A detailed project report on the

Face Mask Detection using CNN

Submitted for the partial fulfilments for the requirement of paper in B.Tech. 5th Semester

BEC 508: Design Project Phase – I

Submitted By:

Devanand Yadav
Enroll. No. : U1951044
B.Tech. CSE 5th Sem.

Under the supervision of

Dr. Mahendra Tiwari
(Assistant Professor)

Department of Electronics and Communication Faculty of Science, University of Allahabad Prayagraj – 211 002 (INDIA)

Department of Electronics and Communication
(JK Institute of Applied Physics and Technology)
University of Allahabad, Prayagraj – 211 002

Declaration

This is to certify that the project work entitled **Face Mask Detection using CNN** is a bonafide work carried out by me bearing University Enrolment Number **U1951044** a student of B.Tech. Computer Science & Engineering (*CSE)*, 5th Semester in the Department of Electronics and Communication, University of Allahabad, Prayagraj (INDIA), under the esteemed supervision of **Dr. Mahendra Tiwari (Assistant Professor)**.

I declare that the work presented here is carried out by me and has not been submitted anywhere else for the award of any degree/certificates.

Signature
(*Devanand Yadav*)

Work presented in this report has been done under my supervision.

Signature
(*Dr. Mahendra Tiwari*)

ACKNOWLEDGMENT

I would like to express my deep and genuine gratefulness to my supervisor, **Dr. Mahendra Tiwari**, Thank you for allowing me to do the project report and providing invaluable guidance throughout this project report. His supervision and support in this project have a great role in the completion of this project on time.

I would also like to thank all those Machine Learning practitioners whose reports/research papers and implementation ideas helped me to think about Machine Learning and implementing the CNN model from another perspective.

ABSTRACT

Facial recognition has been widely used for security and other law enforcement purposes. However, since the COVID-19 pandemic, many people around the world had to wear face masks. This thesis introduces a neural network system, which can be trained to identify people's facial features while half of their faces are covered by face masks. The Convolutional Neural Network (CNN) model using the transfer learning technique has achieved remarkable accuracy even the original dataset is very limited. One large Face mask detection dataset was first used to train the model, while the original much smaller Face mask detector dataset was used to adapt and finetune this model that was previously generated. During the training and testing phases, network structures, and various parameters were adjusted to achieve the best accuracy results for the actual small dataset. Our adapted model was able to achieve a 98.5% accuracy with achievement in validation of 94.3% in 16 epochs.

Keywords: Artificial Intelligence (AL); Machine learning (ML); Deep neural learning (DL); Convolutional Neural Network Model (CNN); Artificial Neural Networks (ANN)

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 Introduction to Project

The field of Artificial Intelligence (AI) research has advanced significantly in recent years, especially in the area of machine learning. Any newly developed technology is inseparable from the term AI. Without AI it is very difficult nowadays to make any significant progress in terms of technical innovation. AI is being considered as the next big thing that will change the world tremendously. The use of neuroscience in marketing research is a new field that offers tremendous promise, and how neurosciences require discipline that discipline requires great experience and costly technology, which is generally not found in marketing research companies.

This project is based on Machine learning using Convolutional Neural Network (CNN) and by MaxPooling each layer for the most perfect area of images for better training. This uses Sequential Learning in which losses are measured by **categorical crossentropy** method and optimizer is **adam.**

1.1.1 AI, ML, and DL

AI (Artificial Intelligence)

AI is a broad field that includes ML and DL. AI is the theory and development of computer systems able to perform tasks normally requiring human intelligence. Another definition AI is a branch of computer science with the simulation of intelligent behaviors in computers. Artificial intelligence is a science like mathematics or biology. It studies ways to build intelligent programs and machines that can creatively solve problems, which has always been considered a human prerogative.

ML (Machine Learning)

Machine Learning is the study of algorithms and computer models used by machines in order to perform a given task. ML is a subset of AI that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. In ML, there are different algorithms (e.g. neural networks) that help to solve problems.
In order of learning process of the machine, there are three components:

- Dataset
- Features
- ML Algorithms
  - Supervised Learning
  - Unsupervised Learning
  - Semi-supervised Learning
  - Reinforcement Learning

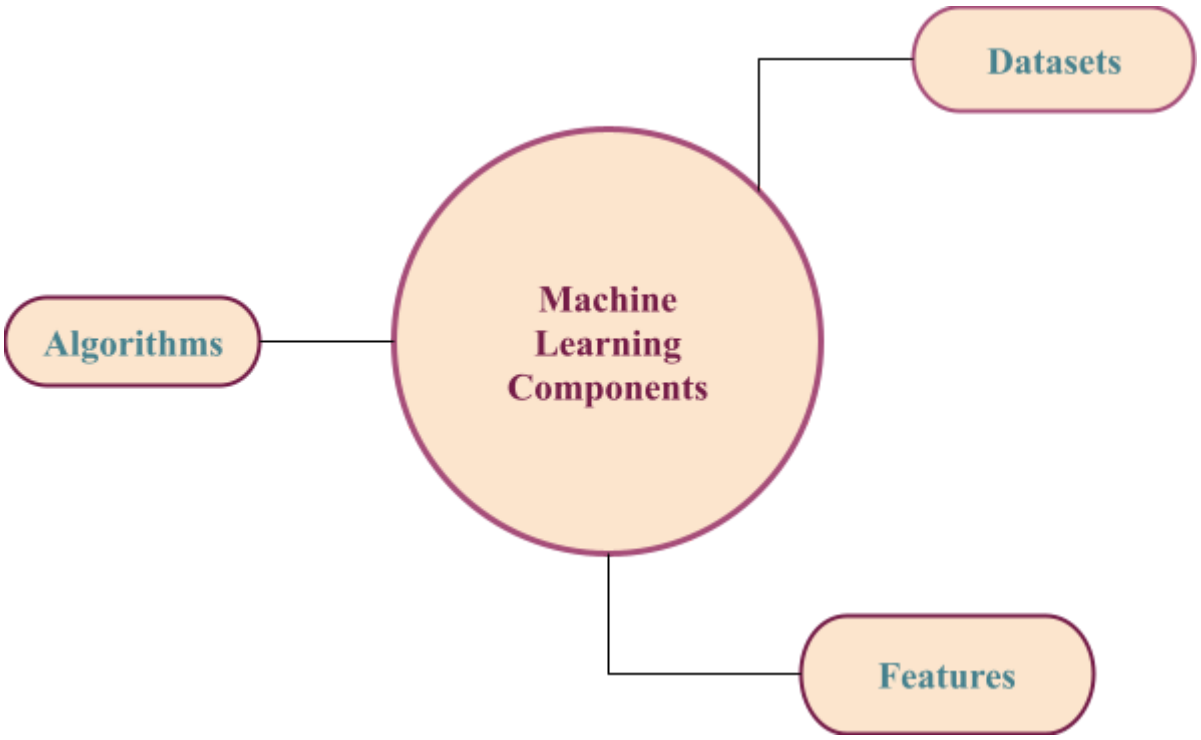


Figure: Flow chart shows components of ML

DL (Deep Learning)

Deep Learning is the subset of ML, which uses the neural networks to analyze different factors with a structure that is similar to the human neural system. Deep learning is a class of machine learning algorithms inspired by the structure of the human brain. Deep learning algorithms use complex multi-layered neural networks, where the level of abstraction increases gradually by non-linear transformations of input data.

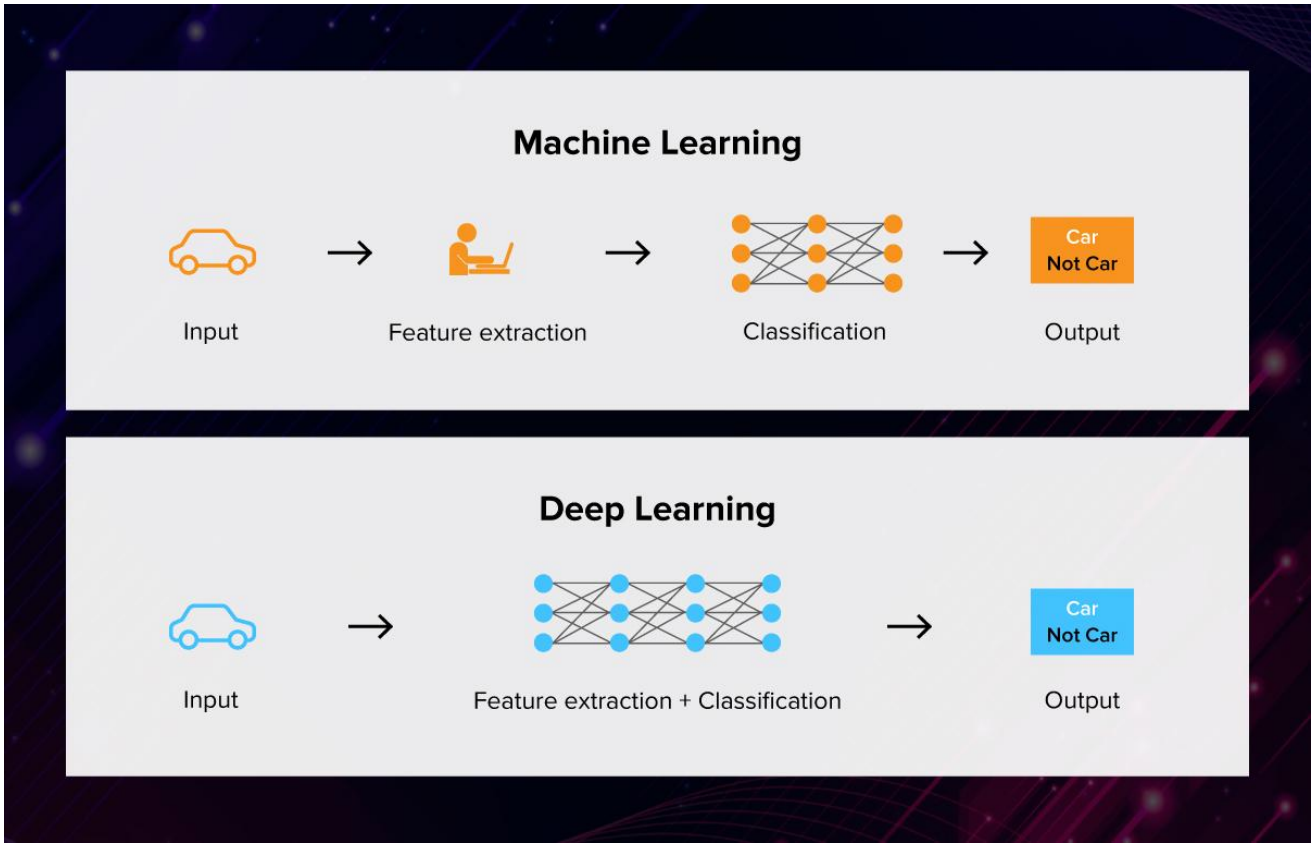
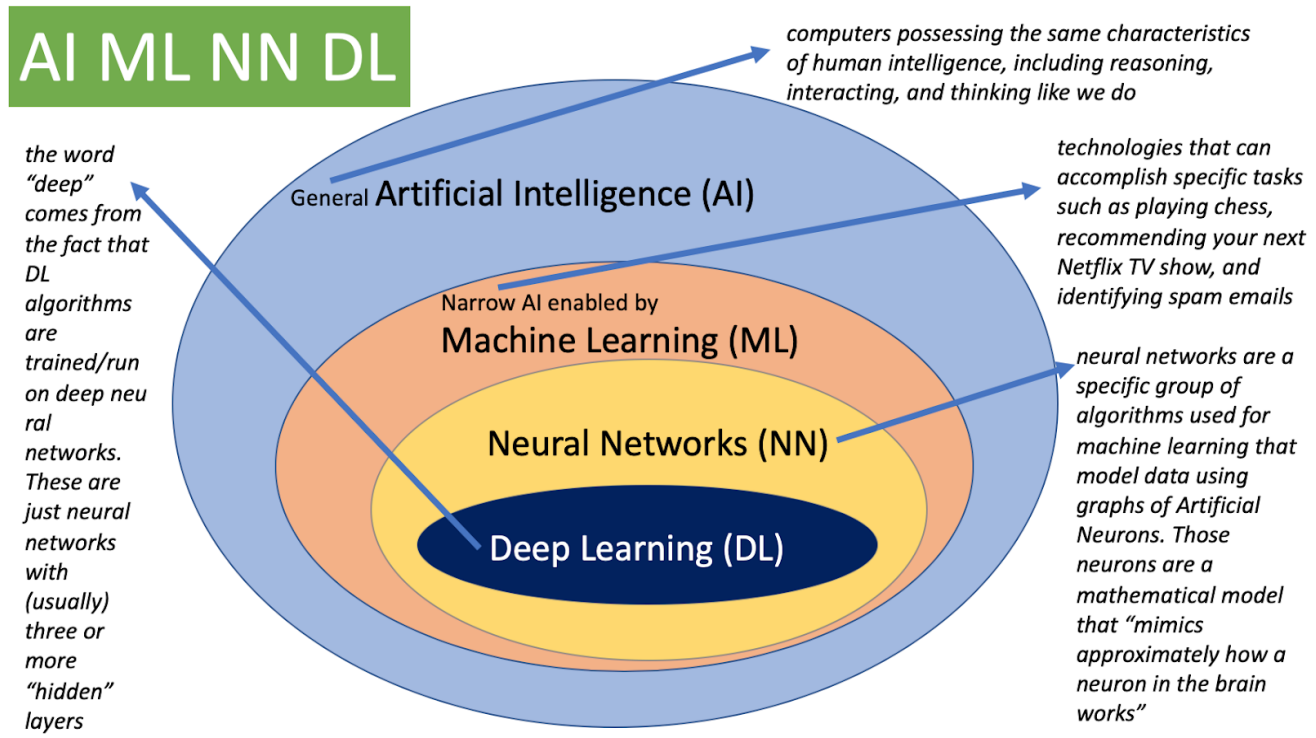
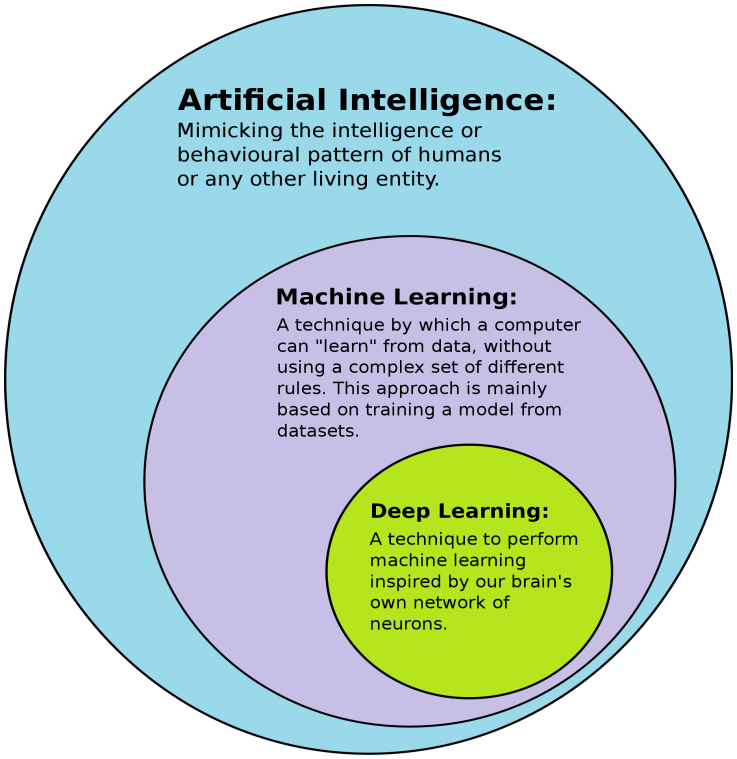


Figure: Machine learning and Deep learning

Relationship Between AI, ML, and DL





Venn's Diagram of AI, ML, and DL

1.2 Objectives

This is for detection of Face Mask using Machine learning techniques using Sequential Learning with the help of CNN (Convolutional Neural Network). In which it detects the person is putting a mask or not on his mouth.

1.3 2. Literature Survey

In The followed part, we briefly present the existing related works on the classification and tracking of face masks. There are several approaches are used for facial masks detection. For instance, used electromagnetic and radiometry techniques for facial masks detection. employed deep neural networks (ANN) using machine learning techniques in Facial Masks detection. Also, the comparison was made between ELM ANN and BP ANN based on performance measurements. Neural Networks are used to exact information from ultrasound to classify the abnormal lesions. An island-based model for classification of face mask and distinguishing between various classes of face feature detection using artificial neural network. That artificial neural network to detect the abnormality masks lesions based on edge. characteristics, shape, and darkness of a lesion. An ultrasound imaging system in order to reduce the dependency of the operator. Linear Discriminant Analysis to classify the informal face mask feature detection using texture and morph metric parameters. presented a paper on face detection segmentation by using genetic algorithm and ANFIS classifier for locating face feature detection and made a comparative analysis between various classifiers.

R. Bhuiyan, S. Khushbu, S. Islam (2020) have published one paper in which the proposed system aims for recognizing the masked and faces are rendered using the advanced YOLOv3 architecture. YOLO (You Only Look Once), uses the learning algorithm Convolution Neural Network (CNN). YOLO establishes a connection with CNN through hidden layers, through research, easy algorithm retrieval, and can detect and locate any type of image. Execution begins by taking 30 unique images of the dataset into the model after combining the results to derive action-level predictions. It gives excellent imaging results and also good detection results. This model is applied to a live video to check the fps rate of the model inside the video and its detection performance with masked/unmasked two layers. Inside the video, our model has impressive outputs with an average fps of 17. This system is more efficient and faster than other methods using their own data set.

CHAPTER 2: ANALYSIS

2.1 Methodology

2.1.1 Supervised Learning

Since supervised learning is the best and most common technique of machine learning nowadays, in this paper, supervised learning techniques were used to achieve the best performance results. Supervised machine learning algorithms to learn by example. The term supervised learning comes from the concept of training the dataset. The training dataset always consists of input images, which are also always combined with their proper outputs. During the training process, any patterns in the data which relate to the selected outputs will be examined by using the supervised learning algorithm. After training, a supervised learning algorithm will take in new unidentified inputs and will be able to identify the new inputs with their correct labels based on the preceding training data. The main purpose of a supervised learning model is to presume the correct label for newly presented input data. A supervised learning algorithm has a simple equation that can be seen in Equation 1. In this equation, Y is the predicted output and x is the input value. This function is used mainly to connect input features to their predicted output which is created by the model during the training process.

$$Y = f(x)$$

Equation 1: Supervised Learning Algorithm Equation

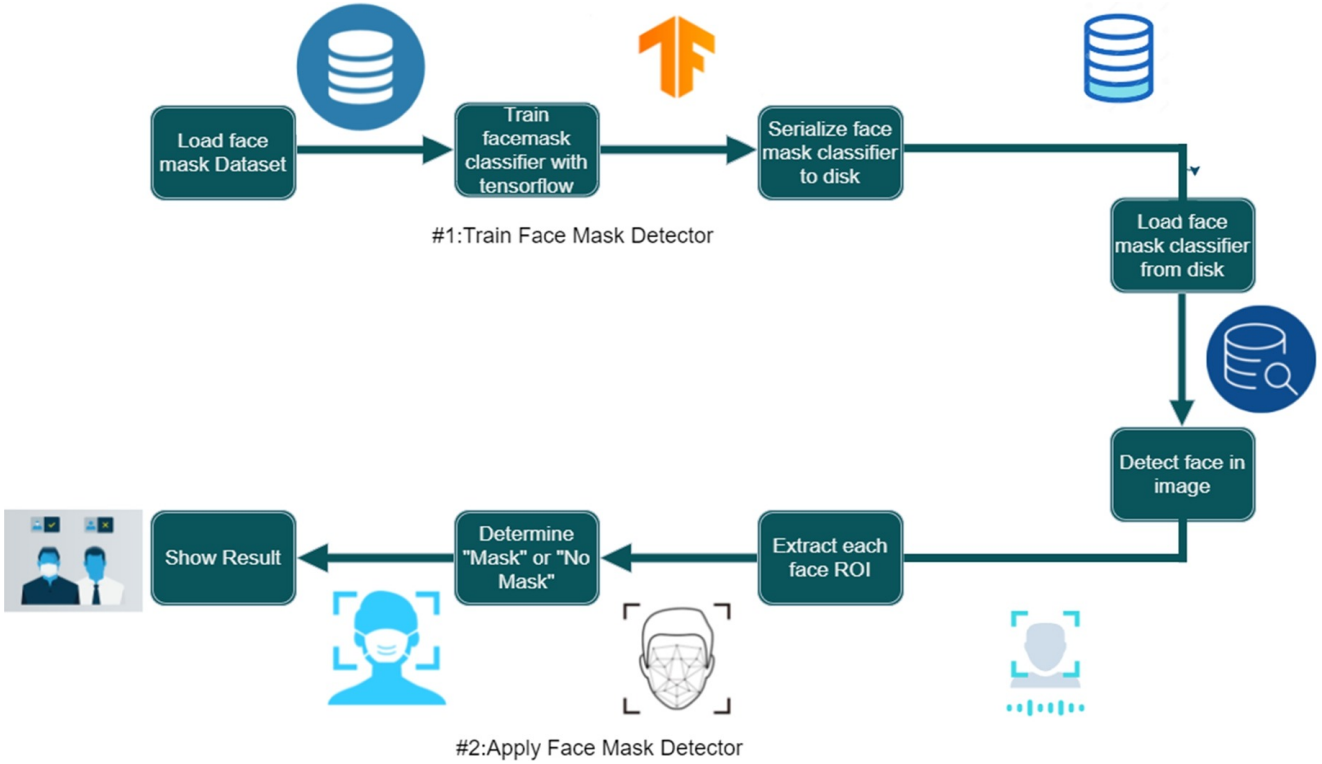


Figure: Process of the flow of data

2.1.2 Convolutional Neural Network (CNN)

The convolutional layer performs as the main building block of the CNN process, which does most of the computing operations. In this layer, the CNN usually detects basic features, such as the shape of edges from the input image. In this project report, three convolutional layers were applied 32 and 64 with alternatively to achieve the best performance of the model. Conservatively, the first layer is responsible for capturing the low-level features, such as color, gradient coordination, edges, and angles. When the architecture goes through another convolutional layer, the output of the first layer becomes the input of the second layer. With more layers being added, the architecture adapts to high-level features, such as a combination of curves and straight edges. Each convolutional layer had two other parameters, which are kernel size and stride. In this experiment, the kernel size was set to 3x3, which helps in detecting differently sized features in the input image which will lead to different sized feature maps. Likewise, the amount of movement between applications of the filter to the input image is referred to as the stride and is almost always balanced in height and width sizes. Thus, in this experiment, the stride dimensions were set to 1. After every convolutional step, ReLU has been used. ReLU stands for Rectified Linear Unit, which is a non-linear operation. ReLU is an element-by-element operation that has to be applied to each pixel. In the feature map, ReLU restores all negative pixel values by zero. The purpose of ReLU is to present non-linearity in the CNN model since most of the real-world data would be non-linear. As the model is building, applying all these operations, and adding more convolutional layers, the architecture goes through activation maps that produce more convoluted features to fully understand the features of human faces

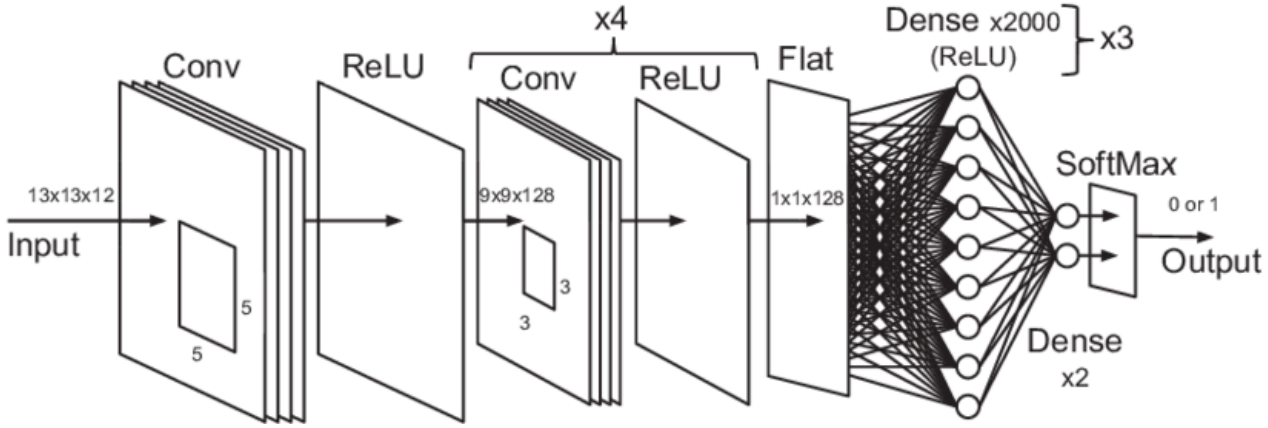
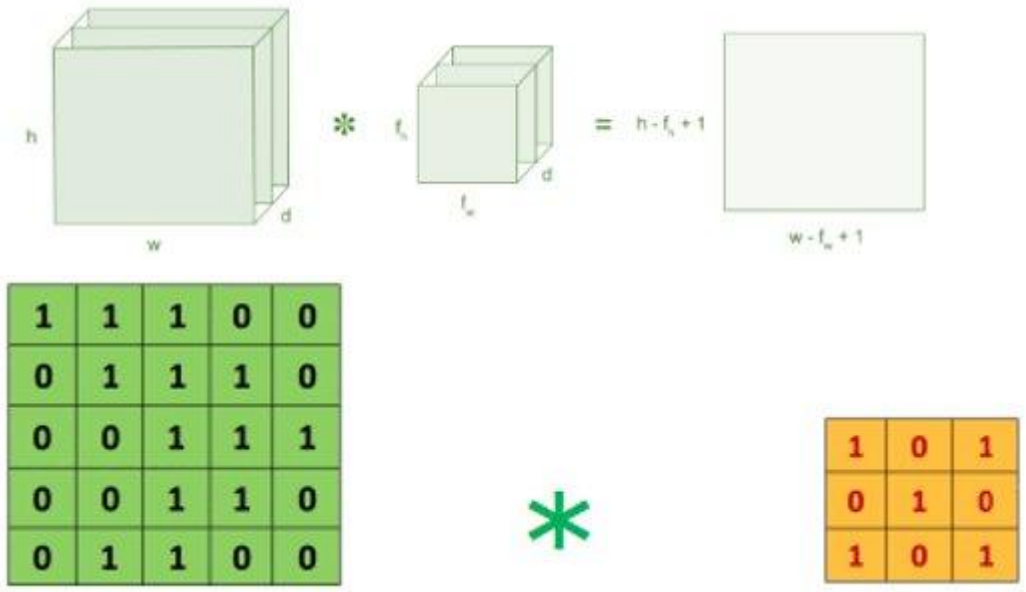


Figure: Example of a CNN

Convolution - Process

- An image matrix (volume) of dimension **(**h **x** w **x** d**)**
- A filter **(**f_h **x** f_w **x** d**)**
- Outputs a volume dimension **(**h **-** f_h **+ 1) x (**w **-** f_w **+1) x 1**



2.1.3 Max Pooling

In this project report, to build the convolutional neural network, 2x2 max-pooling was used to decrease the three-dimensional size of the convoluted feature. 2x2 max-pooling is beneficial for decreasing the computing power necessitated to process the data through dimensionality reduction. Furthermore, it is useful for extracting major features, therefore maintaining the effective process of training the model. Max pooling also returns the highest value from the image's part that was 29 covered by the kernel. Using max pooling can also eliminate noisy activations and reduces dimensionality. The convolutional layer and the pooling layer together help the model to understand image features.

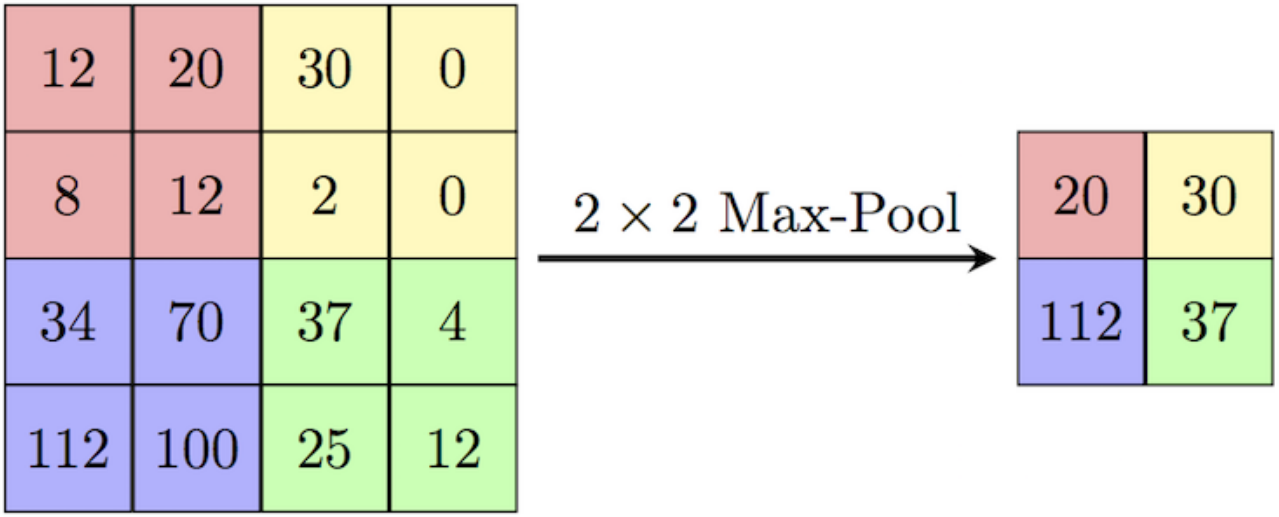


Figure: shows an example of how 2x2 max-pooling is performed

6.2.4 Fully Connected Layer

In this methodology, another two fully connected layers were added, which helped in learning a non-linear mixture of the high-level face features which were obtainable during the process of the kernel. The **SoftMax activation** function was used after the fully connected layers. SoftMax function works by transferring a vector of numbers into a vector of probabilities. SoftMax results in normalizing the outcomes as well as converting those outcomes into probabilities that must add up to 1.0. Since input images were converted into a multi-layer neural network, flattening an image should then occur. The flattened output is then fed to a feed-forward neural network and backpropagation is applied to every step of the training process. The main purpose of using a fully connected layer is to use high-level features for classifying the input image into either people wearing a mask or people not wearing a mask.

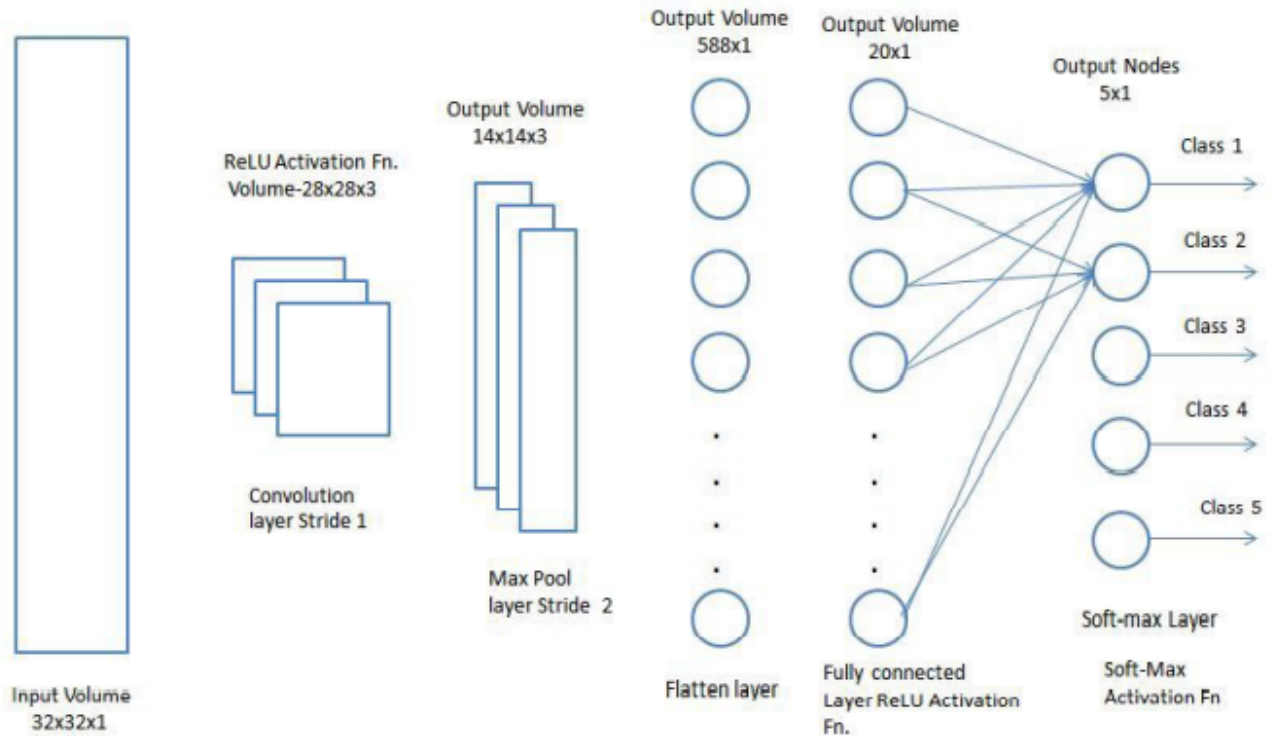


Figure: Fully Connected Layer Architecture

Flattening

Flattening is a process of converting a 2-D matrix into a 1-D Column matrix.

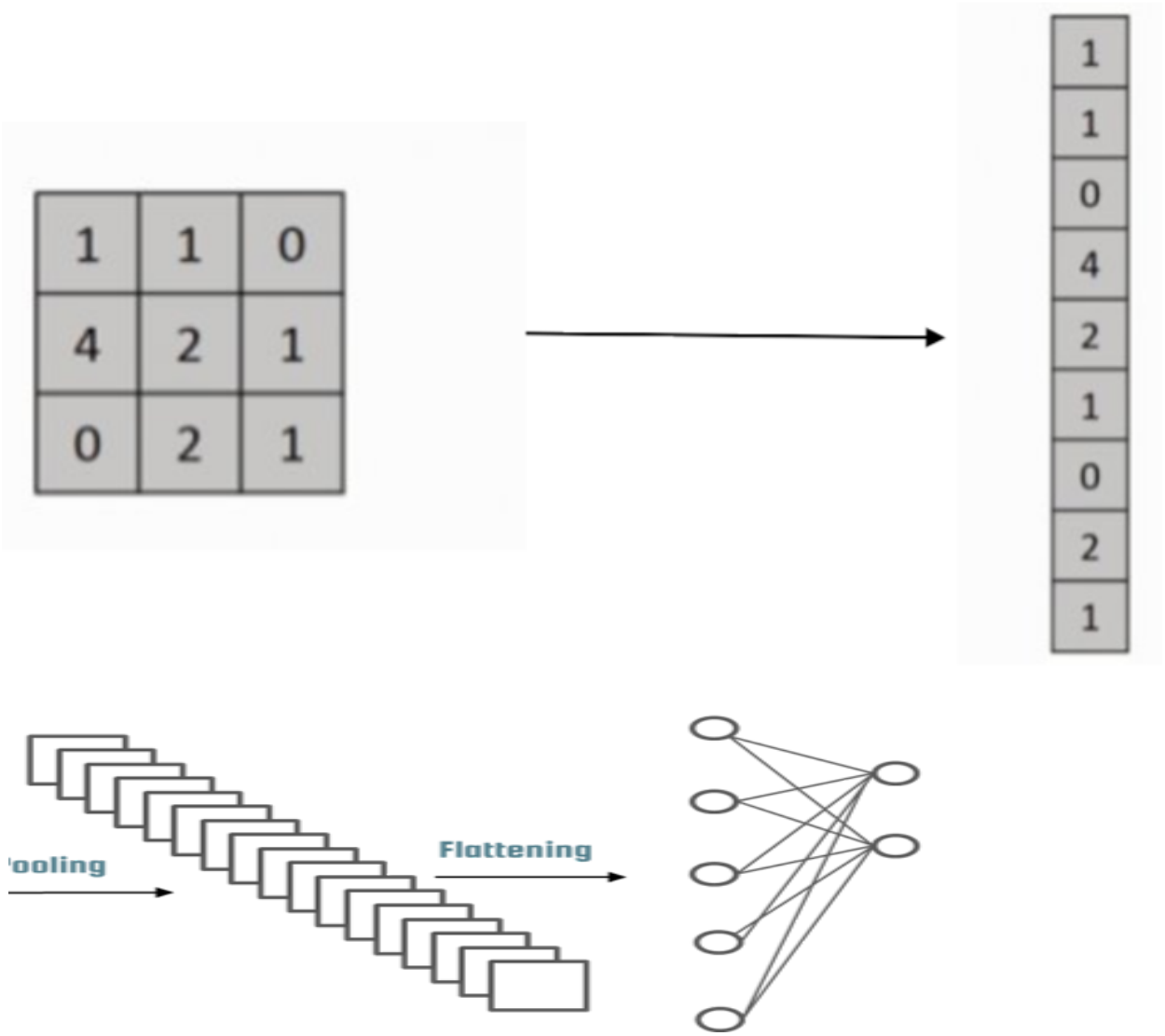


Figure: Flattening of Convoluted layer

2.2 Training

The training process is also known as backpropagation. The training step is very important for the model to find the weights that best accurately represent the input data to match its correct output class. Thus, these weights are constantly updated and moved towards their optimal output class. In this study, a Face Mask Detection Dataset was used to train the CNN model.

2.2.1 Adam Optimizer

During the training process of this paper, Adam optimizer has been applied to the CNN model. Adam is an adaptive learning rate optimization algorithm that has been proposed especially for training deep neural networks. Adam optimizer works by calculating each learning rate according to different parameters within the model.

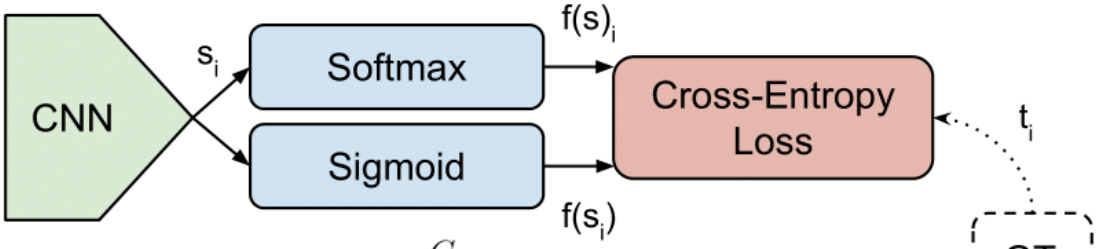
2.2.2 Categorical Crossentropy Loss

Categorical crossentropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one. Formally, it is designed to quantify the difference between two probability distributions.

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

where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value, and outsize is the the number of scalar values in the model output.

This loss is a very good measure of how distinguishable two *discrete probability distributions* are from each other. In this context, y_i is the probability that event i occurs and the sum of all y_i is 1, meaning that exactly one event may occur.
The minus sign ensures that the loss gets smaller when the distributions get closer to each other.



$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Categorical Crossentropy Loss

2.2.3 Running Epochs

An epoch refers to each cycle that a model takes through the full training dataset. For example, by feeding the neural network with training data for more than one epoch, the result should become better in terms of predicting the given unseen data, which is the test data. In this approach, 20 epochs were applied to achieve the best accuracy results of predicting the test data. Table 1 shows the impact of epochs on the loss percentage.

Epochs	Percentage of Loss	Percentage of Accuracy	Percentage of Val_accuracy

1	55.26	70.70	40.10
5	14.32	94.27	90.53
10	5.14	98.18	92.29
15	3.07	98.71	92.95
20	3.36	98.77	93.83

Table 1: Epoch number, loss percentage, percentage of accuracy and validation accuracy

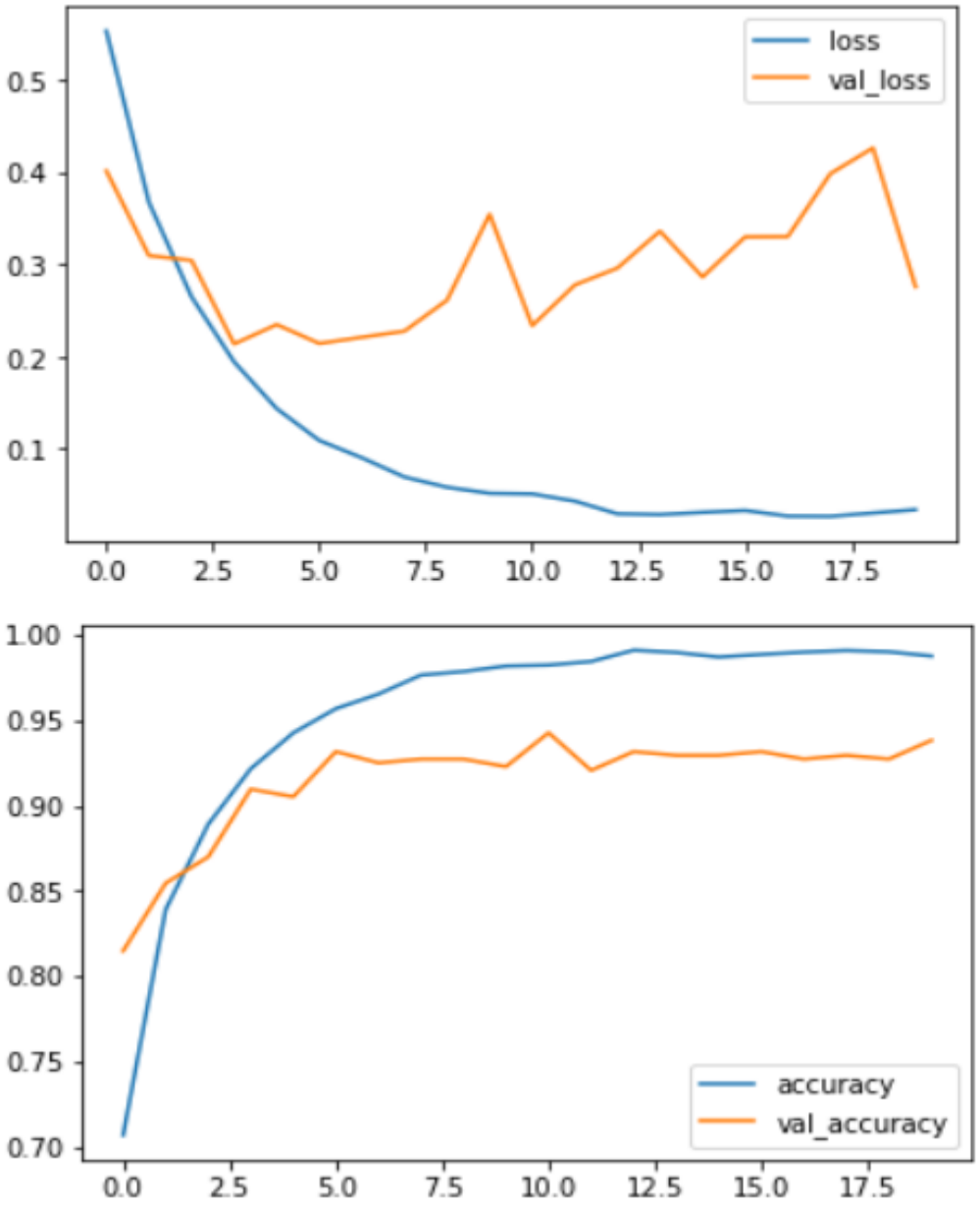


Figure: Graph of loss, val_loss vs epochs and accuracy, val_accuracy vs epochs respectively

CHAPTER 3: DESIGN

3.1 Dataset

Datasets of with mask and without mask are taken from Kaggle in which about 3700 with mask images and 3800 without mask images.

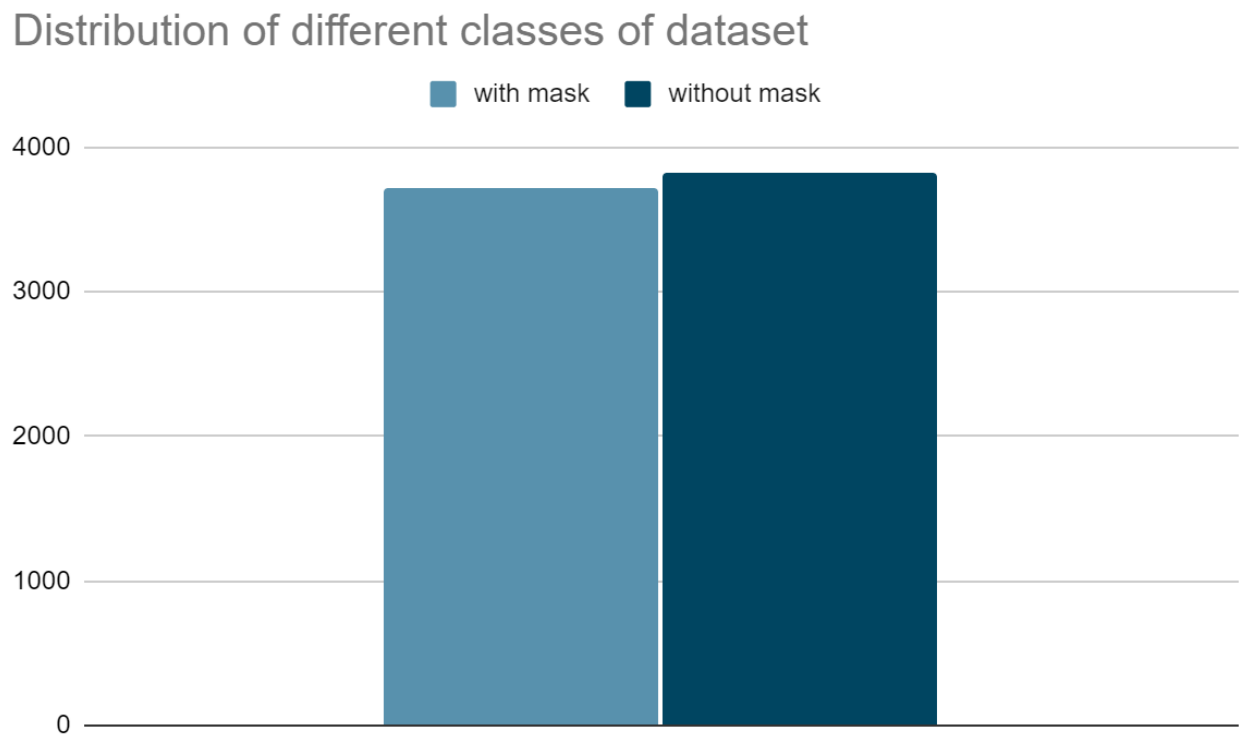


Figure : Classes Distribution of Face Mask Detection Dataset
 # with mask : 3725 images
 # without mask : 3828 images

3.1.1 Dataset Pre-processing Image Datagenerator

In machine learning, performing data pre-processing is a very significant step that helps in improving the quality of data to help the extraction of important understandings from the data. In data pre-processing, the data is getting cleaned and organized to become proper for building and 25 training a model. For both datasets, images were converted to grayscale instead of color channels. The main purpose of using grayscale is to simplify the algorithm as well as reducing computational requirements to help in the extracting descriptors process.

```
image_dataset_from_directory(
        directory,
        labels="inferred",
```

```
    label_mode="int",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(244, 244),
        shuffle=True,
        seed=None,
    validation_split=None,
        subset=None,
        interpolation="bilinear",
    follow_links=False,
)
```

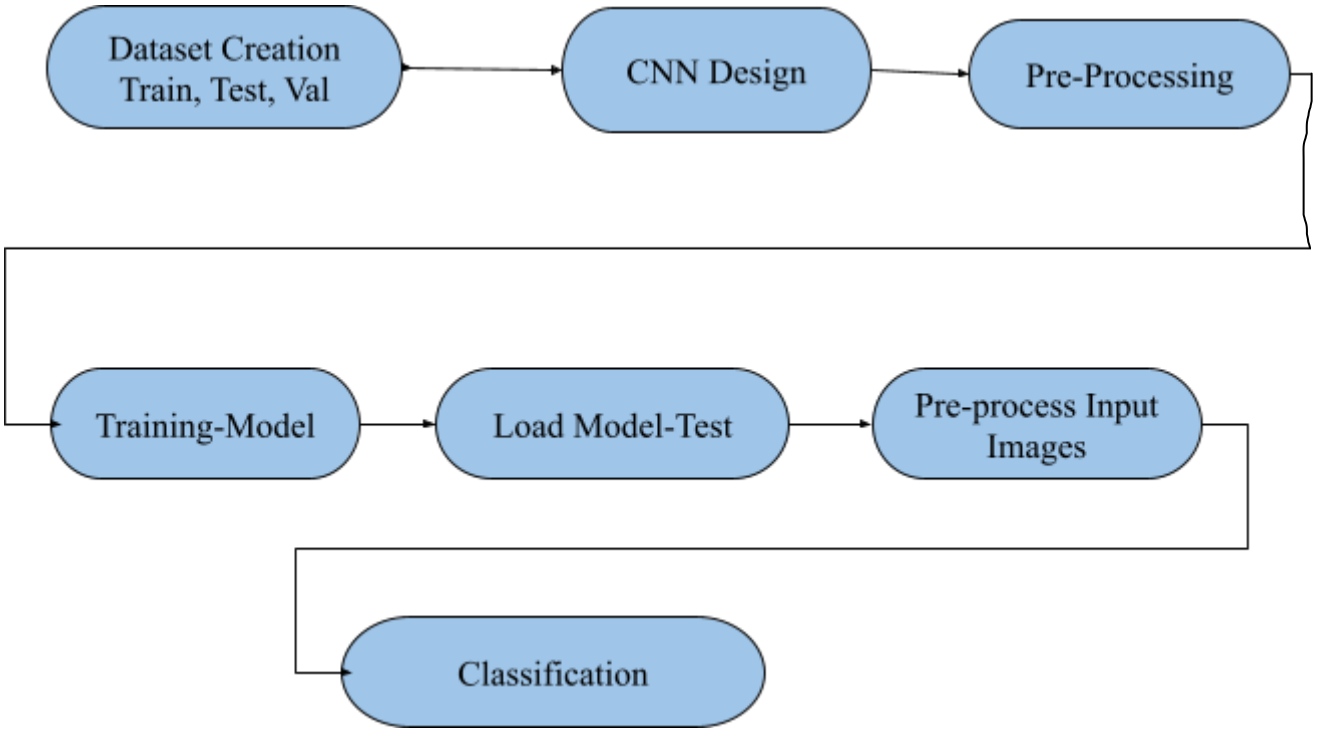
```
image_gen = ImageDataGenerator(preprocessing_function= tf.keras.applications.mobilenet_v2.preprocess_input)

train = image_gen.flow_from_dataframe(dataframe= train_set,x_col="filepaths",y_col="labels",
                                      target_size=(244,244),
                                      color_mode='grayscale',
                                      class_mode="categorical", #used for Sequential Model
                                      batch_size=32,
                                      shuffle=True            #for the shuffle data
                                     )

test = image_gen.flow_from_dataframe(dataframe= test_set,x_col="filepaths", y_col="labels",
                                     target_size=(244,244),
                                     color_mode='grayscale',
                                     class_mode="categorical",
                                     batch_size=32,
                                     shuffle=True
                                    )

val = image_gen.flow_from_dataframe(dataframe= val_set,x_col="filepaths", y_col="labels",
                                    target_size=(244,244),
                                    color_mode= 'grayscale',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=True
                                   )
```

```
Found 5287 validated image filenames belonging to 2 classes.
Found 1812 validated image filenames belonging to 2 classes.
Found 454 validated image filenames belonging to 2 classes.
```

3.2 Image Classification

Image classification is a Supervised Learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos.

3.2.1 Block Diagram - Workflow of Image Classification CNN



3.3 Algorithm for Face Mask Detection

Algorithm For Face Mask Detection

Input: Dataset including faces with and without masks
Output: Categorized image depicting the presence of face mask
for each image in dataset do

- Visualize the image in two categories and label them
- Convert the RGB image to Gray-scale image using OpenCV
- Resize the gray-scale into 200x200
- Normalize the image and convert it into a 4-dimensional array using NumPy

end
for building the CNN model do

- Add a Convolution layer of 200 filters using Keras
- Add the 2nd Convolutional layer of 100 filters
- Insert a Fatten layer to the network classifier as keras.Flatten
- Add a Dense layer of 64 neurons as keras.dense
- Add the final Dense layer with 2 outputs for 2 categories

End

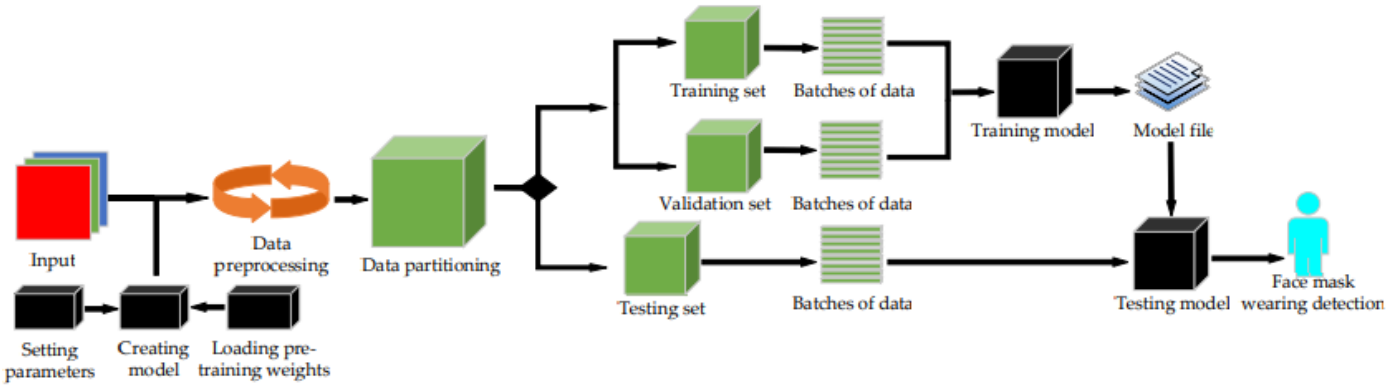


Figure: Flow chart of the Proposed model of Face Mask Detection

3.4 Result

This project model have achieved an accuracy of 98.77%, validation accuracy of 93.83%, and with loss of 3.36% in 20 epochs using Supervised Learning in Deep Learning with Convolutional Neural Network (CNN).

The results are more of what was expected of the model. The mask recognition is implemented using the camera as a medium and shows accurate results. When the person's face is in the camera frame, the model will detect the face, and a green or a red frame will appear over the face. A person who is not wearing a mask will get a red frame over his face in-camera while the person who is wearing a mask will get a red frame. The result is also visible written on the top left of the resulting frame. A percentage match can also be seen on the top of the resulting frame. The model works even if the side view of the face is visible to the camera. It can also detect more than one face in a single camera frame. Overall, the model shows accurate results.

Figure: *Graph of Training Loss and Accuracy vs Epochs number*

CHAPTER 4: CODING AND IMPLEMENTATION

4.1 Conding and Implementation

Training and Validating the model

Importing importing libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
```

```
import warnings
warnings.filterwarnings(action="ignore")
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# importing sklearn libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# importing tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import MaxPooling2D, Dense, Dropout,Flatten, Conv2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import TensorBoard,EarlyStopping
```

Downloading datasets from Kaggle

```
with_mask_dir=r'../input/face-mask-dataset/data/with_mask'
without_mask_dir=r'../input/face-mask-dataset/data/without_mask'
filepaths = []
labels= []
dict_list = [with_mask_dir, without_mask_dir]
for i, j in enumerate(dict_list):
    flist=os.listdir(j)
    for f in flist:
        fpath=os.path.join(j,f)
        filepaths.append(fpath)
        if i==0:
          labels.append('with_mask')
        else:
          labels.append('without_mask')

Fseries = pd.Series(filepaths, name="filepaths")
Lseries = pd.Series(labels, name="labels")
mask_data = pd.concat([Fseries,Lseries], axis=1)
mask_df = pd.DataFrame(mask_data)
print(mask_df.head())
print(mask_df["labels"].value_counts())
```

```
                                           filepaths     labels
0  ../input/face-mask-dataset/data/with_mask/with...  with_mask
1  ../input/face-mask-dataset/data/with_mask/with...  with_mask
2  ../input/face-mask-dataset/data/with_mask/with...  with_mask
3  ../input/face-mask-dataset/data/with_mask/with...  with_mask
4  ../input/face-mask-dataset/data/with_mask/with...  with_mask
without_mask    3828
with_mask       3725
Name: labels, dtype: int64
```

```
# shape of dataset
mask_df.shape
```

```
(7553, 2)
```

Generation of Images for training, validation, and testing using sklearn library module train_test_split

```
train_set, test_images = train_test_split(mask_df, test_size=0.3, random_state=42)
test_set, val_set = train_test_split(test_images, test_size=0.2, random_state=42)
```

Generate batches of tensor image data with real-time data augmentation by using ImageDatagenerator from keras.preprocessing.image library.

```
image_gen = ImageDataGenerator(preprocessing_function= tf.keras.applications.mobilenet_v2.preprocess_input)

train = image_gen.flow_from_dataframe(dataframe= train_set,x_col="filepaths",y_col="labels",
                                      target_size=(244,244),
                                      color_mode='grayscale',
                                      class_mode="categorical", #used for Sequential Model
                                      batch_size=32,
                                      shuffle=True            #for the shuffle data
                                     )

test = image_gen.flow_from_dataframe(dataframe= test_set,x_col="filepaths", y_col="labels",
                                     target_size=(244,244),
                                     color_mode='grayscale',
                                     class_mode="categorical",
                                     batch_size=32,
                                     shuffle=True
                                    )

val = image_gen.flow_from_dataframe(dataframe= val_set,x_col="filepaths", y_col="labels",
                                    target_size=(244,244),
                                    color_mode= 'grayscale',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=True
                                   )
```

```
Found 5287 validated image filenames belonging to 2 classes.
Found 1812 validated image filenames belonging to 2 classes.
Found 454 validated image filenames belonging to 2 classes.
```

```
classes=list(train.class_indices.keys())
for datas in classes:
    print(datas)
```

```
with_mask
without_mask
```

Function to show images

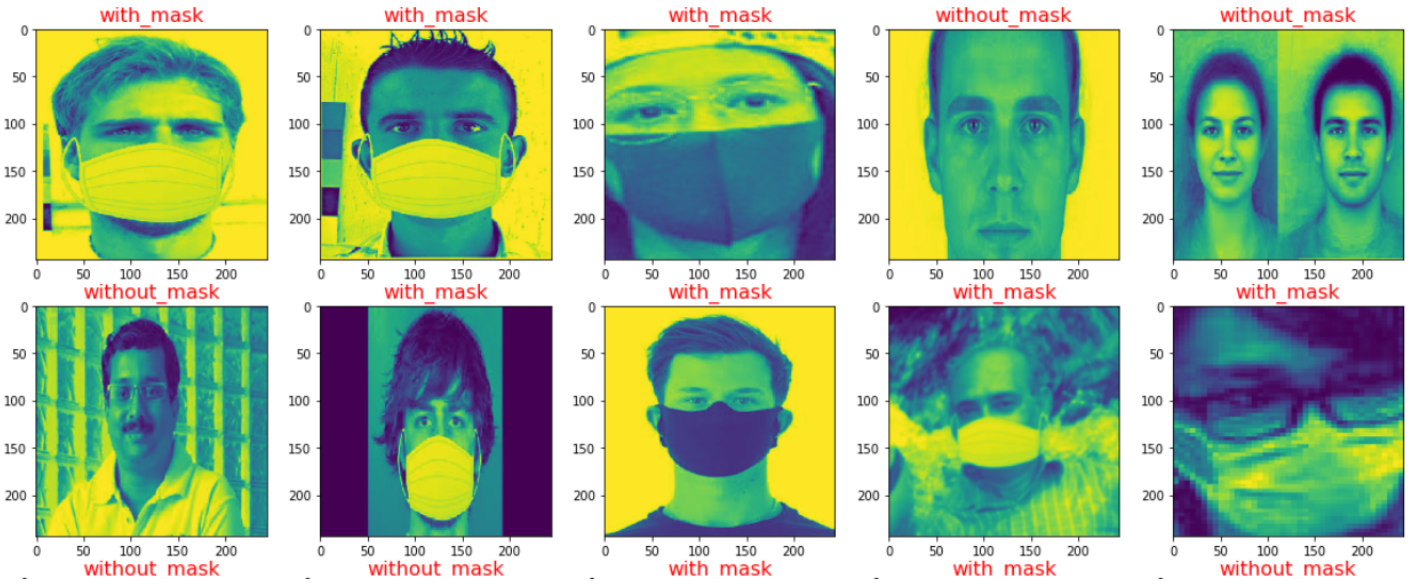
```
# funciton to show images
def show_images(image_gen):
    test_dict = test.class_indices
    classes = list(test_dict.keys())
    images, labels=next(image_gen) # get a sample batch from the generator
    plt.figure(figsize=(20,20))
    length = len(labels)
    if length<25:
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5,5,i+1)
        image=(images[i]+1)/2 #scale images between 0 and 1
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color="red",fontsize=16)
        plt.axis('on')
    plt.show()
```

Training Images and Validation Images

Testing images

```
print("Training Images...\n")
show_images(train)
```

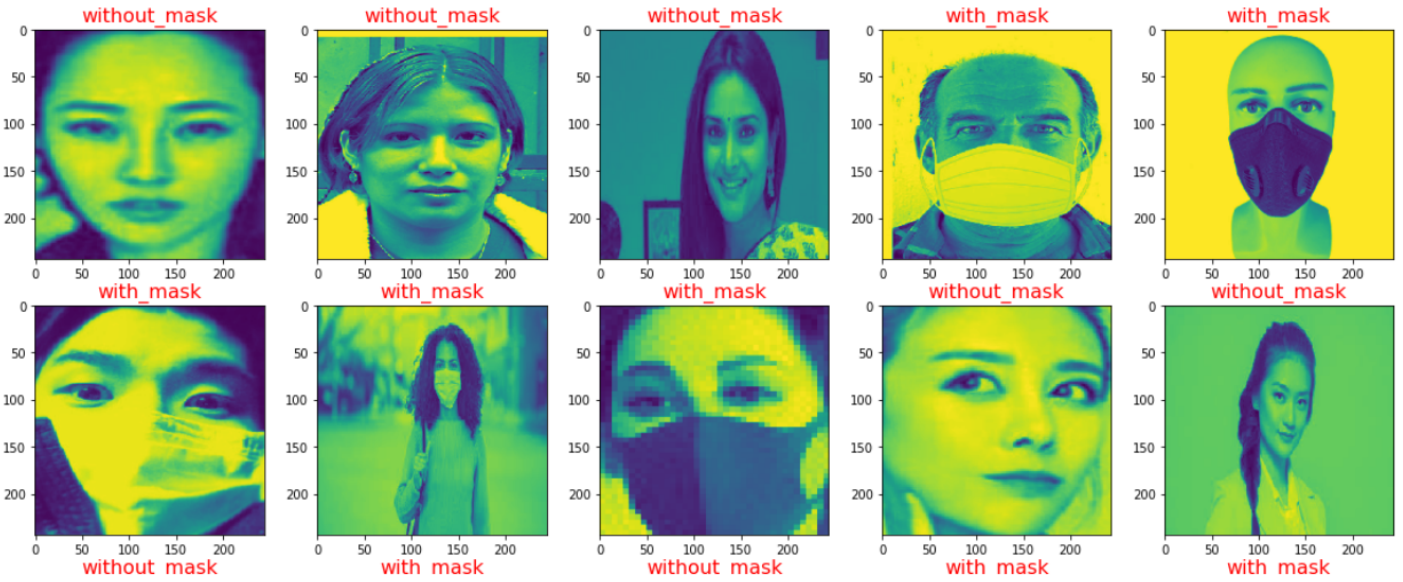
```
Training Images...
```



Validation Images

```
print("Validation Images...\n")
show_images(val)
```

```
Validation Images...
```



This function plot the loss and accuracy vs epochs of the model by the training and validating data generated

```
def plot_loss_and_accuracy(history):
    history_df = pd.DataFrame(history)
    history_df.loc[0:, ['loss', 'val_loss']].plot()
    history_df.loc[0:, ['accuracy', 'val_accuracy']].plot()
```

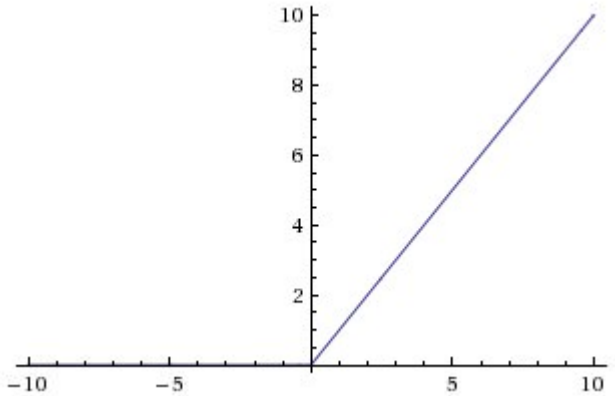
Model of CNN

Prepare a CNN model and MaxPooling each layer using Sequential Learning model with the help of Keras.

```
import keras
from tensorflow.keras import layers

model_CNN_2 = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3,3), activation='relu', padding='same', input_shape=(244,244,1)),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(2, activation='softmax')
])

model_CNN_2.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model_CNN_2.optimizer.lr=0.001

model_CNN_2.summary()
Model: "sequential"
```

ReLU Function

Dense - Fully connected layer,
Units - Number of nodes present in a hidden layer
Activation Function - rectifier linear activation function (ReLU)



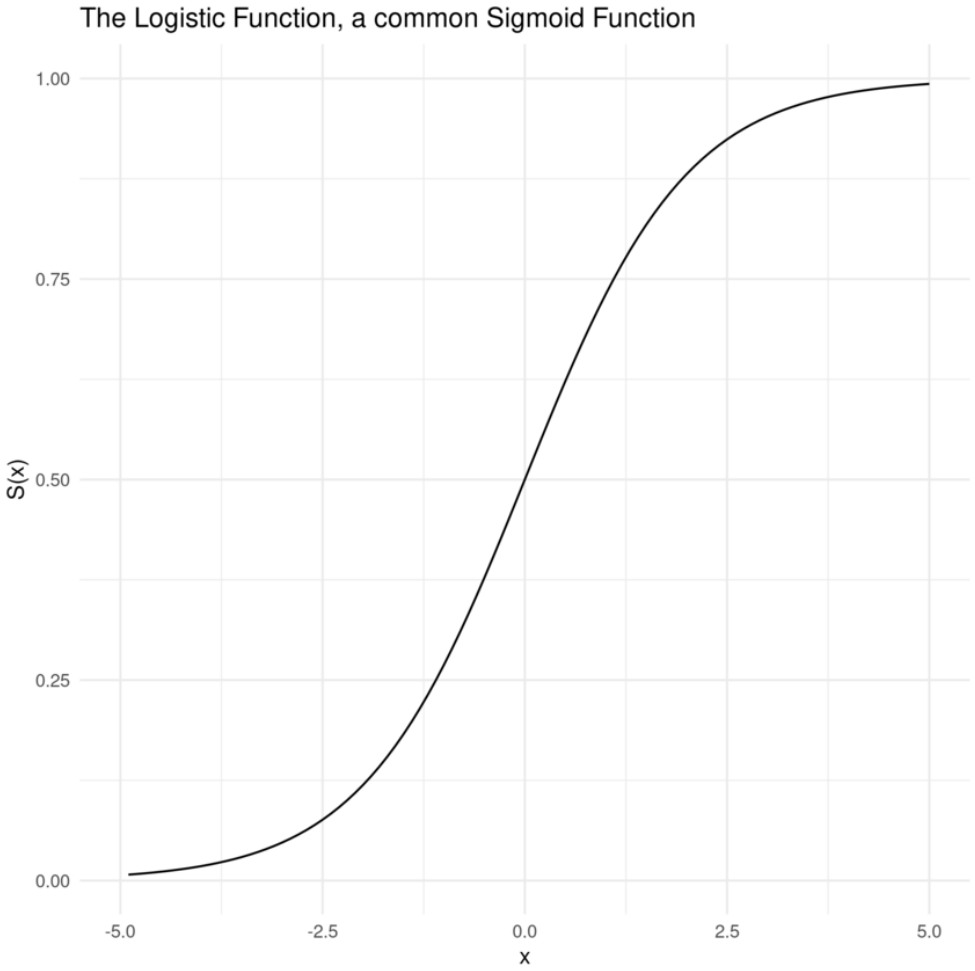
The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value, x it returns that value back. So it can be written as $f(x) = max(0, x)$. It's surprising that such a simple function (and one composed of two linear pieces) can allow your model to account for non-linearities and interactions so well. But the ReLU function works great in most applications, and it is very widely used as a result.

Sigmoid Function

A Sigmoid function is a mathematical function that has a characteristic S-shaped curve. There are a number of common sigmoid functions, such as the logistic function, the hyperbolic tangent, and the arctangent.

In Machine Learning Sigmoid function is normally referred to as the Logistic function,

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$



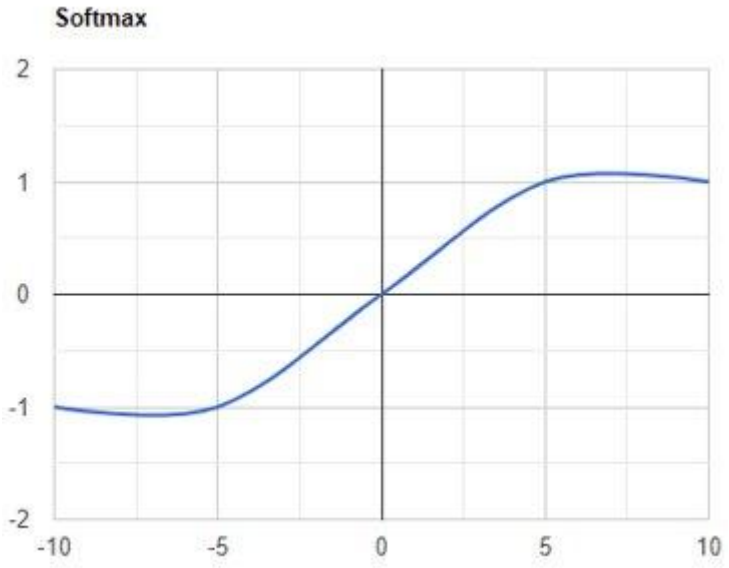
All sigmoid functions have the property that they map the entire number line into a small range such as between 0 and 1, or -1 and 1, so one use of a sigmoid function is to convert a real value into one that can be interpreted as a probability.
Activation function -Sigmoid, gives binary output 0 or 1.

Softmax Function

- Softmax activation function will be applied in the last layer of the Neural network, instead of ReLU, tanh, Sigmoid.

- It is used to map the non-normalized output of a network to a probability distribution over the predicted output class. That is it converts outputs of the last layer into an essential probability distribution.

$$softmax(L_n) = e^{L_n} / \left|\left| e^{L} \right|\right|$$



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 244, 244, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 122, 122, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 120, 120, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 60, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 58, 58, 32)        18464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 29, 29, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 29, 29, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 26912)             0
_________________________________________________________________
dense (Dense)                (None, 64)                1722432
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 1,759,842
Trainable params: 1,759,842
Non-trainable params: 0
_________________________________________________________________
```

Figure : Summary of the CNN model

Train the model

```
history_CNN = model_CNN_2.fit(train, validation_data= val, epochs=20,verbose=1)
```

```
Epoch 1/20
166/166 [==============================] - 163s 976ms/step - loss: 0.5526 - accuracy: 0.7070 - val_loss: 0.4010 - val_accuracy: 0.8150
Epoch 2/20
166/166 [==============================] - 157s 948ms/step - loss: 0.3668 - accuracy: 0.8392 - val_loss: 0.3091 - val_accuracy: 0.8546
Epoch 3/20
166/166 [==============================] - 160s 963ms/step - loss: 0.2644 - accuracy: 0.8892 - val_loss: 0.3039 - val_accuracy: 0.8700
Epoch 4/20
166/166 [==============================] - 160s 963ms/step - loss: 0.1938 - accuracy: 0.9217 - val_loss: 0.2133 - val_accuracy: 0.9097
Epoch 5/20
166/166 [==============================] - 162s 974ms/step - loss: 0.1432 - accuracy: 0.9427 - val_loss: 0.2343 - val_accuracy: 0.9053
Epoch 6/20
166/166 [==============================] - 161s 966ms/step - loss: 0.1087 - accuracy: 0.9569 - val_loss: 0.2138 - val_accuracy: 0.9317
Epoch 7/20
166/166 [==============================] - 159s 957ms/step - loss: 0.0899 - accuracy: 0.9654 - val_loss: 0.2206 - val_accuracy: 0.9251
Epoch 8/20
166/166 [==============================] - 160s 962ms/step - loss: 0.0690 - accuracy: 0.9765 - val_loss: 0.2272 - val_accuracy: 0.9273
Epoch 9/20
166/166 [==============================] - 161s 968ms/step - loss: 0.0579 - accuracy: 0.9786 - val_loss: 0.2608 - val_accuracy: 0.9273
Epoch 10/20
166/166 [==============================] - 159s 955ms/step - loss: 0.0514 - accuracy: 0.9818 - val_loss: 0.3538 - val_accuracy: 0.9229
Epoch 11/20
166/166 [==============================] - 157s 943ms/step - loss: 0.0507 - accuracy: 0.9824 - val_loss: 0.2331 - val_accuracy: 0.9427
Epoch 12/20
166/166 [==============================] - 159s 954ms/step - loss: 0.0429 - accuracy: 0.9845 - val_loss: 0.2773 - val_accuracy: 0.9207
Epoch 13/20
166/166 [==============================] - 159s 959ms/step - loss: 0.0291 - accuracy: 0.9911 - val_loss: 0.2956 - val_accuracy: 0.9317
Epoch 14/20
166/166 [==============================] - 160s 965ms/step - loss: 0.0283 - accuracy: 0.9898 - val_loss: 0.3356 - val_accuracy: 0.9295
Epoch 15/20
166/166 [==============================] - 163s 980ms/step - loss: 0.0307 - accuracy: 0.9871 - val_loss: 0.2859 - val_accuracy: 0.9295
Epoch 16/20
166/166 [==============================] - 161s 970ms/step - loss: 0.0327 - accuracy: 0.9887 - val_loss: 0.3294 - val_accuracy: 0.9317
Epoch 17/20
166/166 [==============================] - 157s 948ms/step - loss: 0.0266 - accuracy: 0.9900 - val_loss: 0.3297 - val_accuracy: 0.9273
Epoch 18/20
166/166 [==============================] - 158s 952ms/step - loss: 0.0264 - accuracy: 0.9909 - val_loss: 0.3979 - val_accuracy: 0.9295
Epoch 19/20
166/166 [==============================] - 157s 945ms/step - loss: 0.0300 - accuracy: 0.9902 - val_loss: 0.4258 - val_accuracy: 0.9273
Epoch 20/20
166/166 [==============================] - 159s 956ms/step - loss: 0.0336 - accuracy: 0.9877 - val_loss: 0.2754 - val_accuracy: 0.9383
```

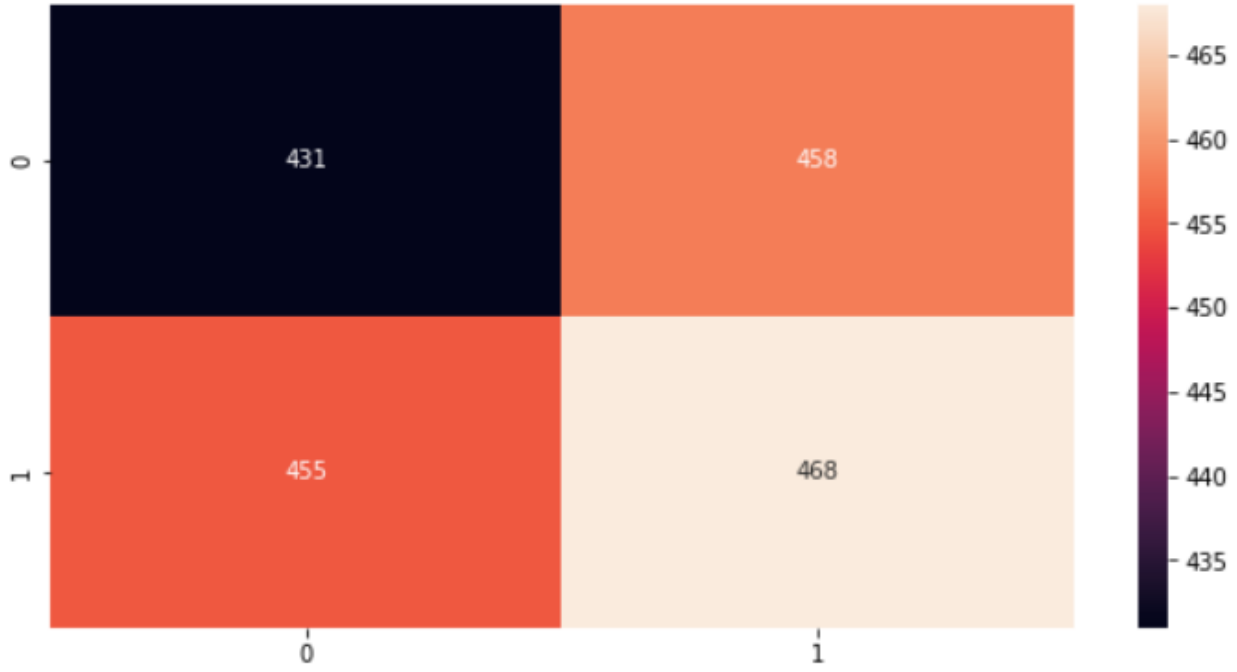
Confusion Matrix

```python
pred = model_CNN_2.predict(test)
pred = np.argmax(pred, axis=1) #pick class with highest  probability

labels = (train.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred2 = [labels[k] for k in pred]

y_test = test_set.labels # set y_test to the expected output
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

   with_mask       0.49      0.48      0.49       889
without_mask       0.51      0.51      0.51       923

    accuracy                           0.50      1812
   macro avg       0.50      0.50      0.50      1812
weighted avg       0.50      0.50      0.50      1812
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
plt.figure(figsize = (10,5))
cm = confusion_matrix(y_test, pred2)
sns.heatmap(cm, annot=True, fmt = 'g')
```

```
<AxesSubplot:>
```



Saving the Model for further use

```
model_json = model_CNN_2.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)
model_CNN_2.save_weights('model.h5')
print('Saved model to disk successfully.')
```

```
Saved model to disk successfully.
```

4.2 Experimental Setup

4.2.1 System And Software setups

- Operating System : Windows 11 pro with intel i7 10th gen. Processors

- IDE: Pycharm, Jupyter-notebook and Jupyter-labs, VSCode, Google Colab, and Kaggle Jupyter-labs
- GPU: Google colab GPU and TPU, AMD Radeon 610 with 2GB Graphic memory in my system
- Software: Anaconda
- Package Manager: pip3 or pip, conda

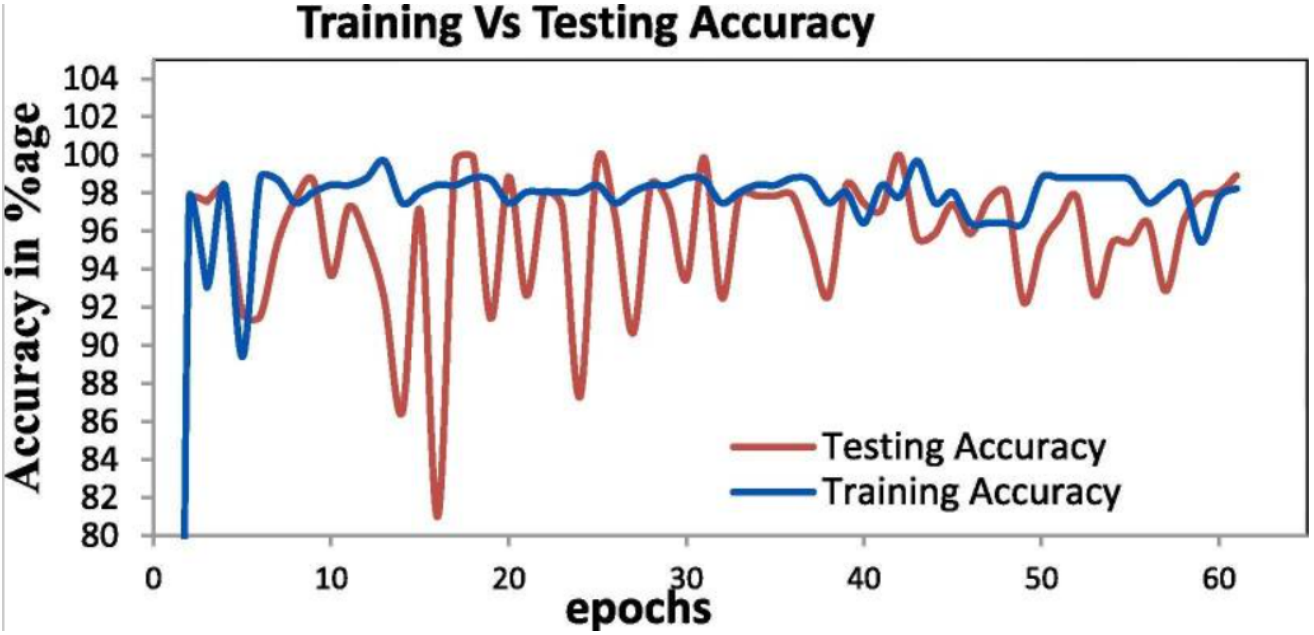
4.2.3 Packages, modules, libraries

- Numpy - to do mathematical work and matrix operations
- Pandas - to data framing
- Scikit Learn - to pre-processing of images
- Matplotlib - to plot graphs and images
- Tensorflow - to implement machine learning model
- Keras - to implement CNN model
- Os - to handle files
- OpenCV - to the real-time image capturing

4.3 Error Handling and Parameter Passing

4.3.1 Controlling Overfitting

To address RQ5 and avoid the problem of overfitting, two major steps are taken. First, we performed data augmentation. Second, the model accuracy is critically observed over 60 epochs both for the training and testing phase. The observations are reported in below figure



It is further observed that model accuracy keeps on increasing in different epochs and get stable after epoch = 3 as depicted graphically in the above figure.

4.3.2 Callback Checkpoints

Initialize a callback checkpoint to keep saving the best model after each epoch while training. It is called before fitting the model for training.

```
from keras.callbacks import TensorBoard, ModelCheckpoint
checkpoint = ModelCheckpoint('model_CNN_2-{epoch:03d}.model',monitor='val_loss',verbose=0,save_best_only=True,mode='auto')
```

CHAPTER 5: TESTING

5.1 Testing the model

In this very last step of testing, sample testing images were passed through the convolutional neural network, and the predicted and actual true classes were compared. The model achieved 91.5% accuracy results after applying 30 epochs. Figure 27 shows the code segment with the best accuracy result.

Testing with the help of images from datasets provided by Kaggle for face mask detection.

```
correct = 0
total = 0
with torch.no_grad():
    for i in tqdm(range(len(test_X))):
        real_class = torch.argmax(test_y[i])
        model_out = net(test_X[i].view(-1, 1, 100, 100))
[0]  # returns a list,
        predicted_class = torch.argmax(model_out)

        if predicted_class == real_class:
            correct += 1
        total += 1
print("Accuracy: ", round(correct/total, 3))
```

```
100%|██████████| 755/755 [00:06<00:00, 111.20it/s]

Accuracy:  0.915
```

Figure : Accuracy result of the trained Model

```
import numpy as np
from keras.models import model_from_json
from keras.preprocessing import image

json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()

model = model_from_json(loaded_model_json)
model.load_weights('model.h5')
print('Loaded model from disk successfully')


def classify(img_file):
    img_name = img_file
    test_image = image.load_img(img_name, target_size=(64, 64))

    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)

    if result[0][0] == 1:
        prediction = 'With Mask'
    else:
        prediction = 'Without Mask'
    #print(prediction, img_name)
```

Testing with help of OpenCV for real-time face mask detection

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
```

```
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)

    # initialize our list of faces, their corresponding locations,
    # and the list of predictions from our face mask network
    faces = []
    locs = []
    preds = []

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the confidence is
        # greater than the minimum confidence
        if confidence > 0.5:
            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = frame[startY:endY, startX:endX]
```

```
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)

            # add the face and bounding boxes to their respective
            # lists
            faces.append(face)
            locs.append((startX, startY, endX, endY))

    # only make a predictions if at least one face was detected
    if len(faces) > 0:
        # for faster inference we'll make batch predictions on *all*
        # faces at the same time rather than one-by-one predictions
        # in the above `for` loop
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    # return a 2-tuple of the face locations and their corresponding
    # locations
    return (locs, preds)

# load our serialized face detector model from disk
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model("mask_detector.model")

# initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
```

```
    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        # include the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        # display the label and bounding box rectangle on the output
        # frame
        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

5.2 Test Reports

In this projrect, a performance accuracy of 98.77% was achieved from using the Deep Supervised Learning technique, including Convolutional Neural Network (CNN) and Transfer Learning. The approach did not just

stop at finding out the accuracy percentage but also printing out the arguments of the maxima with a given data, and results were tested and found accurate as well.

Another result, when I run this project on my system Dell Vostro 3490 with Intel i7 10th generation processor CPU with 8GB RAM and 2GB graphics card of AMD Radeon 610, and get a result of accuracy of 99.34% and with validation accuracy of 99.35%, in which loss of 2.73%, and validation loss of 2.72% in 20 epochs.

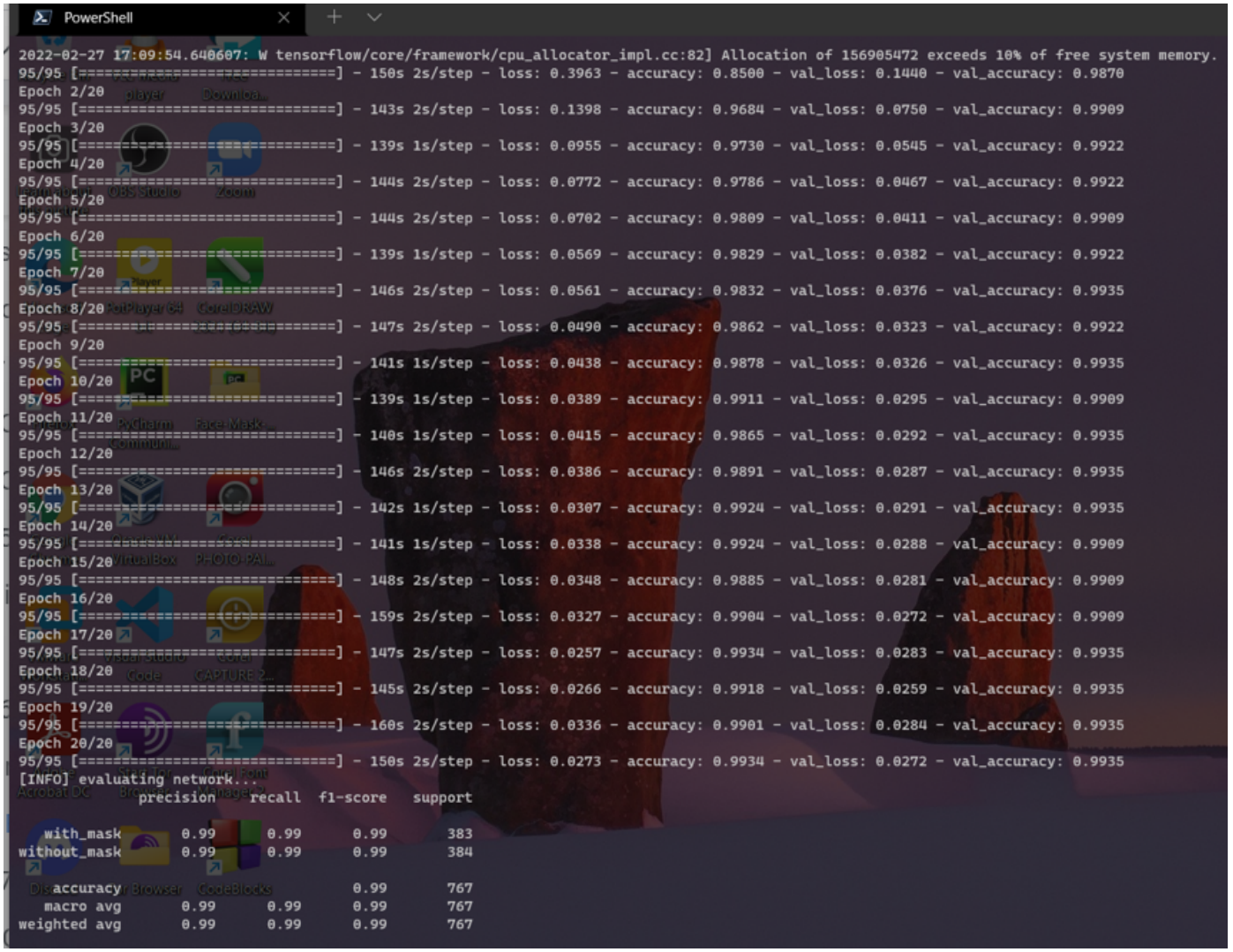
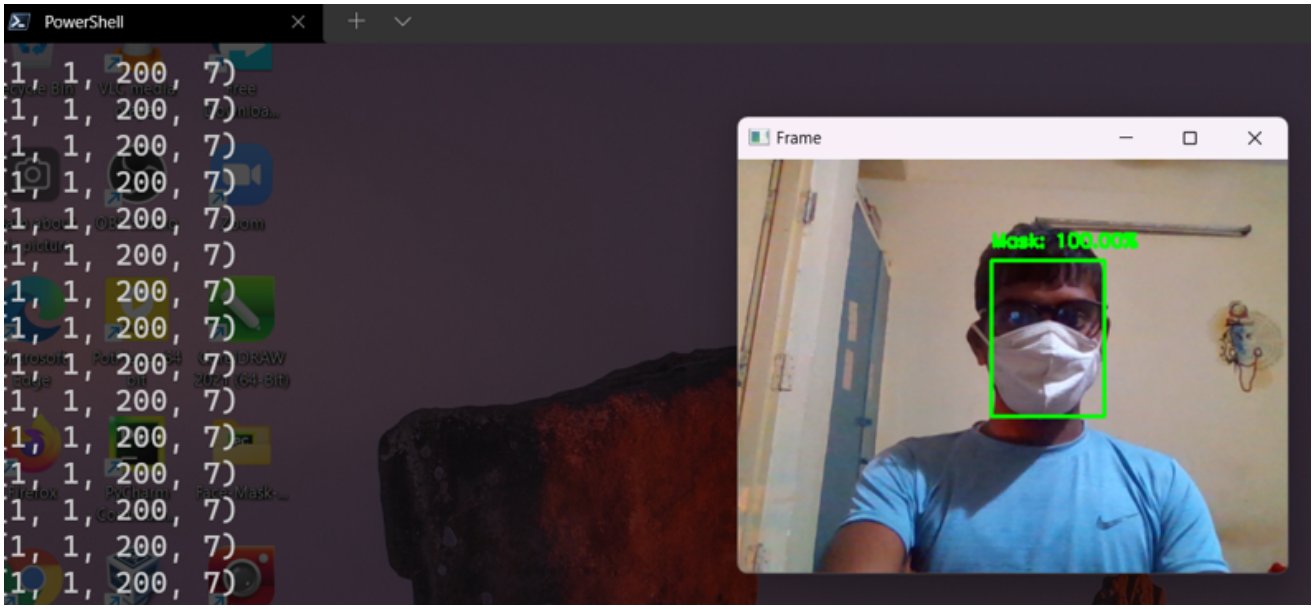


Figure: Results of the Training the model

Below images are the testing images showing detection with or without a mask along with accuracy.


Training Loss and Accuracy
Loss/Accuracy
Epoch #
train_loss
val_loss
train_acc
val_acc
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


PowerShell
1, 1, 200, 7)
Frame

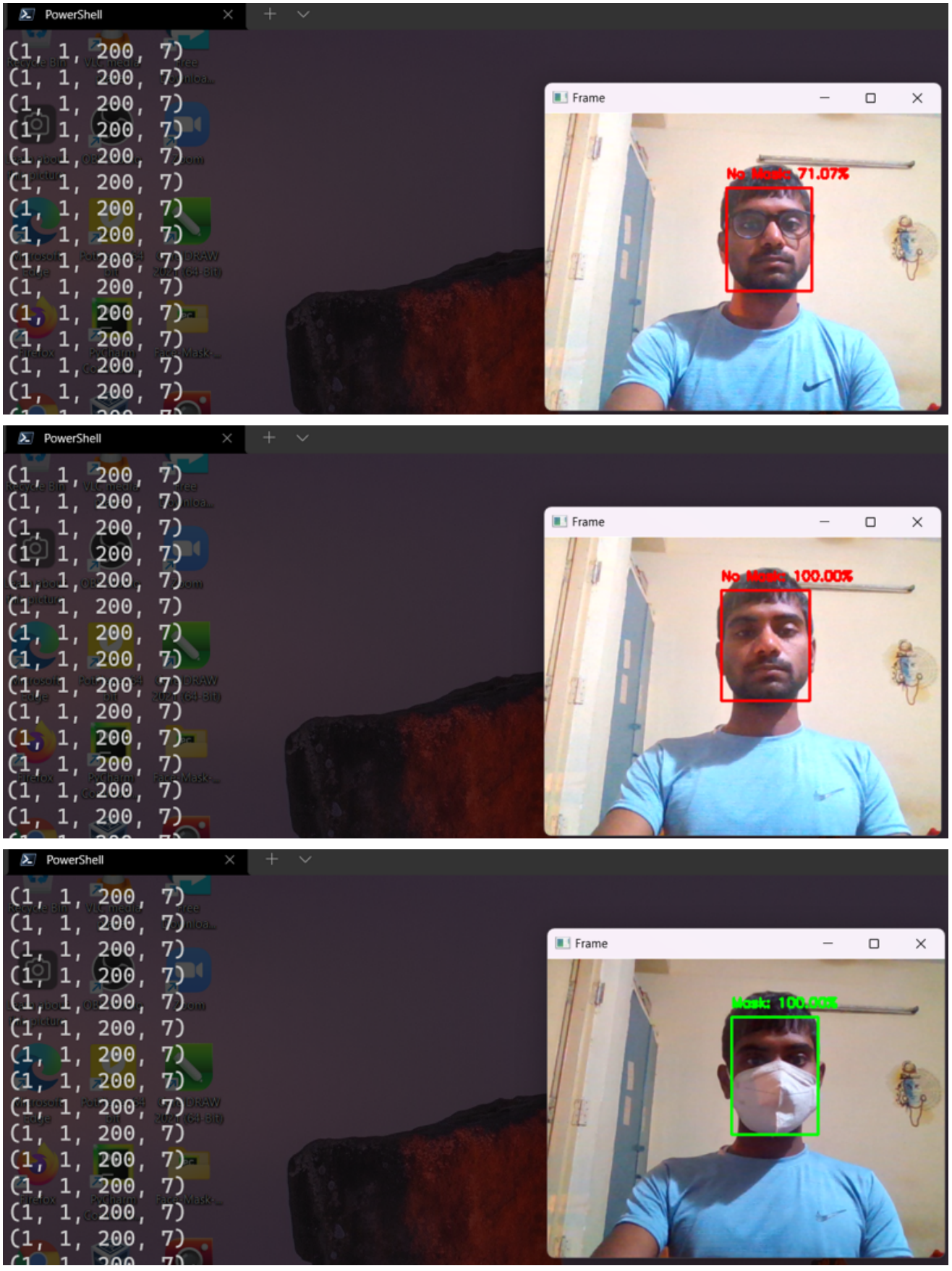


Figure: Above four images shows with mask and without mask along with their accuracy.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE FOR FURTHER DEVELOPMENT

6.1 Conclusion

This project report presented a study on facial recognition and face mask detection through deep learning techniques by building a CNN model using transfer learning and fine-tuning techniques. This process gave accurate and quick results for facial recognition security systems despite the fact that half of the faces are covered with face masks. The test results show a high accuracy rate in identifying individuals wearing a face mask, not wearing a face mask, and wearing a face mask but in an incorrect manner. The model was able to achieve a 98.5% of performance accuracy, and achieve 94.3% of validation accuracy, which is a significant achievement. Moreover, the study presented a useful tool in fighting the spread of the COVID-19 disease by allowing all individuals to wear a face mask while performing biometric authentication. Facial recognition with face masks is becoming more and more important over the past year due to the spread of the COVID-19 virus. Our future works include alarm if somebody is not wearing a face mask properly, and detection of the social distancing

6.2 Future Scope

Finally, the work opens interesting future directions for researchers. Firstly, the proposed technique can be integrated into any high-resolution video surveillance device and is not limited to mask detection only. Secondly, the model can be extended to detect facial landmarks with a facemask for biometric purposes.

REFERENCES

[1] Dr. Adrian Rosebrock, “Deep Learning for Computer Vision with Python”, Practitioner Bundle, PUBLISHED BY PYIMAGESEARCH, 2017, pp. 179-290

[2] X. Lu, A. K. Jain, and D. Colbry. Matching 2.5 d face scans to 3d models. IEEE transactions on pattern analysis and machine intelligence, 28(1):31{43, 2005.

[3] View References (ieee.org)

[4] **Covid-19 Face Mask Detection Using TensorFlow, Keras, and OpenCV | IEEE Conference Publication | IEEE Xplore**

[5] M. Loey, G. Mangogaran, T. M.H.N., and K. N.E.M., "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic," National Library of Medicine, 1 January 2021. [Online]. Available: **https://pubmed.ncbi.nlm.nih.gov/32834324/.** Z. Elhamraoui, "Fine-tuning in Deep Learning," Artificial Intelligence, 23 June 2020. [Online]. Available: **https://ai.plainenglish.io/fine-tuning-in-deep-learning 909666d4c151.**

[6] J. L. Q. Y. a. Z. L. S. Ge, "Detecting Masked Faces in the Wild with LLE-CNNs," IEEE Conference on Computer Vision and Pattern Recognition, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8099536**.**

[7] C. Kanan and G. Cottrell "Color-to-Grayscale: Does the Method Matter in Image Recognition?" PLoS ONE vol. 7 no. 1 pp. e29740 2012.

[8] "Changing Colorspaces — Opencv-Python Tutorials 1 Documentation" 2020 [online] Available: Opencv-python-tutroals.readthedocs.io.

[9] M. Hashemi "Enlarging smaller images before inputting into the convolutional neural network: zero-padding vs. interpolation" Journal of Big Data vol. 6 no. 1 pp. 2019.

[10] S. Ghosh N. Das and M. Nasipuri "Reshaping inputs for the convolutional neural network: Some common and uncommon methods" Pattern Recognition vol. 93 pp. 79-94 2019.

[11] R. Yamashita M. Nishio R. Do and K. Togashi "Convolutional neural networks: an overview and application in radiology" Insights into Imaging vol. 9 no. 4 pp. 611-629 2018.

[12] "Guide to the Sequential model - Keras Documentation" 2020 [online] Available: Faroit.com.

[13] C. Nwankpa W. Ijomah A. Gachagan and S. Marshall "Activation Functions: Comparison Of Trends In Practice And Research For Deep Learning" 2020 [online] Available: https://arxiv.org/abs/1811.03378.2020.

[14] "Keras documentation: MaxPooling2D layer" 2020 [online] Available: Keras.io.

[15] "prajnasb/observations" 2020 [online] Available: https://github.com/prajnasb/observations/tree/master/experiements/data.

[16] https://www.kaggle.com/omkargurav/face-mask-dataset

[17] "TensorFlow White Papers" TensorFlow 2020 [online] Available: https://www.tensorflow.org/about/bib.

[18] "Keras documentation: About Keras" 2020 [online] Available: Keras.io.

[19] "OpenCV" 2020 [online] Available: Opencv.org.

[20] D. Meena and R. Sharan "An approach to face detection and recognition" 2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE) pp. 1-6 2016.

[21] M. Burugupalli, "IMAGE CLASSIFICATION USING TRANSFER LEARNING AND CONVOLUTION NEURAL NETWORKS," July 2020. [Online].

[22] Z. Elhamraoui, "Fine-tuning in Deep Learning," Artificial Intelligence, 23 June 2020. [Online]. Available: https://ai.plainenglish.io/fine-tuning-in-deep-learning-909666d4c151.

[23] Prakharry, "Intuition of Adam Optimizer," 24 October 2020. [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

[24] H. &. G. P. &. S. M. K. &. M.-M. R. &. B. P. &. S. V. Panwar, "A Deep Learning and GradCAM based Color Visualization Approach for Fast Detection of COVID-19 Cases using Chest X-ray and CT-Scan Images.," Researchgate, 2020. [Online]. Available: https://www.researchgate.net/publication/343508866_A_Deep_Learning_and_GradCAM_based_Color_Visualization_Approach_for_Fast_Detection_of_COVID19_Cases_using_Chest_X-ray_and_CT-Scan_Images/.

[25] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," Towards Data Science, 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networksthe-eli5-way-3bd2b1164a53.

[26] V. Gupta and S. Mallick, "Face Recognition: An Introduction," Learn OpenCV, 16 April 2019. [Online]. Available: https://learnopencv.com/face-recognition-an-introduction-forbeginners/.

[27] R. Materese, "NIST Launches Studies into Masks' Effect on Face Recognition Software," NIST, 4 August 2020. [Online]. Available: https://www.nist.gov/newsevents/news/2020/07/nist-launches-studies-masks-effect-face-recognition-software.

[28] P. Nagrath, R. Jain, A. Madan, R. Arora, P. Kataria, and J. Hemanth, "SSDMNV2: A real time DNN-based face mask detection system using single-shot multi-box detector and MobileNetV2,"

Sustainable cities and society, March 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7775036/#sec0085

[29] O. Gurav, "Face Mask Detection Dataset," 2020. [Online]. Available: https://www.kaggle.com/omkargurav/face-mask-dataset

[30] S. Patnaik, "Face mask detector," 2021. [Online]. Available: https://www.kaggle.com/spandanpatnaik09/face-mask-detectormask-not-mask-incorrectmask.