PNEUMONIA DETECTION USING DEEP LEARNING TECHNIQUES

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech

in

Electronics and Communication Engineering

By

Harshvardhan Sekar (Roll No. 108118045)

Vishal M (Roll No. 108118107)

ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI – 620015

MAY 2022

ABSTRACT

Pneumonia is among the most common infections, and it is difficult to recognise due to a lack of experts. Every year, it kills roughly seven hundred thousand children and affects seven percent of the earth's populous, roughly speaking, 450 million people.It is the most dangerous sickness for children under the age of five. This medical issue is diagnosed most of the times through examination of

Thoracic X-rays, which is very hard even for a professional radiologist. There is a need to increase diagnosis precision and accuracy. This can be accomplished by using deep neural networks, notably convolutional neural networks, shortly called as CNN Architecture (with transfer learning), for the accurate and precise identification and classification of pneumonia. In our project, we implement three Convolutional Neural Network architectures – VGG-16, Densenet-201, and EfficientNet-B0 for the detection of three different classes of pneumonia (bacterial, viral, COVID), trained on digital Thoracic X-ray pictures which might assists the professional radiologists in their decision-making process.

Keywords : Pneumonia Detection, Convolution neural network, transfer learning, VGG-16, DenseNet-201, EfficientNet-B0, Deep learning

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the following people for guiding us through this course and without whom this project and the results achieved from it would not have reached completion.

Dr.P.Sudharsan, Assistant Professor, Department of Electronics and Communication Engineering, for helping us and guiding us in the course of this project. Without his guidance, we would not have been able to successfully complete this project. His patience and genial attitude is and always will be a source of inspiration to us.

Dr.P.Muthuchidambaranathan, the Head of the Department, Department of Electronics and Communication Engineering, for allowing us to avail the facilities at the department.

We are also thankful to the faculty and staff members of the Department of Electronics and Communication Engineering, our individual parents and our friends for their constant support and help. caption

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Pneumonia is a disease that is caused by microorganisms such as bacteria, viruses, and fungi. Pneumonia originates from the greek word ”pneumon” that translates to lung in english. Consequently, pneumonia refers to a form of lung illness. Pneumonia is a condition that causes inflammation on one or both lung parenchymas. Food aspiration and chemical exposure, on the other hand, are two other causes of pneumonia. Pneumonia can be attributed to the micro-organisms infecting the lungs, thereby resulting in the alveoli in the lungs to fill up with fluid or pus, consequently limiting carbon dioxide (CO2) and oxygen (O2) exchange between the blood and the lungs, making it difficult for affected people to breathe. Some of the symptoms of pneumonia are shortness of breath,cough, fever and chest pain. Additionally, patients with diseases such as HIV/AIDS, diabetes, chronic respiratory diseases, cardiovascular diseases, cancer, hepatic disease, and other comorbidities, are at risk of pneumonia.

1.1 Manual Diagnosis of Pneumonia

The best method for diagnosing pneumonia is now chest X-rays. However, pneumonia X-ray pictures are often unclear and misclassified as other diseases or benign abnormalities. Moreover, specialists may wrongly classify pneumonia photos, causing patients to receive improper treatment which will worsen their condition. The decisions made by radiologists while diagnosing pneumonia have been found to have significant discrepancies. Radiologists are also in short supply in low-resource countries (LRCs), particularly in rural areas. As a result, there is a pressing demand for computer-aided diagnosis (CAD) systems that can assist radiologists in quickly recognizing distinct kinds of pneumonia from chest X-ray pictures.

1.2 AI Driven Pneumonia Diagnosis

A plethora of biomedical issues are now adopting AI-based solutions such as, brain tumor identification, breast cancer detection, and many more. One type of deep learning approach called Convolutional Neural Networks (CNN) has shown significant promise in image categorization and is therefore been widely adopted by the scientific community. Deep Learning techniques for examination of chest X-rays are gaining popularity since they are simple to employ with cost effective imaging

techniques. Additionally, there is an abundance of data to train various machine-learning models. Convolutional neural networks have been used in the identification of pneumonia by several research groups.

We implemented four efficient Deep Learning CNN Architectures - VGG-16, DenseNet-201, Xception and EfficientNet-B0 for Pneumonia detection using transfer learning after analyzing the performance of different Convolutional Neural Network models for categorizing the chest X-Rays images into one of the four categories: Normal, Bacterial, Viral and COVID-19 Pneumonia.

CHAPTER 2

LITERATURE REVIEW

Pneumonia detection is a technology that uses chest X-ray pictures to predict whether a person has pneumonia. It represents a significant advancement in computer vision and medical image processing.

2.1 Brief Overview of Pneumonia Detection Technqiues

Many earlier studies have stressed the use of AI models, especially Convolutional Neural Networks (CNNs), which have a demonstrated ability to extract valuable features in picture classification tasks. This feature extraction process calls for transfer learning methods, where CNN models are pre-trained to learn generic features on large datasets such as ImageNet. These pre-trained models are then applied to the desired job. The availability of pre-trained CNN models such as Alex-Net, VGG-Net, Xception, Res-Net, Efficient-Net, and Dense-Net greatly facilitates the extraction of relevant features.

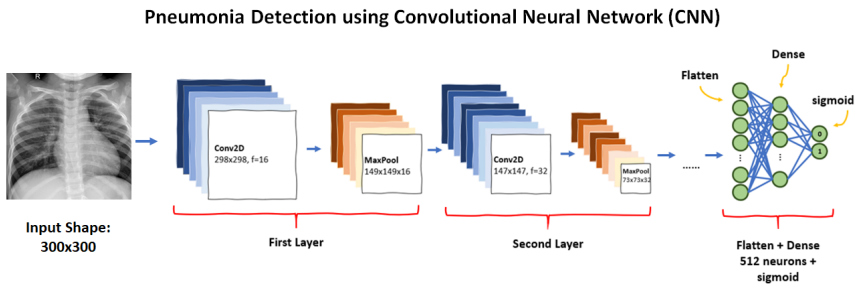

Figure 2.1: Basic CNN Architecture

2.2 Limitations in Training Data

However, several research publications have indicated that the amount of data available for pneumonia picture classification tasks is frequently limited. Due to the scarcity of data, we may have to hand-engineer features if we train our own custom neural network from scratch, as well as the risk of overfitting, in which our model's hyperparameters may well fit the training data, but they may not be an accurate representation of any new test data we work with.

2.3 Tackling Limitations using Transfer Learning

According to research, training a neural network from scratch for Computer Vision applications with sparse input is not advised (in our case, sparsity of Thoracic X-Ray images). Instead, transfer learning can be applied, which involves utilising pre-trained models (that have already been trained on big image datasets like ImageNet or for other Computer Vision tasks) with a set of known weights for pneumonia identification. In most cases, if there is only a small dataset to train on, these pre-trained models function admirably, and we can simply change the final softmax classification layer for our unique image classification needs, such as the number of picture classes in our dataset. If we had a slightly larger dataset, though, we could freeze the weights of most layers while training a few of the final layers and our bespoke softmax classifier.

Figure 2.2: General Classification Method

Increasing the diversity of the training data using various picture augmentation techniques such as mirroring, rotating, cropping, and vertical and horizontal shifting is one solution to the problem of sparse data that has been discussed frequently in the literature on pneumonia detection. This results in more unique image entries for the neural network to train on, resulting in improved performance.

2.4 Societal Relevance

With the advancement of computational power and more affordable access to medical imaging technology, the use of deep learning algorithms in medical image processing and computer-aided diagnosis to automatically detect and assist doctors and professional experts in diagnosing diseases like pneumonia has exploded. Implementing DL models, (Convolutional Neural Networks) for the categorization of pneumonia utilizing state-of-the-art CNN architectures applied by transfer learning has gained a lot of attention. These trials have consistently produced reliable results that match or even exceed those of radiologists and other medical experts.

CHAPTER 3

METHODOLOGY

3.1 Dataset

3.1.1 Chest X-Ray Images (Pneumonia) Dataset

The data is partitioned into train, test, and Val folders. Each folder has subfolders for either Pneumonia or Normal image category.

Dorsal-Ventral chest X-ray images were picked from samples of children patients whose age is in the range of one to five years old at a Guangzhou Medical Center in Guangzhou, China.

All chest x-ray imaging was carried out as a routine medical treatment for the patients. Quality Control was performed on all chest radiographs by examination of the scans. Scans that were of low quality or unreadable were removed.

The photos' diagnoses were then graded by a couple of experts after which it is approved for training the DL model. Another expert validates to make sure there were no grading problems in the evaluation set.

3.1.2 COVID-19 Detection X-Ray Dataset

Data scientists are looking for trends and strategies to speed up testing and diagnostics following the Corona-virus (COVID19) outbreak. COVID19, bacterial and viral-pneumonia patients, and fit persons are all represented in this dataset. This dataset was uploaded in the hopes of detecting a pattern that might be useful to medical practitioners.

3.1.3 COVID-19 Radiography Database

Researchers from universities spanning several countries such as Qatar, Bangladesh, Pakistan, and Malaysia, collaborated to compile a data repository of chest x-ray images of different classes such as covid-19 positive cases, as well as Normal and Viral Pneumonia images, with the assistance of the medical professionals.

3.1.4 Pneumonia (Virus) vs COVID-19

The COVID-19 X-Ray dataset and the Pneumonia (Virus) Chest X-Ray dataset were used to create this dataset.

This dataset is intended to aid other researchers in developing a more detailed diagnosis method for COVID-19 patients. Pneumonia induced by COVID-19 is distinguished from Pneumonia Virus in general.

3.1.5 The Final Dataset

Upon compilation of the four above datasets that have been examined, the resulting dataset has a split of seventy percent for the training set, fifteen percent for the validation set, and fifteen percent for the testing set

Our dataset is tailored to solve the four-class classification problem. The images in our dataset are grouped into the following classes:

* Normal Images : 2914
* Bacterial Pneumonia : 2503
* Viral Pneumonia : 2502
* COVID-19 : 2502

3.2 Pre-Processing

To divide the data into training, testing, and validation sets, data segregation is used.

To meet the input requirements for CNN architectures, data preprocessing is used. All chest x-ray images are scaled to 224 by 224 pixels in size.

We've put together our final dataset and divided it into three parts: Seventy percent for the training set, fifteen percent for the validation set, and another fifteen percent for the test set.

CHAPTER 4

Activation Functions

The activation function of a node or neuron in an artificial neural network determines the output of that node or neuron for a given input or combination of inputs. The output of one neuron is propagated as an input to the next neuron. This process continues several times until the required solution to the problem statement is obtained.

Activation functions convert the values into the required range, such as 0 to 1, -1 to 1, and so on. It is determined by the activation function selected. The employment of a logistic activation function (sigmoid function) in the real number domain, for example, would translate all inputs into the range of 0 to 1.

4.1 Sigmoid Function

$$f(x) = 1/(1 + e - x)$$

is the mathematical formula for sigmoid non-linearity. as depicted in the image below. It ”squashes” a real-valued number into a range of 0 to 1. Large negative numbers, in particular, are converted to 0 and large positive ones to 1. Since it has an intuitive interpretation as a neuron’s firing rate: from not firing at all (0) to fully saturated firing (1) at an established maximum frequency, the sigmoid function has seen widespread use in the past.

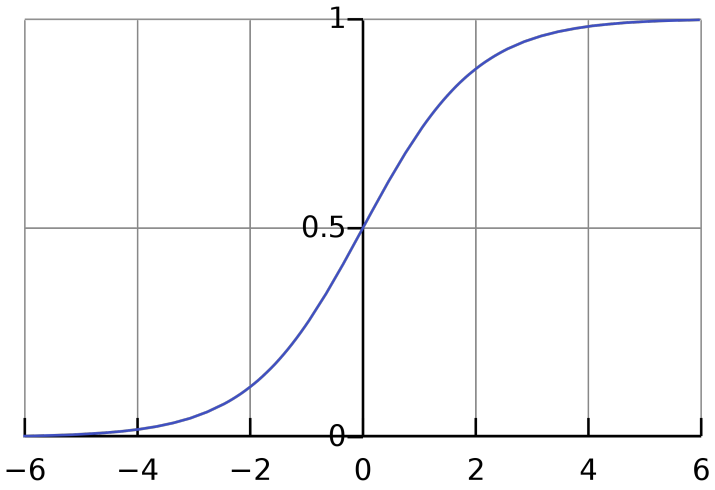

Figure 4.1: Sigmoid Activation Function

Gradients are saturated and killed by sigmoid functions. When the neuron's activity saturates at 0 or 1, the gradient in these areas is almost zero, which is a frequent property of the sigmoid. This local gradient will be multiplied by the gate output for the entire objective during backpropagation. As a result, if the local gradient is very modest, the gradient is effectively "killed," and very little signal will pass through the neuron to its weights and recursively to its data. In order to avoid saturation, an additional penalty will be applied when the weights of sigmoid neurons are initialised. If the initial weights are too large, for example, most neurons will get saturated, and the network will struggle to learn.

4.2 ReLU - Rectified Linear Unit

It's the most commonly utilised activation method. Because it is employed in nearly all convolutional neural networks. From the bottom, ReLU is 50

It is simple to optimise models having linear operations. ReLU tends to operate effectively on most issues since it shares many of the qualities of linear functions. The main problem is that the derivative is undefined at x = 0, which we can solve by setting it to 0 at x = 0. However, when x = 0, the gradient is zero, and learning is impossible.

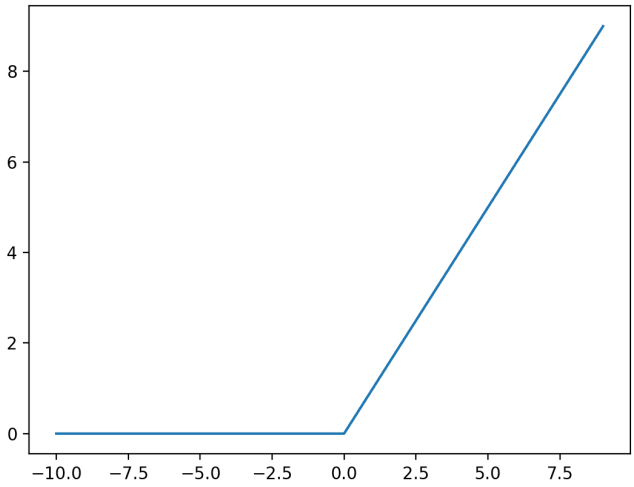

Figure 4.2: ReLU Activation Function

4.3 Softmax Function

The sigmoid function is simple to use, and ReLUs will not eliminate the effect during your training. However, they aren't much assistance when it comes to classification problems; the sigmoid function can only handle two classes, which isn't what we want, and we need something more. The softmax function restricts the outputs of each unit to be between 0 and 1. It also separates each output into equal halves so that the summation of the outputs equals one. The softmax function returns a categorical probability distribution, which indicates the likelihood that any of the classes is true.

CHAPTER 5

CNN Architectures

5.1 General CNN Architecture

CNNs are essentially feed-forward ANNs with two constraints: firstly, the neurons of the same filter are bounded to small patches of the picture to retain geometrical structure, secondly, the CNN makes use of shared weights to reduce the total number of parameters in the model

CNN is made up of three layers: first type of layer called convolution layer facilitates the learning of features. A second layer called as max-pooling or sub-sampling layer is implemented for downsampling the image and decreasing dimensionality, thereby reducing computational effort, finally a fully connected layer is used to classify the images into the respected classes. The architectural overview of CNN is shown in the diagram below.

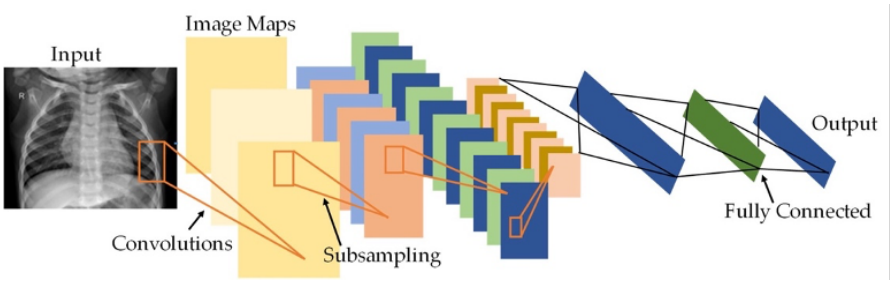

Figure 5.1: General CNN Architecture

CNNs have gone through a lot of changes and improvements over the years. As a result, we now have a variety of CNN models to choose from, the most important of which are discussed below.

5.2 VGG-16

- VGG-16 has 3 fully-connected layers and 13 convolutional layers in its architecture. It, like AlexNet, utilised ReLUs as activation functions. There were 138 million parameters in VGG-16. Along with VGG-16, a deeper version, VGG-19, was also built.

- Year of Publication: 2014

Visual Geometry Group (VGG) is responsible for the implementation fo VGG-16 and VGG-19,with an idea that piling up more layers onto a CNN would improve its efficiency.

- Unique selling proposition: First among the deeper CNNs.

The VGG-16 architecture is depicted in the graphic below.

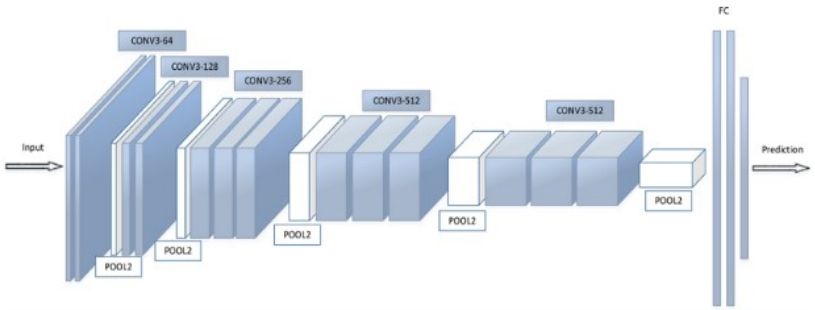

Figure 5.2: VGG-16 Architecture

5.3 DenseNet

Because it does not train redundant feature maps, Dense-Net, abbreviated as Dense Convolutional Network, requires fewer parameters than a typical CNN. DenseNet is composed of 12 incredibly thin filters which results in smaller number of new feature maps. The original input image and loss function grandients for each layer are directly accessible to densenet, thereby decreasing the computational cost by a huge amount.

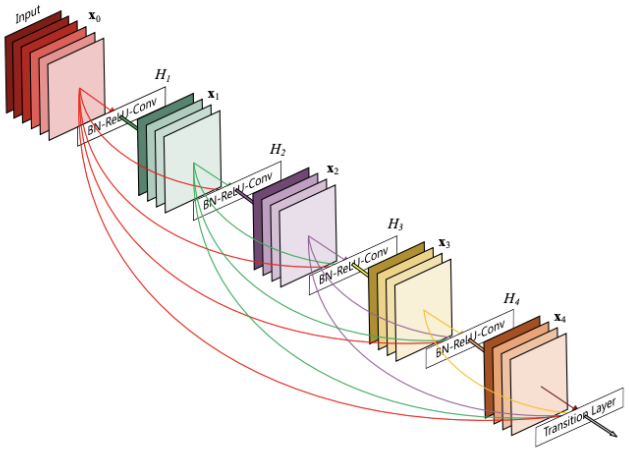

Figure 5.3: DenseNet Architecture

5.4 EfficientNet

Efficient-Net is a CNN architecture that provides an efficient scaling method by making use of a compound coefficient that consistently scales network breath, depth with a set of predetermined scaling coefficients.

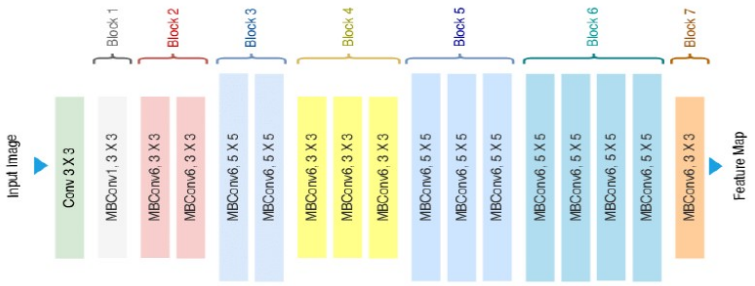

Figure 5.4: EfficientNet Architecture

5.5 Xception

• Architecture: Xception uses only depthwise separable convolution layers in its convolutional neural network architecture. It has 23 million characteristics and was 71 layers deep. It was built on the basis of Inception-v3 .

• Year of Release: 2016

• About: Xception was significantly influenced by Inception-v3, albeit it used depth-wise separable convolutions instead of convolutional blocks.

• Unique selling point: Xception is a CNN built entirely of depth-wise separable convolutional layers.

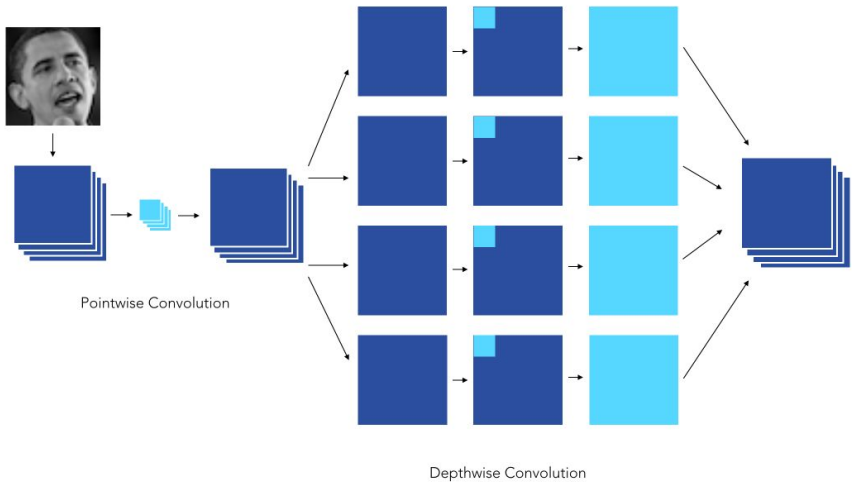

Figure 5.5: Xception Architecture

5.6 Transfer Learning and Fine Tuning

• A pre-trained model is network which saves weights that have been previously trained on a large dataset, typically a large-scale image classification problem.

• When you apply what you learned while tackling one problem to a different but related one, this is known as transfer learning.

• If the new dataset is small, only train the network's last layers to avoid overfitting and leave the other layers alone. Remove the final layers of the pre-trained network. Make additional layers. Retraining should only be done on the new layers.

5.7 ImageNet Dataset

• In transfer learning applications, ImageNet is the most extensively utilised dataset.

• Most advanced Convolutional Neural Network models used in Computer Vision tasks are first pre-trained on ImageNet, a big dataset of pictures.

• ImageNet is a massive collection of annotated pictures used in computer vision research. It has over 14 million photos, over 21,000 image groupings, and over 1 million images with bounding box annotation.

CHAPTER 6

Loss Functions & Optimizers

6.1 Loss

Loss is the cost of making a faulty forecast; it is a metric that represents how inaccurate the model's prediction was on a single example. The loss is zero if the model's forecast is perfect; else, the loss is bigger. We train the model to identify the optimum combination of weights and biases that on average cause the least amount of loss. The accompanying diagram, for example, depicts a a model with low loss on the right hand side and the model with the higher loss on the right hand side. The blue lines reflect predictions, while the arrows represent the loss.

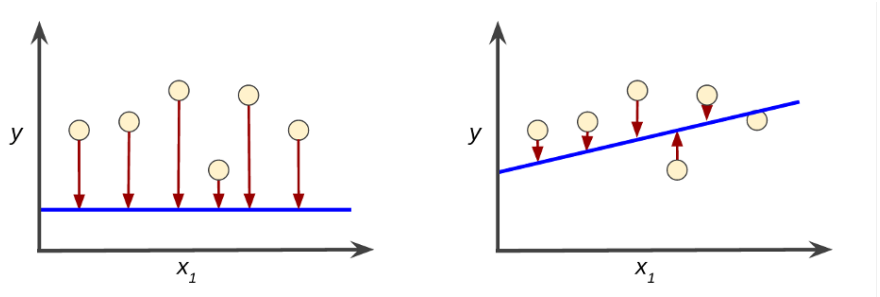

Figure 6.1: High Loss vs Low Loss Diagram

6.2 Squared Loss Function

The mean square error is the average of the model's squared loss per example across the entire dataset. To find MSE, sum all the squared losses corresponding to each case and then divide by the number of examples:

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prediction(x))^2$$

Figure 6.2: Mean Square Error

Where:

• Consider the example (x,y):

• x is the set of features that the model utilises to create predictions (for example, age, gender, and so on).

• The label for the example is y. (for example, temperature).

• The weights and bias, with the collection of features x, determine prediction(x).

• D is a dataset that contains numerous (x,y) pairs of labelled samples.

• In D, N is the number of examples.

MSE is a regularly used loss function in machine learning, however it is neither the sole or the optimal loss function in all situations.

6.3 Cross-Entropy

The function of loss Machine learning makes heavy use of cross-entropy. Cross-entropy is the difference between two probability distributions for a given random variable or set of events.

6.4 Categorical Cross-Entropy

Categorical cross-entropy is a loss function in multi-class classification applications. These are issues when an example can only fit into one of several categories, and the model must decide which one to use.

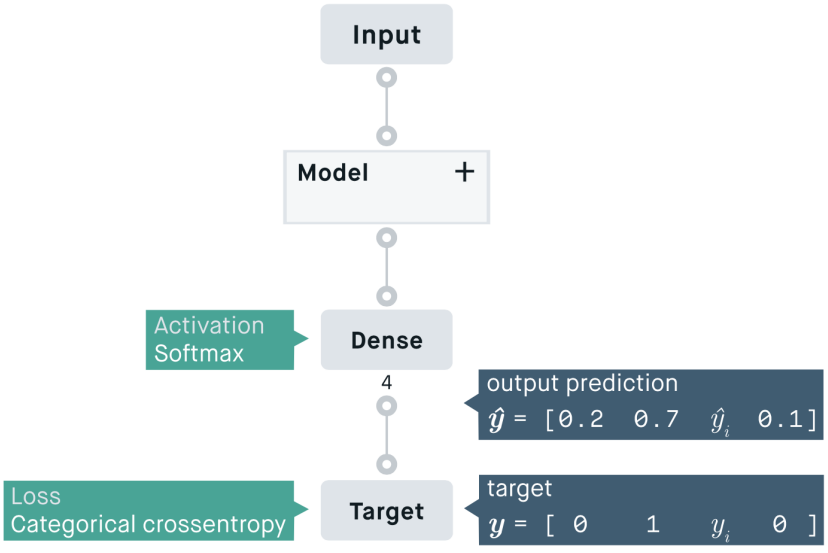

Figure 6.3: Categorical Cross Entropy

6.5 Categorical Cross Entropy Math

The categorical cross-entropy loss function computes an example's loss by adding the following values:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 6.4: Categorical Cross Entropy Loss Function

• Where y_i is the appropriate target value and output size is the number of scalar values in the model output, • y_i is the ith scalar value in the model output.

• This loss is an excellent indicator of how different two discrete probability distributions are from one another. In this case, yi represents the likelihood that event I will occur, and the sum of all yi equals 1, implying that only one event will occur. The minus sign assures that when the distributions come closer together, the loss gets smaller.

6.6 Optimizers

Optimizers are algorithms or techniques for minimising an loss function (error function) or growing manufacturing efficiency. Optimizers are mathematical capabilities which might be primarily based totally at the learnable parameters of the model, which include Weights and Biases. Optimizers help in figuring out a way to alter the weights and gaining knowledge of charge of a neural community with a purpose to minimise losses.

6.7 Adam Optimizer

Adam optimization is a variation of stochastic gradient descent that may be used to replace weights greater fast than conventional stochastic gradient descent.

The Adam Optimization Technique is useful for a variety of reasons:

• It has a small memory footprint and is computationally efficient.

• It is unaffected by gradient diagonal rescaling.

- It's ideal for issues involving a lot of data and parameters.
- It's good for problems with a lot of noise or sparse gradients.
- They have intuitive hyperparameter interpretation and require little adjustment.

6.8 Weight Decay – Regularization Technique

- One strategy to avoid overfitting is to keep our models as simple as possible.
- We can use weight decay to prevent overfitting.
- Adding the squared sum of all the parameters to the loss function is one technique to penalise increasing complexity. However, this could result in a significant increase in the loss function.
- To avoid this, we divide the sum of the squares of the parameters by a smaller value, which we call weight decay.

CHAPTER 7

Proposed Model

We have used Tensorflow framework for our model. We have collected four different dataset each having any combination of our four classes.

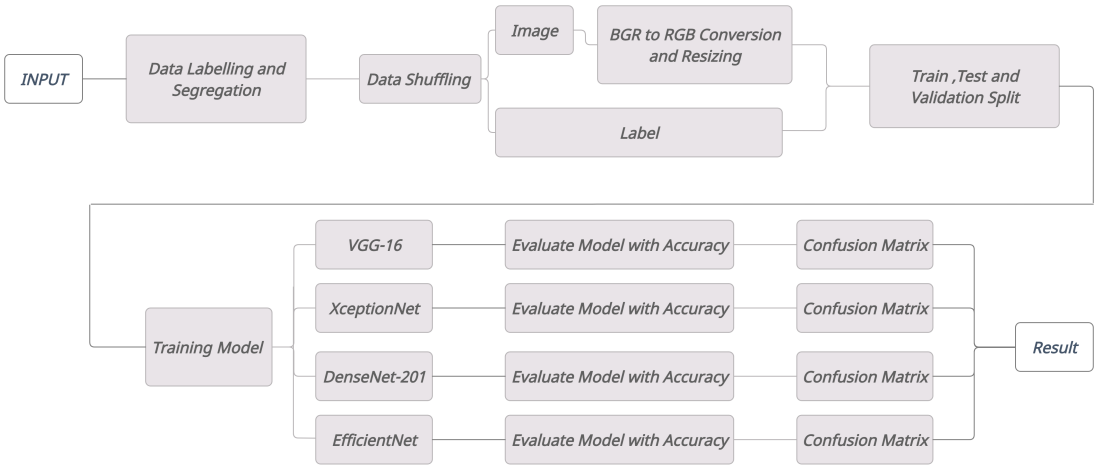

Figure 7.1: Overall Flow of the Model

7.1 Pre-Processing

We loop through each and every directory of the all the datasets. We tried to use all images in JPEG format. We haev albelled all the four classes as follows:

$ClassName$	$LabelNumber$
Normal	0
Bacterial-Pneumonia	1
Viral-Pneumonia	2
COVID-19	3

Table 7.1: Class Table

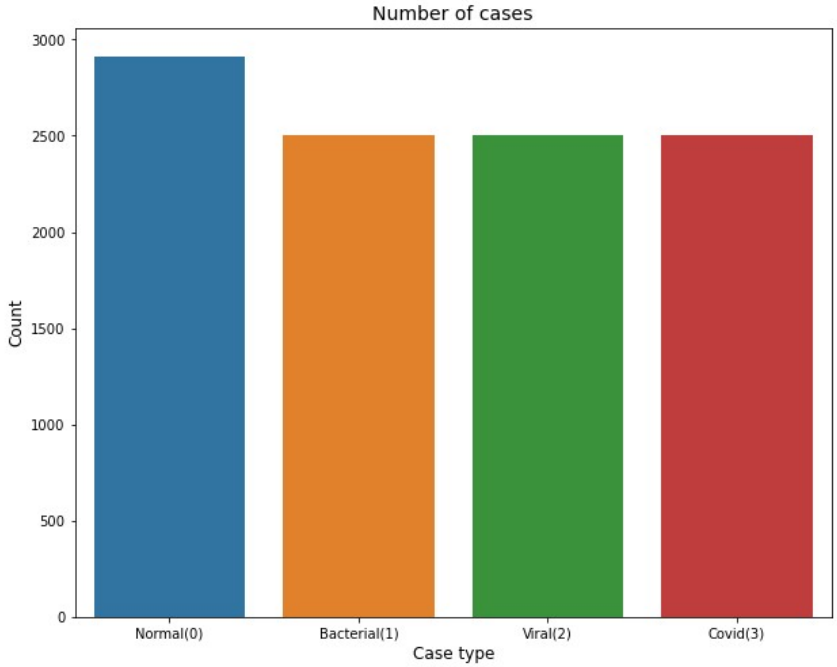

Figure 7.2: Distribution of Dataset

We now compile all the images together into a list call data. Now we convert the dataset into a dataframe using Pandas Library having columns : image and label. Now, we shuffle the dataset and the resulting dataset is represented as follows:

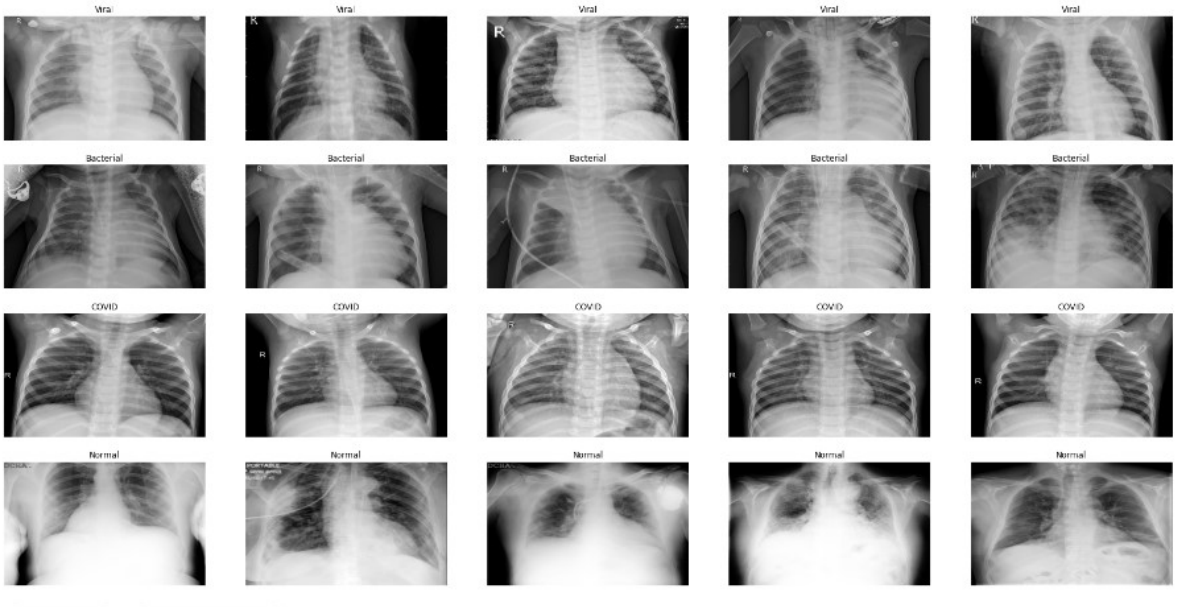

Figure 7.3: Sample Chest X-Ray Images

	image	label
0	../input/covid19-detection-xray-dataset/ValDat...	2
1	../input/chest-xray-pneumonia/chest_xray/chest...	1
2	../input/chest-xray-pneumonia/chest_xray/chest...	1
3	../input/covid19-detection-xray-dataset/ValDat...	0
4	../input/chest-xray-pneumonia/chest_xray/chest...	1
5	../input/covid19-radiography-database/COVID-19...	3
6	../input/chest-xray-pneumonia/chest_xray/chest...	2
7	../input/chest-xray-pneumonia/chest_xray/chest...	1
8	../input/covid19-detection-xray-dataset/NonAug...	3
9	../input/chest-xray-pneumonia/chest_xray/chest...	0

Figure 7.4: Sample Dataframe Entries

Now, we iterate through all the rows of the dataframe and we try to read the image using cv2 package.

$img = cv2.imread(str(img))$

$img = cv2.resize(img, (224, 224))$

We then resize the image into 224x224 pixels. Now, we convert the image into 3 channel RGB.

$img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)$

Now we add the labels of the corresponding image into a list called label_data and the image into a list called image_data. These two lists will have a one-one relationship.

We now split the image and the labelled data into three catagories: image_train, image_val, and image_test & label_train, label_val, and label_test.

The training, validation and test split will be in the ratio of 70:15:15.

7.2 Training the Model

Now we have used different pre-trained models such as VGG-16, DenseNet-201, Efficient-B0, and Xception. We initialize the weights using ImageNet weights (These are the weights of the different Neural Network Layers upon training on the ImageNet dataset).

Now, we use the $dense$ function in order to classify the output among the four categories.

We now compile our model using categorical cross entropy as our loss function

and Adam Optimizer with a learning rate of 10^{-4} , and decay 10^{-5}. We measure accuracy as our primary metric.

We use an early stopping monitor to stop the model once have reached our target accuracy. Having batch size as 16, we try to fit the model with our training data and validation data. We iterate for 10 epochs with the early stopping monitor as the callback.

Now, we try to evaluate our model against the test dataset which provides loss and accuracy.

We use the Confusion Matrix to evaluate the classification model's performance, focusing on recall, precision, and F1-score.

CHAPTER 8

Confusion Matrix and Evaluation Metrics

8.1 Confusion Matrix

The Confusion Matrix is used to determine how well a machine learning classification performs. It has a matrix representation. The Confusion Matrix allows you to compare actual and expected values. N is the number of classes or outputs, while N is the confusion matrix. We have four classes in our problem statement. As a result, we have a 4x4 matrix of perplexity. True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) are the four terms used in the Confusion Matrix (FN).

• True Positive – This indicates that the actual and projected values are identical.

• False Negative - This occurs when the actual number is positive but the model predicts a negative result.

• False Positive - The actual value is negative, but the model predicts a positive result.

• True Negative – This indicates that the actual and projected numbers are identical.

Following image represents a 4x4 Confusion Matrix:

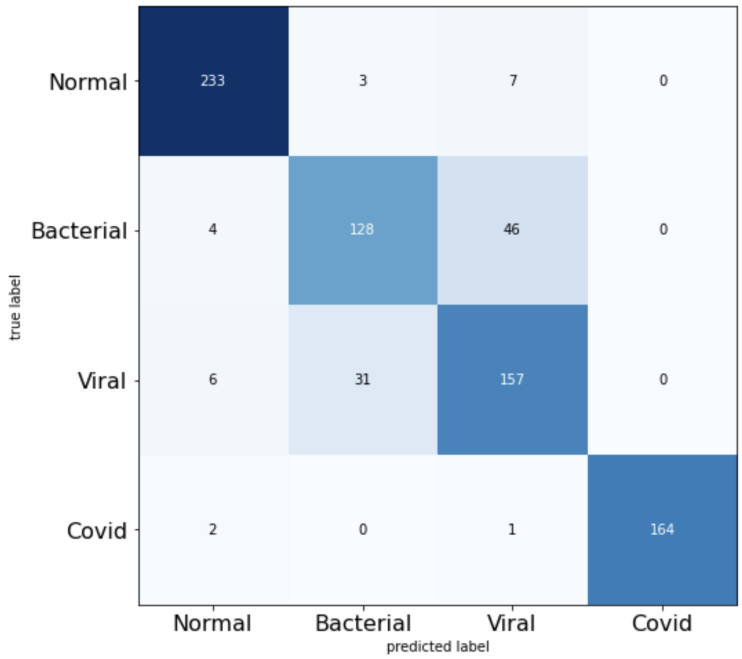

Figure 8.1: 4x4 Confusion Matrix

8.2 Evaluation Metrics

Accuracy: The percentage of correct guesses in the total number of predictions.

$$Accuracy = \frac{TruePositives + TrueNegatives}{N}$$

Figure 8.2: Accuracy Equation

Precision is the percentage of accurately identified affirmative cases.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Figure 8.3: Precision Equation

Recall: The proportion of genuine positive cases that are discovered correctly.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Figure 8.4: Recall Equation

F1 Score: For a classification task, the F1 Score is the harmonic mean of precision and recall values.

$$F = 2 * \frac{Precision_p * Recall_p}{Precision_p + Recall_p}$$

Figure 8.5: F1-Score

CHAPTER 9

Results and Conclusions

9.1 Results

9.1.1 Accuracy

$CNN Architecture$	$Accuracy$
VGG-16	90.1
Xception	89.77
DenseNet	87.2
EfficientNet-B0	84.5

Table 9.1: Accuracy yield by the 4 CNN Architectures

9.1.2 Evaluation Metrics : Recall, Precision and Accuracy

$ClassLabel$	$Recall$	$Precision$	$F1 - Score$
Normal	99	95	96.95
Bacterial-Pneumonia	77	89	82.56
Viral-Pneumonia	86	75	80.12
COVID-19	98	99	98.49

Table 9.2: Metrics of All Classes obtained using VGG-16 model

$ClassLabel$	$Recall$	$Precision$	$F1 - Score$
Normal	96	98	97.0
Bacterial-Pneumonia	81	84	82.47
Viral-Pneumonia	83	79	80.95
COVID-19	100	98	99.0

Table 9.3: Metrics of All Classes obtained using XceptionNet model

$ClassLabel$	$Recall$	$Precision$	$F1-Score$
Normal	96	95	95.49
Bacterial-Pneumonia	72	79	75.33
Viral-Pneumonia	81	74	77.34
COVID-19	98	100	98.98

Table 9.4: Metrics of All Classes obtained using DenseNet- 201 model

$ClassLabel$	$Recall$	$Precision$	$F1-Score$
Normal	93	90	91.5
Bacterial-Pneumonia	73	71	72.0
Viral-Pneumonia	66	70	68.0
COVID-19	99	99	99.0

Table 9.5: Metrics of All Classes obtained using EfficientNet- B0 model

9.2 Conclusions

• We have proposed a four-class classification model and were able to achieve high performance on the collected datasets namely Chest X-Ray Images (Pneumonia) ,COVID-19 Detection X-Ray , COVID-19 Radiography Database and Pneumonia (Virus) vs COVID-19 Dataset.

• We have implemented the four best CNN Architectures and have obtained an accuracy of 90.1% for VGG-16 , 89.77% for XceptionNet, 87.2% for DenseNet-201 and 84.5% for EfficientNet-B0.

• We have used pretrained weights of ImageNet to perform Transfer Learning on the implemented models.

• Recall, Precision, and F1 Score were utilised as our Evaluation Metrices to evaluate the performance of the classification model.

APPENDIX A

CODE ATTACHMENTS

```
#get data
data_dir1 = Path('../input/chest-xray-pneumonia/chest_xray/chest_xray
    ')
data_dir2 = Path('../input/covid19-detection-xray-dataset')
data_dir3 = Path('../input/covid19-radiography-database/COVID-19
    _Radiography_Dataset')
data_dir4 = Path('../input/pneumonia-virus-vs-covid19/
    Pneumonia_and_COVID19')

data = []
covid_data = []
normal_data = []
bacterial_data = []
viral_data = []

"""Load data in from the each dataset. As of now, we are not
    sperating the data into training, test, and validation, but into
    different lists based on the diagnosis."""

loop_dir1 = ['test', 'train', 'val']
for i in loop_dir1:
    normal_dir1 = data_dir1 / i / 'NORMAL'
    pneumonia_dir1 = data_dir1 / i / 'PNEUMONIA'
    normal_cases1 = normal_dir1.glob('*.jpeg')
    pneumonia_cases1 = pneumonia_dir1.glob('*.jpeg')
    for img in normal_cases1:
        normal_data.append((img,0))
        if len(normal_data) >= 2500:
            break
    for img in pneumonia_cases1:
        if "bacteria" in str(img):
            bacterial_data.append((img,1))
            if len(bacterial_data) >= 2500:
                break
        elif "virus" in str(img):
            viral_data.append((img,2))
            if len(viral_data) >= 2500:
                break
del loop_dir1

loop_dir2 = ['NonAugmentedTrain', 'ValData']
for i in loop_dir2:
    normal_dir2 = data_dir2 / i / 'Normal'
    bacterial_dir2 = data_dir2 / i / 'BacterialPneumonia'
    viral_dir2 = data_dir2 / i / 'ViralPneumonia'
```

[

]

```
    covid_dir2 = data_dir2 / i / 'COVID-19'
    normal_cases2 = normal_dir2.glob('*.jpeg')
    bacterial_cases2 = bacterial_dir2.glob('*.jpeg')
    viral_cases2 = viral_dir2.glob('*.jpeg')
    covid_cases2 = covid_dir2.glob('*.jpeg')
    for img in normal_cases2:
        normal_data.append((img,0))
    for img in bacterial_cases2:
        bacterial_data.append((img,1))
        if len(bacterial_data) >= 2500:
            break
    for img in viral_cases2:
        viral_data.append((img,2))
    for img in covid_cases2:
        covid_data.append((img,3))
        if len(covid_data) >= 2500:
            break
del loop_dir2

len(os.listdir('../input/covid19-radiography-database/COVID-19
    _Radiography_Dataset/COVID/images'))

normal_dir3 = Path('../input/covid19-radiography-database/COVID-19
    _Radiography_Dataset/Normal/images')
covid_dir3 = Path('../input/covid19-radiography-database/COVID-19
    _Radiography_Dataset/COVID/images')
viral_dir3 = Path('../input/covid19-radiography-database/COVID-19
    _Radiography_Dataset/Viral_Pneumonia/images')
normal_cases3 = normal_dir3.glob('*')
viral_cases3 = viral_dir3.glob('*')
covid_cases3 = covid_dir3.glob('*')

count = 0
for img in normal_cases3:
    normal_data.append((img,0))
    count+=1
    if len(normal_data) >= 2500:
        break
for img in covid_cases3:
    covid_data.append((img,3))
    if len(covid_data)>= 2500:
        break
for img in viral_cases3:
    viral_data.append((img,2))
    if len(viral_data)>= 2500:
        break
gc.collect()

loop_dir4 = ['TEST', 'TRAIN']
for i in loop_dir4:
    viral_dir4 = data_dir4 / i / 'PNEUMONIA_(VIRUS)'
    viral_cases4 = viral_dir4.glob('*.jpeg')
    covid_dir4 = data_dir4 / i / 'COVID-19'
    covid_cases4 = covid_dir4.glob('*.jpeg')
    for img in viral_cases4:
        viral_data.append((img,2))
```

[

]

```
93          if len(viral_data) >= 2500:
94              break
95      for img in covid_cases4:
96          covid_data.append((img,3))
97          if len(covid_data) >= 2500:
98              break
99
100 print('number of normal cases: ' + str(len(normal_data)))
101 print('number of bacterial cases: ' + str(len(bacterial_data)))
102 print('number of viral cases: ' + str(len(viral_data)))
103 print('number of covid cases: ' + str(len(covid_data)))
104
105 gc.collect()
106
107 del data_dir1, data_dir2, data_dir3, data_dir4
108 # save all data into single list
109
110 data += normal_data
111 del normal_data
112 data += bacterial_data
113 del bacterial_data
114 data += viral_data
115 del viral_data
116 data += covid_data
117 del covid_data
118
119 #Convert to dataframe and shuffle.
120
121 data = pd.DataFrame(data, columns=['image', 'label'],index=None)
122 #shuffle
123 data = data.sample(frac=1.).reset_index(drop=True)
124 #print
125 data.head()
126
127 #Visualize counts
128 cases_count = data['label'].value_counts()
129 print(cases_count)
130
131 # Plot the results
132 plt.figure(figsize=(10,8))
133 sns.barplot(x=cases_count.index, y= cases_count.values)
134 plt.title('Number of cases', fontsize=14)
135 plt.xlabel('Case type', fontsize=12)
136 plt.ylabel('Count', fontsize=12)
137 plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Bacterial(1)
    ', 'Viral(2)','Covid(3)'])
138 plt.show()
139 del cases_count
140
141 #show sample
142 viral_samples = (data[data['label']==2]['image'].iloc[:5]).tolist()
143 bacterial_samples = (data[data['label']==1]['image'].iloc[:5]).tolist
    ()
144 normal_samples = (data[data['label']==0]['image'].iloc[:5]).tolist()
145 covid_samples = (data[data['label']==3]['image'].iloc[:5]).tolist()
146
```

[

]

```
# Concat the data in a single list and del the above 3 lists
samples = viral_samples + bacterial_samples + normal_samples  +
    covid_samples
del viral_samples, normal_samples, bacterial_samples, covid_samples

# Plot the data
f, ax = plt.subplots(4,5, figsize=(30,15))
for i in range(20):
    img = imread(samples[i])
    ax[i//5, i%5].imshow(img, cmap='gray')
    if i<5:
        ax[i//5, i%5].set_title("Viral")
    elif i>=5 and i<10:
        ax[i//5, i%5].set_title("Bacterial")
    elif i>=10 and i<15:
        ax[i//5, i%5].set_title("COVID")
    else:
        ax[i//5, i%5].set_title("Normal")
    ax[i//5, i%5].axis('off')
    ax[i//5, i%5].set_aspect('auto')
plt.show()

print(data.shape)
gc.collect()

"""This code takes care of that by resizing the images, turning
    grayscale images(they only have 1 layer so we need to convert them
    ) into colored ones (which have 3 layers), and setting all images
    to the same color so color won't be a confounding variable."""

image_data = []
label_data = []
for index, d in data.iterrows():
    img = d['image']
    l = d['label']
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] ==1:
        img = np.dstack([img, img, img])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(l, num_classes=4)
    image_data.append(img)
    label_data.append(label)
image_data = np.array(image_data)
label_data = np.array(label_data)
del data

gc.collect()

"""Here we finally split the data into training, validation, and test
    ."""
```

A.2 Training, Testing and Validation Split

[

]

```
image_train, image_validate, image_test = np.split(image_data, [int(.7*len(image_data)), int(.85*len(image_data))])
label_train, label_validate, label_test = np.split(label_data, [int(.7*len(label_data)), int(.85*len(label_data))])
print('number of training images and labels: ' + str(len(image_train)) + ' and ' + str(len(label_train)))
print('number of validation images and labels: ' + str(len(image_validate)) + ' and ' + str(len(label_validate)))
print('number of test images and labels: ' + str(len(image_test)) + ' and ' + str(len(label_test)))
del image_data, label_data

gc.collect()
```

A.3 Training DeepLearning Models

```
#Model Time
#VGG- 16
from keras.applications.vgg16 import VGG16
vgg16_weights = '../input/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5'
vgg16_model = VGG16(weights=vgg16_weights)
new_vgg16 = Sequential()
for layer in vgg16_model.layers[:-1]:
    new_vgg16.add(layer)
new_vgg16.add(Dense(4, activation='softmax'))
del vgg16_model, vgg16_weights
#new_vgg16.summary()

new_vgg16.compile(loss='categorical_crossentropy',
                  optimizer= Adam(lr=0.0001, decay=1e-5),
                  metrics=['accuracy'])
gc.collect()

early_stopping_monitor = EarlyStopping(
    monitor='val_acc',
    min_delta=0,
    patience=4,
    verbose=0,
    mode='max',
    baseline=None,
    restore_best_weights=True
)

batch_size = 16
with tf.device('/gpu:0'):
    new_vgg16.fit(image_train, label_train, \
                  validation_data=(image_validate, label_validate), \
                  epochs=10, callbacks=[early_stopping_monitor], batch_size=batch_size, verbose = 0)

test_loss, test_score = new_vgg16.evaluate(image_test, label_test, batch_size=16)
print("Loss on test set: ", test_loss)
```

[

]

```
print("Accuracy on test set: ", test_score)

# Get predictions
preds = new_vgg16.predict(image_test, batch_size=16)
preds = np.argmax(preds, axis=-1)

# Original labels
orig_test_labels = np.argmax(label_test, axis=-1)

print(orig_test_labels.shape)
print(preds.shape)

# Get the confusion matrix
cm  = confusion_matrix(orig_test_labels, preds)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.
    Blues)
plt.xticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
plt.yticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
plt.show()

#Confusion Matrix

nn, nb, nv, nc, bn, bb, bv,bc, vn, vb, vv,vc,cn,cb,cv,cc = cm.ravel()

#precision
normal_precision = nn/(nn+bn+vn+cn)
bacterial_precision = bb/(nb+bb+vb+cb)
viral_precision = vv/(nv+bv+vv+cv)
covid_precision = cc/(nc+bc+vc+cc)
print("Precision of the model when dealing with normal pneumonia is
    {:.2f}".format(normal_precision))
print("Precision of the model when dealing with bacterial pneumonia
    is {:.2f}".format(bacterial_precision))
print("Precision of the model when dealing with viral pneumonia is
    {:.2f}".format(viral_precision))
print("Precision of the model when dealing with covid pneumonia is
    {:.2f}".format(covid_precision))

normal_recall = nn/(nn+nb+nv+nc)
bacterial_recall = bb/(bn+bb+bv+bc)
viral_recall = vv/(vn+vb+vv+vc)
covid_recall = cc/(cn+cb+cv+cc)
print("Recall of the model when dealing with normal pneumonia is {:.2
    f}".format(normal_recall))
print("Recall of the model when dealing with bacterial pneumonia is
    {:.2f}".format(bacterial_recall))
print("Recall of the model when dealing with viral pneumonia is {:.2f
    }".format(viral_recall))
print("Recall of the model when dealing with covid pneumonia is {:.2f
    }".format(covid_recall))

gc.collect()

```

[

]

```
#DenseNet-201

pretrained_model3 = tf.keras.applications.DenseNet201(input_shape
    =(224,224,3),include_top=False,weights='imagenet',pooling='avg')
pretrained_model3.trainable = False

inputs3 = pretrained_model3.input
x3 = tf.keras.layers.Dense(128, activation='relu')(pretrained_model3.
    output)
outputs3 = tf.keras.layers.Dense(4, activation='softmax')(x3)
newmodel = tf.keras.Model(inputs=inputs3, outputs=outputs3)

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

gc.collect()
gc.collect()

batch_size = 16
with tf.device('/gpu:0'):
    newmodel.fit(image_train, label_train, \
                 validation_data=(image_validate, label_validate),
                     \
                 epochs=10, callbacks=[early_stopping_monitor],
                     batch_size=batch_size, verbose = 0)

test_loss, test_score = newmodel.evaluate(image_test, label_test,
    batch_size=16)
print("Loss on test set: ", test_loss)
print("Accuracy on test set: ", test_score)

# Get predictions
preds = newmodel.predict(image_test, batch_size=16)
preds = np.argmax(preds, axis=-1)

# Original labels
orig_test_labels = np.argmax(label_test, axis=-1)

print(orig_test_labels.shape)
print(preds.shape)

gc.collect()

# Get the confusion matrix
cm = confusion_matrix(orig_test_labels, preds)
plt.figure()
plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True,cmap=plt.cm.
    Blues)
plt.xticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
plt.yticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
plt.show()

nn, nb, nv, nc, bn, bb, bv,bc, vn, vb, vv,vc,cn,cb,cv,cc = cm.ravel()

```

[

]

```
gc.collect()

#precision
normal_precision = nn/(nn+bn+vn+cn)
bacterial_precision = bb/(nb+bb+vb+cb)
viral_precision = vv/(nv+bv+vv+cv)
covid_precision = cc/(nc+bc+vc+cc)
print("Precision of the model when dealing with normal pneumonia is {:.2f}".format(normal_precision))
print("Precision of the model when dealing with bacterial pneumonia is {:.2f}".format(bacterial_precision))
print("Precision of the model when dealing with viral pneumonia is {:.2f}".format(viral_precision))
print("Precision of the model when dealing with covid pneumonia is {:.2f}".format(covid_precision))

normal_recall = nn/(nn+nb+nv+nc)
bacterial_recall = bb/(bn+bb+bv+bc)
viral_recall = vv/(vn+vb+vv+vc)
covid_recall = cc/(cn+cb+cv+cc)
print("Recall of the model when dealing with normal pneumonia is {:.2f}".format(normal_recall))
print("Recall of the model when dealing with bacterial pneumonia is {:.2f}".format(bacterial_recall))
print("Recall of the model when dealing with viral pneumonia is {:.2f}".format(viral_recall))
print("Recall of the model when dealing with covid pneumonia is {:.2f}".format(covid_recall))

gc.collect()

#Xception

from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.optimizers import Adam

base = Xception(include_top=False, weights='imagenet', input_shape=(128,128,3))
x = base.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
head = Dense(4, activation='softmax')(x)
model = Model(inputs=base.input, outputs=head)
new_vgg16 = model
new_vgg16.summary()

new_vgg16.compile(loss='categorical_crossentropy',
                  optimizer= Adam(lr=0.0001, decay=1e-5),
                  metrics=['accuracy'])

gc.collect()

early_stopping_monitor = EarlyStopping(
    monitor='val_acc',
    min_delta=0,
    patience=4,
    verbose=0,
```

[

]

```
    mode='max',
    baseline=None,
    restore_best_weights=True
)

batch_size = 16
with tf.device('/gpu:0'):
    new_vgg16.fit(image_train, label_train, \
                  validation_data=(image_validate, label_validate), \
                  epochs=10, callbacks=[early_stopping_monitor], batch_size=batch_size, verbose = 0)

test_loss, test_score = new_vgg16.evaluate(image_test, label_test, batch_size=16)
print("Loss on test set: ", test_loss)
print("Accuracy on test set: ", test_score)

# Get predictions
preds = new_vgg16.predict(image_test, batch_size=16)
preds = np.argmax(preds, axis=-1)

# Original labels
orig_test_labels = np.argmax(label_test, axis=-1)

print(orig_test_labels.shape)
print(preds.shape)

# Get the confusion matrix
cm  = confusion_matrix(orig_test_labels, preds)
plt.figure()
plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True,cmap=plt.cm.Blues)
plt.xticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'], fontsize=16)
plt.yticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'], fontsize=16)
plt.show()

nn, nb, nv, nc, bn, bb, bv,bc, vn, vb, vv,vc,cn,cb,cv,cc = cm.ravel()

#precision
normal_precision = nn/(nn+bn+vn+cn)
bacterial_precision = bb/(nb+bb+vb+cb)
viral_precision = vv/(nv+bv+vv+cv)
covid_precision = cc/(nc+bc+vc+cc)
print("Precision of the model when dealing with normal pneumonia is {:.2f}".format(normal_precision))
print("Precision of the model when dealing with bacterial pneumonia is {:.2f}".format(bacterial_precision))
print("Precision of the model when dealing with viral pneumonia is {:.2f}".format(viral_precision))
print("Precision of the model when dealing with covid pneumonia is {:.2f}".format(covid_precision))

normal_recall = nn/(nn+nb+nv+nc)
```

[

]

```
bacterial_recall = bb/(bn+bb+bv+bc)
viral_recall = vv/(vn+vb+vv+vc)
covid_recall = cc/(cn+cb+cv+cc)
print("Recall of the model when dealing with normal pneumonia is {:.2f}".format(normal_recall))
print("Recall of the model when dealing with bacterial pneumonia is {:.2f}".format(bacterial_recall))
print("Recall of the model when dealing with viral pneumonia is {:.2f}".format(viral_recall))
print("Recall of the model when dealing with covid pneumonia is {:.2f}".format(covid_recall))

#EfficientNet

!pip install efficientnet

import efficientnet.keras as efn
from tensorflow.keras.callbacks import Callback
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.metrics import Recall, Precision

# Model
## Define the base model with EfficientNet weights
model = efn.EfficientNetB0(weights = 'imagenet',
                           include_top = False,
                           input_shape = (224, 224, 3))

## Output layer
x = model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation="relu")(x)
x = Dense(32, activation="relu")(x)
predictions = Dense(4, activation="relu")(x)

gc.collect()

model = Model(inputs=model.input, outputs=predictions)

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
batch_size = 16
with tf.device('/gpu:0'):
    model.fit(image_train, label_train, \
                    validation_data=(image_validate, label_validate),
                        \
                    epochs=10, callbacks=[early_stopping_monitor],
                        batch_size=batch_size, verbose = 0)

test_loss, test_score = model.evaluate(image_test, label_test,
    batch_size=16)
print("Loss on test set: ", test_loss)
print("Accuracy on test set: ", test_score)
```

[

]

```
270
271 # Get predictions
272 preds = new_vgg16.predict(image_test, batch_size=16)
273 preds = np.argmax(preds, axis=-1)
274
275 # Original labels
276 orig_test_labels = np.argmax(label_test, axis=-1)
277
278 print(orig_test_labels.shape)
279 print(preds.shape)
280
281 # Get the confusion matrix
282 cm = confusion_matrix(orig_test_labels, preds)
283 plt.figure()
284 plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True,cmap=plt.cm.
    Blues)
285 plt.xticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
286 plt.yticks(range(4), ['Normal', 'Bacterial', 'Viral','Covid'],
    fontsize=16)
287 plt.show()
288
289 nn, nb, nv, nc, bn, bb, bv,bc, vn, vb, vv,vc,cn,cb,cv,cc = cm.ravel()
290
291 #precision
292 normal_precision = nn/(nn+bn+vn+cn)
293 bacterial_precision = bb/(nb+bb+vb+cb)
294 viral_precision = vv/(nv+bv+vv+cv)
295 covid_precision = cc/(nc+bc+vc+cc)
296 print("Precision of the model when dealing with normal pneumonia is
    {:.2f}".format(normal_precision))
297 print("Precision of the model when dealing with bacterial pneumonia
    is {:.2f}".format(bacterial_precision))
298 print("Precision of the model when dealing with viral pneumonia is
    {:.2f}".format(viral_precision))
299 print("Precision of the model when dealing with covid pneumonia is
    {:.2f}".format(covid_precision))
300
301 normal_recall = nn/(nn+nb+nv+nc)
302 bacterial_recall = bb/(bn+bb+bv+bc)
303 viral_recall = vv/(vn+vb+vv+vc)
304 covid_recall = cc/(cn+cb+cv+cc)
305 print("Recall of the model when dealing with normal pneumonia is {:.2
    f}".format(normal_recall))
306 print("Recall of the model when dealing with bacterial pneumonia is
    {:.2f}".format(bacterial_recall))
307 print("Recall of the model when dealing with viral pneumonia is {:.2f
    }".format(viral_recall))
308 print("Recall of the model when dealing with covid pneumonia is {:.2f
    }".format(covid_recall))
309
310 #End
```

[

]

CHAPTER 10

REFERENCES

1. Ashitosh Tilve, Shrameet Nayak, Saurabh Vernekar, et al., Pneumonia Detection Using Deep Learning Approaches International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020.

2. P Naveen, B Diwan, Pre-trained VGG-16 with CNN Architecture to classify X-Rays images into Normal or Pneumonia, International Conference on Emerging Smart Computing and Informatics (ESCI), 2021.

3. Dimpy Varshni, Kartik Thakral, Lucky Agarwal, et al., Pneumonia Detection Using CNN based Feature Extraction, IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2019.

4. Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning, Published Online, 2017.

5. Shubhangi Khobragade, Aditya Tiwari, C.Y. Patil,et al., Automatic Detection of Major Lung Diseases Using Chest Radiographs and Classification by Feed-forward Artificial Neural Network, 1st IEEE International Conference on Power Electronics. Intelligent Control and Energy Systems (ICPEICES), 2016.

6. Mohammad Farukh Hashmi, SatyarthKatiyar, Avinash G Keskar, et al., Efficient Pneumonia Detection in Chest X-ray Images Using Deep Transfer Learning, Published Online, 2020.

7. Sammy V. Militante, Nanette V. Dionisio, Brandon G. Sibbaluca, Pneumonia Detection through Adaptive Deep Learning Models of Convolutional Neural Networks, 11th IEEE Control and System Graduate Research Colloquium (ICSGRC), 2020.

8. Özlem Polat, Detection of Pediatric Pneumonia from X-Ray Images using ResNet50 and GAL Networks, International Conference on Innovations in Intelligent Systems and Applications (INISTA), 2021.

9. Tawsifur Rahman, Muhammad E. H. Chowdhury, AmithKhandakar, et al., Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection using Chest X-ray, Published Online, Appl. Sci., 2020.

10. Sheikh Md Hanif Hossain, S M Raju, Amelia Ritahani Ismail, Predicting Pneumonia and Region Detection from X-Ray Images using Deep Neural Network, Published Online, 2021.

11. Abdullahi Umar Ibrahimi, Mehmet Ozsozi, Sertan Serte, et al., Pneumonia Classification Using Deep Learning from Chest X-ray Images During COVID-19, Published on Springer Nature, 2021.

12. Devansh Srivastav, Akansha Bajpai, Prakash Srivastava, Improved Classification for Pneumonia Detection using Transfer Learning with GAN based Synthetic Image Augmentation, 11th International Conference on Cloud Computing, Data Science Engineering (Confluence), 2021.

13. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

14. Karen Simonyan Andrew Zisserman (Visual Geometry Group, Department of Engineering Science, University of Oxford), Very Deep Convolutional Networks for Large-Scale Image Recognition, International Conference on Learning Representations, 2015.

15. Gao Huang, Zhuang Liu, Laurens van der Maaten, et al., Densely Connected Convolutional Networks, published in the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).