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

In the present era for security and other law enforcement agencies have widely used facial recognition technology. As per the situation of COVID-19 (Coronavirus) spreading overall in the world day to day, every government and WHO gives alert and promoting to wear a mask, and keep social distancing. As the population strictlincreasesng, checking whether a person is wearing a mask or not is so challenging. So here come the Face Mask Detection model comes more feasible and helpful for detecting a person is wearing a mask or not in real-time. In the training, validation, and testing phases CNN structures and various parameters during compilation and fitting the model for training and validation as image generation, number of epochs, batch sizes, kernel size, and so on were adjusted to achieve the best accuracy result. This model was able to achieve an accuracy of 98.77% with achievement in the validation of 93.83% in 20 epochs, these results were achieved when running this model on Jupyter-Notebook on Kaggle and Google Colab. When this model was run on my system, I achieved an accuracy of 99.73% with a loss of 2.32%, and validation accuracy of 99.72% with a validation loss of 2.33% in 20 epochs.

Keywords: Sequential Learning, Artificial Intelligence (AI), Machine learning (ML), Deep neural learning (DL), Convolutional Neural Network Model (CNN), Supervised Learning.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 Introduction to Project

Machine learning has made significant advances in recent years in the field of artificial intelligence (AI). AI is an essential part of any newly developed technology. Today, making significant progress in technical innovation is very difficult without AI. There is a growing recognition that artificial intelligence will change the world tremendously. In the present era, AI is most helpful in recognizing COVID-19 from X-ray reports with more accuracy and precision, in which Machine Learning and Deep Learning come role in more effective way. As the gradually increasing of COVID-19 (Coronavirus), the COVID protocols provided by WHO and each government like wearing masks, social distancing, sanitization and staying home, and so on should be followed by persons strictly. But there is a problem to check each and every person whether the person is wearing a mask or not, this is more challenging work. That's why here comes the importance of concepts of Deep Learning which comes under Machine Learning and it comes under Artificial Intelligence.

This project is based on Deep learning using a 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.** The Face Mask Detection model can recognize a person in real-time with an accuracy of 99.73% which is most effective.

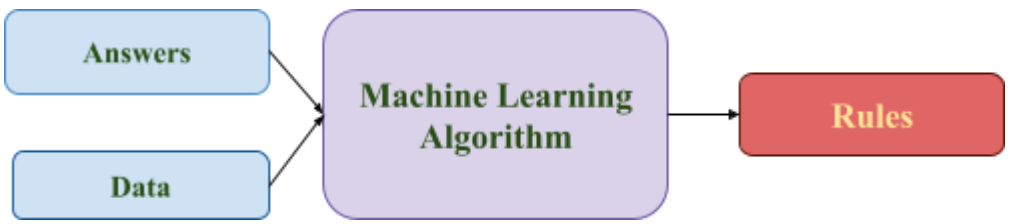
1.1.1 AI, ML, and DL

AI (Artificial Intelligence)

Artificial Intelligence (AI) is a vast field that includes Machine Learning (ML) and Deep Learning (DL). Artificial Intelligence is the theory of learning human behaviours and think Logically and process of computer to be intelligent like human being. Another definition AI is a branch of computer science with the simulation of intelligent behaviors in computers. It studies ways to build intelligent programs and machines that can creatively solve problems, an individual's right to self-determination has long been regarded as a human right.

ML (Machine Learning)

Machine Learning (DL) deals with the study of algorithms and computer models used by machines in order to perform a given task. ML is a subset of AI that gives structures the capability to mechanically learn and enhance from experience without being explicitly programmed. In Machine Learning, we are giving inputs Answers along with data and passes into a machine-learning algorithm to give the rules associated with the inputs.



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

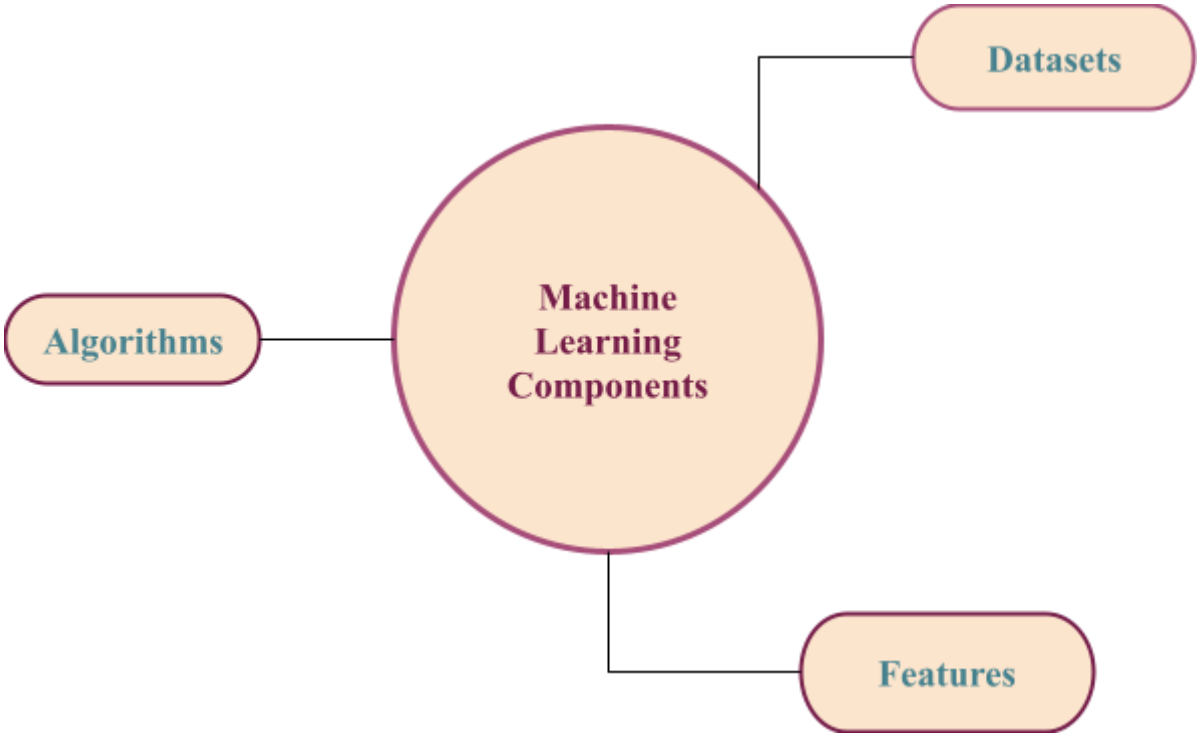


Figure: Flow chart shows components of ML

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 subset of Machine Learning which is inspired by human brain, which having millions of Brain Neural Networks (BNN). The level of abstraction of deep learning algorithms increases by nonlinearly transforming input data through complex, multilayered neural

networks.

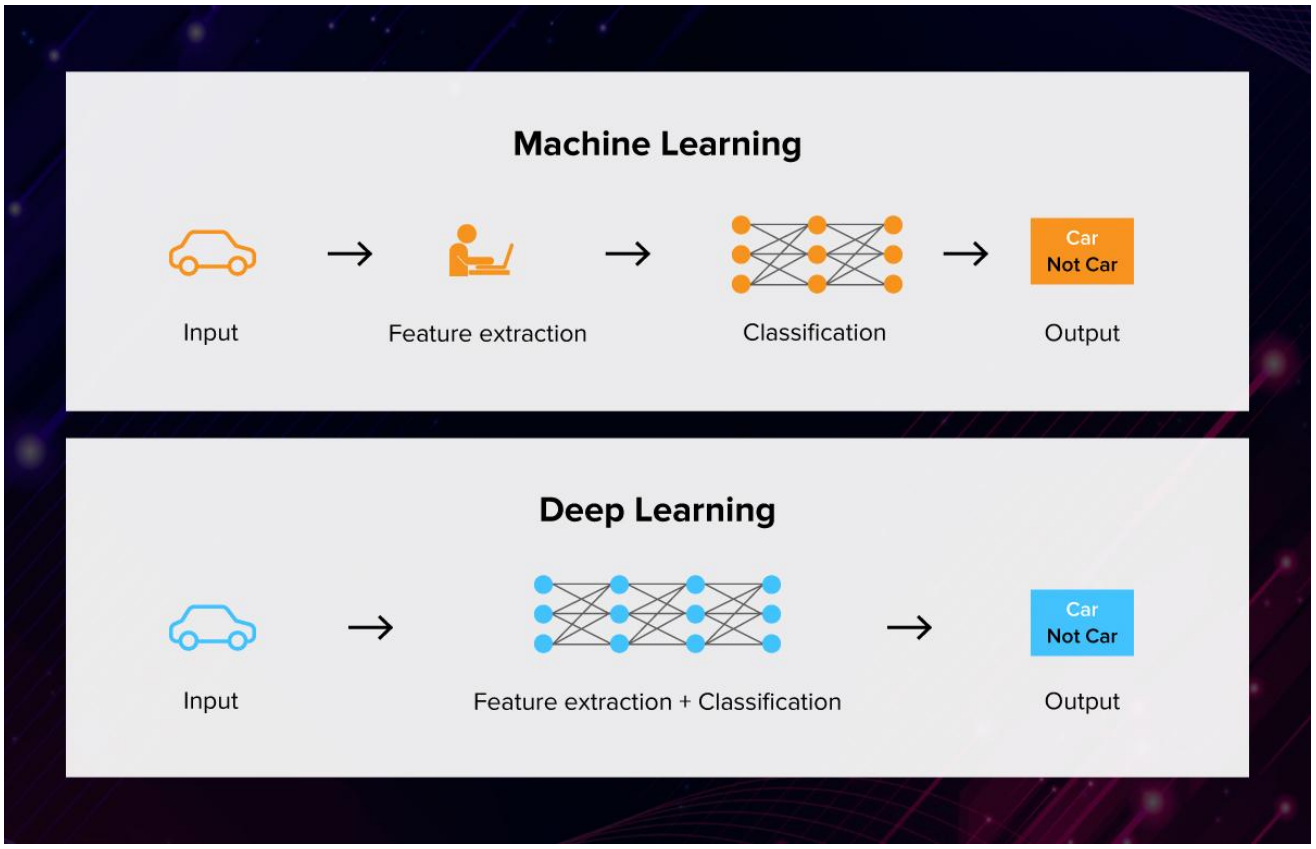
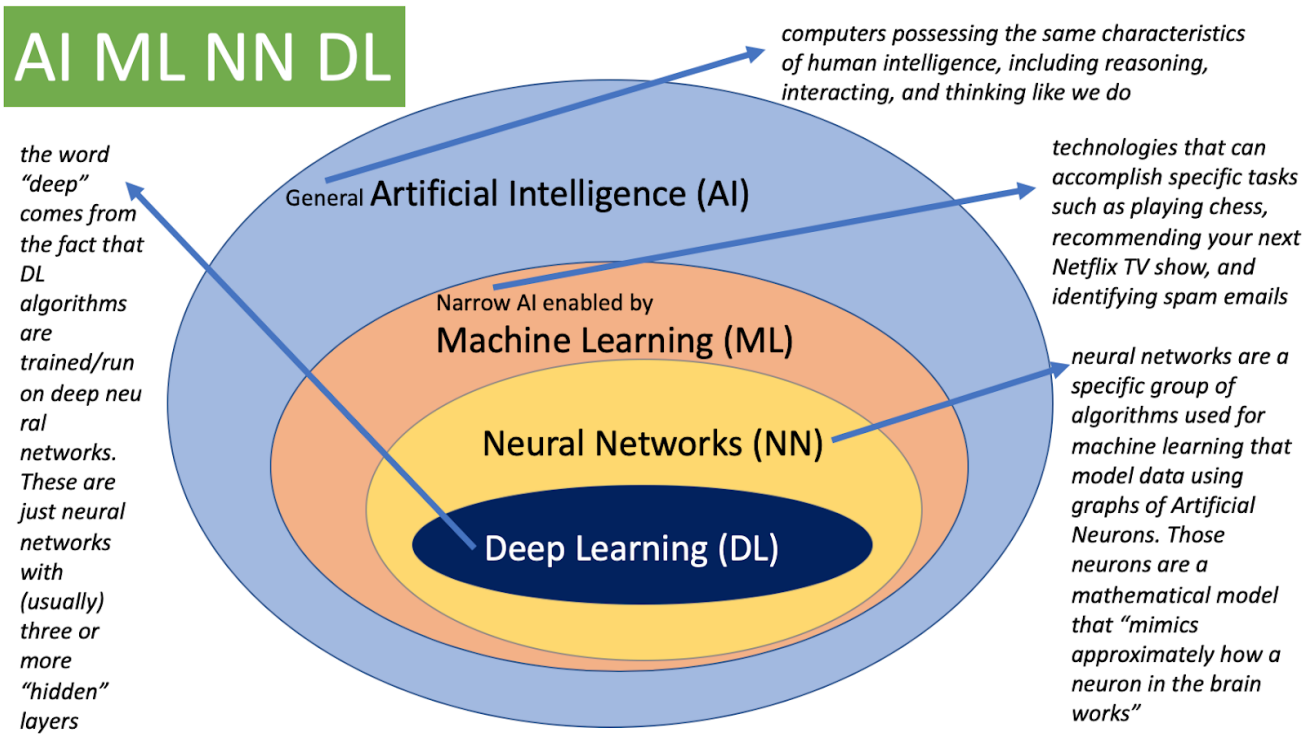
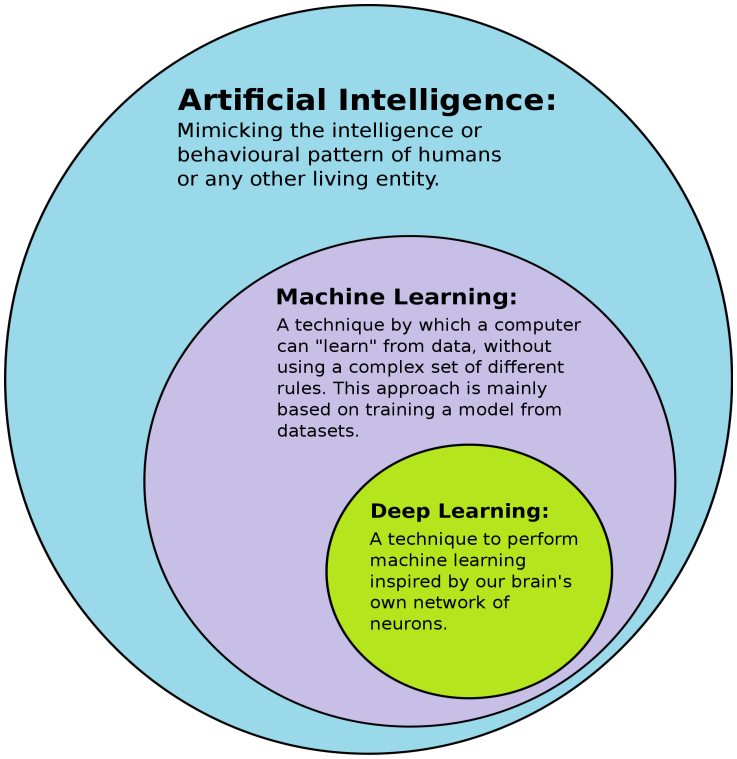


Figure: Machine learning and Deep learning

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. It can also detect Face Mask on real-time video whether a person is wearing a mask or not.

1.3 2. Literature Survey

Presented below is a brief overview of relevant works on the classification and tracking of face masks. The detection of these masks can be achieved in several ways. Some of the techniques use electromagnetic and radiometry signals. Machine learning techniques were employed in the detection of facial masks using deep neural networks (DNNs). Furthermore, performance measurements were conducted to compare ELM ANN and BP ANN. Using neural networks, ultrasound images are analyzed to determine whether they are abnormal lesions.

The proposed system uses YOLOv3 architecture to recognize masked faces and the masked faces are rendered using the advanced system proposed by R. Bhuiyan, S. Khushbu, and S. Islam. Convolution Neural Network (CNN) is a learning algorithm used by YOLO (You Only Look Once) which is a pre-trained model. CNN is used by YOLO through hidden layers, through research, and with easy algorithm retrieval, allowing for easy

detection and location of any type of threat and can detect and locate any type of image. Following the breakdown of the dataset into 30 unique images, 30 predictions are derived using the results. The results are excellent on both counts, both in terms of imaging and in terms of detecting. It is tested using a live video to determine the fps rate of the model inside the video and how well it can detect the unmasked/masked layers. With an average frame rate of 17, our model produces impressive results in the video. Other methods using their own data set are less efficient and slower than this system.

CHAPTER 2: ANALYSIS

2.1 Methodology

2.1.1 Supervised Learning

Supervised learning is a method for developing artificial intelligence (AI) that involves training an algorithm in which a labeled input data set refers to a set of output data. Learning from a dataset is referred to as supervised learning. There is always a training dataset that is composed of input images, together with their correct outputs. By using the supervised learning algorithm to examine the data patterns associated with the outputs, any patterns relating to the selected outputs will be detected. Based on the previously trained data, a supervised learning algorithm will be able to identify new unknown inputs with their correct labels after taking in new information. In supervised learning models, new input data is essentially presumed to have the correct label. In Equation 1, we see a simple supervised learning algorithm. A predicted output value Y is attached to an input value x as shown in equation 1. When a model is trained, input features are connected to the predicted output, which is created by the model.

Supervised Learning is further classified into two categories:

- Classification
- Regression

$$Y = f(x)$$

Equation 1: Supervised Learning Algorithm Equation

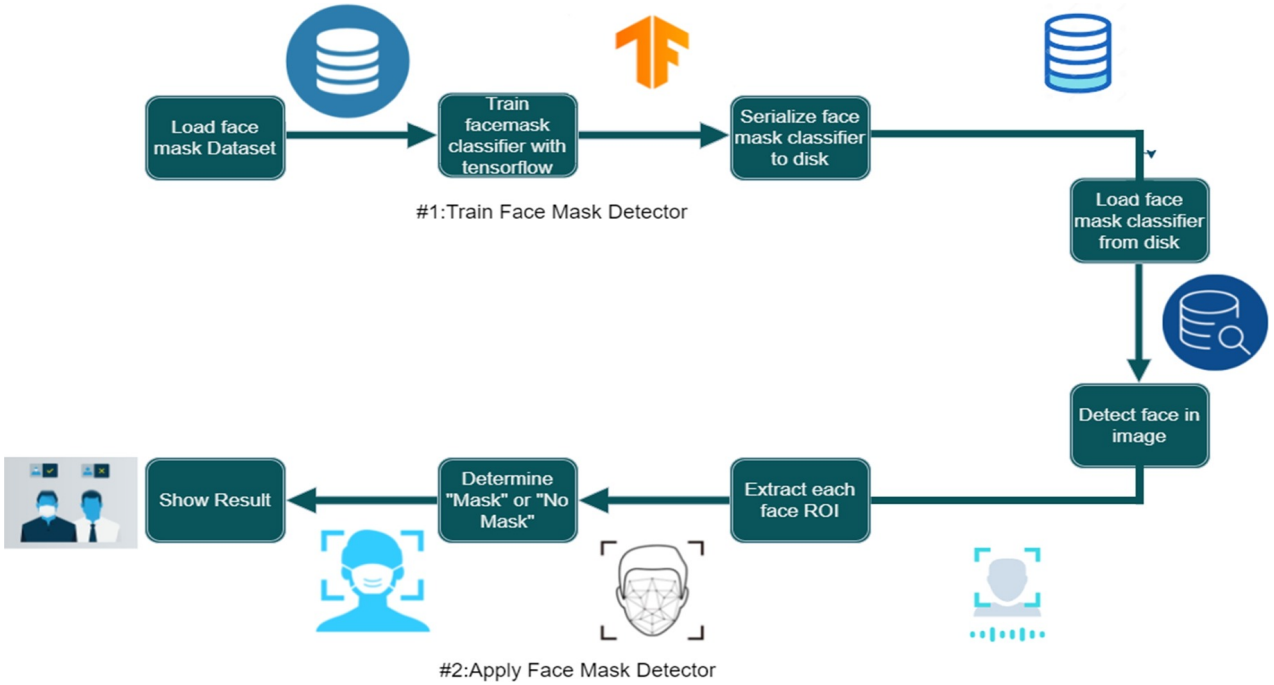


Figure: Process of the flow of data

Sequential Learning

The sequential concept describes how you learn parts of a task one after the other. An example of sequential learning would be learning one part of the task before moving on to another. Human behavior is fundamentally driven by serial organization, which is conducive to the psychology of human beings.

2.1.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN) are Deep Learning (DL) algorithms that give weight to various aspects of the image (learnable weights and biases) and are able to differentiate one object from another.

- By using convolutional neural networks, the right features are automatically extracted from the data.
- Spatial features (Arrangement of pixels) are captured by the CNN.
- CNN also follows parameters sharing, applies a single filter in a different part of a single image.
- CNN has convolution type of hidden layers.
- Activation functions include convolution and pooling. In which pooling are of two types MaxPooling and MinPooling.
- CONVOLUTION: Filters (or other image processing codes) are applied to the input image (kernel) to produce the output.
- POOLING: The process of picking a maximum value from the selected region is Max pooling and vice versa.

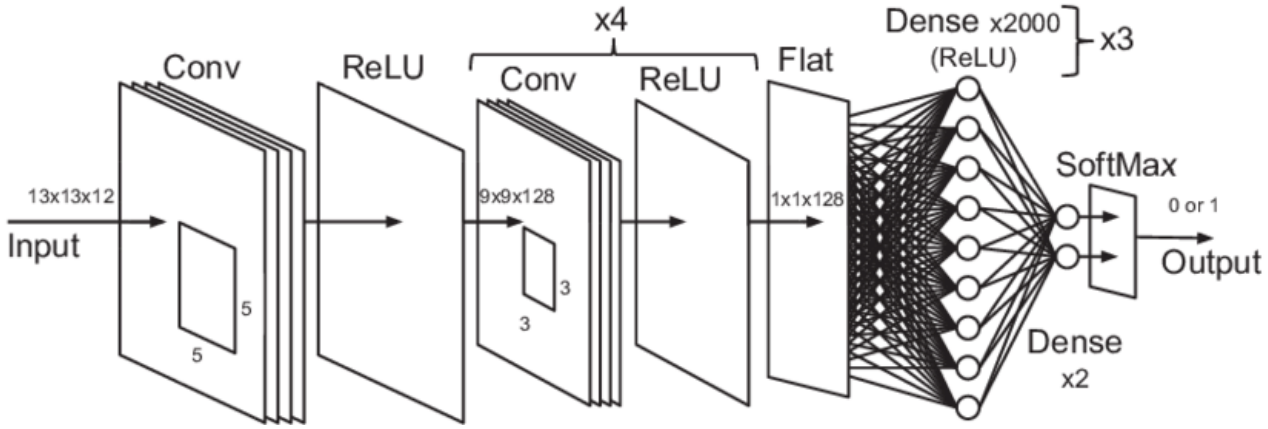
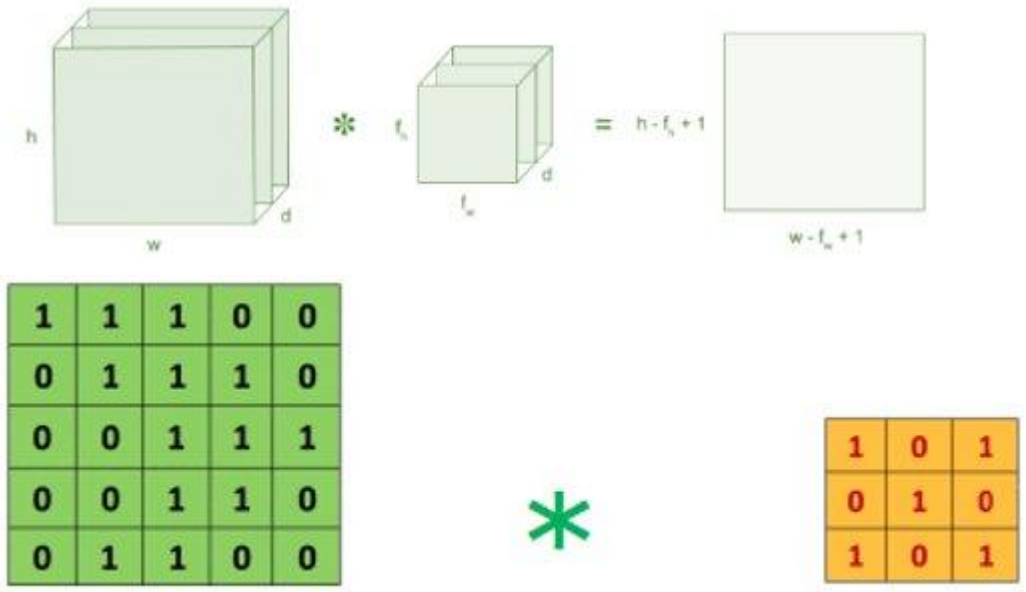


Figure: Example of a CNN

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

For this project report, 2x2 max-pooling was used to reduce the three-dimensional size of the convolutional neural network. Through dimensionality reduction, 2x2 pooling will reduce the computing power that is required to process the data. In addition, it can be used to extract major features, which allows the model to be trained effectively. A maximum pooling technique will also return the highest value from the image's part that was covered by the kernel, in this project kernel is taken as (3, 3). Additionally, max-pooling can lead to reduced dimensionality and noise reduction. Combining the convolution layer and the layer of pooling allows the model to learn features about the image.

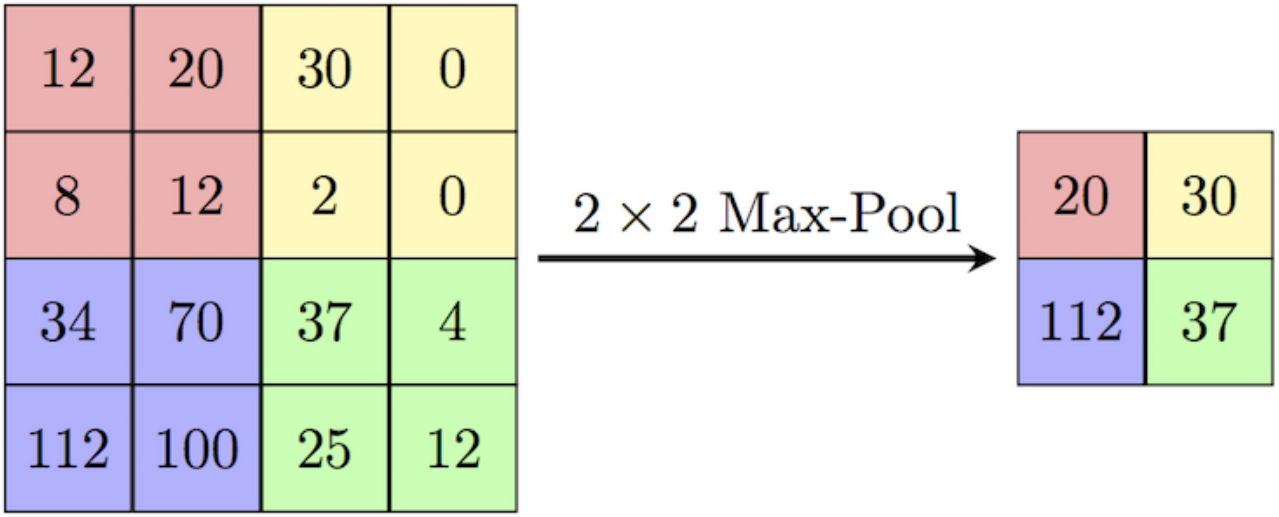


Figure: shows an example of how 2x2 max-pooling is performed

2.1.4 Fully Connected Layer

Additionally, two additional fully connected layers were introduced as part of this methodology which allowed a non-linear mix of high-level face features obtainable during the kernel process to be acquired. After fully connected layers, SoftMax activation was used. By using the SoftMax function, a vector of numbers can be converted into a vector of probabilities. After the SoftMax algorithm has normalized results into probabilities of 1.0, the results are converted to probability. The output image should then be flattened since it was converted into a multilayer neural network. In every step of the training process, backpropagation is applied to the flattened output of a feed-forward neural network. Fully connected layers are used to recognize people wearing masks from people who are not wearing masks by using high-level feature analysis.

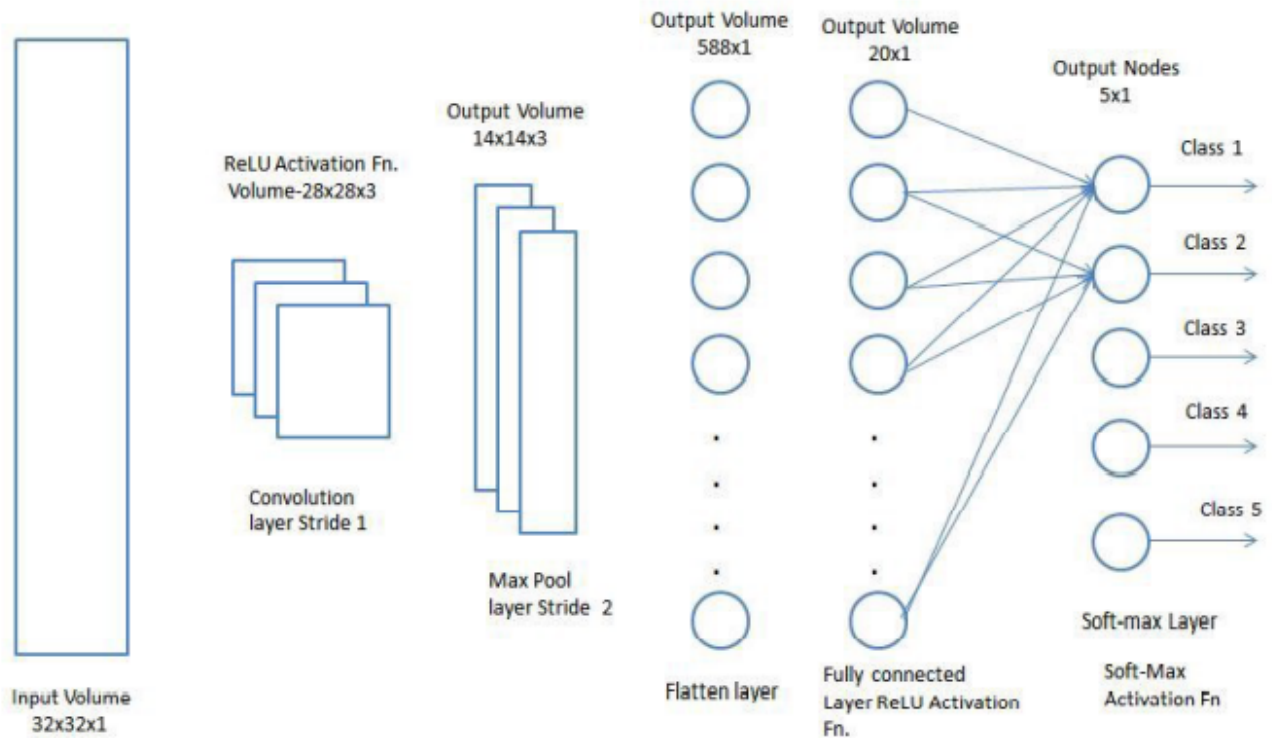


Figure: Fully Connected Layer Architecture

Flattening

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

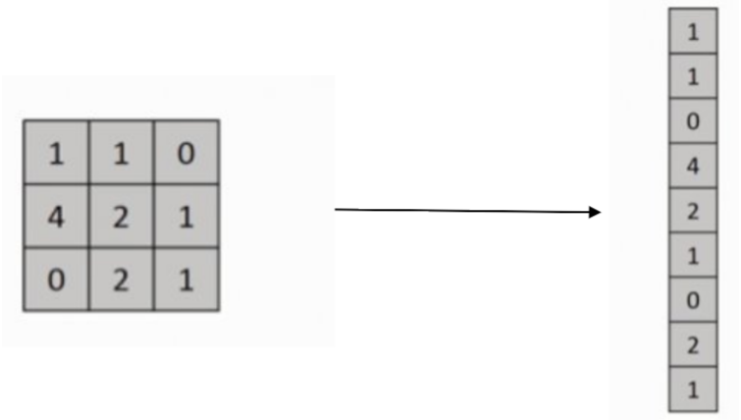


Figure: Flattening of 3 x 3 matrix into column matrix

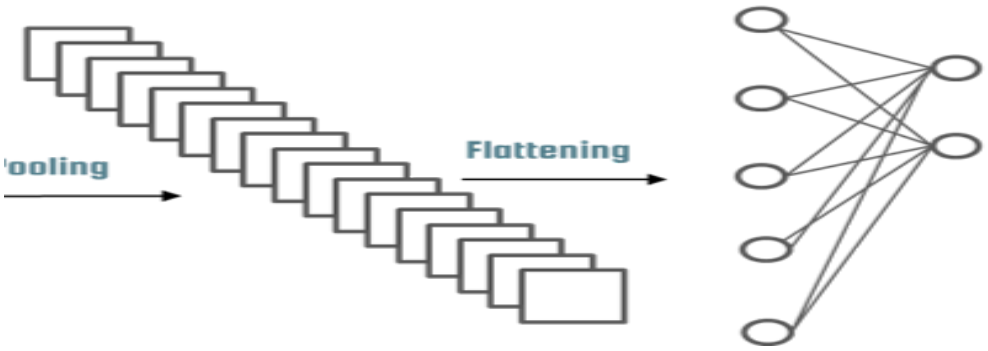


Figure: Flattening of Convoluted layer

2.1.5 Implementation of CNN

Feed Forward

- Set of input features and random weights
- Weights will be optimized by backpropagation

Back Propagation

- Calculating error between predicted output and target output and use Gradient descent method to update weights.

Gradient Descent

- It is a Machine Learning Algorithm.
- It operated iteratively to find the optimal values for its parameters, user-defined learning rate, and initial parameter values.

Vanishing & Exploding Gradient

- It is very common problem in every Neural Network, which is associated with Backpropagation.
- Weights of the network are updated through backpropagation by finding gradients.
- When the number of the hidden layer is high, then the gradient vanishes or explodes as it propagates backward. It leads to instability in-network, unable to learn from training.
- An explosion occurs when gradients with values greater than one are multiplied through layers of the network repeatedly.
- A redesign of the network, Long-Short Term Memory networks, Gradient clipping, etc. can be used to solve the problem.

2.2 Training

The training process is also referred to as backpropagation. For a model to correctly predict an output class, it must train to find weights that reflect the input data most accurately. As a result, these weights are continuously updated and moved towards their optimal output class. In this project, the CNN model was trained using a Face Mask Detection Dataset. Training occurs over epochs and each epoch is split into batches.

Epoch - In a training dataset, an epoch is one pass through each row
Batch - Batches are a series of samples considered by the model within an epoch before weights are updated.

2.2.1 Adam Optimizer

In this project report, the CNN model was trained using the Adam optimizer. An adaptive learning rate optimization algorithm called Adam has been proposed to specifically train deep neural networks. Analyzing each learning rate within the model, Adam optimizer calculates each rate using different parameters.

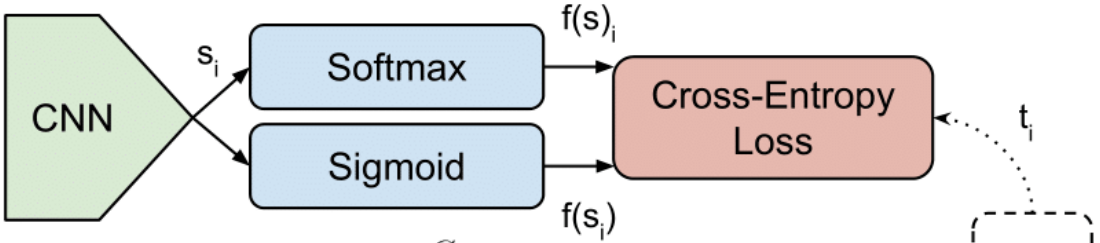
2.2.2 Categorical Crossentropy Loss

In multi-class classification tasks, categorical crossentropy is used as a loss function. A model must decide which category an example belongs to out of all the possible categories in these tasks.
Basically, it is used to measure the difference between two probability distributions.

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

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

Two discrete probability distributions may be distinguished from each other very well with this loss. In this context, y_i represents the probability that event i occurs and the sum of all y_i is 1, meaning that exactly one event may occur.
In this case, the minus sign ensures that the loss increases as the distributions get nearer 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

Whenever a model runs through a training data set, an epoch is taken, i.e., every image is iterated through a convoluted layer every time. The neural network may become better at predicting given unseen data, which is the test data, if it is fed with training data over several epochs, for example. To determine the most accurate results, the data was predicted over a period of 20 epochs. In Table 1, epochs are shown to impact loss percents.

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

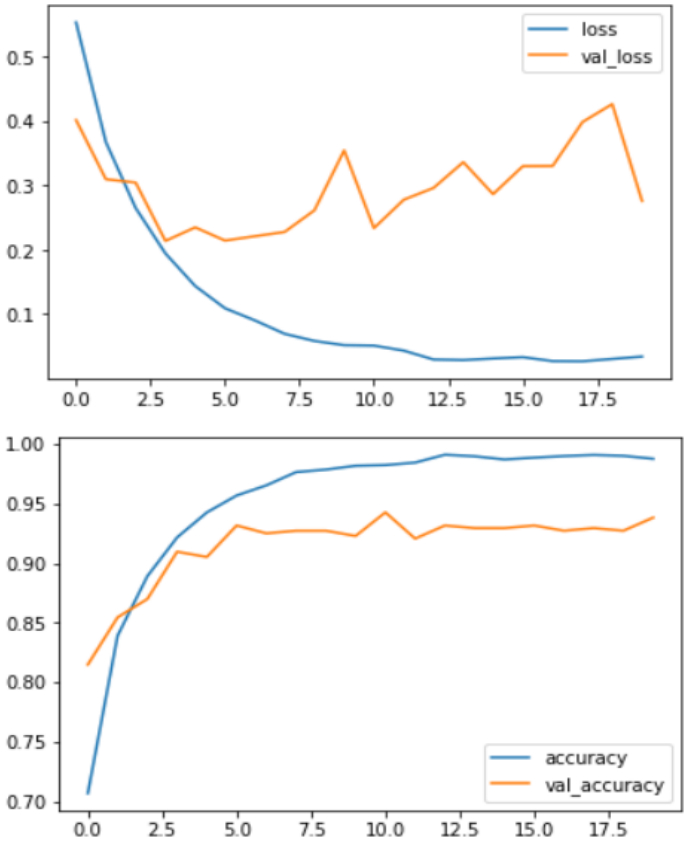


Figure: Graph of loss, val_loss vs epochs and accuracy, val_accuracy vs epochs respectively

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.

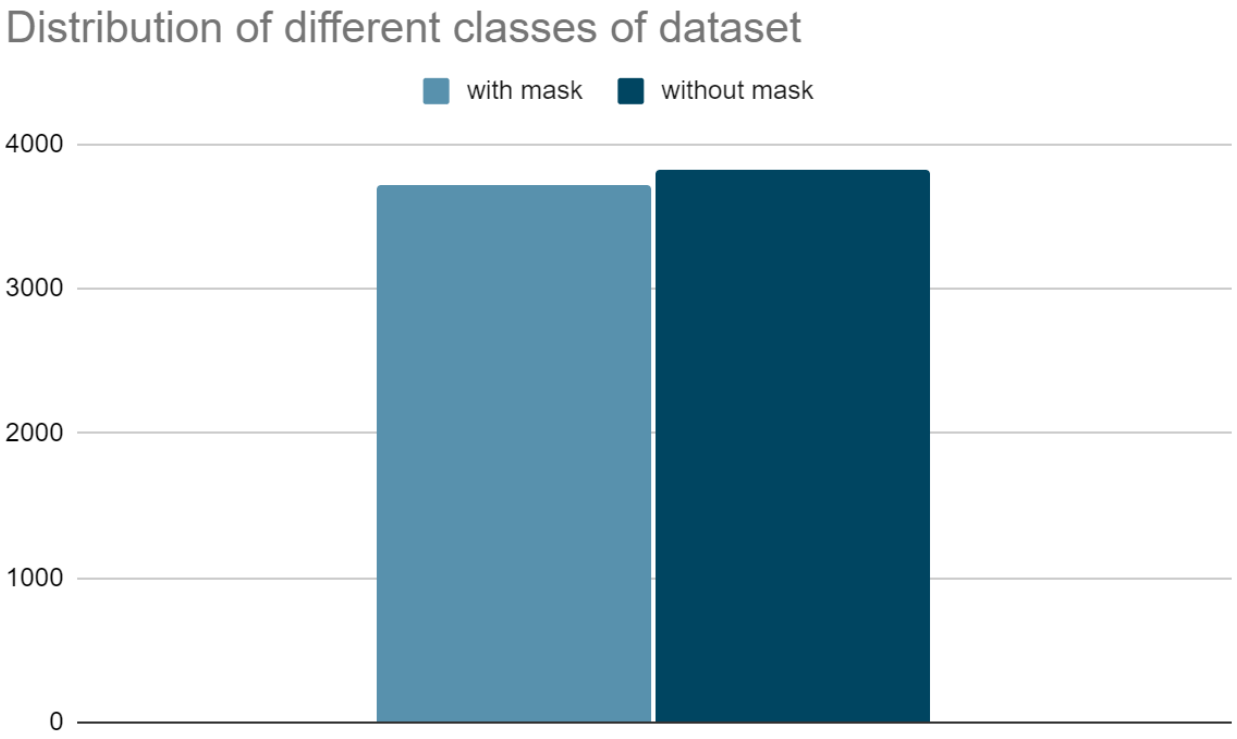


Figure : Classes Distribution of Face Mask Detection Dataset
 # with mask : 3725 images
 # without mask : 3828 images

3.1.1 Dataset Pre-processing Image Datagenerator

A pre-processing step, such as resizing the image, grayscaling the image, or rescaling the image is very significant in machine learning because it helps improve the quality of the data, which can then be used to extract important insights from the data. As part of data preprocessing, the data is cleaned and organized to prepare it for model building and training. As part of data preprocessing, the data is cleaned and organized to prepare it for model building and training. Utilizing grayscale reduces the computational requirements and simplifies the algorithm to aid the extraction of descriptors.

```
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,
)
```

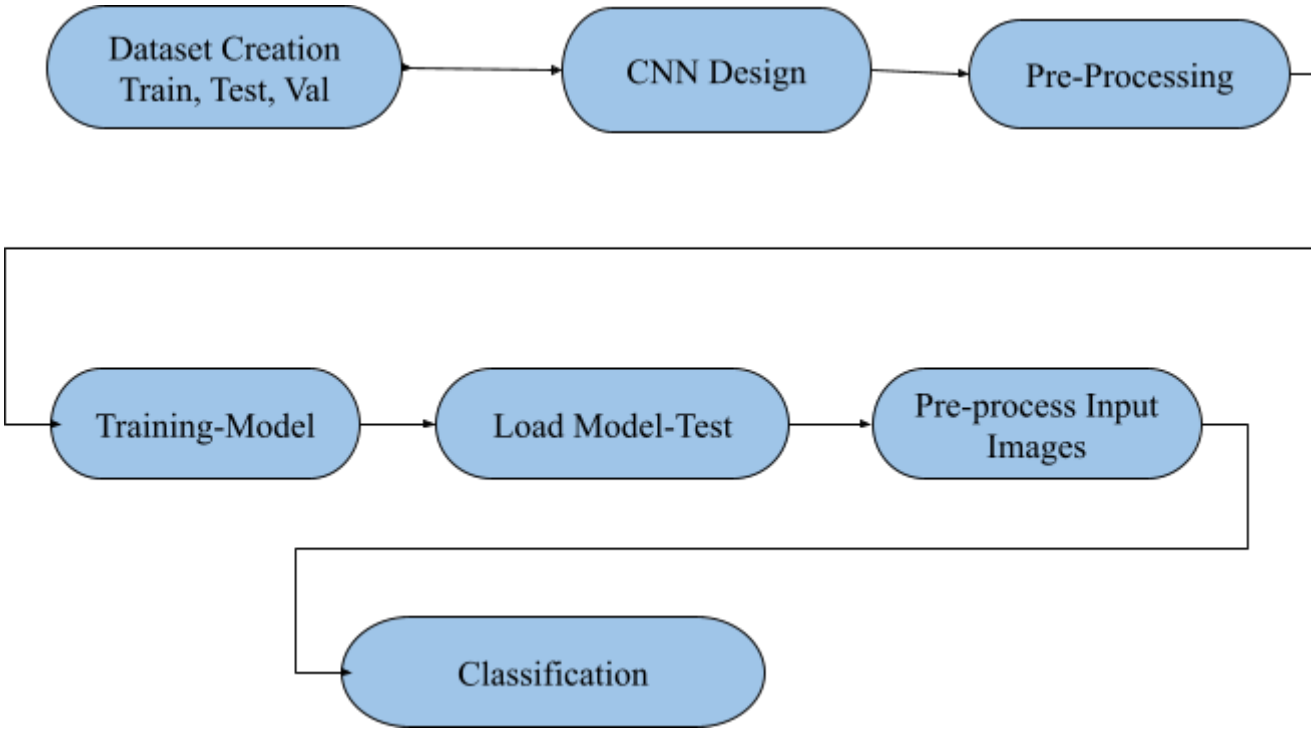
```
image_gen = ImageDataGenerator(preprocessing_function= tf.keras.applications.mobilenet_v2.preprocess_input)

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

As a Supervised Learning problem, image classification involves defining classes (objects to identify in images) and training models to recognize them from labeled examples. Using labeled example photos, a model is trained to recognize objects in images based on a defined set of target classes.

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 masks and without masks
Output: The image depicts a face mask in a categorized manner
for each image in dataset do

- Visualization of the image in two categories and label them
- Create a grayscale image by converting RGB images to OpenCV
- Resizing the grayscale image into 244x244
- Normalization of the image and convert it into a 4-dimensional array using NumPy

end
for building the CNN model do

- Create a layer of 64 convolution filters using Keras
- Add the 2nd Convolutional layer of 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 as with mask and without mask

End

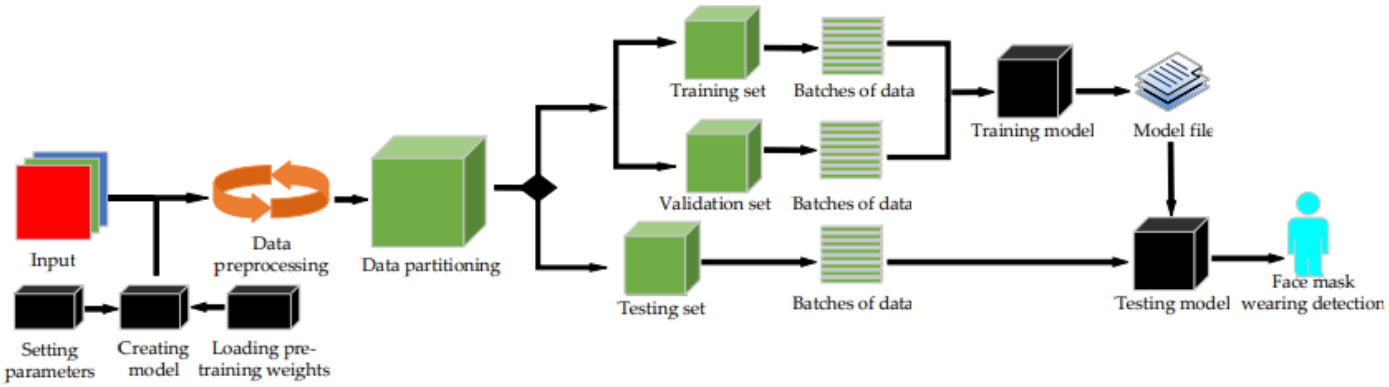


Figure: Flow chart of the Proposed model of Face Mask Detection

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 on Jupyter-Notebook on Kaggle and Google Colab using Supervised Learning in Deep Learning with Convolutional Neural Network (CNN) but when I have run on my system Dell Vostro with Intel Core i7 10th generation with 2GB of AMD Radeon 610 and Intel Integrated Graphic card, the model achieved an accuracy of 99.73% with a loss of 2.32%, and validation accuracy of 99.72% with validation loss of 2.33% in 20 epochs.

According to the model, the results are more what was expected. Using the camera, mask recognition is implemented and produces accurate results. A green frame or a red frame will be displayed over the person's face when the person's face is detected in the camera frame which shows the person is with a mask or without a mask respectively. Those who are not wearing a mask will have a red frame over their faces, whereas those who are wearing masks will have a green frame. In the resulting frame, you can also see the result written on the top left. On the top of the resulting frame, you can also see a percentage match. When the camera is able to view the side view of the face, it will work. Overall, the model is quite good.

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

CHAPTER 4: CODING AND IMPLEMENTATION

4.1 Coding 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)
```

By using the ImageDatagenerator from keras.preprocessing.image library, you can generate batches of tensor image data along with real-time data augmentation.

```
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

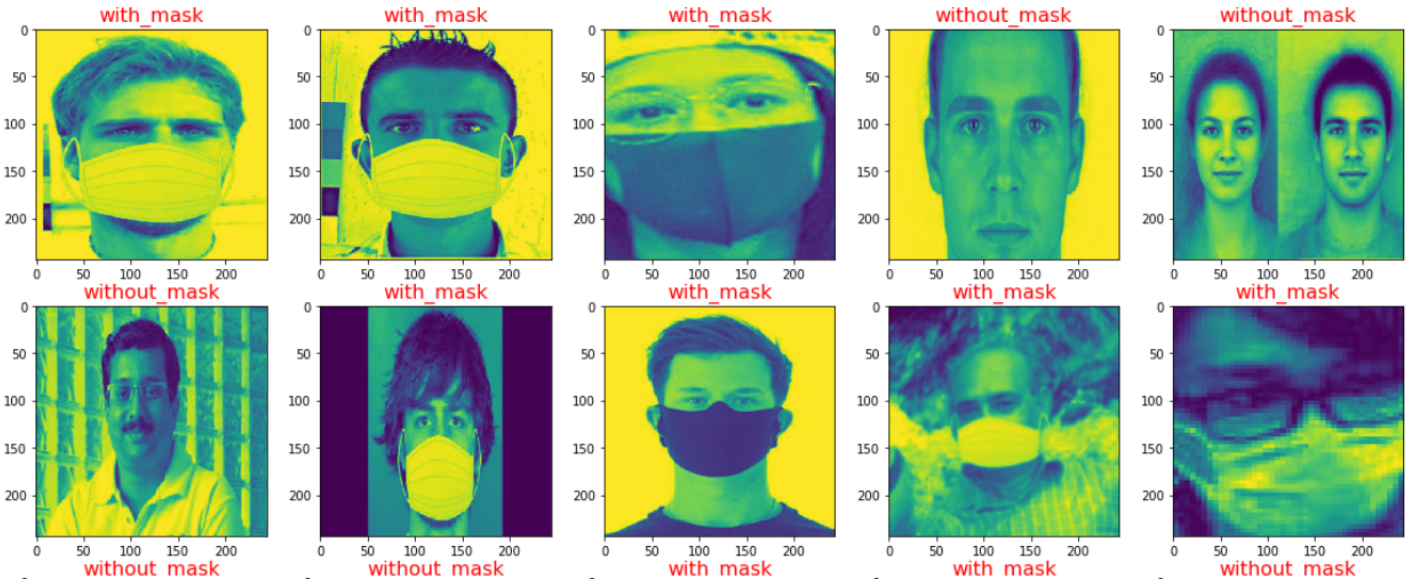
```
# funciton to show images
def show_images(image_gen):
    test_dict = test.class_indices
    classes = list(test_dict.keys())
    images, labels=next(image_gen) # get a sample batch from the generator
    plt.figure(figsize=(20,20))
    length = len(labels)
    if length<25:
        r=length
    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)
```

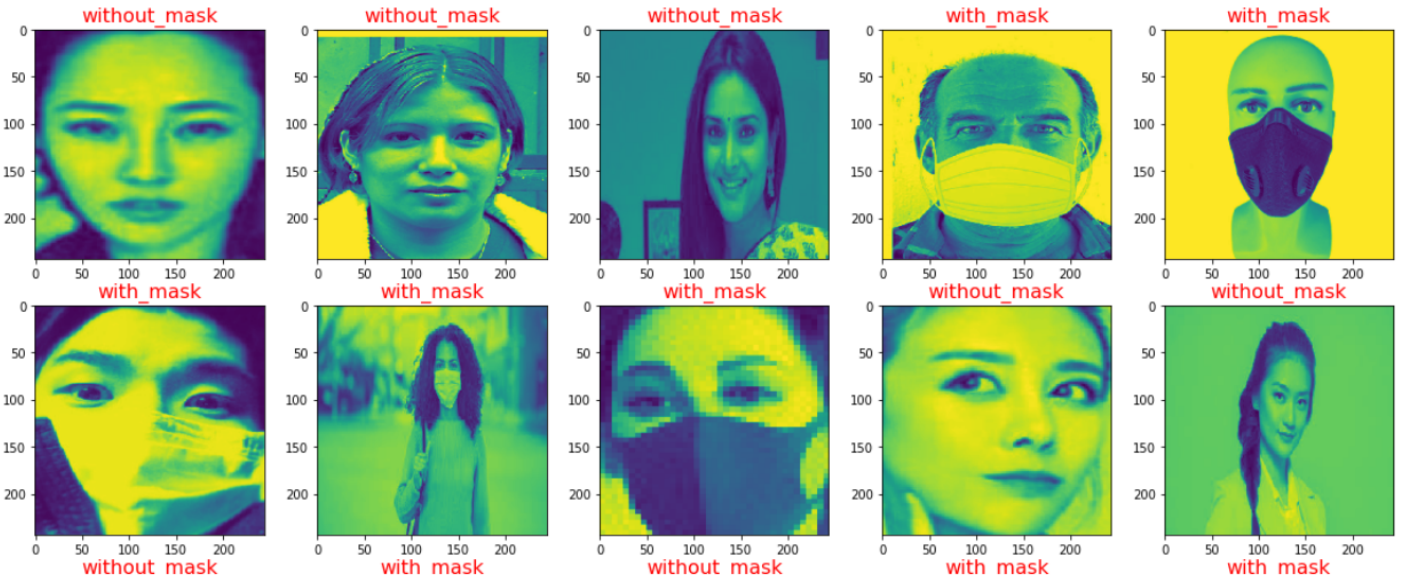
```
Training Images...
```



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()
```

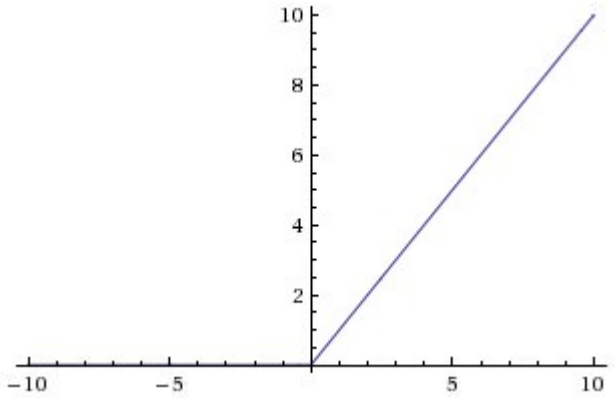
Model of CNN

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)



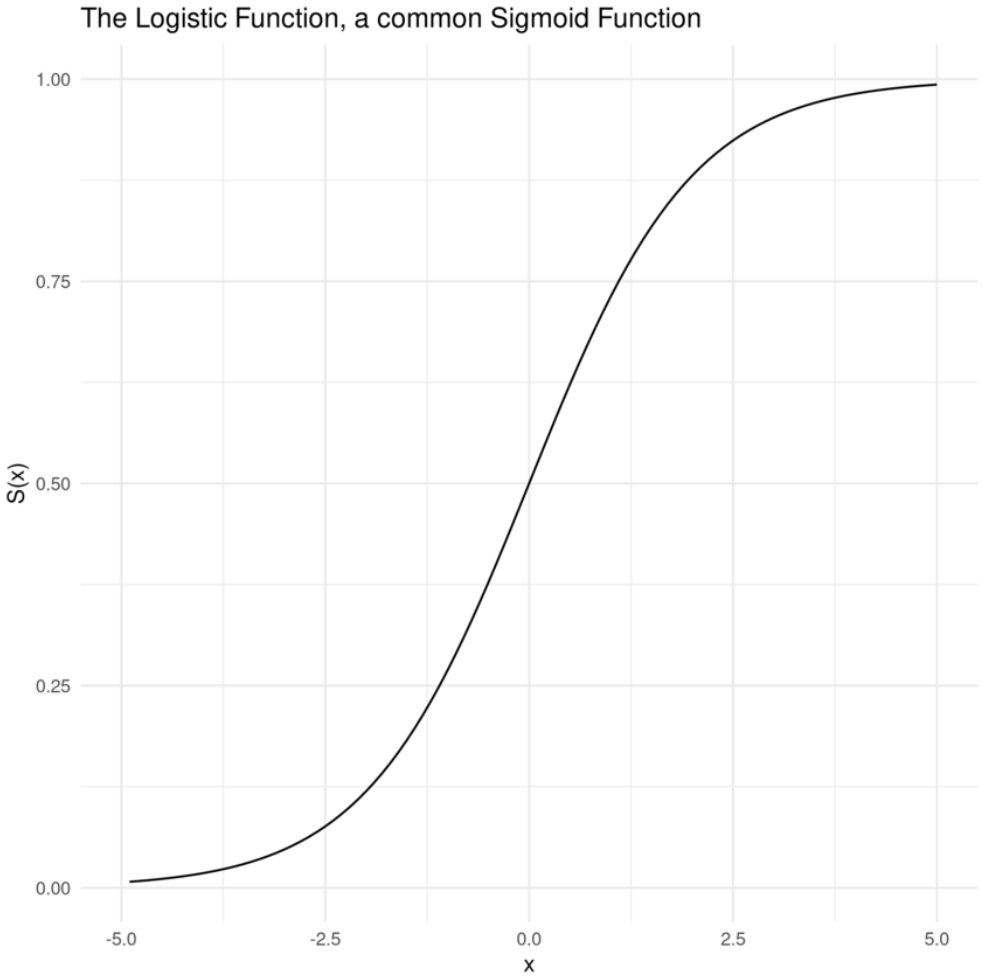
Deep learning (DL) models primarily use Rectified Linear Units (ReLU) as activation functions. If any negative value is input, x returns 0, but if any positive value is input, x returns the value that was given. So ReLu function can be written as $f(x) = max(0, x)$. I find it remarkable that such a simple function (and one based on only two lines) is able to capture nonlinearities and interactions so well. In most applications, however, ReLU is a very useful function, so it is commonly used.

Sigmoid Function

In mathematics, a sigmoid function is a function whose curve has an S-shape. As well as the logistic function, hyperbolic tangents, and arctangents, there are a number of common sigmoid functions.

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

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



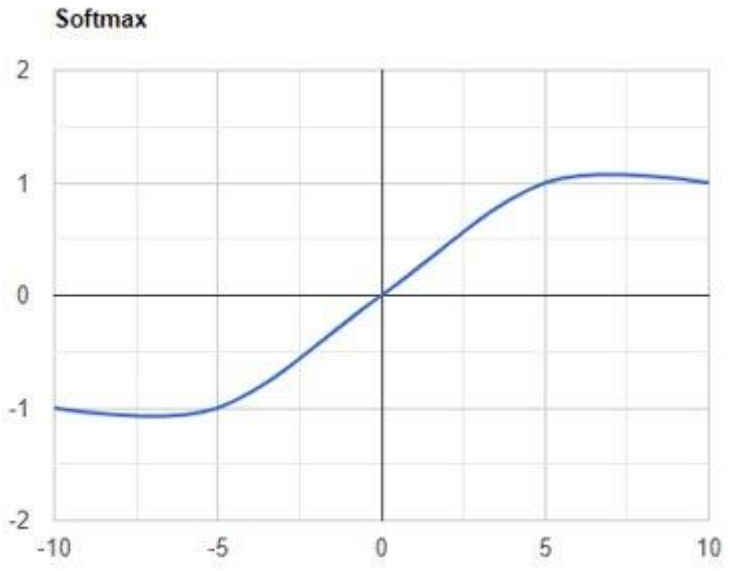
A sigmoid function maps the entire number line into a small range such as 0 or 1, or -1 or 1, so it can be used to approximate a probability out of real values.
Activation function -Sigmoid, gives binary output 0 or 1.

Softmax Function

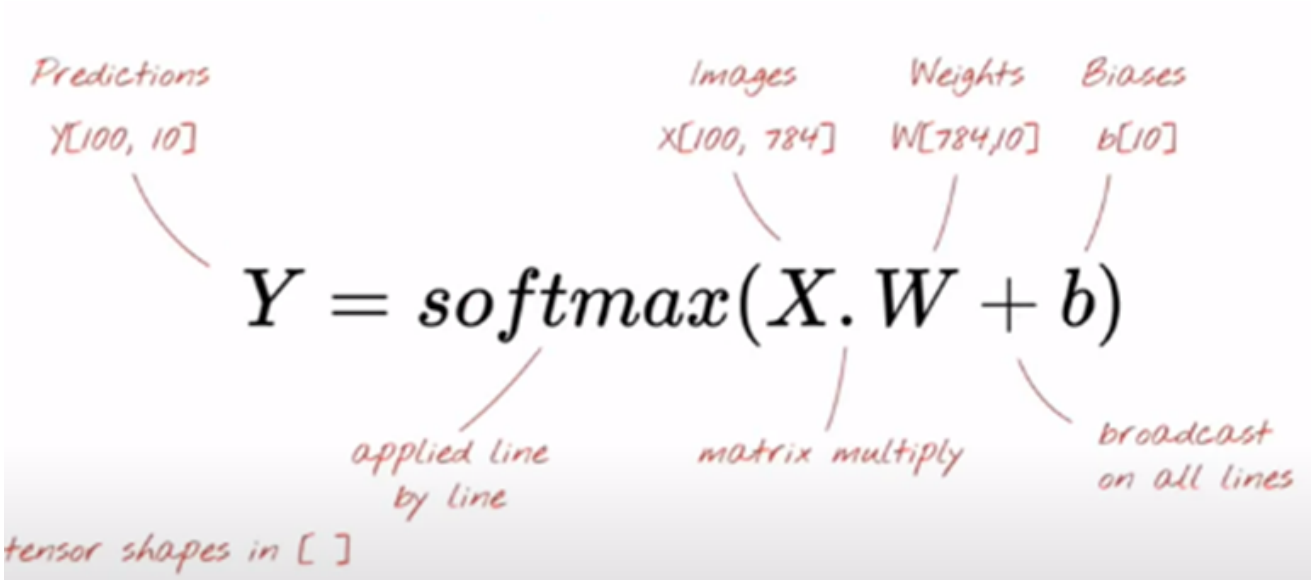
- Softmax activation function will be used instead of ReLU, TANH, and Sigmoid in the last layer of the neural network.

- By using the softmax function, the non-normalized output of a network can be translated into a probability distribution over the predicted output class. It calculates the probability distribution from the output of the last layer.

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



L = X.W + b



```
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
```

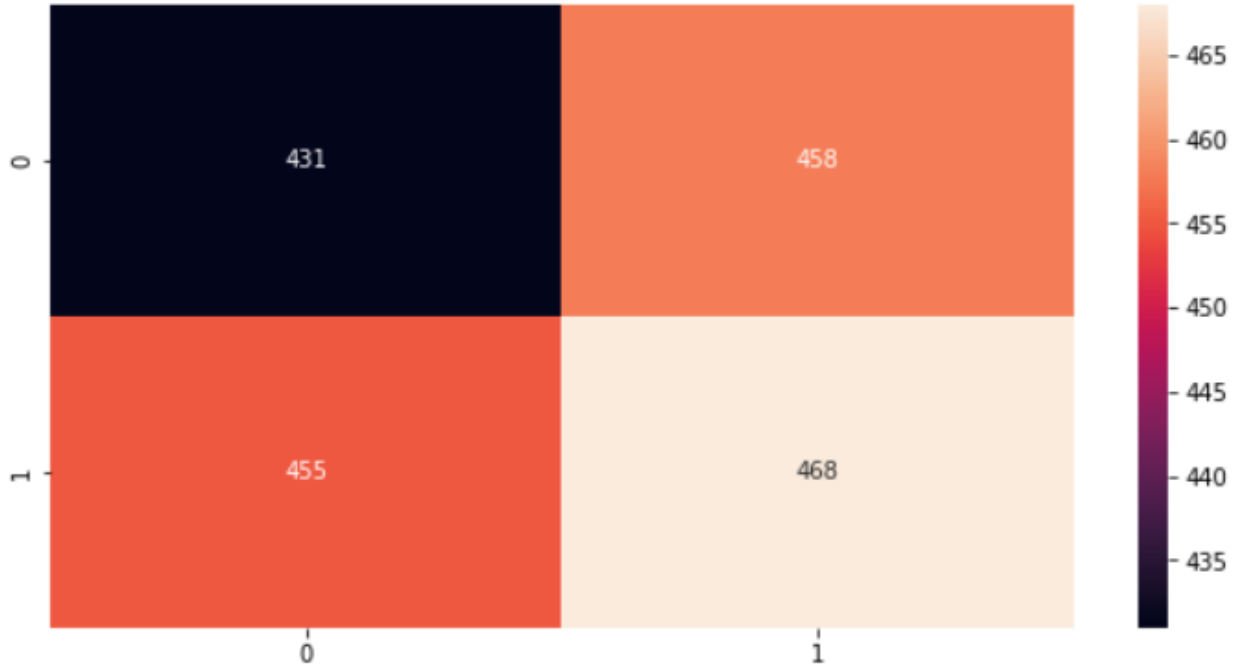
Confusion Matrix

```
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

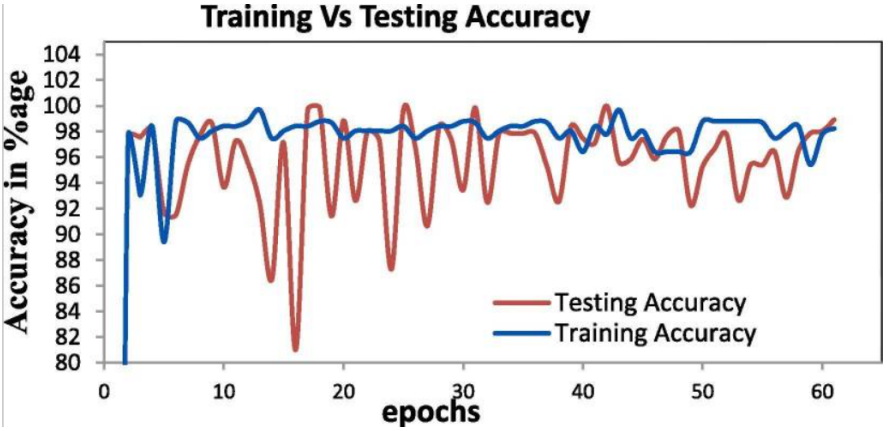
4.2.3 Packages, modules, libraries

- 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

Overfitting can be avoided in two ways. The first step is to augment the data. Secondly, over 60 epochs of training and testing, the accuracy of the model is observed critically. Below is a picture showing the observations



Additionally, as shown in the above figure, model accuracy increases with each epoch and becomes stable after epoch = 3.

4.3.2 Callback Checkpoints

Create a callback checkpoint to continue saving the best model after every epoch.

```
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

At the very end of the testing process, sample images were passed through a Convolutional Neural Network (CNN), and the comparisons were made between the predicted and the actual true classes. After applying 20 epochs, 99.73% of the predictions were accurate.

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.

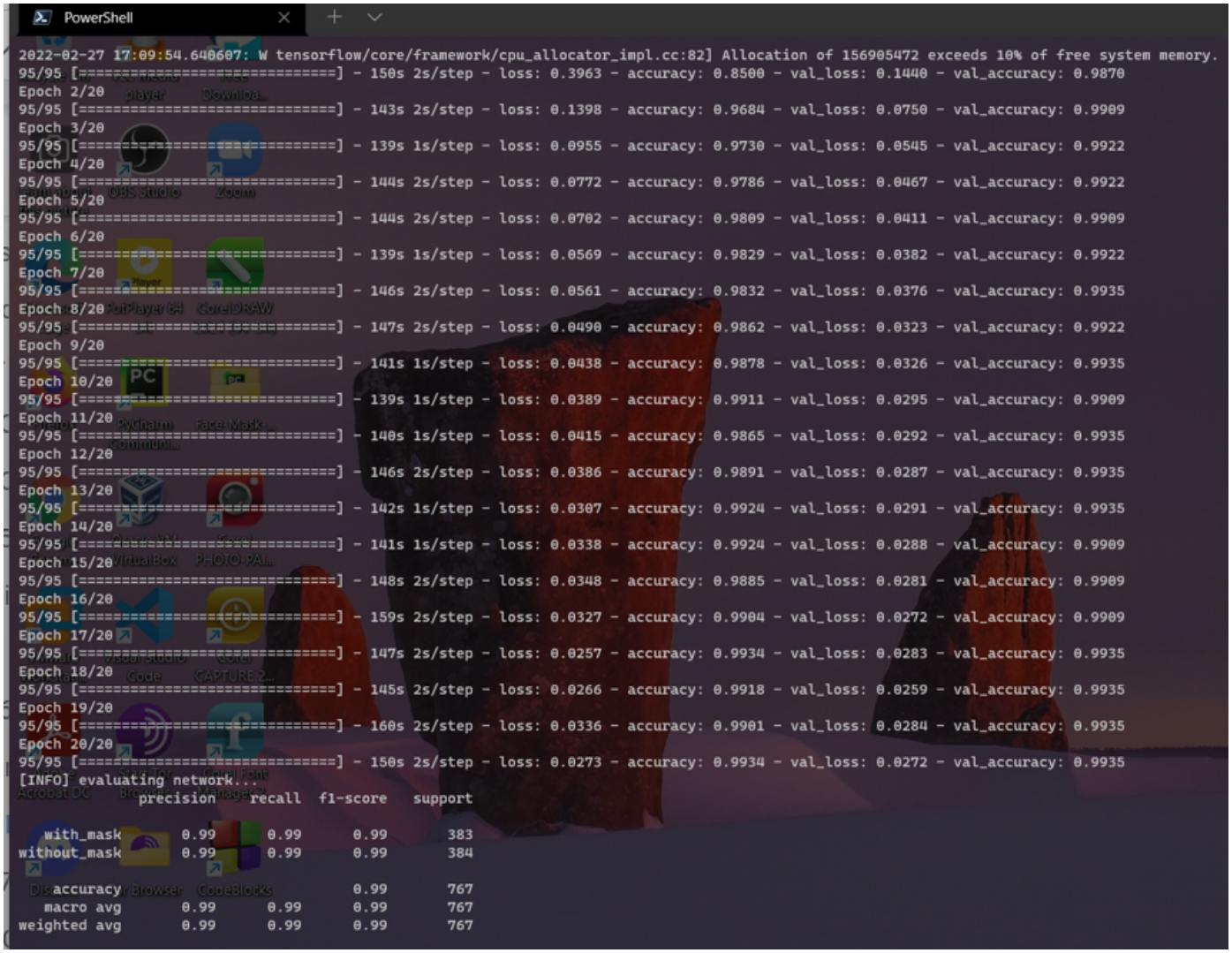
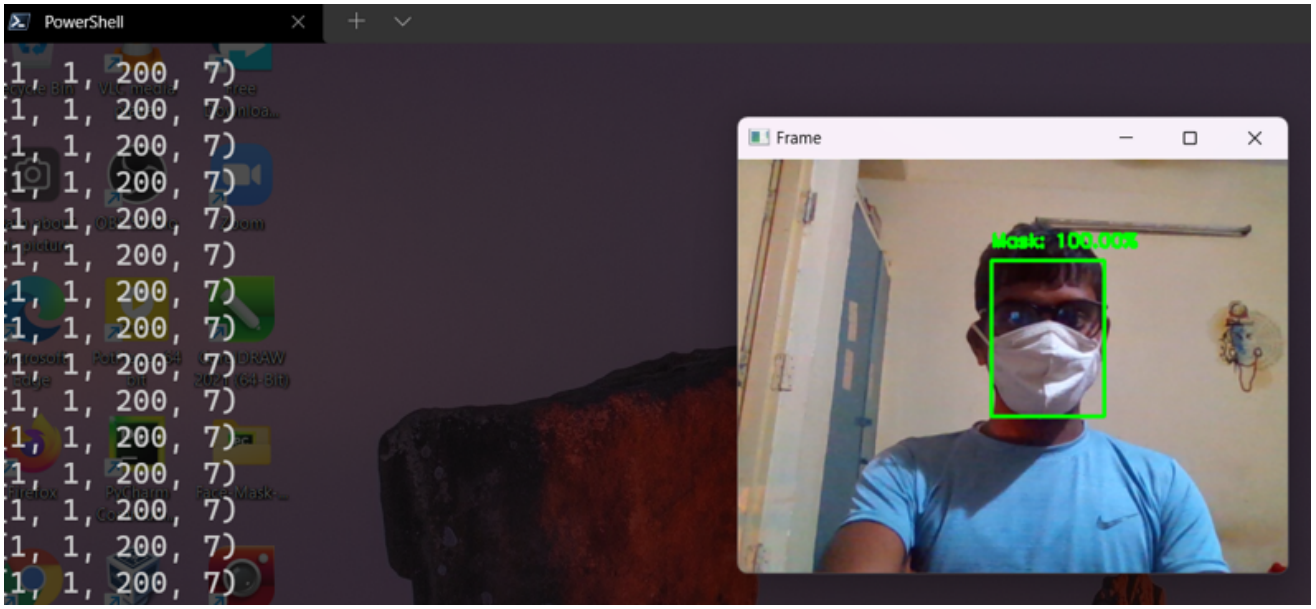


Figure: Results of the Training the model

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

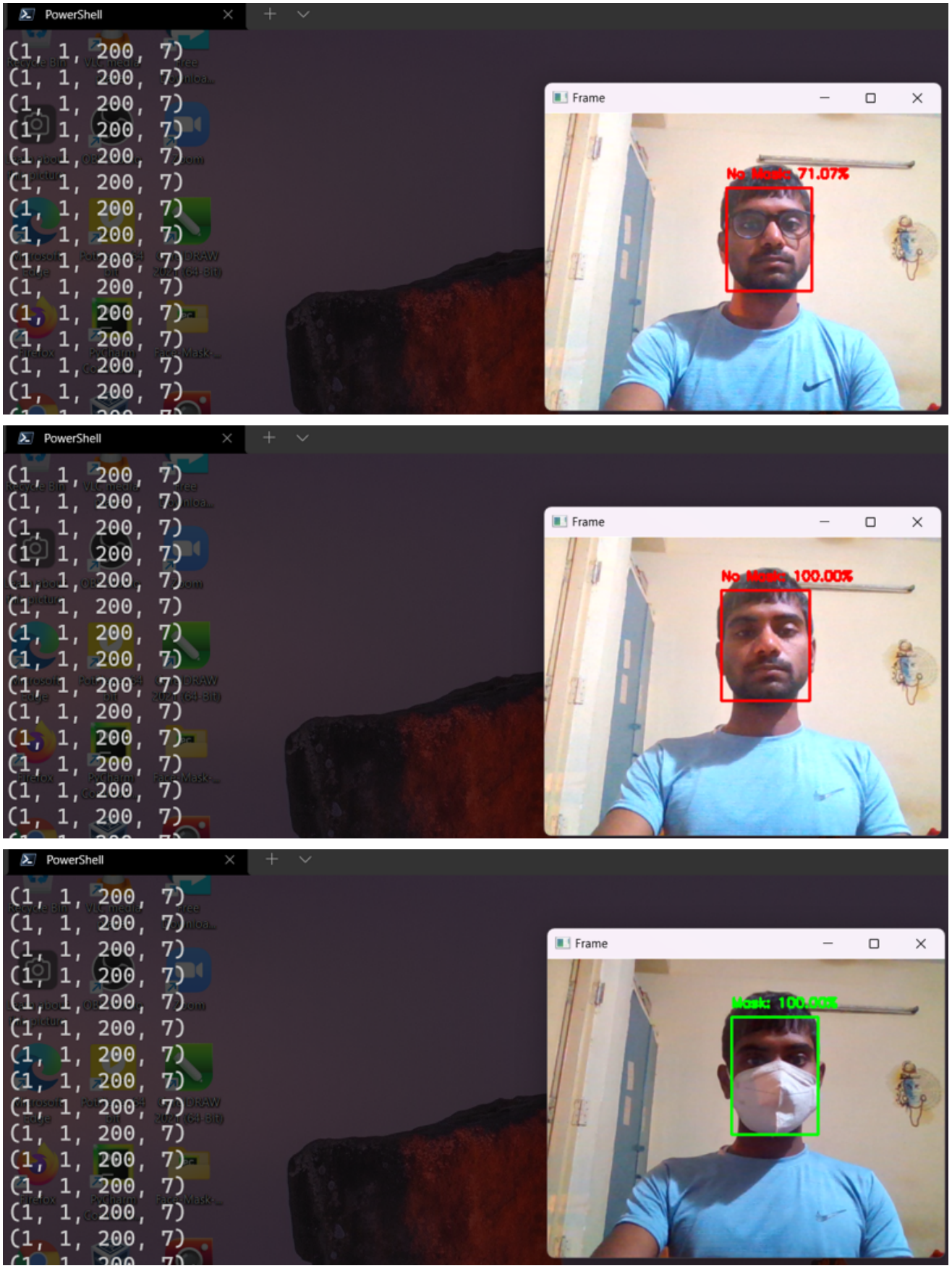


Figure: Above four images shows with mask and without mask along with their accuracy.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE FOR FURTHER DEVELOPMENT

6.1 Conclusion

This project report has presented a study of Face Mask Detection from images or from real time videos using CNN which is the concept of Deep Neural Network (DNN) using Supervised Learning. This model was able to achieve an accuracy of 98.77% with achievement in validation of 93.83% in 20 epochs, these results were achieved when running this model on Jupyter-Notebook on Kaggle and Google Colab. When this model was run on my system, I achieved an accuracy of 99.73% with a loss of 2.32%, and validation accuracy of 99.72% with a validation loss of 2.33% in 20 epochs. Further, COVID-19 could be fought by allowing people to wear face masks while performing biometric authentication. This study presented a useful tool for controlling the spread of the disease. Since the spread of the COVID-19 virus last year, facial recognition with face masks has become increasingly critical. One of our future projects involves detecting social distancing when a face mask is not worn properly and alarming when it is not worn properly.

6.2 Future Scope

The work opens up new avenues of research for the future. Firstly, the proposed technique can be integrated into any high-resolution video surveillance device and is not limited to mask detection only. A second extension of the model is the detection of facial landmarks for biometric purposes while wearing a mask. Face Mask detection can be implement on Respberry pi or arduino for automate the checkups on public places like railway station, airport, tourist places, hospitals, school and colleges and so on.

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.