmentation. In case a new dataset contains insufficient data to train a complete model from scratch, transfer learning is often utilized in these circumstances with the aid of data augmentation, which is crucial. Hence, in what follows, we will focus on utilizing the first workflow in our research.

The first workflow is employed in this investigation without adding finetuning since the fitting problem is questioned and it requires a very low learning rate, which will increase the training time. ImageNet [RDS15], a massive database of more than 1.2 million high-resolution images, is deployed to pretrain the transfer learning models that are employed in the experiments. We can't utilize the pretrained model instantly to extract image features for classification since the images in ImageNet are different from our dataset. Pretrained parameters on our dataset must be tuned instead to provide better image features. Another modification would be deployed to the last fully connected layer, the outputs would be scaled to the number of our classification challenge instead of 1,000 in the image net. This would tune the model's ability to identify the images with the highest scoring in each class, which makes it a useful tool for image classification. Figure 4.7 shows how features from a model that has been trained on ImageNet with 1000 classes may be helpful to launch a model intended to recognize four classes of x-ray images.

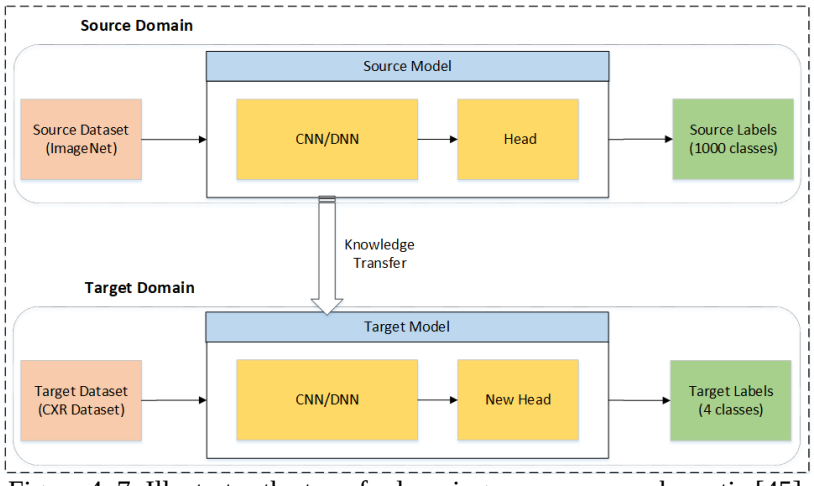


Figure 4. 7: Illustrates the transfer learning process as a schematic [45].

4.3.3 EffcientNetB0

It falls under the convolutional neural network family, and this family efficiently scales up when it comes to layer width, layer depth, input resolution, various permutations, and combinations of these factors. It leverages features creation from illustrations that can be later transferred into the classifier. This helps it to form the backbone of various other models, and one is EfficientNet, which is an object detection model family. EfficientNet is implemented in Keras, which is abstracted, and hence customized data sets can be loaded and the training of EfficientNet can be done in just a few lines of code.

This happens to be a simplistic compounded method of scaling, not much like conventional practice, which scales these factors arbitrarily. EfficientNet network scales uniformly network depth and resolution width with some fixed scaling coefficients set. As an example, to use 2^N times more computation resources, then the depth of the network is incremented by α^N, image size and width are incremented as well by γ^N and β^N, respectively. Here α, β, and γ happen to be constant coefficients which are yielded by a small grid search over their small original model, which makes sense, since if the input image is bigger, the network requires more layers for increasing receptive field as well as more channels for capturing better-grained shapes over the bigger image. Thus, we have a certain relationship between network depth and width [46]. Figure 4.8 illustrates the difference between EfficientNet scaling method and conventional methods.

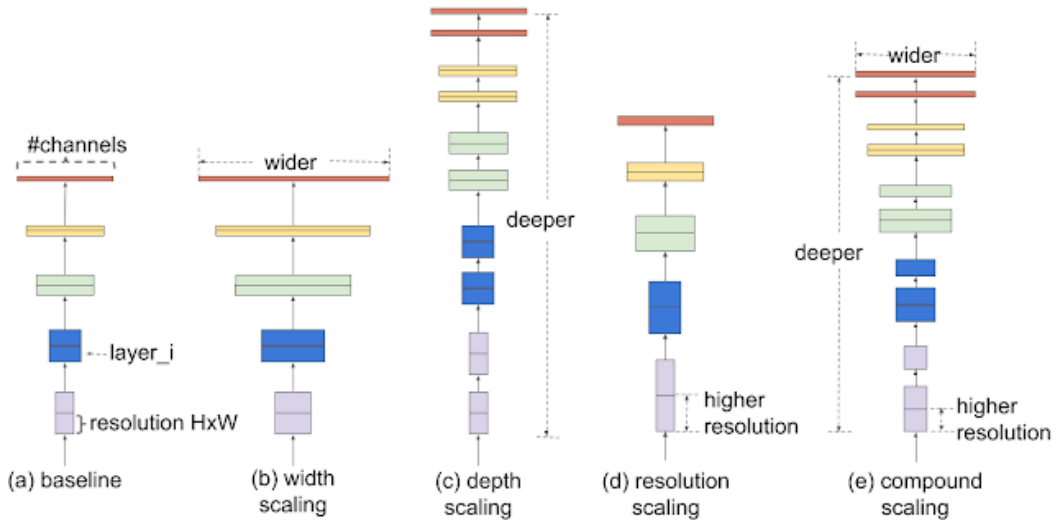


Figure 4. 8: Illustrates Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is EfficientNet compound scaling method that uniformly scales all three dimensions with a fixed ratio [46].

EfficientNet Architecture

The baseline network is also a factor when talking about the effectiveness of model scaling. Hence, for further performance improvement, a fresh baseline network was created through the performance of neural architecture search with the help of the AutoML MNAS Framework, which optimizes both efficiency and accuracy (FLOPS). The resultant architecture makes use of mobile inverted bottleneck convolution (MBConv), the same as MnasNet and MobileNetV2 networks. However, it is still large data due to the increased FLOP budget. Scaling up the baseline network would be obtaining a model family known as EfficientNets.

EfficientNetB0's core is the MBConv block. It is a residual block that is inverted to limit the number of trainable parameters. The MBConv's squeeze and excitation blocks provide weights to the MBConv block channels to help in the feature extraction process. Scaling in the EfficientNetB0 baseline includes both width and depth scaling as well as resolution. A blend of linear and sigmoid activation functions is employed in EfficientNetB0's swish activation function. Moreover, in order to keep the negative values, the swish activation function is used. EfficientNetB0 network which has depth of 132 layers and 5.3 million parameters, 240 x 240 is the input size for images used by it, figure 4.9 presents the EfficientNetB0 baseline design with MBConv's Blocks.

It employs seven blocks of inverted residual. A combination of squeeze and excitation blocks, as well as the swish activation function, are all employed. MBConv chunks are used by EfficientNet. There are two inputs to each MBConv block; data is the first, and block arguments are the second. The data is collected from the last layer. Input and output filters, expansion and squeezing ratios, and other MBConv block parameters are included in the block arguments. The goal of the expansion phase is to widen the layer. The kernel size specified in the block arguments is used in the depth-wise convolution phase. In the Squeeze and Excitation phase, the global average pooling is used to extract the global features. The squeeze ratio is used to reduce the number of channels [46]. The output phase applies convolution operation using the output filters mentioned in the block arguments. Figure 4.10 shows an example of the EfficientNetB0 full structure.

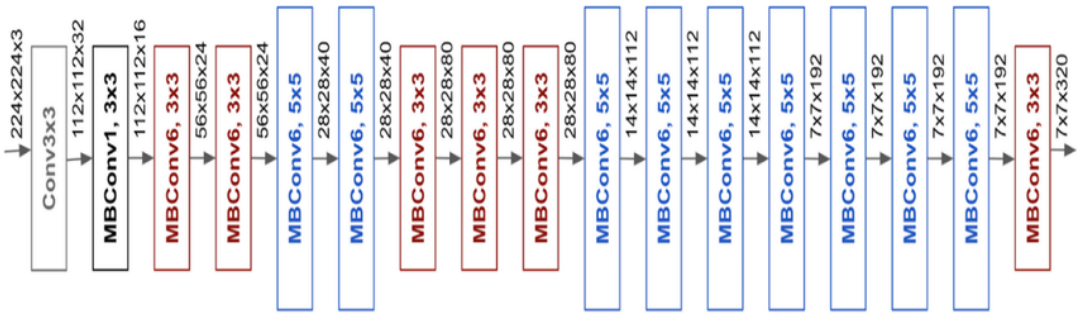


Figure 4. 9: Illustrates the structure of EfficientB0 with MBconv [47].

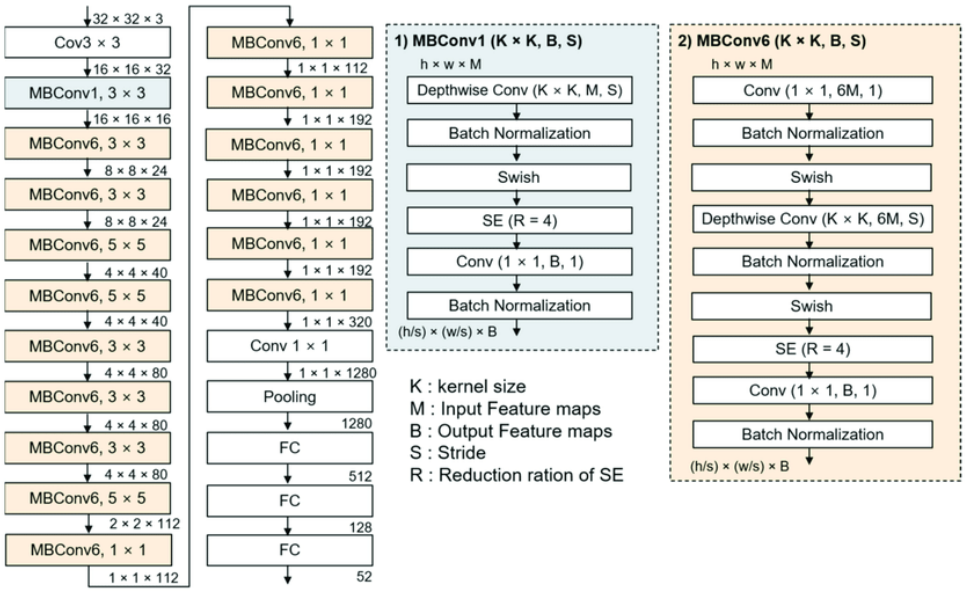


Figure 4. 10: Illustrates a full structure of EfficientNetB0 [47].

4.3.4 Xception

An Inception module computes numerous distinct transformations (as indicated in figure 4.11 below) over the same input before aggregating all the output, allowing the model to pick which features to take and by how much.

There is one issue because of convolutions, it is still computationally inefficient. These convolutions occur not just spatially but also across depth. As a result, for each new filter, convolution must be performed over the input depth to generate only a single output map. Thus, the depth becomes a significant bottleneck in the DNN.

This depth can be lowered by doing a 1x1 convolution across it. This convolution takes the spatial information from numerous channels and compresses it to a smaller dimension. For example, the

input of size 128x128x100 (with 100 feature maps) can be compressed down to 128x128x30 using 30 1x1 filters.

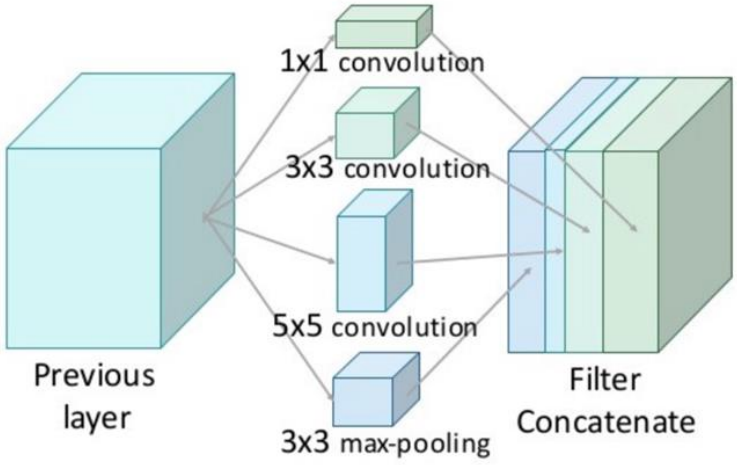


Figure 4. 11: illustrates the structure of the Inception module.

Xception stands for "extreme inception," and it takes the concepts of Inception to their logical conclusion. In Inception, 1x1 convolutions were utilized to compress the original input, and different types of filters were employed on each depth space from each of those input spaces. This process is simply reversed by Xception. Instead, it initially applies the filters to each depth map before compressing the input space with a 1x1 convolution across the depth. This approach is nearly equivalent to a depth-wise separable convolution, which has been utilized in neural network building since 2014. One further distinction exists between Inception and Xception, the existence or absence of a nonlinearity following the first operation. Both processes in the Inception model are followed by a ReLU non-linearity, but it is not deployed in Xception .

The Extreme Inception model, often known as the Xception model, is an “extreme” improved version of CNN's Inception model. In the Inception model, deep convolutional layers and broader convolutional layers work in parallel, this model is divided into two tiers, each with three convolutional layers. The Xception model, unlike the Inception model, has two levels, one of which has a single layer, this layer divides the output into three parts and sends it to the following series of filters. The first level contains a single convolutional level of 1x1 filter, whereas the second level has three convolutional levels of 3x3 filters. Figure 4.12 depicts the architecture of the Xception network model. The Xception model incorporates depth-wise and pointwise

convolution, whereas a typical deep CNN model handles spatial and channel distribution. The depth-wise Separable Convolution is the feature that defines the Xception model [48].

The Xception model is an 81-layer deep CNN that was inspired by Google's Inception model and is based on an extreme interpretation of the Inception model. Its design consists of depth-wise separable convolutional layers layered on top of each other. It has 22.9 million parameters, 240 x 240 is the input size for images used by it. The model's pre-trained version is trained on millions of photos from the ImageNet database. Furthermore, this model provides comprehensive representations of its utility for a wide variety of images and can identify hundreds of object types. The Xception model is extremely useful in picture recognition and classification. In this study, the Xception model is used among other DL models to assess the x-ray dataset in order to classify the lung's infectious diseases, Chollet's work [49] demonstrates the advancement of Xception above earlier CNN models.

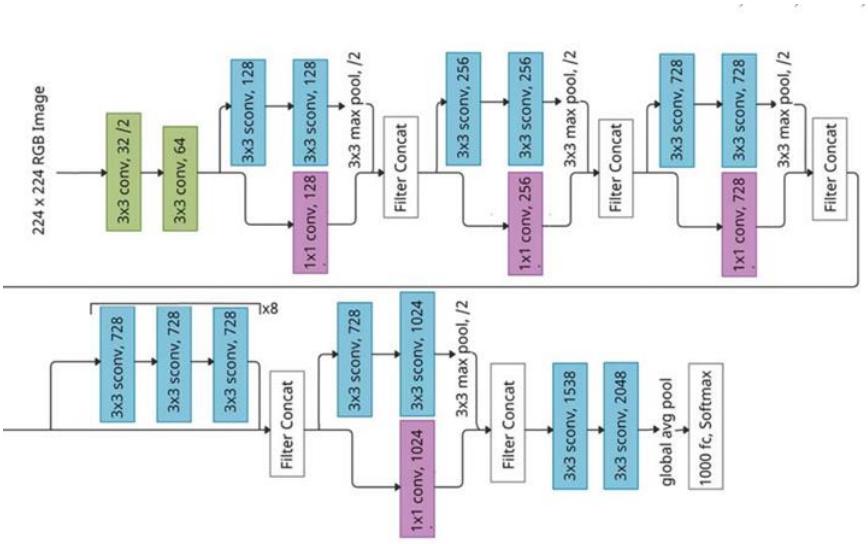


Figure 4. 12: Illustrates the structure of Xception Network [48].

4.3.5 NasNetLarge

NasNet or neural architecture search, is an abbreviation for new research architecture and happens to be a machine learning model. The main principle differs from models like GoogleNet, which is going to bring a major disruption in AI quite soon.

Generally, it was thought to find the most efficient combination of parameters in the provided search space of output channels, filter sizes, number of layers, etc. This is an example of reinforcement learning, and the reward after each search is the accuracy of this architecture

on the provided data set. Neural architecture engineering is automated by neural architecture search, as shown in figure 4.13, this happens to be an algorithm that finds the best algorithm for achieving the best turnout for a certain assignment.

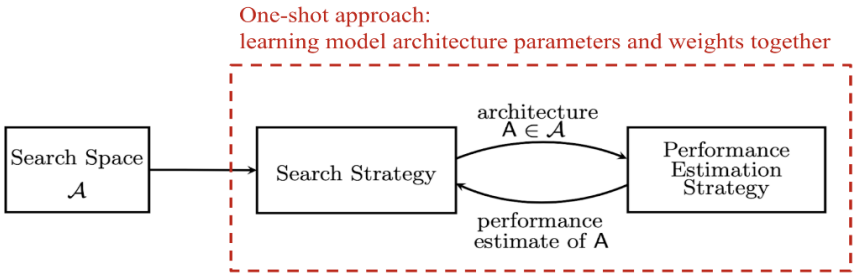


Figure 4. 13: illustrates the search strategy process in NasNet as a schematic diagram.

Search Space:

In simple terms, it defines the space of all possible architectures to look for, concerning a given problem. The NAS search space can be seen as a subspace of the definition of this neural network. Given a deep learning problem, there are two fundamental groups of search spaces as shown below in figure 4.14:

a) Global search space:
   i) This allows a lot of freedom related to the arrangement of operations and, hence, it permits all types of architecture.
   ii) It has a set of permissible operations like convolutions, dense layers with activation, pooling, and various global average pooling with varied hyperparameter settings such as filter height, filter width, and several filters.
   iii) It does not permit repetitive architectural patterns.
   iv) It comes up with certain constraints, like not inheriting dense layers before the first set of convolution operations or beginning a neural network as the first operation with pooling.

b) Cell-based search space:
   i) This focuses on deciphering the architecture of specific cells, which are combined for an entire neural network assembling.
   ii) It is based upon the inference that a lot of effectively designed manual architecture comes up with fixed structure reputations, which we also know as cells. A perfect example of this can be seen in the GoogleNet image.

iii) This also consists of minute-size cells which are stacked for farming larger architecture.

iv) The main advantage of this is that it forms architectures that are smaller but more efficient. However, they can also be combined into bigger architectures.

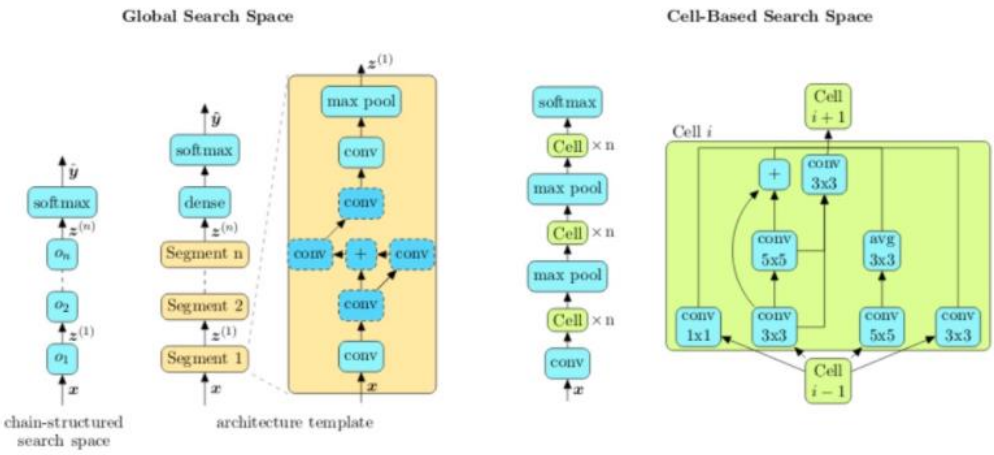


Figure 4. 14: Presents the Global search space and Cell-based search space.

Search Strategy

NAS is based on search strategies. The search strategy leverages the identification of the most efficient architecture for estimation of performance and bypasses bad architecture testing. There are many strategies, such as grid search, random search, and gradient-based strategies, as well as reinforcement learning strategies and evolutionary algorithms.

The NasNet is a machine learning model that stands for neural architecture search network, The Google ML group's NAS concept was used to accomplish this proposal. In order to design the convolutional architecture known as a "NASNet architecture", Zoph et al. worked on finding the best convolutional layer (or "cell") on the CIFAR-10 dataset and then applied this cell to the ImageNet dataset by stacking together additional copies of this cell, each with their own parameters [50] .NasNetLarge is a CNN architecture that obtained a top-1 82.7% accuracy on the ImageNet dataset. It uses a reinforcement learning search method to find the best architecture configurations. It is made up of both reduced and normal cells as shown in fig 4.15. The normal cells are convolutional cells that return a two-dimensional feature map. Reduction cells are convolution cells that return a feature map with a two-fold reduction in feature map and breadth [50].

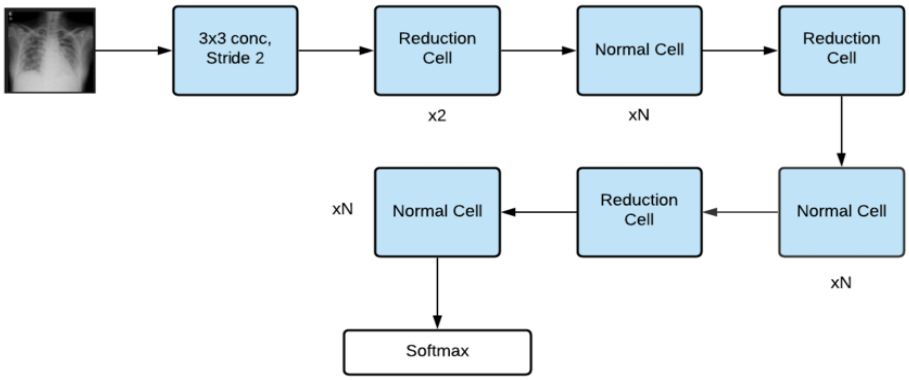


Figure 4. 15: illustrates the structure of NasetLarge with Normal and Reduction Cells [50].

4.3.5 Deploying Transfer Learning Approach on The Pretrained Models

First, pretrained models EfficienNetB0, NasnetLarge, and Xception were downloaded from the Keras application library. Top layers of the models weren't utilized because these models were trained on ImageNet dataset with 1000 classes. The models in this research would only be trained with only 4 classes, so it's recommended as stated earlier to remove the top layers.

Since the features extracted in the last convolutional layer maintain greater generality than the final layer, we have relied on it. As a result, the FC or top layers are not included when the EfficientNet, NasNetLarge, and Xception models were preloaded while preserving their weights from ImageNet. Instead, customized top layers were added, which would fit our dataset problem.

The new fully connected layers were added called "Dense layers" of sizes 512, 256, 128, and 64, respectively. The last fully connected layer was activated using the SoftMax function. All the convolution blocks were frozen while the model is instantiated. During training, freezing prohibits the weights in each frozen layer from being updated. In our research, we have used an average pooling layer on the pretrained model to get the output of the pretrained models to build custom models and train only the added custom top layers to keep the accuracy from the pretrained model. The additional layers related to the classification tasks are then defined in order to provide predictions from the block of features.

First, prior to passing the pooled feature map to the FC layer, the extracted output features from the last feature extraction layer of the pretrained models were flattened so that the pooled feature map is converted into a 1-dimensional vector before passing it to the FC layer. Flatten

layer was the first of the custom layers used to convert the data from pretrained model into a 1-dimensional array for inputting it to the next layer. Then the Dense layer was added with ReLU as activation, Dense layer is a simple layer of neurons in which each neuron receives input from all previous layer's neurons, thus called dense. Dense Layer was used to classify images based on output from convolutional layers. ReLU activation was employed to avoid vanishing gradient. Next, dropout layer was added to prevent overfitting on the training data with a random reduce 0.1%, and the last layer is Dense with 4 node output since our dataset has only 4 classes. Finally, Activation Softmax was used to get probability of each class figures 4.16, 4.17, and 4.18 show the pretrained models after adding the customized top layers

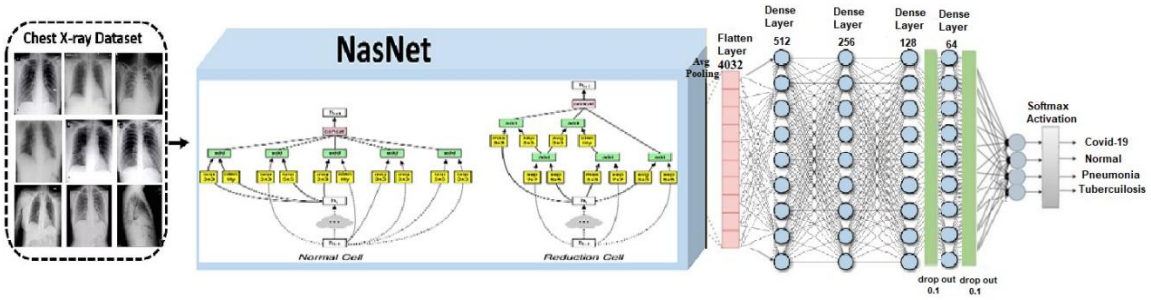


Figure 4. 16: Illustrates the structure of NasNetLarge with the new classification head.

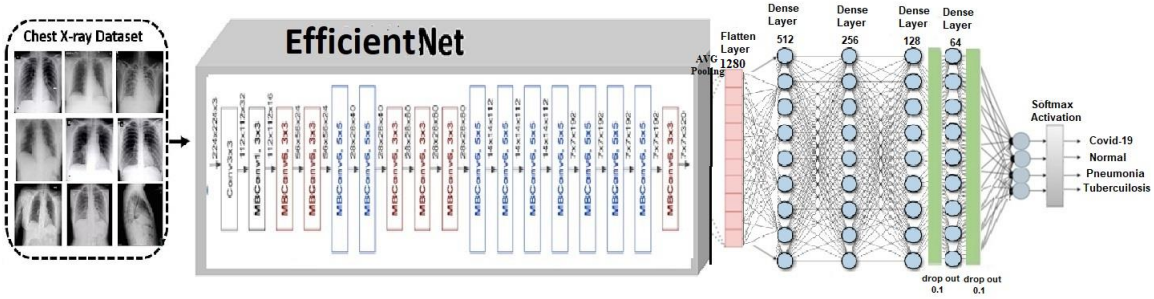


Figure 4. 17: Illustrates the structure of EfficientNetB0 with the new classification head.

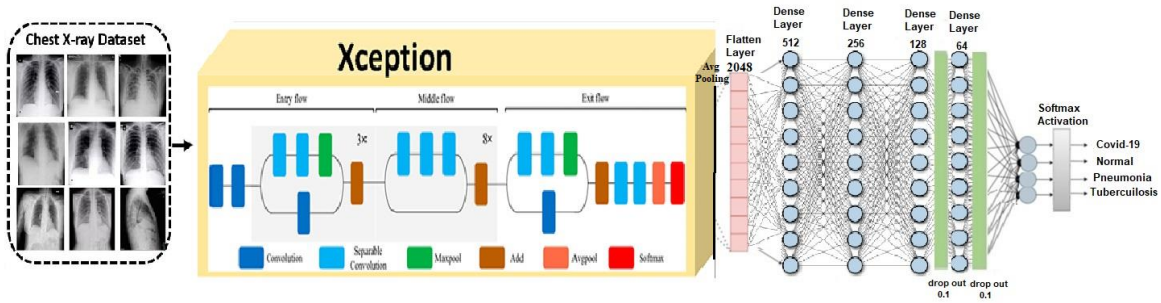


Figure 4. 18: Illustrates the structure of Xception with the new classification head.

4.4 Compiling the models

Compiling the models required the use of two parameters: optimizer and metrics. Adam was the chosen optimizer to be in use. Based on the loss, the optimizer was used to adjust the weights of DL models. Hyperparameters were experimentally chosen through iteratively rerunning the model with different parameters during training or by reviewing previous literature and are discussed briefly in the next section. Accuracy, categorical cross-entropy loss, precision, recall, and F-score were the metrics employed. These metrics were defined in Chapter 4.

4.5 Models' Training

As previously stated, the four-class chest X-ray dataset was downloaded from Kaggle. The dataset went through a couple of preprocessing stages, including data augmentation using the Image Generator class in Keras, which helps to reduce overfitting. This is a critical step since failing to do so has a negative impact on the model's performance. As stated earlier, the dataset split ratio was modified with a ratio of 80:10:10, respectively, for the three subsets: training, validation, and testing, The testing dataset was kept unseen by the model until it was utilized for model evaluation after the training and validation processes were done. The images in the dataset were all scaled to 331x331x3 and 224x224x3 pixels, respectively, to align with the transferred learning model's recommended input sizes.

To compute the findings using TL, three distinct deep learning models of different trainable parameters count, as shown earlier in figure 4.6 section 4.3.1, were downloaded from the keras application library and their top layers were removed by setting toplayer = false, and customized top layers were added. Then the three models were frozen by setting pre_train_model.trainable = False, which prevented the downloaded models from being trained and maintained their weight from ImageNet. The output weights from the frozen pretrained were passed into the added customized top layers, called classification layers. Figure 4.19 illustrates an example of freezing layers and the training process.

An optimizer and appropriate fit methods were utilized to train and validate our models, with each model running about 100 epochs with 178 iterations per epoch and a batch size of 32. An iteration is defined as steps per epoch, i.e., the total number of samples/batche_size. To

properly train deep learning models, we needed to increase the number of samples and epochs. As a result, we utilized 100 epochs and a learning rate (LR) of 0.0001.

Learning Rate (LR) is a training parameter that influences how quickly the model weights are computed. A high LR might cause the model to converge too soon, whereas a low LR can result in more accurate weights (up to convergence) but requires more computing time.

The number of epochs is the number of times a dataset is sent through the NN forward and backward. Applying the performance metric formulae to the outputs of the validation data led to the conclusions, and the registered results show the highest validation values attained. We optimized our proposed models using the Adam optimizer (Adamax).

The Adam optimizer includes integrating the benefits of two other stochastic gradient descent extensions. Adam, in particular, recognizes the advantages of AdaGrad and RMSProp. AdaGrad is an adaptive gradient algorithm that keeps a per-parameter learning rate constant to improve performance when there are sparse gradients (e.g., computer vision problems and natural language). Root Mean Square Propagation (RMSProp), which additionally maintains per-parameter learning rates based on the average of recent magnitudes of gradients for the weights (e.g., how quickly they are changing). This implies that the method performs well on both online and non-stationary issues (e.g., Noisy) [51]. All models' learning rate (LR) values, optimizer and hyperparameters are listed in table 4.3 below. Furthermore, while conducting this study, two strategies, which will be explained and evaluated in the next section, were deployed on all the utilized deep learning models, NasNetLarge, EfficientNetB0, and Xception, during the training process.

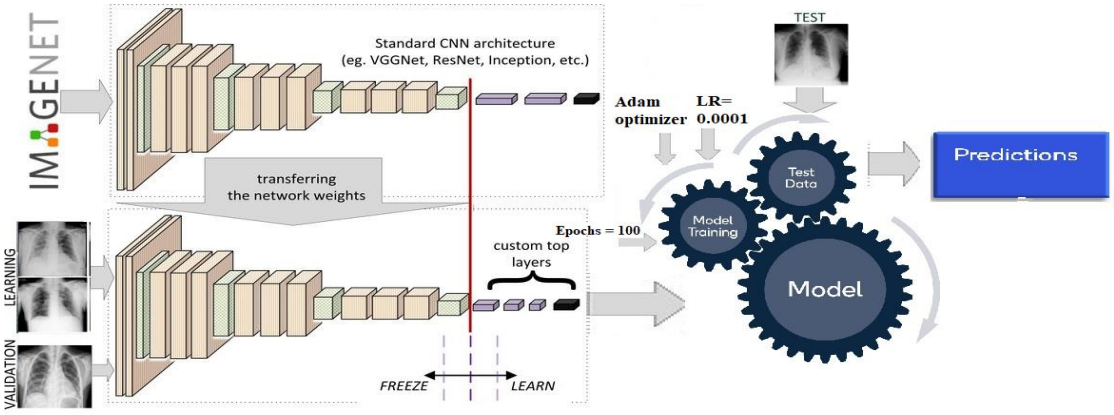


Figure 4. 19: Illustrates the training process after freezing the pretrained models.

	EfficientNet	Xception	NasNetLarge
Number of parameters	4,878,183	22,083,308	87,154,454
Trainable parameters	828,612	1,221,828	2,237,636
Non-trainable parameters	4,049,571	20,861,480	84,916,818

Table 4. 2: Shows the three model's number of parameters.

Hyper Parameters	Value
Learning Rate	0.0001, 0.00001
Epochs	100
Batch Size	32
Loss Function	Categorical cross-entropy
Optimizer	Adam optimizer
Dropout Value	0.1
Training Ratio	0.80

Table 4. 3: Hyper parameters value.

4.5.1 Training Strategies

The chest X-ray dataset with four classes was taken from Kaggle and had an imbalanced count of images in its classes due to the limitation and the lack of enough medical images for researchers to collect images for each class. Since the imbalanced dataset used in this research might promote bias against one or more classes during model training, two distinct training strategies have been adopted and evaluated and compared in this study.

4.5.1.1 Strategy I: Training Models on imbalanced dataset without balancing technique.

The models were trained on the imbalanced dataset without using any balancing techniques during training, and the results were addressed in the next chapter to evaluate the performance of the models.

4.5.1.2 Strategy II: Training Models using balancing Technique.

In the used dataset, we can find that pneumonia accounts for 63% of the total images and 20% for the normal class. The remaining 2 classes each account for 10%. So, it means this dataset is imbalanced. There is not a specific method that counts as right or wrong to address this problem. Different techniques may work better or worse with different problems. Resampling and class weight are the most common balancing techniques.

Resampling (Oversampling and Undersampling)

This method is employed to increase or decrease the sample size for the minority or majority class. When working with an unbalanced dataset, we can use replacement to oversample the minority class. Oversampling is the term used for this method. On the other hand, in undersampling, we can randomly remove rows from the majority class in order to align them with the minority class. After sampling the data, we can get a balanced set of data for both the majority class and the minority class.

Class weights

Regardless of the quantity of the samples from each class in the training set, all the classes are given an average emphasis on gradient updates. As a result, models are prevented from predicting the numerous classes more frequently just because it is more prevalent.

Class weights can be used to make the prediction error higher when an instance of the underrepresented class is erroneously classified. Class weights are used to make the model pay more attention to classes with fewer pictures.

The entire array of classes and their set of unique class labels are sent into the Class Weight function, class_weight Argument can either be None or 'balanced,', if it was none, the class weight will be uniform, if it's 'balanced', in this case output class weights will be calculated according to a mathematical formula:

$$Wj = n\_samples / (n\_classes * n\_samplesj) \quad (4.1)$$

where *Wj* is the weight for each class, *n_samples* is the total number of samples, *n_classes* is the total number of classes, and *n_samplesj* is the total sample count of each class.

A total loss is determined for each batch during model training, and the model parameters are then iteratively updated in a direction that minimizes this loss. By default, each sample contributes equally to the overall loss, hence using class weights make the model pay more attention to underrepresented classes.

When class weighting is turned on, the total is changed to a weighted sum such that each sample contributes to the loss in accordance with its class weight. Class weights are assigned by the Peltarion Platform and are inversely proportional to the class frequencies in the training data [52]. This indicates that samples from the smaller class(es) contribute more to the overall loss. As a result, when the parameters update is carried out, the learning algorithm will focus

equally on the smaller class(es) due to their higher prior probability, preventing the model from overclassifying the higher classes in the count.

Class weight was the chosen method that would fit better with our research problem as it doesn't require altering the portion of the samples in each class rather than paying more attention to the fewer count classes, which might work better due to the significant variance in each class count.

4.6 Experimental parameters

TensorFlow 2.4 and related libraries, such as Keras, are used throughout the study. Python and Keras framework is the primary programming language for the whole implementation and to build the models. These were operated on a Google Colab Pro version with a P100 GPU processor, 16 GB of RAM, and 2 TB of storage. The images of the input classes were augmented in the first usage of the dataset using an API Keras Augmenter to increase the number of images fed to the model during the training phase and achieve the statistical findings by applying augmentation methods like image rotation, skew, and shift. All images, both original and augmented, were provided to Keras. Our suggested multi-classification deep learning models were fed the original and augmented images during the training process. The next table 4.4 provides the hardware and software specifications used to meet the experiment expectations.

Item	**Specification**
Machine name	Google Colab
GPU	Tesla P100-PCIE
GPU Memory	16GB
CPU Model	Intel(R) Xeon(R) CPU @ 2.30GHz
Cache size	46080 KB
CPU MHz	2299.998
Available RAM	32 GB
Disk Space	2TB
Programming Language	Python3
Python libraries	TensorFlow, Keras, NumPy, Pandas, Matplot

Table 4. 4: The experimental hardware and software specifications

Chapter 5

5. Performance Evaluation and Results

In this chapter, the results of each model were evaluated and discussed while being trained and validated then tested using the unseen test datasets. This chapter displays and analyzes the classification findings of lung disease classes in the chest X-Ray images. Before delving into these findings, let us first establish the most important parameters employed in the current research to evaluate and analyze the model's performance while being trained using the training and validation dataset and their results on the test dataset.

The train curve was generated using the training dataset, which gives an indication of how effectively the model is learning, while a holdout validation dataset is used to produce the validation curve, which offers an indication of how well the model generalizes. Moreover, the training and validation loss is the sum of each example's errors in the validation or training sets. In contrast to accuracy, loss is not a percentage. The loss function applied for this research multiclass problem was categorical cross-entropy (CE). The categorical cross-entropy loss function can be defined as in Equation 5.1:

$$L_{CCE} = -\sum_{i=1}^{C} y_i \, log \, P_i \tag{5.1}$$

where C is the number of classes, y_i is the true probability, while P_i is the predicted probability.

Overall, the optimal model is one that generalizes well and is neither overfit nor underfit, overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. Underfitting refers to a model that can neither model the training data nor generalizes to new data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. Additionally, the confusion matrix displays in detail what happens to images following classification.

Two strategies were used in training the models that were discussed in detail in the earlier chapter, and their results were evaluated using various evaluation metrics in this chapter.

5.1 Evaluation Metrics

Metrics for measuring the performance of deep learning classification models are critical. Following the completion of the training phase, the performance of each model on the test data set was assessed and compared using the following five performance metrics: confusion matrix, accuracy (ACC), precision (PPV), sensitivity or recall, and F1-score. This chapter presents percentages for the accuracy, precision, recall, and F-measure

5.1.1 Confusion Matrices

It is the evaluation metric that is the most frequently used in predictive analysis since it is relatively simple to grasp and can be used to calculate other crucial metrics like sensitivity, accuracy, etc. An NxN matrix is used to represent the overall effectiveness of a model when applied to a certain dataset, where N is the number of classes of the classification problem's labels. A 2x2 confusion matrix for binary classification is presented as shown in Figure 5.1, the confusion matrix is made up of statistics such as False Negative (FN), False Positive (FP), True Positive (TP), and True Negative (TN) [53].

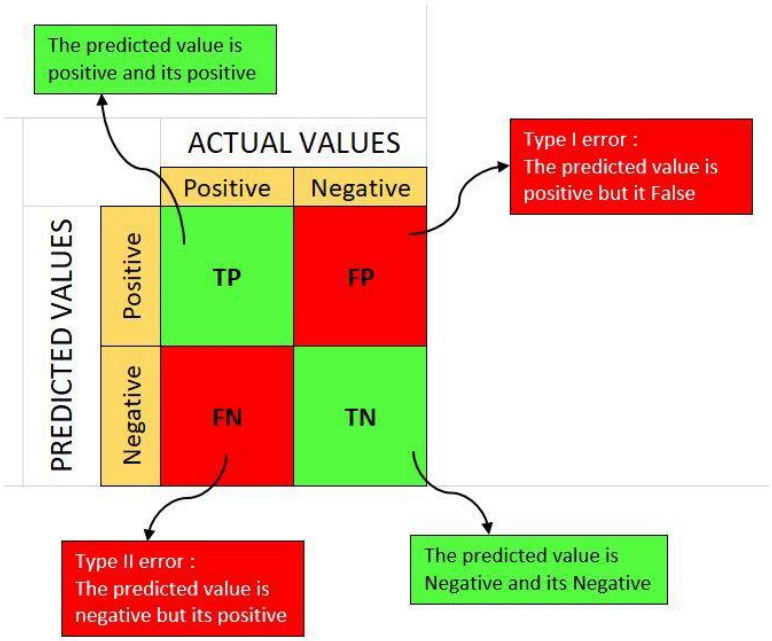


Figure 5. 1: An example of binary confusion matrix [53].

- True Positive (TP): the number of instances that the prediction model correctly classified as positive and was in fact positive.
- False Positive (FP): the number of occurrences that the prediction model classified as positive but was apparently negative.
- False Positive (FN): the number of how many instances the prediction model misclassified as Negative while, they were positive.
- True Negative (TN): the number of instances with negative labels that are genuinely negative according to the prediction model

A Multi-classification problem was addressed in this study which includes COVID-19, Pneumonia, Tuberculosis, and Normal classes; as a result, the model's performance on the test dataset was evaluated using the following 4x4 confusion matrix illustrated in Figure 5.2

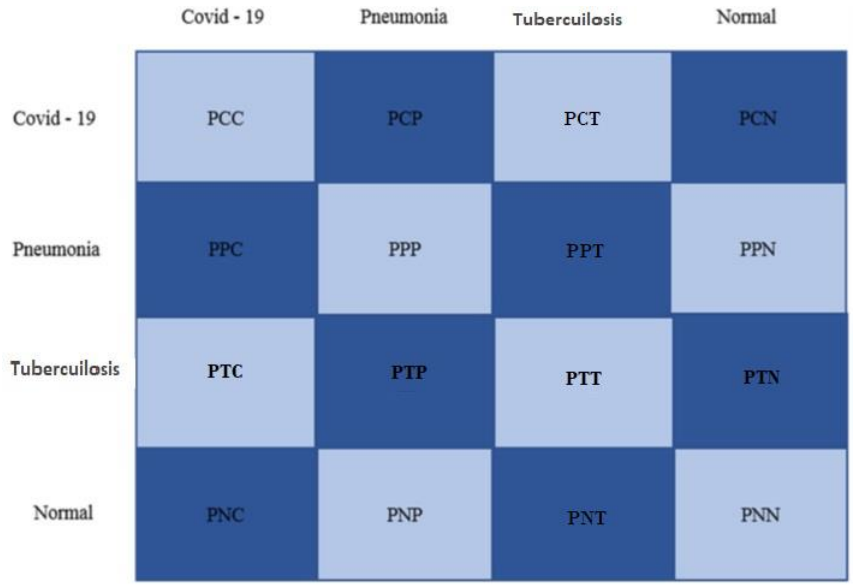


Figure 5. 2: 4X4 confusion matrix.

5.1.2 Accuracy

In order to assess the number of True positives, accuracy is employed. The truer positives a model has, the more accurate it is. When the data set is substantially unbalanced, accuracy measurements can occasionally be deceptive since the real positive will be high for the higher instance class. Therefore, classification metrics should be chosen carefully. Since our problem is multi-classification problem, the categorical accuracy metric computes the mean accuracy rate across all predictions. Generally, Using Eq. 5.2, accuracy may be estimated.

$$Accuracy = \frac{(TP + TN)}{(TP + FP + FN + TN)} \quad (5.2)$$

5.1.3 Precision

Precision is the proportion of accurately predicted positives to total positives anticipated, when classification accuracy appears to be deceiving, measures like precision, recall, and F-1 can provide an accurate assessment of the model. If we have information of TP and FP, we can compute precision. It provides a measurement of the proportion of positive predictions that come true. It shows how well a model predicts a certain class of interest. In essence, precision is the ratio of properly positively identified labeled to all positively labeled samples. Precision may be calculated mathematically as seen in Eq.5.3:

$$Precision = \frac{TP}{TP+FP} \tag{5.3}$$

5.1.4 Recall (Sensitivity)

It serves as a gauge of the proportion of instances that were actually positive and got anticipated to be positive. In fact, this suggests that a certain percentage of real positive situations are supposed to be implicitly projected as negative. The term "recall" is another name for "sensitivity" The following formula may be used to calculate sensitivity mathematically as seen in Eq.5.4:

$$Recall = \frac{TP}{TP+FN} \tag{5.4}$$

5.1.5 The F-measure

The term "F-Measure" refers to the average accuracy and recall that takes both false positives and false negatives into consideration. Accuracy is less useful than F-Measure, particularly when the data distribution is imbalanced. By examining Precision and Recall, it can be observed that the accuracy is more efficient if false positives and false negatives have the same outcome. The following formula is used to determine F-Measure as shown in Eq.5.5:

$$F{-}measure = 2* \frac{(Precision*Recall)}{(Precision+Recall)} \tag{5.5}$$

5.2 Performance of the models

The three different deep learning models after being tuned and trained on the dataset, were evaluated, and compared while using two different training strategies to find if they have good adaptation and performance with the multiclass dataset and if the adjusted training strategy has enhanced the performance.

5.2.1 Strategy I Performance: Training models without any balancing techniques.

In Strategy I, The Three Transfer Learning models have been trained on the preprocessed datasets (normalized and data augmented) and classified using the customized classification head without adding any balancing techniques during training the model with the dataset.

5.2.1.1 Performance of EfficientNetB0

After running both training and validation datasets on the EfficientNetB0 network with 100 epochs, followed by evaluating the model's accuracy by running the test dataset against it.

The following results in figs below were obtained: Notably, the computational time was around 5 hours, including a training time of 170s/per epoch and a testing time of 332 s. EficientnetB0 has 5.3 M parameters, size of 29 MB, and a depth of 132 layers as stated earlier in figure 4.6.

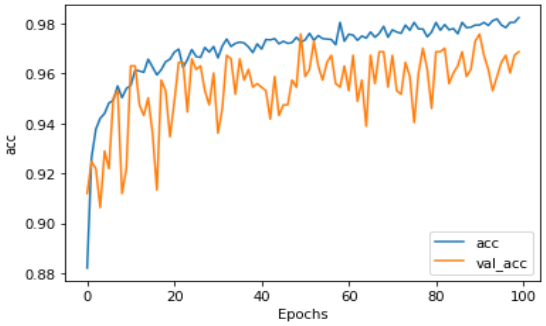


Figure 5. 3: Training & Validation Accuracy EfficientnetB0 with 100 epochs

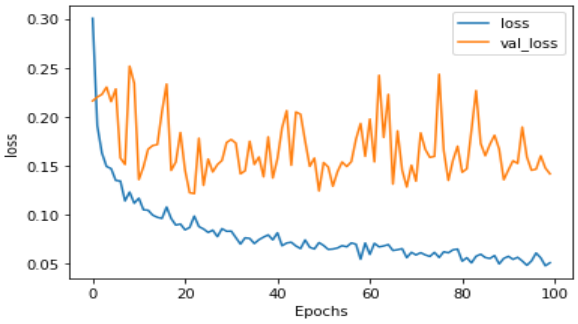


Figure 5. 4: Training & Validation Loss EfficientnetB0 with 100 epochs

.

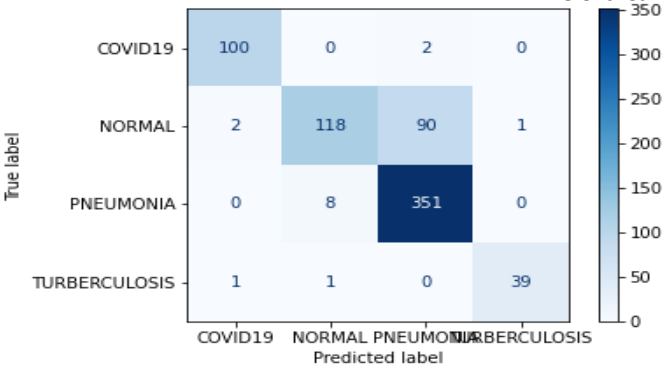


Figure 5. 5: Confusion Matrix EfficientnetB0 strategy I

Training and validation Output Results:

The findings obtained while training EfficientNetB0 are shown in the above figures. In fact, the accuracy values tremendously increased from epoch 0 to epoch 5, reaching maximum values of 95.36 percent for the train data curve and 92.23 percent for the validation data curve, respectively. Following that, the training curve smoothly fluctuates along with a gradual increase to reach the value of 98.63 percent accuracy by the end of epoch 100, but the validation curve is still very erratic around 95 percent (fig 5.3).

There was a dramatic drop in the loss curve, as seen in figure 5.4 for train data between epochs 0 and 5, from 0.30 to a loss of 0.153, and then some stability showed up at a loss of 0.05. On the other hand, the validation data curve showed a noticeable fluctuation between 0.15 and 0.25 until it reached epoch 100, when the volatility is concentrated around 0.15.

The graphs above show a sort of model overfitting due to the imbalanced dataset used during the model training process. Thus, the validation loss plot doesn't converge well at the end of epoch 100. Thus, the overfitting might have affected the test accuracy.

Testing Output Results:

Through running the test dataset on the EfficientNetB0 network, the above confusion matrix has been obtained (Fig 5.5). The labels are Covid-19, Normal, Pneumonia, and Tuberculosis, respectively. The blocks located on the diameter of the matrix show how well each class is distinguished. Pneumonia is the only class that reaches high a accuracy of determination, while some other classes reach a lower accuracy such as Normal. As it is illustrated, "pneumonia" images are significantly higher in volume compared with the other categories. This may explain the more accurate prediction of this category as shown in the confusion matrix. It shows that for the class of images (Normal), 118 images were properly identified as Normal and 90 were inaccurately classified as Pneumonia. The model achieved higher success in predicting Pneumonia and COVID than the other images for the remaining classes.

Accuracy on test data is 85.27% which is considered a good accuracy but might be improved in strategy II as indicated earlier, The results of the model's evaluation using test data are shown in figure 4.6 below, including metrics of recall, precision, F1 score, and accuracy, which were calculated using the results from the obtained convolution matrix listed earlier, with respective average values of 0.92, 0.87, 0.88 and 0.85 with support for 713 test images.

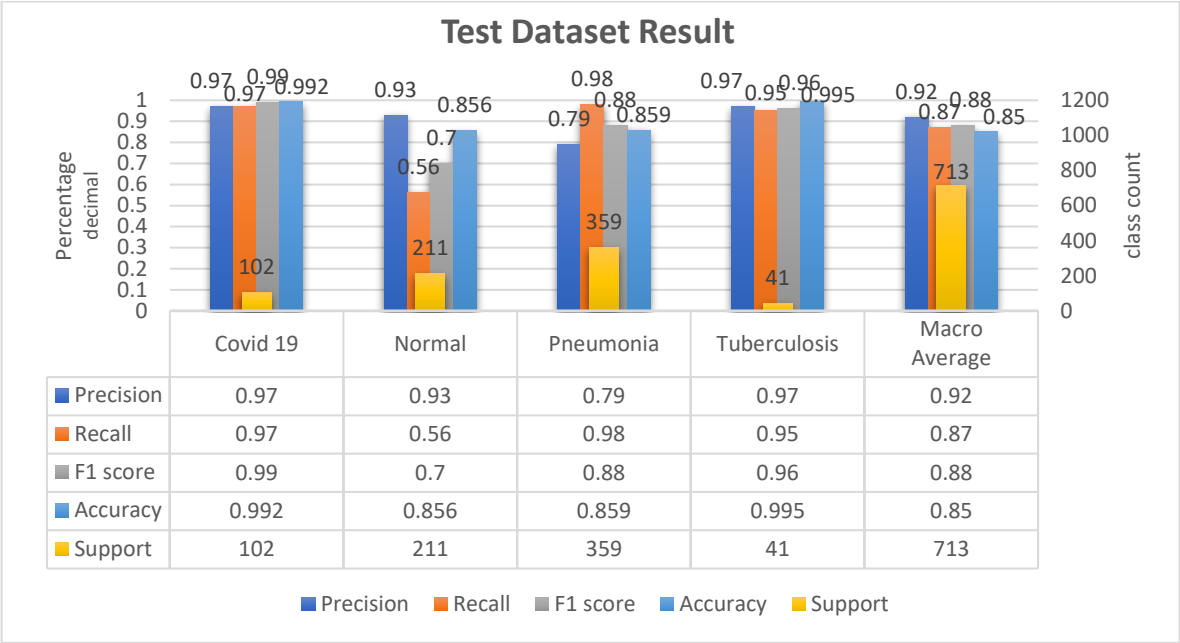


Figure 5. 6: Performance of EfficientNetB0 on the test dataset using Strategy I.

5.2.1.2 Performance of Xception

After running both training and validation datasets on the Xception network with 100 epochs, The following results were obtained by evaluating the model accuracy through running the test dataset against it. Noteworthy, the computational time was less than 5.5 hours, including training time of 173s/per epoch and testing time of 235 s. Xception had 22.9 M parameters, a size of 88 MB and a depth of 81 layers, as stated earlier in figure 4. 6, it is noted that it was quite slower than EfficientNetB0.

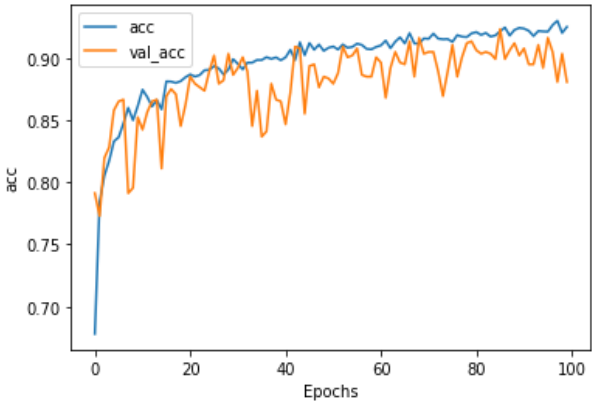


Figure 5. 7: Training & Validation Accuracy Xception with 100 epochs

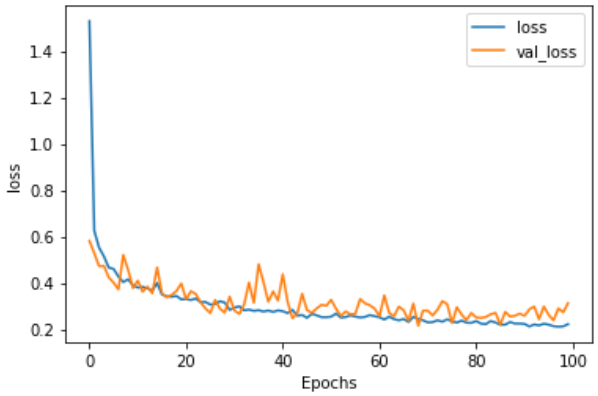


Figure 5. 8: Training & Validation loss Xception with 100 epochs

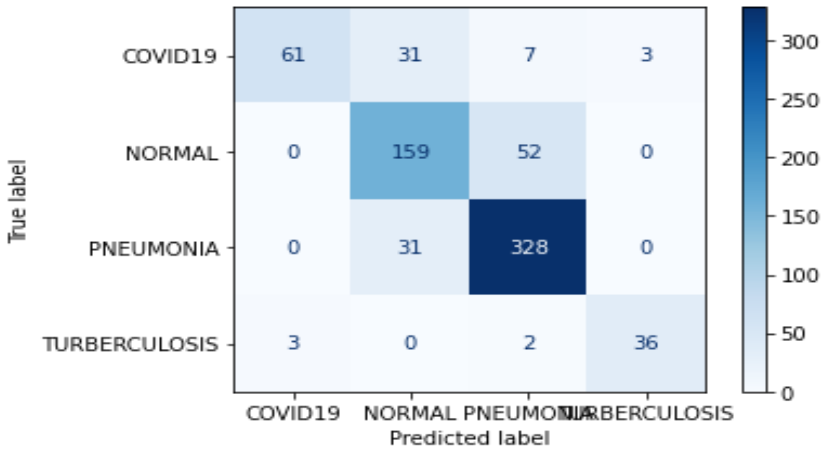


Figure 5. 9: Confusion Matrix Xception strategy I

Training and validation Output Results:

The findings obtained while training the Xception network are shown in the above figures. It is noted that the accuracy of train data is vastly increasing from epoch 0 to epoch 5, where the accuracy is equal to 85.30 percent, then it starts to be stable until epoch 100, where the accuracy is equal to 93.45 percent. For the validation data curve, a quick increase can be observed from epoch 0 to epoch 5, and it fluctuates along with a gradual increase to the end of epoch 100 with a value of 88.63 percent accuracy (fig 5.7).

For the loss curve of train data, as shown in figure 5.8, the values rapidly decrease from epoch 0 to epoch 5, where the value is 0.4. After epoch 5, the value converges to 0.22 for train data. On the other hand, the validation data curve showed a noticeable dropping fluctuating curve from 0.60 until it converges to 0.32 at epoch 100.

Testing Output Results:

Through running the same test dataset on the Xception network, the above confusion matrix has been obtained (Fig 5.9). Similarly, "pneumonia" is the only class that reached high accuracy of determination, while other classes reached lower accuracy, especially COVID-19 with only 61 images were accurately classified, while 31, 7, and 3 images were classified as normal, pneumonia, and tuberculosis respectively. The confusion matrix showed that for the class of images (Normal), 159 images were properly identified as Normal and 52 were inaccurately classified as Pneumonia. The model achieved slightly better success in predicting tuberculosis with 36 images, 3 images as covid and 2 images as pneumonia.

The average accuracy on test data is 82.1%, which is considerably less than the previous model. The model's evaluation using the test dataset is displayed in figure 5.10 below, including metrics of recall, precision, F1 score, and accuracy, with an average of 0.86, 0.79, 0.81, and 0.82 respectively, with support of 713 test images.

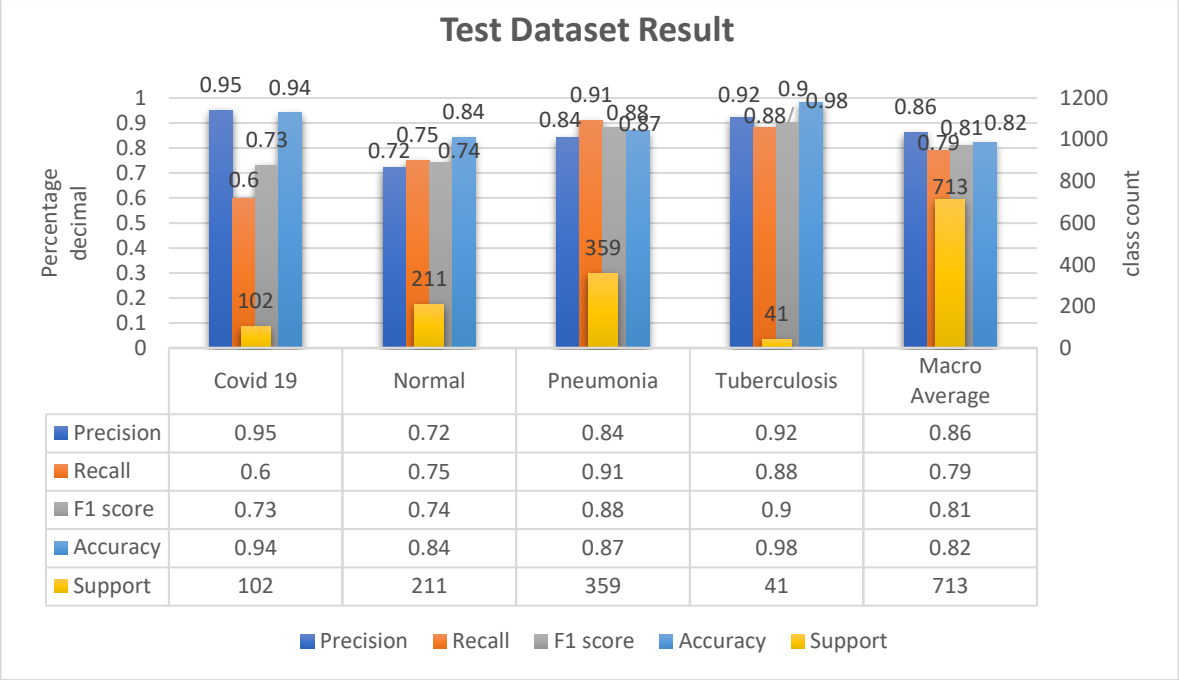


Figure 5. 10: Performance of Xception on the test dataset using Strategy I.

5.2.1.3 Performance of NasNetLarge

The following findings are achieved as shown below after running both the training and validation datasets on the NasNetLarge network with 100 epochs and then evaluating the model using the test dataset. It's noteworthy that the computation took a little over 7 hours. including training time 252s per epoch and testing time 250s, NasNetLarge has 88.9 M parameters and size of 343 MB, and a depth of 533 layers as stated earlier in figure 4.6, it is noted that it was significantly slower than EfficientNetB0 and Xception models.

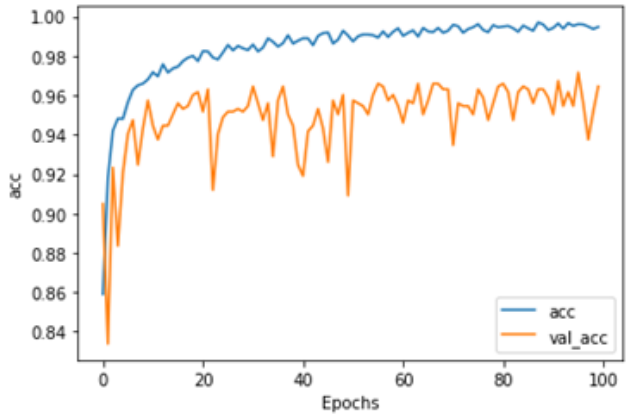


Figure 5. 11: Training & Validation Accuracy NasnetLarge with 100 epochs

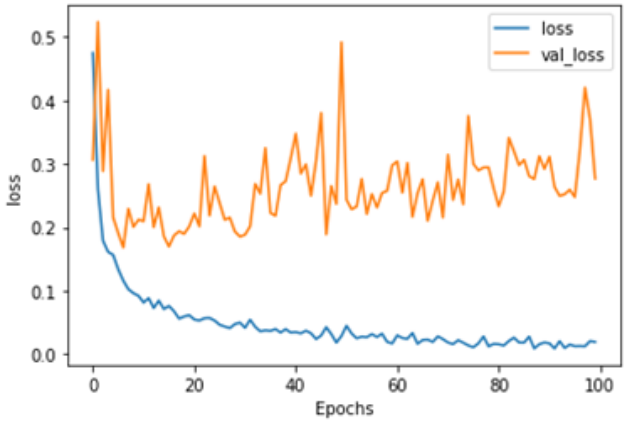


Figure 5. 12: Training & Validation Loss NasnetLarge with 100 epochs

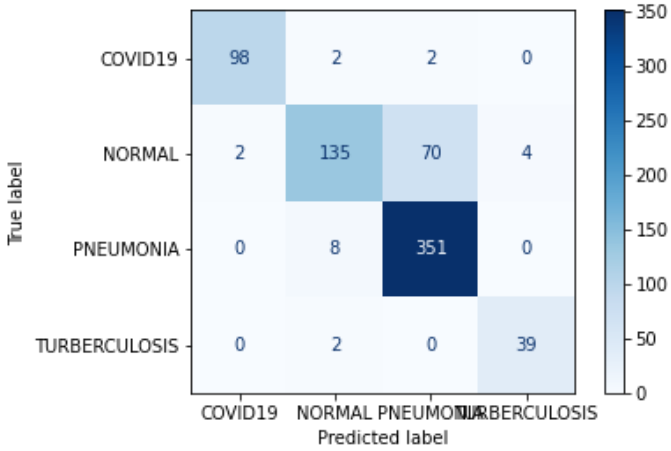


Figure 5. 13: Confusion Matrix NasnetLarge strategy I

Training and validation Output Results:

The train data accuracy curve is seen in the above figure 5.11 to be rapidly growing starting from epoch 0 till epoch 6 to reach an accuracy of 95.17 percent, then beginning to climb steadily until stabilizing at epoch 100 at an accuracy of 98.98 percent. Likewise, the accuracy curve of validation data increased, but with a strong fluctuation to reach an accuracy of 94.84 percent.

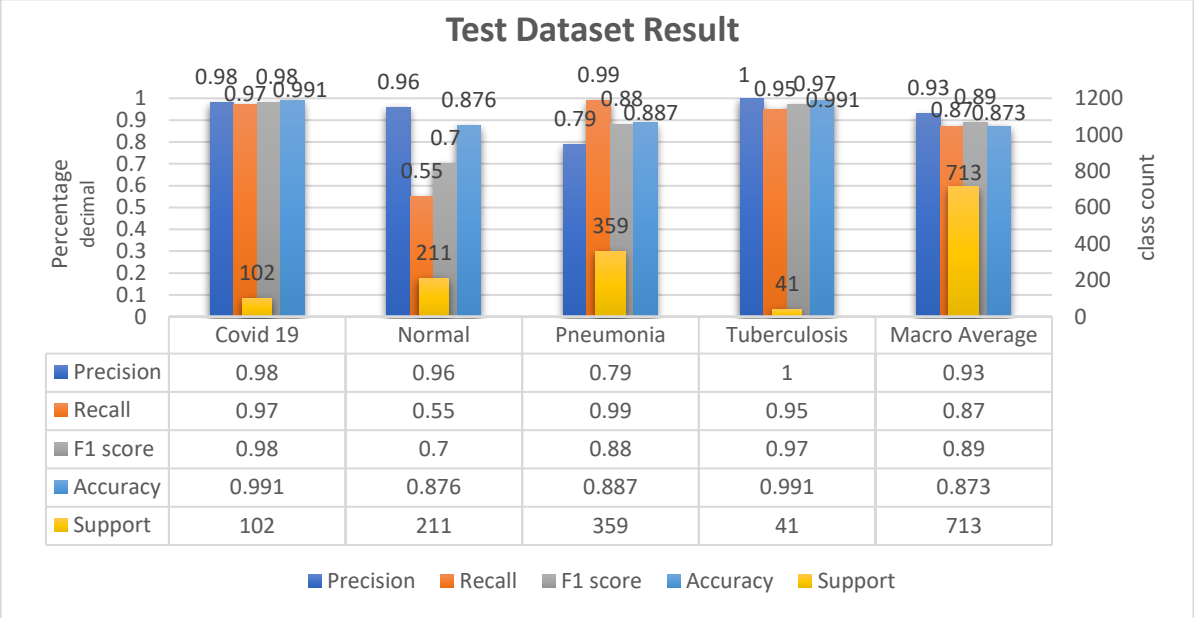
For the train data, the loss curve shows a sharp decline beginning from epoch 0 to epoch 6, where the loss is 0.18, before becoming steady around epoch 100, at which point the loss is equal to 0.012. Similarly, the loss curve of validation data dropped to 20 percent, then slightly increased while strongly fluctuating around a loss of 0.31 at epoch 100, as shown in figure 5.12.

The model plots above showed signs of model overfitting, and that has likely occurred because the utilized dataset during training and validating the model had a class imbalance. Therefore, the training curve showed a higher accuracy of around 4% than the validation accuracy. It is also noticed that the training loss curve is lower at around 5% while the validation loss is strongly fluctuating at a higher percentage of 37.85%, which is a sign of overfitting of the model and might disrupt the test accuracy. This issue might be mitigated by applying strategy II of balancing technique during training the datasets.

Testing Output Results:

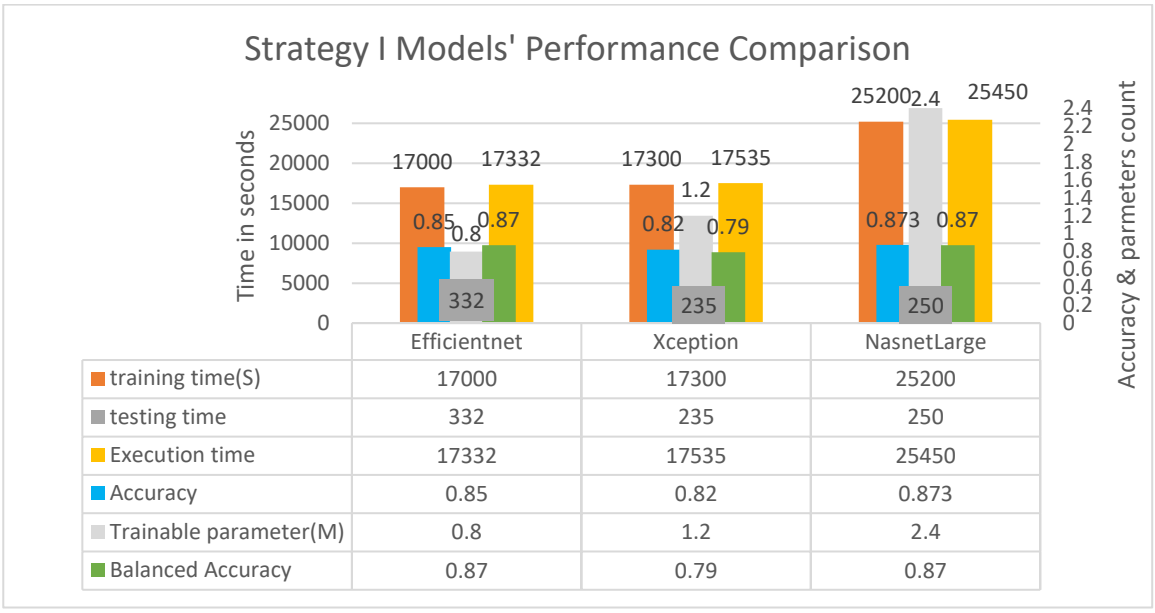
Through running our test dataset on the NasNetlarge network, which consists of 713 x-ray images imbalanced in their class count number, the previous confusion matrix (fig 5.13) has been obtained. As stated, earlier class labels are shown as COVID, Normal, Pneumonia and Tuberculosis, respectively. The blocks on the matrix's diameter display how effectively each class can be distinguished. The darker the diameter block, the more correct the prediction for a disease is. As shown in darker blue, “Pneumonia” is the class that reached a higher accuracy of determination, but it might be only a result of what was illustrated earlier that Pneumonia images are significantly higher in volume compared with the other diseases classes, while for the Normal class, the model could accurately identify 135 images correctly in the Normal class. On the other hand, 60 images were inaccurately labeled as “Pneumonia”. However, these above networks seem to be not effective enough for detecting the other 3 classes in the Multiclass Recognition with the EfficientNet, Xception and NasNetlarge networks. The above confusion matrices show high off-diagonal values, which is due to class imbalance. To put it more simply, there are more examples in the majority class than in the minority class, so the model can't learn much from the minority class, since the majority class dominates the samples.

The accuracy on the test dataset was 87.3%, which is considered a good accuracy but might be improved in strategy II. The results given in Fig 5.14 below show the model’s evaluation using the test dataset, including the metrics recall, precision, F1 score, and accuracy, which were calculated using the results from the obtained convolution matrix listed earlier.



	Covid 19	Normal	Pneumonia	Tuberculosis	Macro Average
Precision	0.98	0.96	0.79	1	0.93
Recall	0.97	0.55	0.99	0.95	0.87
F1 score	0.98	0.7	0.88	0.97	0.89
Accuracy	0.991	0.876	0.887	0.991	0.873
Support	102	211	359	41	713

Figure 5. 14: Performance of NasNetLarge on the test dataset using Strategy I



	Efficientnet	Xception	NasnetLarge
training time(S)	17000	17300	25200
testing time	332	235	250
Execution time	17332	17535	25450
Accuracy	0.85	0.82	0.873
Trainable parameter(M)	0.8	1.2	2.4
Balanced Accuracy	0.87	0.79	0.87

.

Figure 5. 15: Comparison of three model's Accuracy, Processing time and parameters count in strategy I

5.2.2 Strategy II Performance: Training Models using balancing Technique.

In Strategy II, the three Transfer Learning models were trained again, and classification was performed using the new classification head, this time a class weight balancing technique was added during the training of the models as stated in the previous chapter, to try to address the deformity in the numbers of samples of each class that might have led to results and accuracy leaning towards one class over the others.

5.2.2.1 Performance of EfficientNetB0

After running both training and validation datasets on the EfficientNetB0 network while adding the class balance technique with 100 epochs, and then testing the model using the test dataset, the following results were obtained as shown below. Noteworthy, the computational time was about 5 hours, including training time of 173s/per epoch and testing time of 340 s. It is noted that this was quite slower than it was in Strategy I due to adding the class weight technique while training the model.

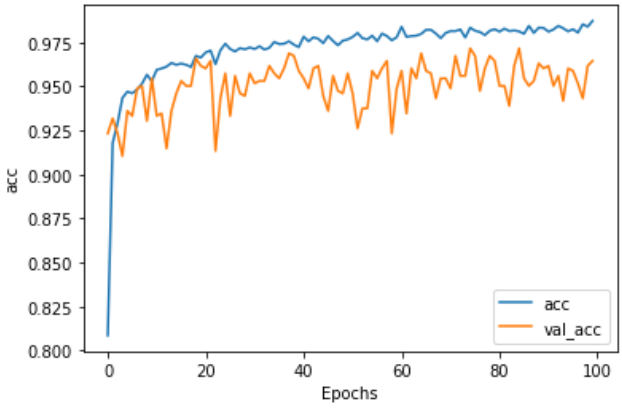


Figure 5. 16: Training & Validation Accuracy EfficientnetB0 with 100 epochs

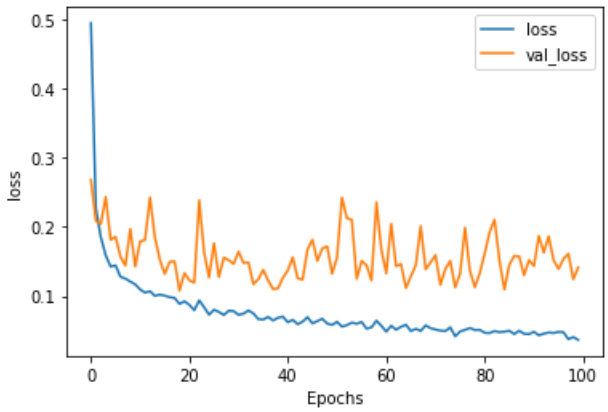


Figure 5. 17: Training & Validation Loss EfficientnetB0 with 100 epochs

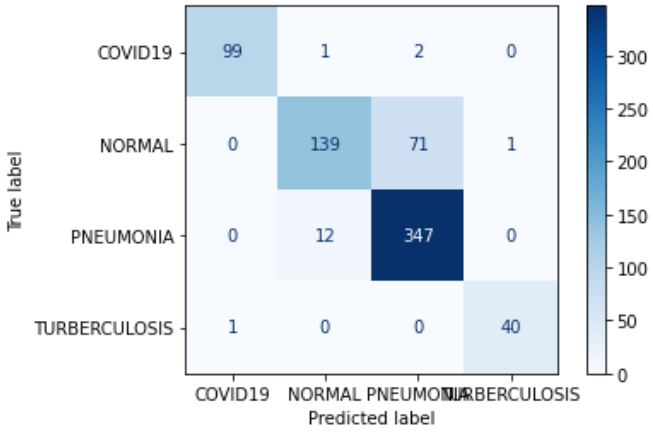


Figure 5. 18: Confusion Matrix EfficientnetB0 strategy II

Training and validation Output Results:

As noticed above in figure 5.16, a significant surge is noted for the training accuracy from epoch 0 with 81.67% accuracy to somewhere in the vicinity of epoch 4 with 94.91% accuracy, and it is followed by mild oscillations with a steady climb between epoch 4 and epoch 100,

reaching a maximum of 98.72 percent. In terms of validation accuracy, it exhibits significant volatility, rising gradually to a value of 96.45 percent at the end of epoch 100.

For the loss curve of train data, the values rapidly decreased from epoch 0 to epoch 5, where the value started at 0.5 and after epoch 5, the value converged to 0.036 for train data. On the other hand, the validation data curve showed a noticeable dropping fluctuating curve from 0.25 until it converges to 0.14 at epoch 100, as seen in the above figure 5.17.

Testing Output Results:

After running our test dataset of 713 Xray images on the EfficientNetB0 network after employing a balancing strategy in training the model, it was noted similarly on the above confusion matrix figure 5.18 that the blocks on the diameter were darker, which suggested that the classes were classified more accurately when strategy II was used, and low-count classes were paid more attention. It is worth noting that a greater proportion of "COVID" ,"Pneumonia", and “Tuberculosis” images were accurately identified.

The model's accuracy on the test data set is 88%, which is noticeably a lot better than strategy I. The model's performance was assessed using the test data set (fig 5.19), and metrics of recall, precision, and accuracy were on an average of 0.93, 0.89, 0.90, and 0.88, respectively.

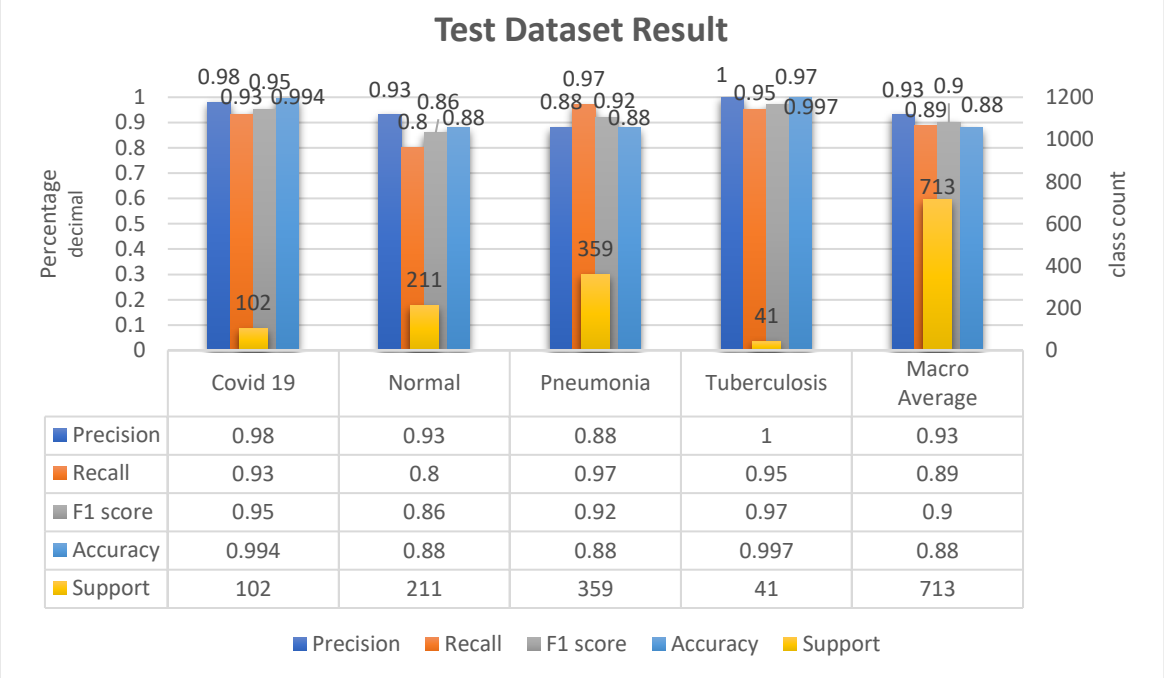


Figure 5. 19: Performance of EfficientNetB0 on the test dataset using Strategy II.

5.2.2.2 Performance of Xception

After running both training and testing datasets on the Xception network with 100 epochs, The following results were obtained by evaluating the model accuracy through running the test dataset against it. Notably, the total computational time was less than 5.5 hours, including training time of 176s/per epoch and testing time of 240 s. It is noted that it was quite slower than it was in Strategy I due to adding class weight technique while training the model.

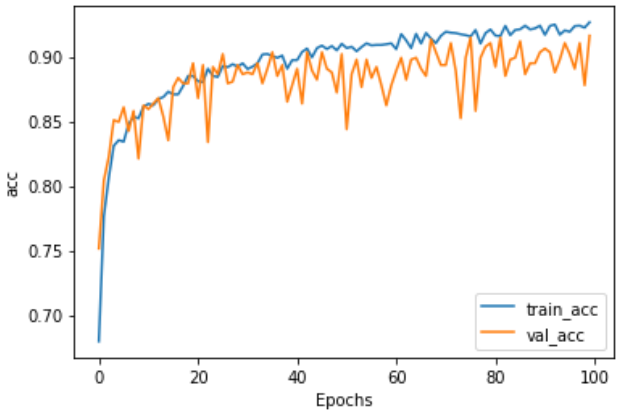


Figure 5. 20: Training & Validation Accuracy Xception with 100 epochs

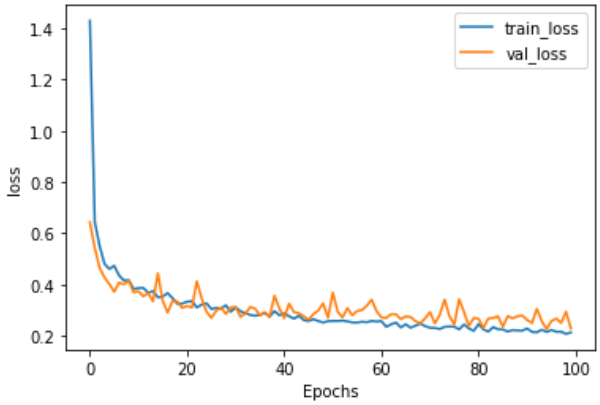


Figure 5. 21: Training & Validation Loss Xception with 100 epochs

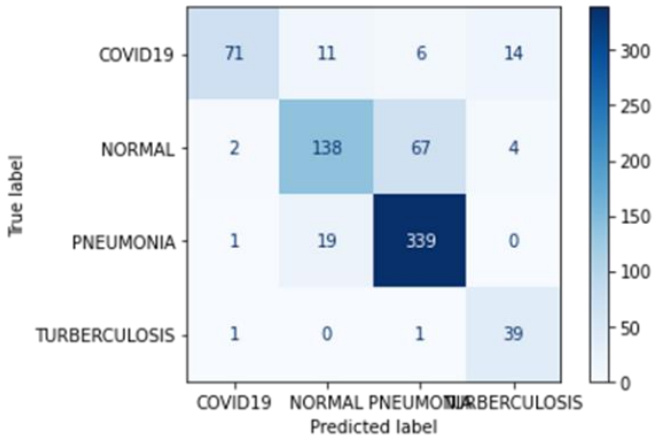


Figure 5. 22: Confusion Matrix Xception strategy II

Training and validation Output Results:

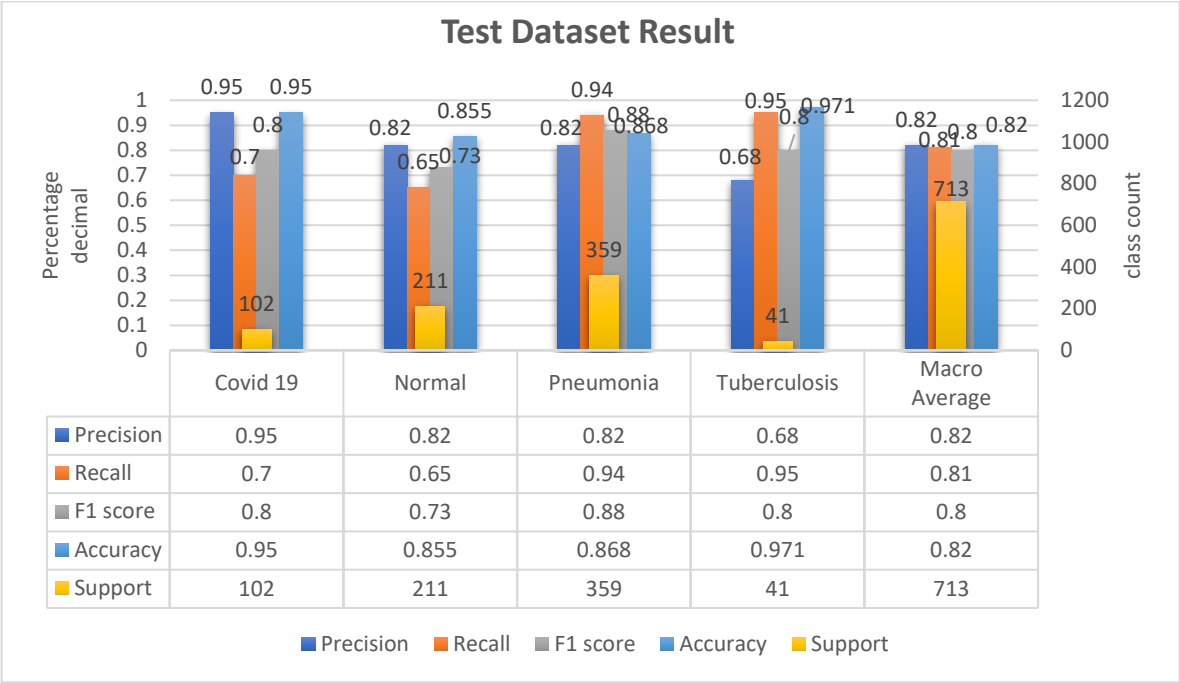
The findings obtained while training the Xception network are shown in the above figures 5.20 and 5.21. A staggering soar is witnessed from epoch 0 with 68.31% accuracy to somewhere towards epoch 5 with 84.01% accuracy, followed by mild fluctuations with a sustained ascent between epoch 5 and epoch 100 until it reaches an accuracy of 92.36%. For the validation data curve, a quick increase can be observed from epoch 0 to epoch 5, and then it strongly fluctuates along with a gradual increase to the end of epoch 100 with a value of 91.63% accuracy.

A good fit could be noticed for the loss curve of training and validation data. Apparently, from epoch 0 to epoch 4, the loss is rapidly falling, equivalent to 50.89 percent, and then begins to converge until epoch 100 at 0.21 and 0.22 for the train data and the validation data, respectively.

Testing Output Results:

Through running the same test dataset on the Xception network, the above confusion matrix has been obtained (Fig 5.22). In strategy II, it is noted that the "Pneumonia" class reached high accuracy of determination, while other classes showed improvement in their accuracy, especially COVID-19 as it paid more attention to minor classes in counts with 71 images were accurately classified, while 11, 6, and 14 images were classified as Normal, Pneumonia and Tuberculosis respectively. The confusion matrix shows that for the class of images (Normal), 138 images were properly identified as Normal and 67 were incorrectly classified as Pneumonia. The model achieved better improvement in predicting Tuberculosis than in Strategy I with 39 images, 1 image as COVID and 1 image as Pneumonia.

The average accuracy on test data did not change from strategy I and remained as low as 82.33%, which is considerably less than the other models, and remained unimproved by strategy II. The model's evaluation using the test dataset is displayed in the table below, including metrics of recall, precision, F1 score, and accuracy with an average of 0.86, 0.79, 0.81, and 0.82 respectively for all classes as represented in figure 5.23 below.



	Covid 19	Normal	Pneumonia	Tuberculosis	Macro Average
Precision	0.95	0.82	0.82	0.68	0.82
Recall	0.7	0.65	0.94	0.95	0.81
F1 score	0.8	0.73	0.88	0.8	0.8
Accuracy	0.95	0.855	0.868	0.971	0.82
Support	102	211	359	41	713

Figure 5. 23: Performance of Xception on the test dataset using Strategy II

5.2.2.3 Performance of NasNetLarge

After running both training and validation datasets on the NasNetLarge network while adding the Strategy II class balance technique with 100 epochs, and then testing the model using the test dataset, the following results were obtained as shown below. Notably, the computational time was about 7 hours. Including a training time 266s/per epoch and a testing time of 250 s. It is noted that it was quite slower than it was in Strategy I due to adding class weight technique while training the model.

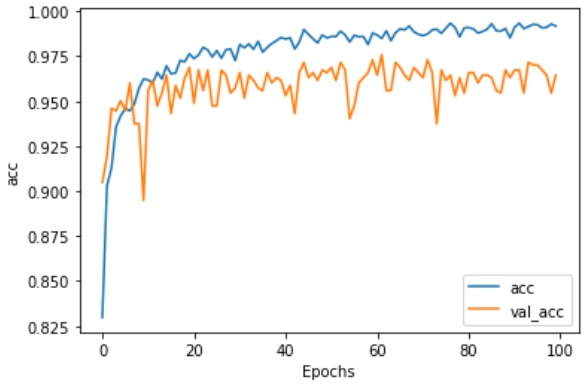


Figure 5. 24: Training & Validation Accuracy Nasnetlarge with 100 epochs

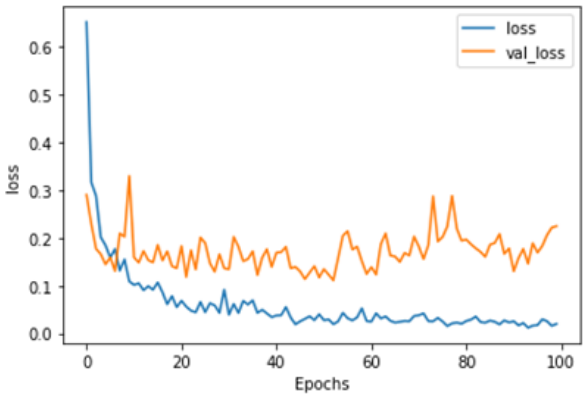


Figure 5. 25: Training & Validation Loss Nasnetlarge with 100 epochs

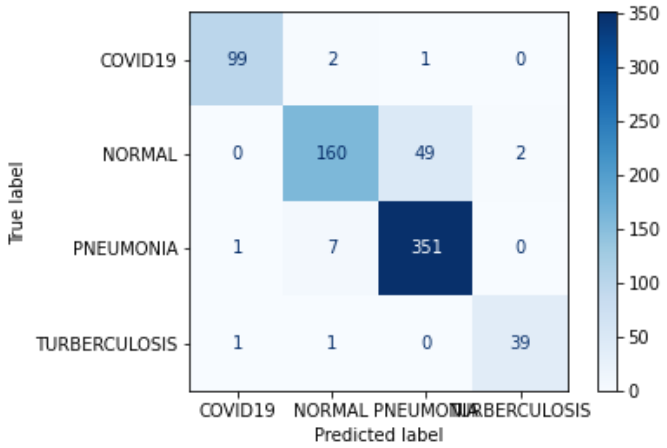


Figure 5. 26: Confusion Matrix NasnetLarge strategy II

Training and validation Output Results:

The resultant train data accuracy curve is growing exponentially, as seen in figure 5.24, until it reaches 94.89 percent. After epoch 20, the accuracy begins to stabilize gradually at 99 percent for the training data and 96.51 percent for validation data. A satisfactory match for the training and validation data curves may be assessed.

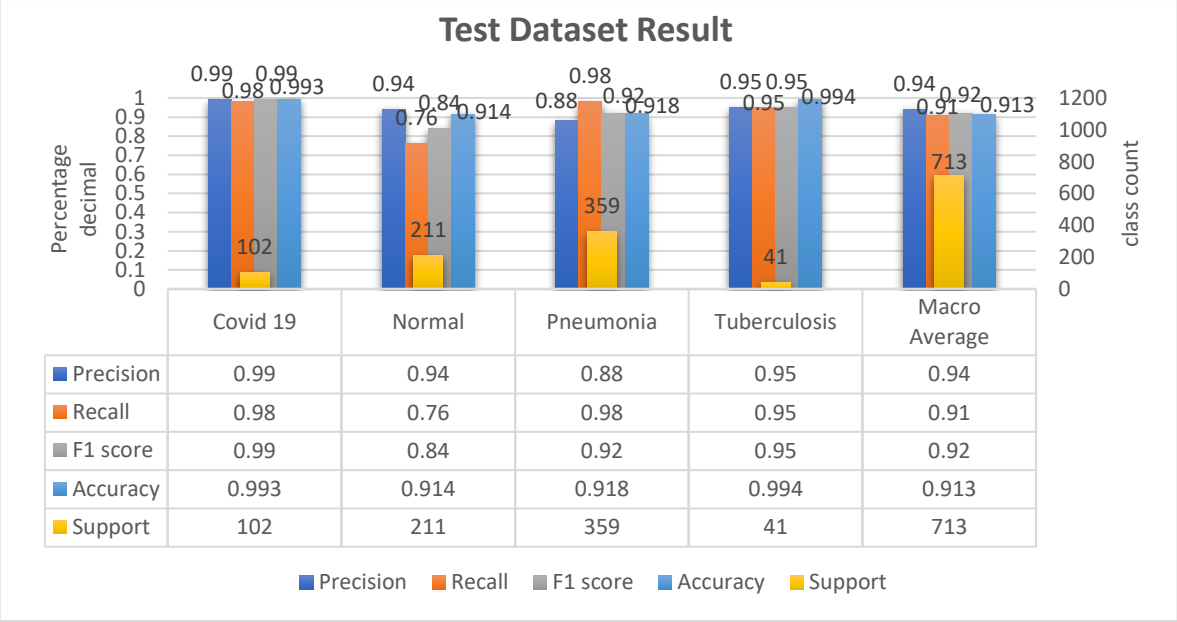
Indeed, as seen in Figure 5.25, the loss is rapidly reducing from epoch 0 to epoch 5, when it is equal to 0.2, and then begins to converge until epoch 100, where it is equal to 0.014, and for the validation loss curve, it falls to 0.2 percent and hovers around it until epoch 100.

Testing Output Results:

Through running our Test dataset of 713 on NasNetlarge network after employing balancing strategies in training the model. Similarly, to EfficientNetB0 and Xception in Strategy II, it is noted that the blocks on the diameter of the convolution matrix obtained are darker, as seen in figure 5.26, which suggests that the classes are classified more accurately when strategy II was used as it pays more attention to the low count classes. It is worth noting that a greater proportion of All classes “COVID” “Pneumonia" “Normal” and Tuberculosis photos were accurately identified. When we see this confusion matrix, we can observe that for the first image class (covid), the model was able to predict 99 images correctly in the covid class, and 2 were labeled as “Normal” and 1 as “Pneumonia”. The model was also able to identify 160 images correctly as Normal while 49 images were labeled as Pneumonia for the second image class (Normal), For the Pneumonia class, 351 were classified correctly while 7 and 1 were

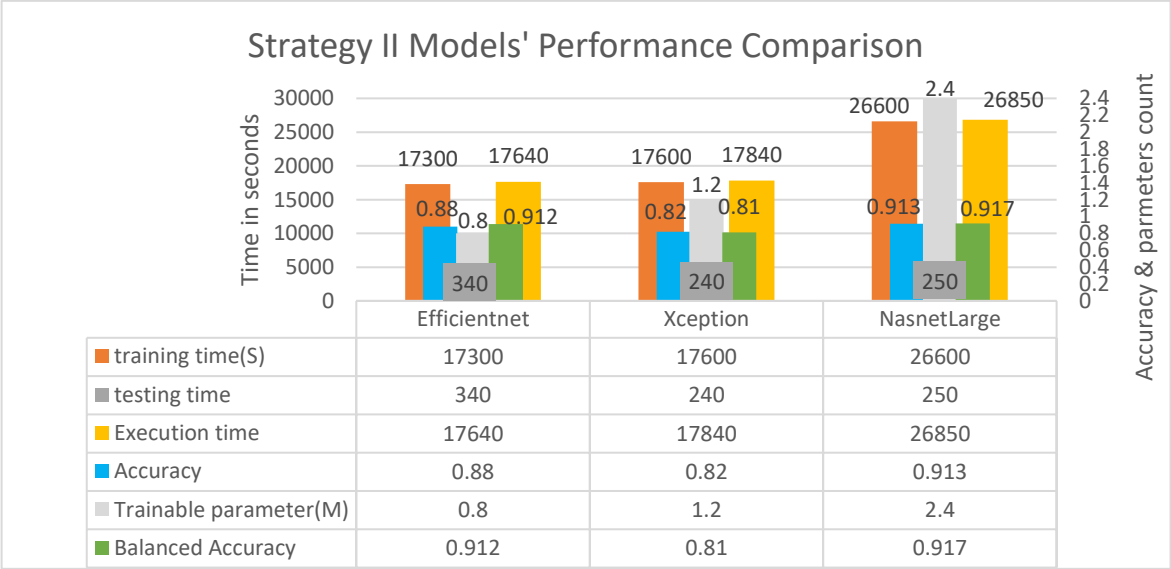
misclassified as Normal and COVID respectively. Lastly, Tuberculosis was almost perfectly classified with 39 while only 1 image was misclassified as Normal and covid as well.

The model's accuracy on test data is 91.3%, which is certainly better than strategy I. The findings shown in the table below indicate the model's evaluation metrics calculated from the resultant confusion matrix after running a test dataset on the model. The recall, precision, F1 score, and accuracy for each class were all represented accordingly in figure 5.27 below with an average of 0.94, 0.91, 0.91, and 0.91, respectively.



	Covid 19	Normal	Pneumonia	Tuberculosis	Macro Average
Precision	0.99	0.94	0.88	0.95	0.94
Recall	0.98	0.76	0.98	0.95	0.91
F1 score	0.99	0.84	0.92	0.95	0.92
Accuracy	0.993	0.914	0.918	0.994	0.913
Support	102	211	359	41	713

Figure 5. 27: Performance of NasNetLarge on the test dataset using Strategy II.



	Efficientnet	Xception	NasnetLarge
training time(S)	17300	17600	26600
testing time	340	240	250
Execution time	17640	17840	26850
Accuracy	0.88	0.82	0.913
Trainable parameter(M)	0.8	1.2	2.4
Balanced Accuracy	0.912	0.81	0.917

Figure 5. 28: Comparison of three model's Accuracy, Processing time and parameters count in strategy II.

In this step, a learning rate of 0.00001 was applied during training in Strategy II. As a result, it is obvious that there was not that much of fluctuation in the training and validation accuracy curves, and it showed a good fit for the loss curve as well for all models as shown below in figures 5.29, 5.30 and 5.31. Nevertheless, test accuracy for EfficientNetB0, Xception and NasNetLarge was 90%, 81% and 90%, respectively which didn't get any improvement except for EfficientNetB0, moreover the training and execution time for all models has increased as shown in figure 5.32.

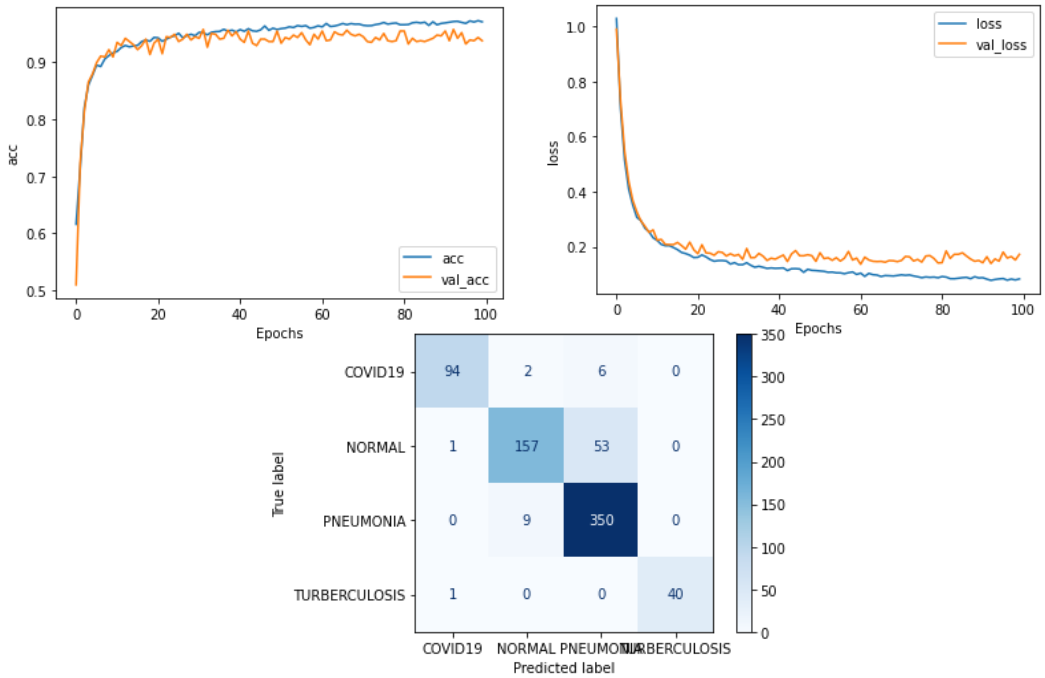


Figure 5. 29: Training & Validation Accuracy, Training &Validation loss, and Confusion Matrix for EfficientNetB0 with 100 epochs at learning rate 0.00001 in Strategy II

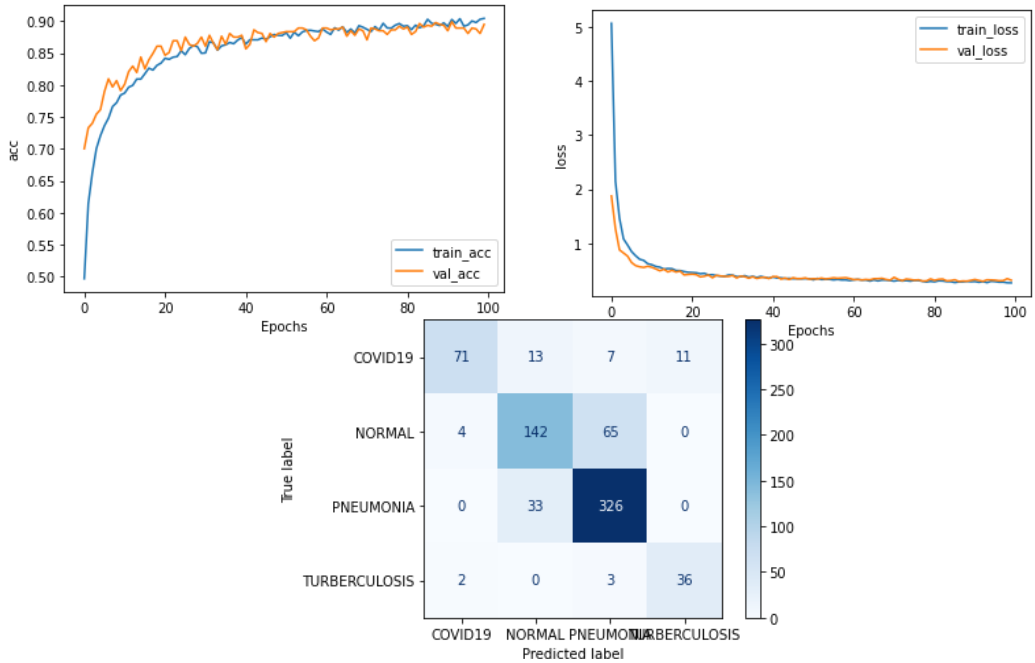


Figure 5. 30: Training & Validation Accuracy, Training &Validation loss and Confusion Matrix for Xception with 100 epochs at learning rate 0.00001 in Strategy II

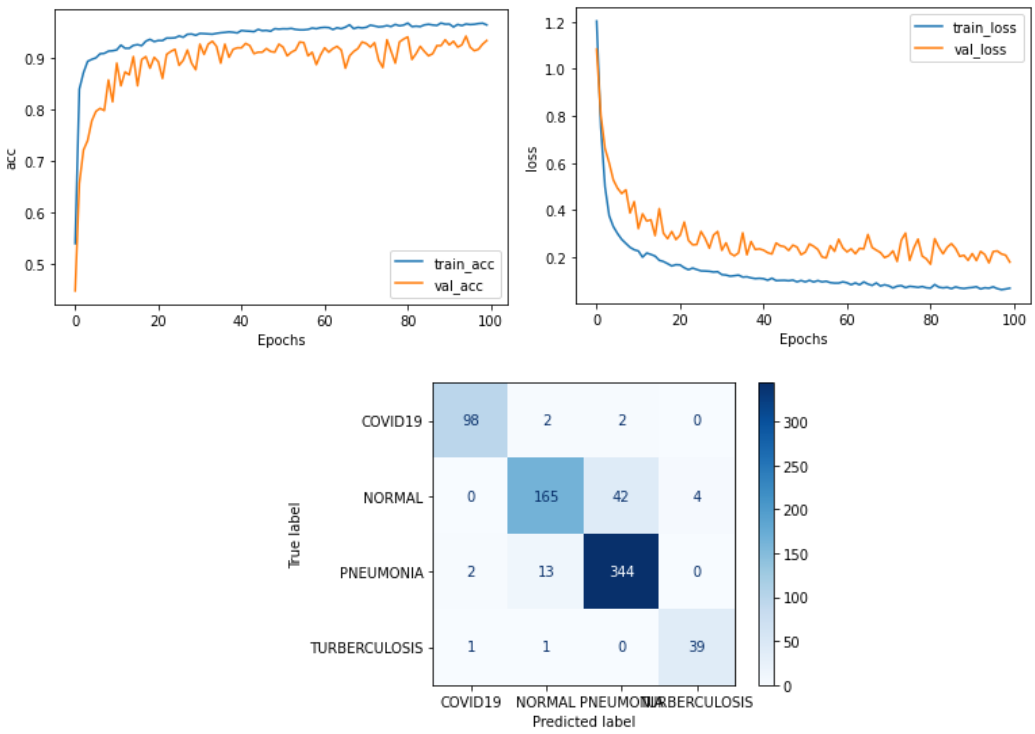
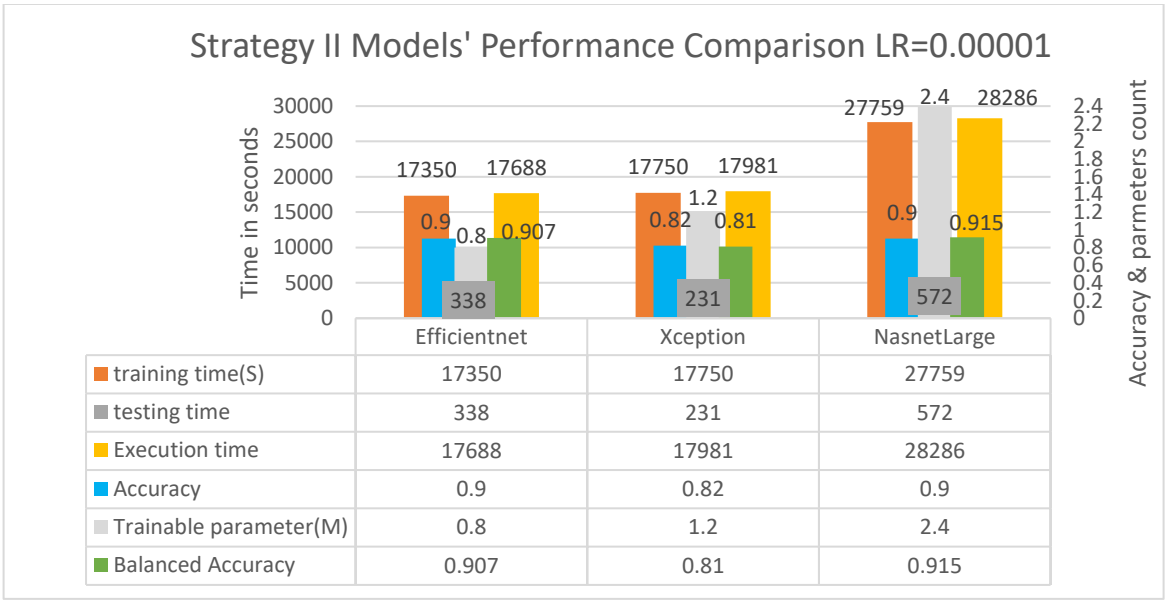


Figure 5. 31: Training & Validation Accuracy, Training &Validation loss, and Confusion Matrix for NasNetLarge with 100 epochs at learning rate 0.00001 in Strategy II



	Efficientnet	Xception	NasnetLarge
training time(S)	17350	17750	27759
testing time	338	231	572
Execution time	17688	17981	28286
Accuracy	0.9	0.82	0.9
Trainable parameter(M)	0.8	1.2	2.4
Balanced Accuracy	0.907	0.81	0.915

Figure 5. 32: Comparison of the three model's Accuracy, processing time and parameters count in strategy II with learning rate 0.00001

5.3 Discussion

Table 5.1 below shows the outcome of Strategy I, where the three models were trained without the use of any balancing techniques. On average, NasNetlarge required for training approximately 228 s/epoch, EfficientNetB0 required approximately 173 s/ epoch, while Xception needed 176 s/epoch. Table 4.2 in Chapter 4 showed that there are fewer trainable parameters in EfficientNetB0; hence, it took the least time to be trained than Nasnetlarg and Xception. Table 5.1 also demonstrates that NasNetLarge surpassed the others with the greatest test accuracy of 86 percent. It achieved an average precision of 91.75 percent, while the average sensitivity and F1 scores for all classes were 94.5 percent and 92.75 percent, respectively. On the other hand, EfficientNetB0 achieved an accuracy of 85.30 percent, a precision of 89.25 percent, and the sensitivity was measured as 91.75 percent, while Xception provided an F1 score of 81 percent, an average precision of 86.75 percent, and the average sensitivity was 79 percent. Table 5.1 clearly illustrates that NasNetLarge outperforms all other models in terms of average accuracy and metrics.

A sensitivity of 98% and 97% for the covid class can be observed for EfficientNetB0 and NasNetlarge models, respectively. This is critical as the model should be able to detect all positive COVID-19 cases to reduce the virus's spread to the community.

In other words, confirmed positive COVID-19 patients would be accurately identified as "COVID-19 positive" 98% and 97% of the time by employing the EfficientNetB0 and NasNetlarge models, respectively. Furthermore, the aforementioned models show a high precision value of 97% and 99% for the COVID class, respectively. This implies that only two normal classes were incorrectly classified as COVID in NasNetlarge, while only two COVID cases were incorrectly classified as pneumonia in EfficientNetB0, as shown in Figures 5.13 and 5.5, respectively.

A similar trend can be depicted in terms of F1-score. Also, one of the very encouraging results is the ability of these models to achieve high sensitivity and precision in the normal class. This ensures that the FPs are minimized not only for the COVID but also for the Pneumonia and Tuberculosis classes and can potentially help alleviate the burden on the healthcare system.

Table 5.2 below displays the outcomes of the suggested approach utilizing Strategy II. As described in Section 5.2.2, the class weight balancing strategy was utilized during model training to improve the performance of the models used in Strategy I. This method noticeably

enhanced the outcomes of these three models. According to Table 5.2, EfficientNetB0 had a maximum test accuracy of 88.13 percent, with average precision, sensitivity, and F1 score of 93 percent, 89 percent, and 90 percent, respectively.

NasNetLarge outperformed EfficientNetB0 with 91 percent accuracy, 94 percent precision, 91 percent sensitivity, and 92 percent F1 score. The Xception, on the other hand, only achieved 82% accuracy and precision, while sensitivity and F1-score were 81% and 80%, respectively. The results in Table 5.2 could also be validated through the confusion matrix obtained earlier.

Moreover, NasNetLarge in Strategy II achieved a precision of 99% for the covid class, as shown earlier in figure 4.27, which means that there were almost no normal or pneumonia classes falsely misclassified as COVID. Furthermore, the model would accurately identify "COVID-19 negative" cases 99% of the time and present a high sensitivity value Thus, confirmed COVID-19 cases would be able to be identified most of the time. Additionally, the model presents a high value of 99% for the F1-score in the COVID class. Indeed, this high value is justified by a low number of FNs. Figure 5.26 depicts that the COVID class has 3 misclassified cases in total as normal or pneumonia. As expected, this is desirable when dealing with such a contagious virus. Furthermore, one important observation is that sensitivity presents high values for pneumonia and tuberculosis classes in all models except in the Xception model, which is slightly less than the other two. This ensures that patients with common bacterial and viral pneumonia will not be misclassified as COVID.

Furthermore, NasNetLarge in Strategy II had achieved an accuracy value of 99.3%, 91.5%, and 99.4% for the COVID-19, Pneumonia, and Tuberculosis classes respectively. The Accuracy values are significantly high for the three diseases, indicating that the model performs very efficiently in detecting the COVID-19 among other lung diseases.

Overall, three separate models were trained in this study using two different strategies. It has been discovered that using the balancing strategy considerably increases the performance of all three models. Though NasNetlarge performs well in both circumstances but with higher training time than the other models due to its higher trainable parameters and depth, the balancing strategy considerably improves EfficientNetB0's and NasNetlarge's performance but did not improve Xception's performance much. Therefore, NasNetlarge is considered the best solution for our problem on the cost of time with an average accuracy of 91.3%. On the other hand, EfficientNetB0 had also reached a good competitive accuracy of 88% with a

lesser amount of time in regard to its lightweight in size and parameters. Thus, it's worth noting that the performance of the models was not totally affected with the higher number of parameters. As noted in Strategy I and Strategy II results, EfficientNetB0 had fewer parameters than Xception and attained better accuracy than it, but Nastnetlarge outperformed both models with the aid of its higher parameter count at the cost of a long training time. A comparison of the accuracy performance of the models is shown below in figure 5.33. It is clearly seen that NasNetLarge outperforms in both strategies. In addition, a comparison of accuracy performance with the recent work that is presented in the literature is shown in below table 5.3.

Strategy I	**EfficientNetB0**			**Xception**			**NasNetLarge**		
Training time	**17000s**			**17300s**			**25800s**		
Metrics	precision	recall	F1-score	Precision	recall	F1-score	Precision	recall	F1-score
COVID19	0.97	0.98	0.98	0.95	0.60	0.73	0.98	0.97	0.99
NORMAL	0.93	0.56	0.70	0.72	0.75	0.74	0.96	0.55	0.70
PNEUMONIA	0.79	0.98	0.88	0.84	0.91	0.88	0.78	0.99	0.88
TURBERCULOSIS	0.97	0.95	0.96	0.92	0.88	0.90	1.0	0.95	0.97
Macro Average	**0.92**	**0.87**	**0.88**	**0.86**	**0.79**	**0.81**	**0.93**	**0.87**	**0.89**
Av. Accuracy	**0.85**			**0.82**			**0.87**		

Table 5. 1: Performance comparison of the three models on the test dataset using Strategy I.

Strategy II	**EfficientNetB0**			**Xception**			**NasNetLarge**		
Training time	**17300s**			**17600s**			**26600s**		
Metrics	precision	recall	F1-score	precision	recall	F1-score	precision	recall	F1-score
COVID19	0.99	0.97	0.98	0.95	0.70	0.80	0.99	0.98	0.99
NORMAL	0.91	0.66	0.77	0.82	0.65	0.73	0.94	0.76	0.84
PNEUMONIA	0.83	0.97	0.89	0.82	0.94	0.88	0.88	0.98	0.92
TURBERCULOSIS	0.98	0.98	0.98	0.68	0.95	0.80	0.95	0.96	0.95
Macro Average	**0.93**	**0.89**	**0.90**	**0.82**	**0.81**	**0.80**	**0.94**	**0.91**	**0.92**
Av. Accuracy	**0.88**			**0.82**			**0.91**		

Table 5. 2: Comparison of the Performance of the three models on test dataset using Strategy II.

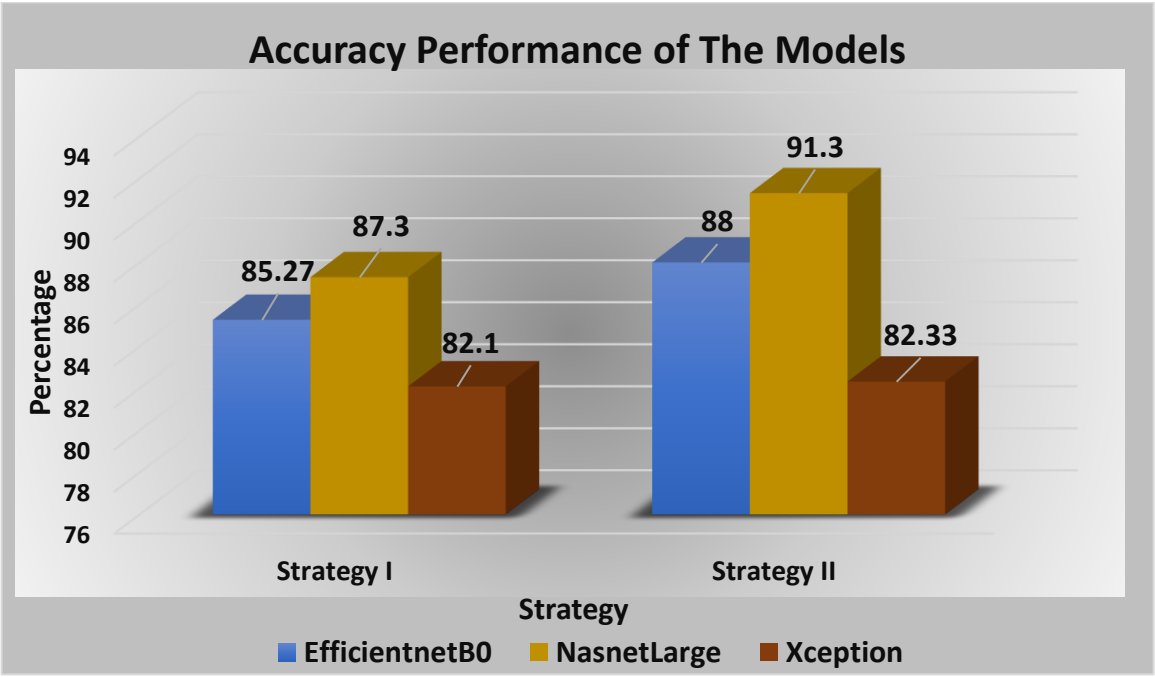


Figure 5. 33: Accuracy performance comparison between Strategy I and Strategy II.

Reference	Year	No. of Classes	Dataset Size	Accuracy
Khan et al. [26]	2020	3	1300	**89.6%**
Ozturk et al. [28]	2020	3	1125	**87.02%**
Hussain et al. [25]	2021	4	7390	**91.2%**
Tiwari et al. [54]	2022	4	9208	**87.32%**
Proposed	**2022**	**4**	**7135**	**91.3%**

Table 5. 3: Comparison with other literatures.

Chapter 6

6. Conclusion and Future Work

6.1 Conclusion

A deep learning-based approach is presented for the classification of COVID-19 and various chest illnesses. The suggested automated method can identify chest infections following the examination of chest X-ray pictures. Pixel normalization is a preprocessing tool that is used to normalize the pixel intensity of images when data is collected from various sources. Furthermore, picture augmentation is used to address the issue of the limitation of available data. Three pre-trained deep learning models, EfficeintnetB0, Xception, and NasnetLarge were selected based on their range of accuracy and size and were tuned and afterward retrained to classify four different chest X-ray classes. This study demonstrates that the class weight balancing strategy during training the models in strategy II has increased the performance of all three models. The NasnetLarge model with strategy II exceeds the other models with a classification accuracy of 91%, and when compared with other research, the proposed technique showed its superiority in performance with 4 classes, since increasing the number of classes increases the complexity of the problem and might affect the performance. EfficientnetB0 with strategy II showed a competitive result with an accuracy of 88% with a lot better execution time dueto its lightweight. that's why EfficientnetB0 might also be a good solution to address the automated detection of COVID-19 in clinics and hospitals that have limited computational power or in rural areas, while Nasnetlarge can be generally utilized.

6.2 Future Work

This research has the potential for future expansion to accommodate databases with more than four classes, including further lung diseases. Alternative lightweight deep learning models can be utilized to reduce the necessary processing time. EfficientNet versions ranging from B1 to B7 may be tried out to solve the research problem, since using the basic version EfficientNetB0 produced a competitive result despite its small size and parameters. Furthermore, for future work, various hyperparameters such as learning rate, dropout rate, batch size, activation functions and optimization approaches, particularly metaheuristic techniques can be tested to pick the best features for classification. K fold cross validation and Finetuning approaches can be experimented as well to enhance the performance of the models.

Bibliography

[1] Hui, D. S., Azhar, E. I., Madani, T. A., Ntoumi, F., Kock, R., Dar, O., ... & Petersen, E. (2020). The continuing 2019-nCoV epidemic threat of novel coronaviruses to global health—The latest 2019 novel coronavirus outbreak in Wuhan, China. International journal of infectious diseases, 91, 264-266.

[2] Muhammad, Y., Alshehri, M. D., Alenazy, W. M., Vinh Hoang, T., & Alturki, R. (2021). Identification of pneumonia disease applying an intelligent computational framework based on deep learning and machine learning techniques. Mobile Information Systems, 2021.

[3] Wei, M., Zhao, Y., Qian, Z., Yang, B., Xi, J., Wei, J., & Tang, B. (2020). Pneumonia caused by Mycobacterium tuberculosis. Microbes and infection, 22(6-7), 278-284.

[4] Yang, X., Yu, Y., Xu, J., Shu, H., Liu, H., Wu, Y., ... & Shang, Y. (2020). Clinical course and outcomes of critically ill patients with SARS-CoV-2 pneumonia in Wuhan, China: a single-centered, retrospective, observational study. *The Lancet Respiratory Medicine*, *8*(5), 475-481.

[5] Cucinotta, D., & Vanelli, M. (2020). WHO declares COVID-19 a pandemic. *Acta Bio Medica: Atenei Parmensis*, *91*(1), 157.

[6] V'Kovski, P.; Kratzel, A.; Steiner, S.; Stalder, H.; Thiel, V. Coronavirus biology and replication: Implications for SARS-CoV-2. Nat. Rev. Microbiol. 2021, 19, 155–170. [CrossRef] [PubMed]

[7] Cau, R., Pacielli, A., Fatemeh, H., Vaudano, P., Arru, C., Crivelli, P., ... & Saba, L. (2021). Complications in COVID-19 patients: Characteristics of pulmonary embolism. *Clinical Imaging*, *77*, 244-249.

[8] Singanayagam, A., Patel, M., Charlett, A., Bernal, J. L., Saliba, V., Ellis, J., ... & Gopal, R. (2020). Duration of infectiousness and correlation with RT-PCR cycle threshold values in cases of COVID-19, England, January to May 2020. *Eurosurveillance*, *25*(32), 2001483.

[9] Kooraki, S., Hosseiny, M., Myers, L., & Gholamrezanezhad, A. (2020). Coronavirus (COVID-19) outbreak: what the department of radiology should know. *Journal of the American college of radiology*, *17*(4), 447-451.

[10] Simpson, S., Kay, F. U., Abbara, S., Bhalla, S., Chung, J. H., Chung, M., ... & Litt, H. (2020). Radiological Society of North America expert consensus statement on reporting chest CT findings related to COVID-19. Endorsed by the Society of Thoracic Radiology, the American College of Radiology, and RSNA. *Journal of thoracic imaging*.

[11] Luján-García, J. E., Moreno-Ibarra, M. A., Villuendas-Rey, Y., & Yáñez-Márquez, C. (2020). Fast COVID-19 and Pneumonia Classification Using Chest X-ray Images. *Mathematics*, *8*(9), 1423. https://doi.org/10.3390/math8091423

[12] Oh, Y., Park, S., & Ye, J. C. (2020). Deep learning COVID-19 features on CXR using limited training data sets. *IEEE transactions on medical imaging*, *39*(8), 2688-2700.

[13] WHO Coronavirus (COVID-19) Dashboard. (2022). Retrieved from https://covid19.who.int/

[14] Chowdhury, M. E., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M. A., Mahbub, Z. B., ... & Islam, M. T. (2020). Can AI help in screening viral and COVID-19 pneumonia?. *IEEE Access*, *8*, 132665-132676.

[15] Narin, A., Kaya, C., & Pamuk, Z. (2021). Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks. *Pattern Analysis and Applications*, *24*(3), 1207-1220.

[16] Apostolopoulos, I. D., & Mpesiana, T. A. (2020). Covid-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks. *Physical and engineering sciences in medicine*, *43*(2), 635-640.

[17] Zhang, J., Xie, Y., Li, Y., Shen, C., & Xia, Y. (2020). Covid-19 screening on chest x-ray images using deep learning-based anomaly detection. *arXiv preprint arXiv:2003.12338*, *27*.

[18] Apostolopoulos, I. D., & Mpesiana, T. A. (2020). Covid-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks. *Physical and engineering sciences in medicine*, *43*(2), 635-640.

[19] Tsiknakis, N., Trivizakis, E., Vassalou, E. E., Papadakis, G. Z., Spandidos, D. A., Tsatsakis, A., ... & Marias, K. (2020). Interpretable artificial intelligence framework for COVID-19 screening on chest X-rays. *Experimental and Therapeutic Medicine*, *20*(2), 727-735.

[20] Sethy, P. K., & Behera, S. K. (2020). Detection of coronavirus disease (covid-19) based on deep features.

[21] Saha, P., Sadi, M. S., & Islam, M. M. (2021). EMCNet: Automated COVID-19 diagnosis from X-ray images using convolutional neural network and ensemble of machine learning classifiers. *Informatics in medicine unlocked*, *22*, 100505.

[22] Mahmud, T., Rahman, M. A., & Fattah, S. A. (2020). CovXNet: A multi-dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization. *Computers in biology and medicine*, *122*, 103869.

[23] Oh, Y., Park, S., & Ye, J. C. (2020). Deep learning COVID-19 features on CXR using limited training data sets. *IEEE transactions on medical imaging*, *39*(8), 2688-2700.

[24] Xu, X., Jiang, X., Ma, C., Du, P., Li, X., Lv, S., ... & Li, L. (2020). A deep learning system to screen novel coronavirus disease 2019 pneumonia. *Engineering*, *6*(10), 1122-1129.

[25] Hussain, E., Hasan, M., Rahman, M. A., Lee, I., Tamanna, T., & Parvez, M. Z. (2021). CoroDet: A deep learning based classification for COVID-19 detection using chest X-ray images. *Chaos, Solitons & Fractals*, *142*, 110495.

[26] Khan, A. I., Shah, J. L., & Bhat, M. M. (2020). CoroNet: A deep neural network for detection and diagnosis of COVID-19 from chest x-ray images. *Computer methods and programs in biomedicine*, *196*, 105581.

[27] Chowdhury, M. E., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M. A., Mahbub, Z. B., ... & Islam, M. T. (2020). Can AI help in screening viral and COVID-19 pneumonia?. *IEEE Access*, *8*, 132665-132676.

[28] Ozturk, T., Talo, M., Yildirim, E. A., Baloglu, U. B., Yildirim, O., & Acharya, U. R. (2020). Automated detection of COVID-19 cases using deep neural networks with X-ray images. *Computers in biology and medicine*, *121*, 103792.

[29] Shankar, K., & Perumal, E. (2021). A novel hand-crafted with deep learning features based fusion model for COVID-19 diagnosis and classification using chest X-ray images. Complex & Intelligent Systems, 7(3), 1277-1293.

[30] Akter, S., Shamrat, F. J. M., Chakraborty, S., Karim, A., & Azam, S. (2021). COVID-19 detection using deep learning algorithm on chest X-ray images. Biology, 10(11), 1174.

[31] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

[32] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, *77*, 354-377.

[33] Chelliah, P. R., Sakthivel, U., & Nagarajan, S. (Eds.). (2021). *Applied Learning Algorithms for Intelligent IoT*. CRC Press.

[34] Lambers, K. (2019). Learning to look at LiDAR: The use of R-CNN in the automated detection of archaeological objects in LiDAR data from the Netherlands. *Journal of Computer Applications in Archaeology*, *2*(1), 31-40.

[35] Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, *9*(4), 611-629.

[36] Costa, C. D. (2020) Best Python Libraries for Machine Learning and Deep Learning. Retrieved from https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

[37] JTIPTJ. (2021). Chest X-Ray (Pneumonia, Covid-19, Tuberculosis). Retrieved from https://www.kaggle.com/datasets/jtiptj/chest-xray-pneumoniacovid19tuberculosis

[38] Mooney, P. (2021). Chest X-Ray Images (Pneumonia). Retrieved from https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

[39] Rahman, T. (2021). Tuberculosis (TB) Chest X-ray Database. Retrieved from https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset

[40] Perchant, P. (2020). Chest X-Ray Images (Pneumonia). Retrieved from https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia

[41] Cohen, J. P., Morrison, P., & Dao, L. (2020). COVID-19 image data collection. *arXiv preprint arXiv:2003.11597*.

[42] Rehman, Z. U., Khan, M. A., Ahmed, F., Damaševičius, R., Naqvi, S. R., Nisar, W., & Javed, K. (2021). Recognizing apple leaf diseases using a novel parallel real-time processing framework based on MASK RCNN and transfer learning: An application for smart agriculture. *IET Image Processing*, *15*(10), 2157-2168.

[43] Komendyak. M. (2021). How to choose the best pre-trained model for your Convolutional Neural Network? Retrieved from: https://data-science-blog.com/blog/2022/04/11/how-to-choose-the-best-pre-trained-model-for-your-convolutional-neural-network

[44] Keras API reference / Keras Applications. Retrieved from https://keras.io/api/applications/

[45] Khan, E., Rehman, M. Z. U., Ahmed, F., Alfouzan, F. A., Alzahrani, N. M., & Ahmad, J. (2022). Chest X-ray classification for the detection of COVID-19 using deep learning techniques. *Sensors*, *22*(3), 1211.

[46] Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.

[47] Mingxing, T. (2019, May). EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. Retrieved from https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[48] Srinivasan, K., Garg, L., Datta, D., Alaboudi, A. A., Jhanjhi, N. Z., Agarwal, R., & Thomas, A. G. (2021). Performance comparison of deep cnn models for detecting driver's distraction. *CMC-Computers, Materials & Continua*, *68*(3), 4109-4124.

[49] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).

[50] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697-8710).

[51] Slovak,I. (2020). Few-shot face recognition using artificial neural networks.

[52] Glossary of Common Terms and API Elements. Retrieved from https://scikit-learn.org/0.23/glossary.html

[53] Suresh, A. (2020, Nov). What is a confusion matrix? Retrieved from https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

[54] Tiwari, A., Sharan, T. S., Sharma, S., & Sharma, N. (2022). Deep learning-based automated multiclass classification of chest X-rays into Covid-19, normal, bacterial pneumonia and viral pneumonia. *Cogent Engineering*, *9*(1), 2105559.