FACE MASK DETECTION USING CONVOLUTIONAL NEURAL NETWORK AND TRANSFER LEARNING

Project report submitted in fulfilment of the requirement for the degreeof Bachelor of Technology

in

Computer Science and Engineering

By

PRATIKSHA (171326)

UNDER THE SUPERVISION OF

MR. PRATEEK THAKRAL
Assistant Professor (Grade-II)

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

DECLARATION BY CANDIDATE

I hereby declare that the work presented in this report entitled **"Face Mask Detection using Convolutional Neural Network and Transfer Learning"** in fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2021 to June 2021 under the supervision of Mr. Prateek Thakral (Assistant Professor Grade-II , Computer science department).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

[illegible]

(Student's Signature)
Pratiksha 171326

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

[illegible]

(Supervisor's Signature)
Mr. Prateek Thakral
Assistant Professor(Grade-II)
Computer Science Department

ACKNOWLEDGEMENT

We are highly indebted to all the members of the **Computer Science & Engineering**, Jaypee University of Information Technology for their guidance and constant supervision as well as providing necessary information regarding the project and for their support in completing the same.

We would like to express our gratitude to **Mr. Prateek Thakral** for his kind cooperation and encouragement throughout the project as well as his valuable time and attention, which helped us in completion of this work. We would also like to thank our panel members **Dr. Himanshu Jindal** and **Mr. Arvind Kumar** for giving us this opportunity to present our work and providing us with the necessary feedbackto improve this work in future.

Lastly, we would like to acknowledge our Institution, **Jaypee University of Information Technology** for providing us this platform to showcase our work till date.

TABLE OF CONTENT

ABSTRACT

In the new world of coronavirus, multidisciplinary efforts have been organized to slow the spread of the pandemic. The AI community has also been a part of these endeavors. In particular, developments for monitoring social distancing or identifying face masks have made-the-headlines. Businesses are constantly overhauling their existing infrastructure and processes to be more efficient, safe, and usable for employees, customers, and the community. With the ongoing pandemic, it's even more important to have advanced analytics apps and services in place to mitigate risk. For public safety and health, authorities are recommending the use of face masks and coverings to control the spread of COVID-19.

Face masks help diminish the transmission of the infection by meddling with the spread of infection loaded droplets ejected from the nose and mouth. Wearing a face mask is one of the precautionary steps an individual can take to decrease the spread of COVID-19.

Face mask detection systems are now increasingly important, especially in smart hospitals for effective patient care. They're also important in stadiums, airports, warehouses, and other crowded spaces where foot traffic is heavy and safety regulations are critical to safeguarding everyone's health. In this simple project, a video camera detects if an individual is wearing a face mask or not in real-time. We have used a prebuilt cascade classifier that detects faces from the input image and identifies the region of interest, which is then fed as input to our designed CNN. The CNN and Transfer learning detects whether the person is wearing the mask or not. The goal here is to train an AI model that is not only accurate but lightweightand performant for real-time inference on the edge.

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
ANN	Artificial Neural Network
nCoV	Novel CoronaVirus
COVID	CoronaVirus Disease
GPU	Graphics Processing Unit
OpenCV	Open Source Computer Vision Library

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

Chapter 01
INTRODUCTION

1.1 Introduction

In the new world of coronavirus, multidisciplinary efforts have been organized to slow the spread of the pandemic.[1] The AI community has also been a part of these endeavors. In particular, developments for monitoring social distancing or identifying face masks have made-the-headlines. Businesses are constantly overhauling their existing infrastructure and processes to be more efficient, safe, and usable for employees, customers, and the community. With the ongoing pandemic, it's even more important to have advanced analytics apps and services in place to mitigate risk. For public safety and health, authorities are recommending the use of face masks and coverings to control the spread of COVID-19.[3] [5]

Face masks help reduce the transmission of the disease by interfering with the spread of virus-laden droplets ejected from the nose and mouth. Wearing a face mask is one of the precautionary steps an individual can take in order to lessen the spread of COVID-19. [2]

Face mask detection systems are now increasingly important, especially in smart hospitals for effective patient care. They're also important in stadiums, airports, warehouses, and other crowded spaces where foot traffic is heavy and safety regulations are critical to safeguarding everyone's health. Also, the absence of large datasets has made this task more cumbersome and challenging.[4]

1.2 Objective of the Minor Project

In this project, we propose a two-stage CNN architecture and transfer learning, where the first stage detects human faces while the second one uses a lightweight image classifier to classify the faces detected in the first stage as 'With Mask' or 'Without Mask' and draws bounding boxes around them along with the confidence score of the predicted category.

1.3 Motivation of the Major Project

The inspiration of the Project was taken from a post uploaded by Mr. Adrian Rosebrock who produced a tutorial on how to build a real time face mask detector using MobileNetV2. As there was no pre-trained classifier to distinguish faces with and without masks, Adrian trained this model with a dataset provided by one of his readers, Prajna Bhandary who created the dataset artificially by using facial landmarks to apply masks to face images, thus creating morphed image dataset. Although the model generalized pretty well, real images obtained from real-world sources like CCTV or surveillance cameras can be much noisier. In this project, we have used several scraping techniques to gather our own dataset and labelled our dataset accordingly. The dataset used here is more of a real world dataset, and hence generalizes pretty well in real world scenarios. At last we applied inception V3 model to increasing the accuracy of the project and to decrease the curve.

1.4 Technical Requirements

- **System Requirements**

  Computational power of individual machines are not sufficient to train big CNNs, hence the model is trained over cloud GPUs. However, data preprocessing was done on local Machine. To run the final working script, the system needs to meet the following requirements:

  1. **Operating system:** Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
  2. **System architecture:** Windows- 64-bit x86, 32-bit x86; MacOS-64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
  3. Minimum 5 GB disk space to download and install the software requirements.

- **Python**

  Python is an interpreted programming language, both high-level and general-purpose. With its prominent use of substantial white space, Python's design philosophy emphasizes code readability. It aims to help programmers write simple logical code for small and large-scale projects with its language constructs and object-oriented approach.

  Python is typed and garbage-collected dynamically. It supports different paradigms of programming, including structured (specifically, procedural), object-oriented, and functional programming. Because of its comprehensive standard library, Python is sometimes defined as a language that includes batteries.

- **Numpy**

  NumPy, which stands for Numerical Python, is a library consisting of objects in a multidimensional array and a series of processing routines for those arrays. Mathematical and logical operations on arrays can be achieved using NumPy. The fundamentals of NumPy, including its architecture and climate, are explained in this tutorial. It also addresses the different functions of the list, indexing types, etc

- **Pandas**

  Pandas is a software library written for data manipulation and analysis in the Python programming language. It provides data structures and operations for the manipulation of numerical tables and time series, in particular. It is free software which has been published under the BSD three-clause license. The name derives from the word "panel data" an econometric term for data sets that contain multiple time span measurements for the same individuals.

- **Matplotlib**

  Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is the Python programming language plotting library and its NumPy numerical mathematics extension. For embedding plots into applications, it offers an object-oriented API using general-purpose GUI toolkits such as Tkinter, wxPython, Qt, or GTK+.

- **Scikit-learn**

  A free software machine learning library for the Python programming language is Scikit-learn (formerly scikits.learn and also referred to as sklearn). It includes numerous algorithms for classification, regression and clustering, including vector support machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interface with the NumPy and SciPy numerical and scientific libraries of Python.

- **Machine Learning**

  As a sub-domain of AI, ML algorithms can be classified and render a machine or software program intelligent enough to be more precise without needing to be clearly programmable and can forecast performance. The key idea behind the operation of these algorithms is to collect input as a dataset and then learn from the output for the respective inputs. Which helps to predict the performance of the algorithms when they obtain the same domain input. It effectively learns the pattern of similarity between the inputs by which the algorithm is trained and implies an output from the input of the test dataset.

- **Deep Learning**

  Deep learning is a form of machine learning (ML) and artificial intelligence (AI) that mimics the way certain kinds of information are acquired by humans. A significant aspect of data science, which involves statistics and predictive modelling, is deep learning. Data scientists who are charged with the compilation, review and evaluation of vast volumes of data are extremely beneficial; deep learning makes this process quicker and simpler. Deep learning can be thought of as a way of automating predictive analytics at its simplest. While conventional machine learning algorithms are linear, in a hierarchy of growing complexity and abstraction, deep learning algorithms are stacked. Computer programs that use deep learning go through almost the same method to classify the dog as the toddler learning. In the hierarchy, each algorithm applies a nonlinear transformation to its input and uses what it learns as an output to construct a statistical model. Iterations continue until an appropriate degree of precision has been achieved by the production. What profoundly influenced the mark was the number of processing layers through which data could move.

  The learning process is monitored in conventional machine learning, and when asking the computer what kinds of things it should be searching for to determine whether a picture contains a dog or does not contain a dog, the programmer must be extremely precise. This is a laborious method called feature extraction, and the success rate of the computer depends entirely on the ability of the programmer to define a feature set for "dog." correctly. The benefit of deep learning is that the software builds the feature set without control by itself. Not only is unsupervised learning easier, but it is usually more precise.

- **Keras and Tensorflow**

  TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications. TensorFlow provides stable Python and C++ APIs, as well as non-guaranteed backward compatible API for other languages.

Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. It is Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks.

- **OpenCV**

  OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. OpenCV is a library of programming functions mainly aimed at real-time computer vision.

- **Neural Networks (ANN)**

  Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute human brains.

  An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

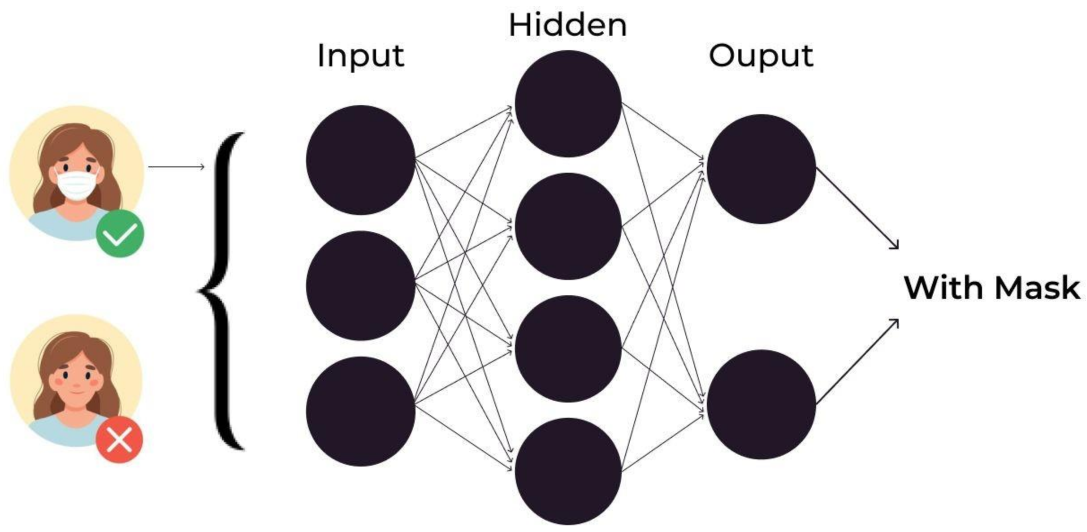


Fig 1.1: Illustration of an ANN

- **Convolutional Neural Network**

  Convolutional Neural Networks (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They have a shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

  Convolutional Neural Network has had groundbreaking results over the past decade in a variety of fields related to pattern recognition; from image processing to voice recognition. The most beneficial aspect of CNNs is reducing the number of parameters in ANN . This achievement has prompted both researchers and developers to approach larger models in order to solve complex tasks, which was not possible with classic ANNs; . The most important assumption about problems that are solved by CNN should not have features which are spatially dependent. In other words, for example, in a face detection application, we do not need to pay attention to where the faces are located in the images. The only concern is to detect them regardless of their position in the given images
  . Another important aspect of CNN, is to obtain abstract features when input propagates toward the deeper layers. For example, in image classification, the edge might be detected in the first layers, and then the

simpler shapes in the second layers, and then the higher level features such as faces in the next layers.[6]

CNNs provide the three basic advantages over the traditional ANNs:

1. Firstly, they have sparse connections instead of fully connected connections which lead to reduced parameters and make CNN’s efficient for processing high dimensional data.
2. Secondly, weight sharing takes place where the same weights are shared across the entire image, causing reduced memory requirements as well as translational invariance. As the same weights are shared across the images, hence if an object occurs in any image it will be detected irrespective of its position in the image.
3. Thirdly, CNN’s use a very important concept of pooling in which the most prominent pixels are propagated to the next layer dropping the rest providing a fixed size output matrix required for classification. [9]

1.5 Project Deployment

The System developed here is meant to be deployed as a software in embedded systems. Hence, a user interface has not been created for the same. Further enhancements can be done by embedding in RaspberryPi's Camera Module.

Chapter 02
LITERATURE SURVEY

The inspiration of the Project was taken from a post uploaded by Mr. Adrian Rosebrock who produced a tutorial on how to build a real time face mask detector using MobileNetV2. MobileNetV2 is a lightweight network, with its biggest advantage being the fact that such lightweight networks can be embedded easily into raspberry pi and camera module. As there was no pre-trained classifier to distinguish faces with and without masks, Adrian trained this model with a dataset provided by one of his readers, Prajna Bhandary who created the dataset artificially by using facial landmarks to apply masks to face images, thus creating morphed image dataset.

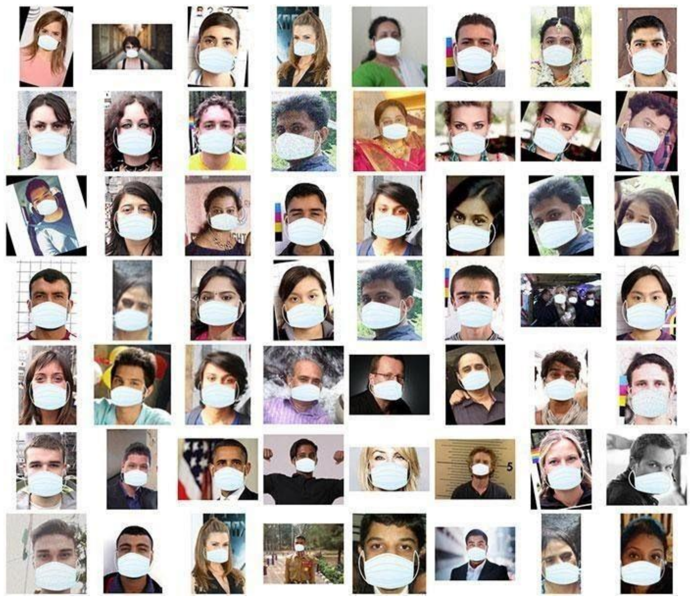


Fig 2.1: Dataset of morphed images created by Prajna Bhandary

Even though it was a synthetic dataset and was built with a single mask type, it seems to generalize pretty well for other kinds of masks. However, real images obtained from real-world sources like CCTV or surveillance cameras can be much noisier. In this project, we have used several scraping techniques to gather our own dataset and labelled our dataset accordingly.

The dataset used here is self-collected via web-scraping (in Python, using beautiful soup) more of a real-world dataset, and hence generalizes pretty well in real world scenarios.

Fig 2.2: Dataset of real images collected via scraping techniques

Our dataset consists of over 5300 images (Excluding augmented data) as compared to a relatively small dataset of around 1400 images, used by Mr. Rosebrock. Also, instead of using MobileNetV2, we have limited our scope to using simple Convolutional Neural Network.

Study of Research Paper: Understanding of a Convolutional Neural Network

(Saad ALBAWI , Tareq Abed MOHAMMED ,Department of Computer Engineering
Faculty of Engineering and Architecture, Istanbul Kemerburgaz University
Istanbul, Turkey)

Elements of a CNN:

- **Convolutional Layers**

  Major advantage of Convolutional layers is that they have sparse connections instead of fully connected connections which lead to reduced parameters and make CNN’s efficient for processing high dimensional data.

  Secondly, weight sharing takes place where the same weights are shared across the entire image, causing reduced memory requirements as well as translational invariance. As the same weights are shared across the images, hence if an object occurs in any image it will be detected irrespective of its position in the image.

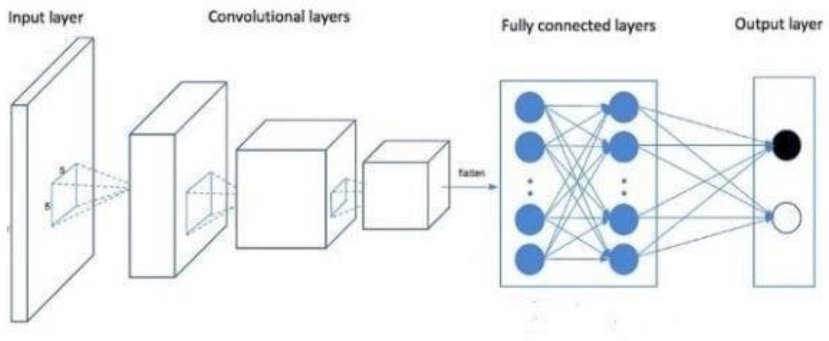


Fig 2.3: Convolutional v/s Fully Connected Layers

- **Non-linearity**

  The next layer after the convolution is non-linearity. The nonlinearity can be used to adjust or cut-off the generated output. This layer is applied in order to saturate the output or limit the generated output. The Rectified Linear Unit (ReLU) has been used more often for the following reasons:

  1. ReLU has simpler definitions in both function and gradient.
  2. The saturated function such as sigmoid and tanh cause problems in the back propagation. As the neural network design is deeper, the gradient signal begins to vanish, which is called the “vanishing gradient”. This happens since the gradient of those functions is very close to zero almost everywhere but the center. However, the ReLU has a constant gradient for the positive input. Although the function is not differentiable, it can be ignored in the actual
  3. The ReLU creates a sparser representation. because the zero in the gradient leads to obtaining a complete zero. However, sigmoid and tanh always have non-zero results from the gradient, which might not be in favor for training.

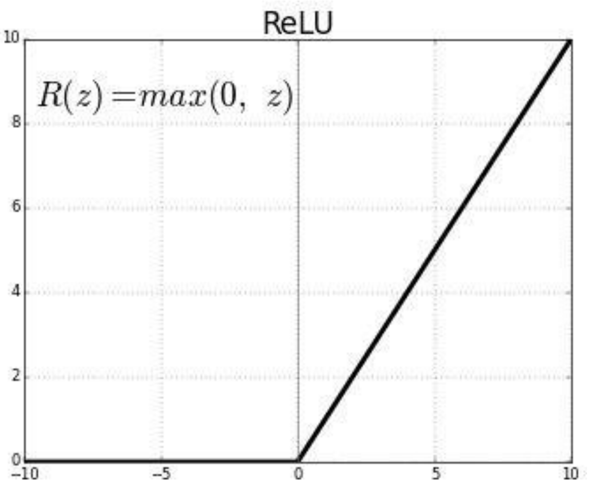


Fig 2.4: ReLU Activation Function

- **Striding**

  CNN has a lot of ways to decrease the trainable parameters, and at the same time reduce some of the side effects. One of these is stride. While performing simple convolution, the next layer's node has lots of overlaps with their neighbors. We can manipulate the overlap by controlling the stride. Fig. 2.5 , shows a given 7×7 image. If we move the filter one node every time, we can have a 5x5 output only. Note that the output of the three left matrices in Fig. 2.5 , have an overlap (and three middle ones together and three right ones also). However, if we move and make every stride 2, then the output will be 3x3. Put simply, not only overlap, but also the size of the output will be reduced.

  For an image of size N×N dimensions and the filter of size F×F, the size of the output image O is given by :

  $$O = 1 + (N-F)/S$$

  Here, N is the input size, F is the filter or kernel size, and S is the stride.

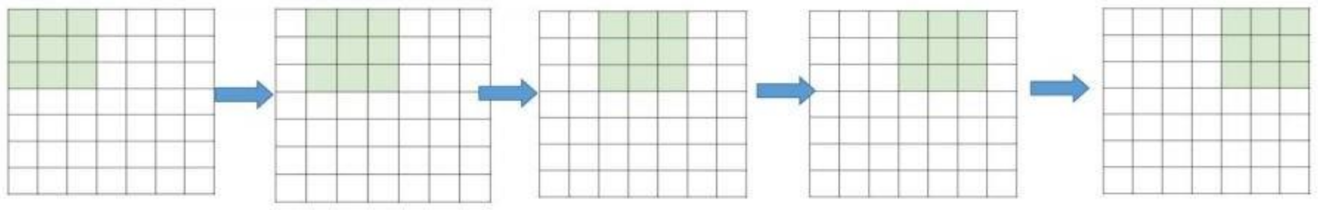


Fig 2.5: For Stride = 1, the filter moves once for each connection

- **Padding**

  One of the drawbacks of the convolution step is the loss of information that might exist on the border of the image. Because they are only captured when the filter slides, they never have the chance to be seen. A very simple, yet efficient method to resolve the issue is to use zero-padding. The other benefit of zero padding is to manage the output size. For example, in Fig. 2.5 , with N=7 and F=3 and stride 1, the output will be 5×5 (which shrinks from a 7×7 input).

  However, by adding one zero-padding, the output will be 7×7, which is exactly the same as the original input (The actual N now becomes 9). The modified formula including zero-padding can be given as:

  $$O = 1 + (N+2P-F)/S$$

  Where P is the number of the layers of the zero-padding, N is the input size, F is the filter size, and S is the stride size.

- **Features of CNN**

  Firstly, they have sparse connections instead of fully connected connections which lead to reduced parameters and make CNN's efficient for processing high dimensional data.

  Secondly, weight sharing takes place where the same weights are shared across the entire image, causing reduced memory requirements as well as translational invariance. As the same weights are shared across the images, hence if an object occurs in any image it will be detected irrespective of its position in the image.

  Thirdly, CNN's use a very important concept of pooling in which the most prominent pixels are propagated to the next layer dropping the rest providing a fixed size output matrix required for classification. [9]

- **Pooling Layers**

  The main idea of pooling is down-sampling in order to reduce the complexity for further layers. In the image processing domain, it can be considered as similar to reducing the resolution. Pooling does not affect the number of filters. Max-pooling is one of the most common types of pooling methods. It partitions the image to sub-region rectangles, and it only returns the maximum value of the inside of that sub-region. One of the most common sizes used in max-pooling is 2×2. As in Fig. 2.6 , when pooling is performed in the top-left 2×2 blocks (pink area), it moves 2 and focuses on the top-right part. This means that stride 2 is used in pooling. To avoid down-sampling, stride 1 can be used, which is not common. It should be considered that down-sampling does not preserve the position of the information. Therefore, it should be applied only when the presence of information is important (rather than spatial information). Moreover, pooling can be used with non-equal filters and strides to improve the efficiency.

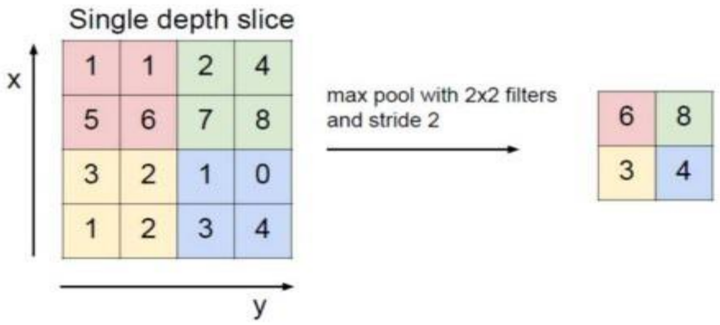


Fig 2.6: Dataset of real images collected via scraping techniques

- **Fully-Connected Layers**

  The fully-connected layer is similar to the way that neurons are arranged in a traditional neural network. Therefore, each node in a fully-connected layer is directly connected to every node in both the previous and in the next layer. Each of the nodes in the last frames in the pooling layer are connected as a vector to the first layer from the fully-connected layer. These are the most parameters used with the CNN within these layers, and take a long time in training. The major drawback of a fully-connected layer, is that it includes a lot of parameters that need complex computations in training examples. Therefore, we try to eliminate the number of nodes and connections. The removed nodes and connection can be satisfied by using the dropout technique [7] . For example, LeNet and AlexNet designed a deep and wide network while keeping the computational complex constant. [11]

Chapter 03
SYSTEM DEVELOPMENT AND PERFORMANCE ANALYSIS

3.1 Data Warehousing and Data Preprocessing

Data warehousing is the process of constructing and using a data warehouse. A data warehouse is constructed by integrating data from multiple heterogeneous sources that support analytical reporting, structured and/or ad hoc queries, and decision making. Sources of Data:

- Kaggle's dataset for real v/s fake face image detection.
- Prajna Bhandary's dataset of morphed images.
- Some other GitHub Repositories.
- Uncleaned raw images scrapped from various other sources. (Web Scraping using BeautifulSoup)

Data preprocessing is a data mining technique used to transform the raw data in a useful and efficient format. This helps in reducing the complexity and increases the accuracy of the applied algorithm.

- **Conversion of images from RGB to grayscale-** In the specified problem statement, color isn't necessary to recognize and interpret an image i.e. Grayscale is good enough to extract useful information and reduce computational complexity.

- **Resizing images-** One important constraint that exists in CNN, is the need to resize the images in the dataset to a unified dimension i.e. images must be preprocessed and scaled to have identical widths and heights before fed to the learning algorithm.

- **Data Augmentation-** Another common pre-processing technique involves augmenting the existing dataset with perturbed versions of the existing images by performing transformations such as scaling, rotations, etc. to enlarge the dataset and expose the classifier to a wide variety of variations of the images. It makes the model more robust to slight variations, and hence prevents the model from overfitting.

- **Normalizing image inputs-** Data normalization ensures that each input image follows a similar data distribution. An alternative to this is standardization, which causes the input dataset to follow standard normal distribution.

Description of the finally created dataset:
Balanced dataset of *5,300 images* with two classes: **with_mask** and **without_mask**.

3.2 Training the Network

Various versions of improved Network are as follows:

3.2.1 Initially designed Network

The initially designed network consisted of two Convolutional layers, two max pooling layers, one flattening and one densely connected layer.

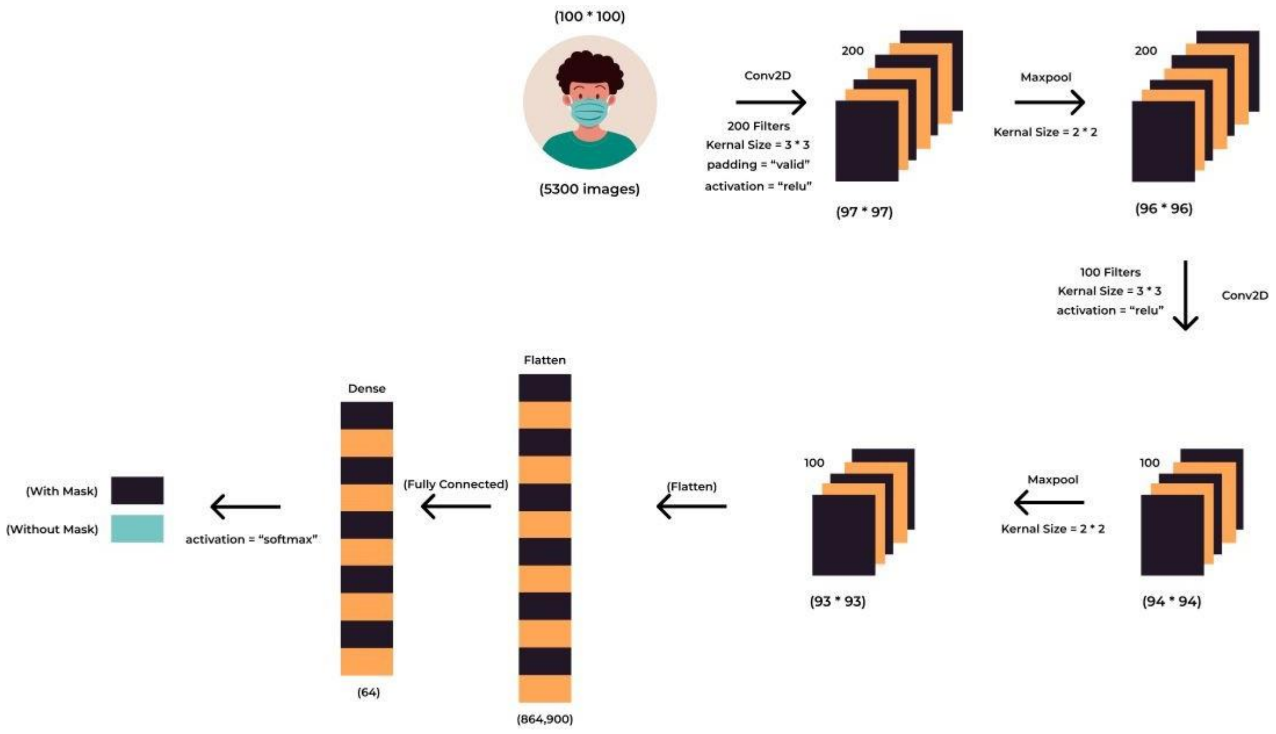
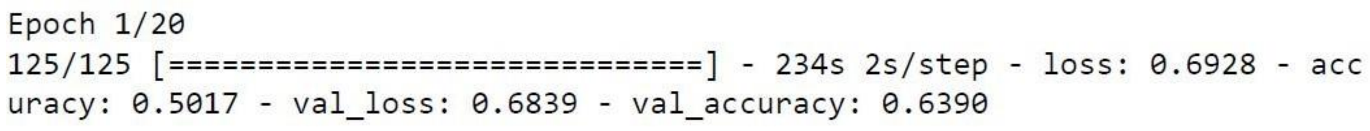


Fig 3.1: Initial Network Architecture

Number of Epochs: 20
Learning rate: 1e-3
Batch Size: 32

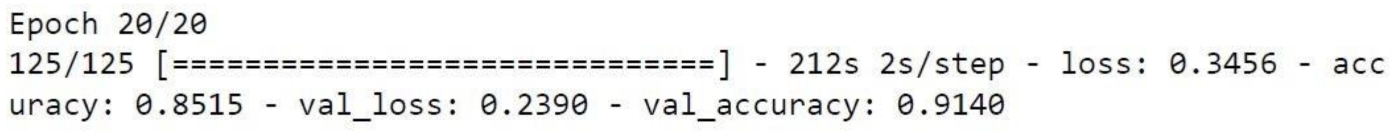
Train-Test Split: 70% Training; 20% Validation; 10% Test

Started With:



```
Epoch 1/20
125/125 [==============================] - 234s 2s/step - loss: 0.6928 - acc
uracy: 0.5017 - val_loss: 0.6839 - val_accuracy: 0.6390
```

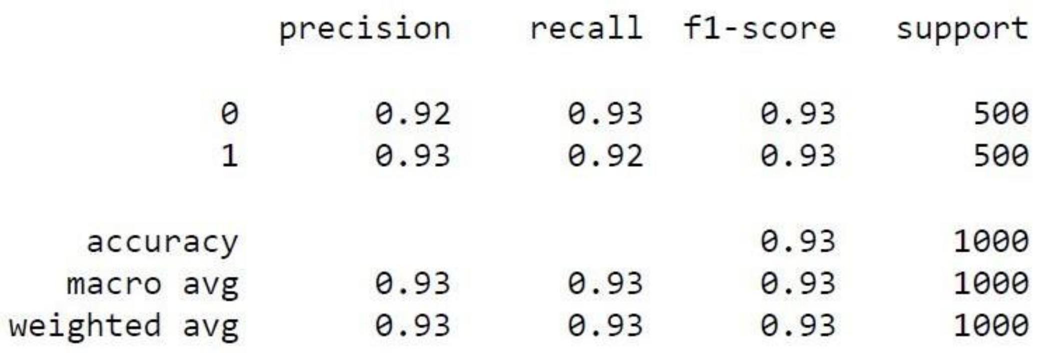
Ended On:



```
Epoch 20/20
125/125 [==============================] - 212s 2s/step - loss: 0.3456 - acc
uracy: 0.8515 - val_loss: 0.2390 - val_accuracy: 0.9140
```

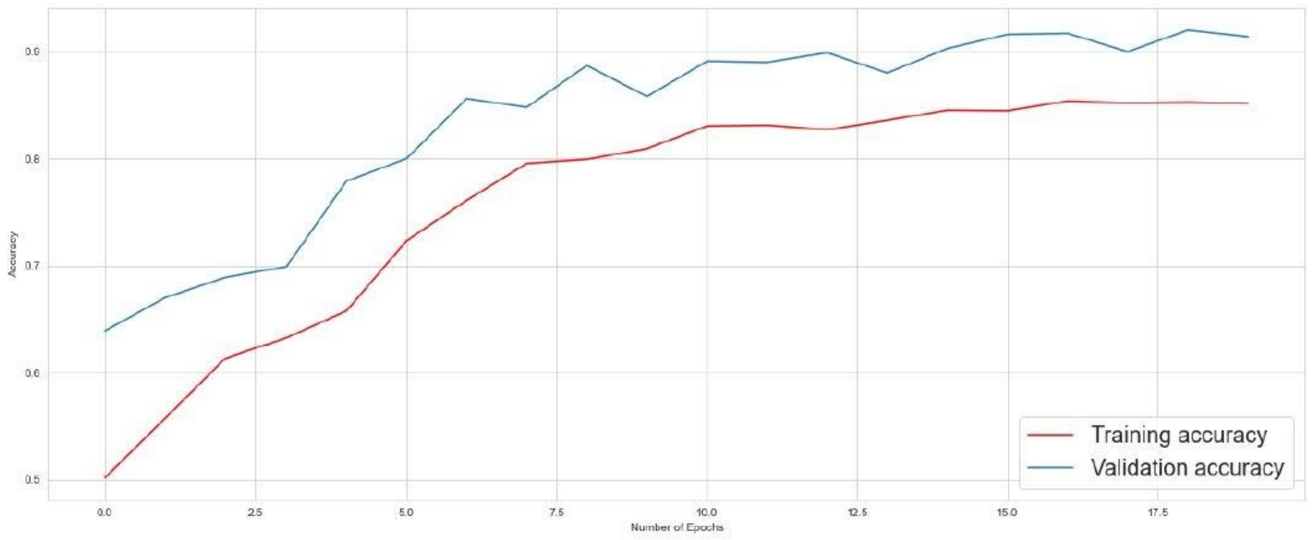
Validation Accuracy: 91.4% (Good)
Validation Loss: 23.9% (VERY HIGH)

Table 3.1: Phase 1 Classification Report



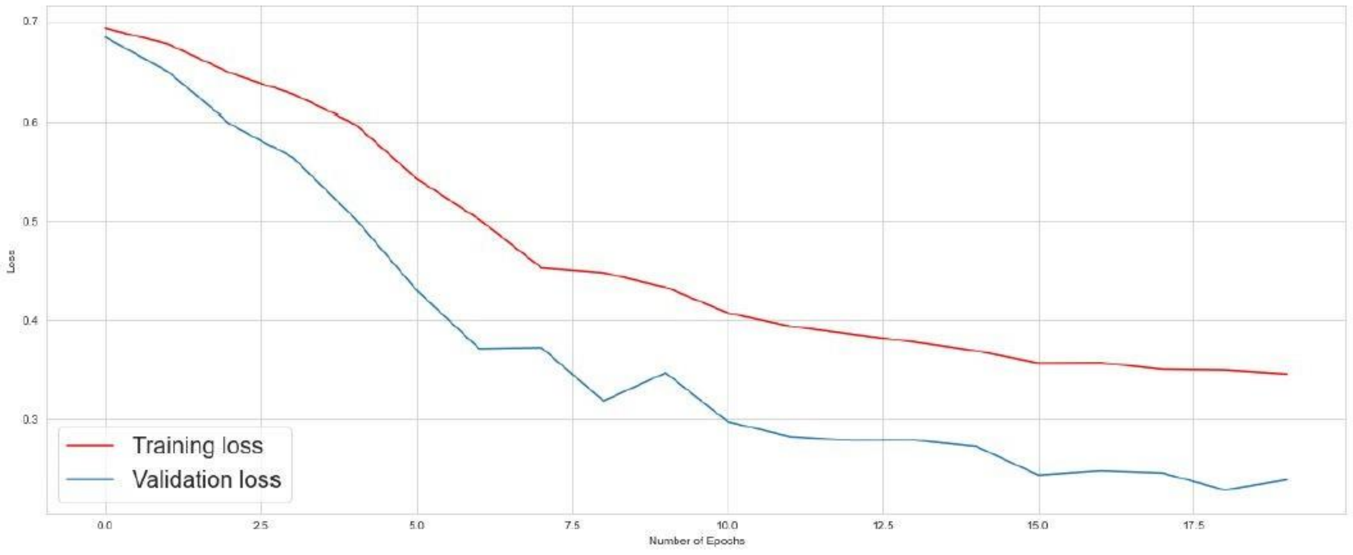
```
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       500
           1       0.93      0.92      0.93       500

    accuracy                           0.93      1000
   macro avg       0.93      0.93      0.93      1000
weighted avg       0.93      0.93      0.93      1000
```

Accuracy Curve:



Graph 3.1: Accuracy Curve

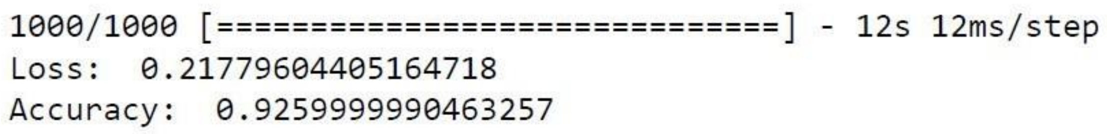
Loss Curve:



Graph 3.2: Loss Curve

Since Validation Accuracy was better than Training accuracy, we say that our model generalized well over the dataset. It appeared that the model suffered from a high bias problem.
To resolve this issue, we increased the size of the network.

Performance on the test data verified our conclusions:



```
1000/1000 [==============================] - 12s 12ms/step
Loss:  0.21779604405164718
Accuracy:  0.9259999990463257
```

Validation Accuracy: 92.5% (Good)
Validation Loss: 21.7% (VERY HIGH)

3.2.2 Second Phase of the designed Network

After further changes, our improvised network consisted of three Convolutional layers, three max pooling layers, one flattening and one densely connected layer.

Also, we altered the train test split to get some better insights.

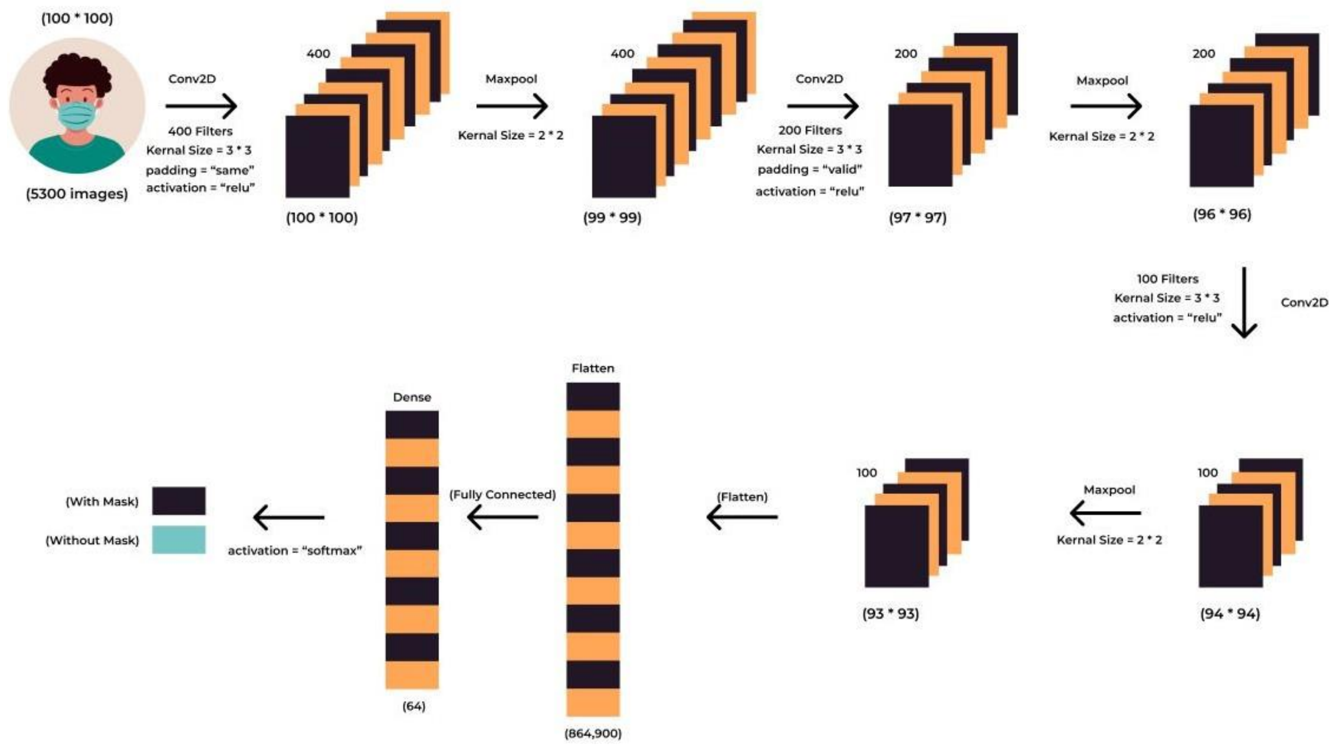
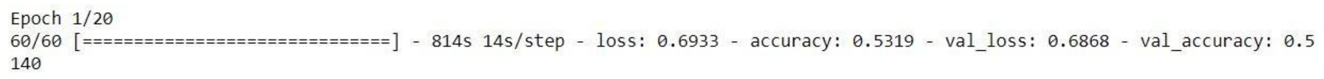


Fig 3.2: Phase 2 Network Architecture

Number of Epochs: 20
Learning rate: 1e-3
Batch Size: 32

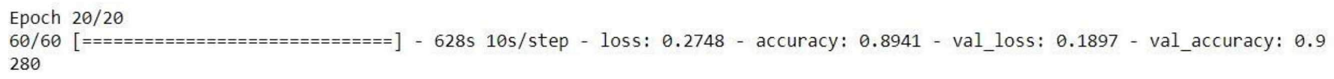
Train-Test Split: 72.5% Training; 12.75% Validation; 15% Test

Started With:



```
Epoch 1/20
60/60 [==============================] - 814s 14s/step - loss: 0.6933 - accuracy: 0.5319 - val_loss: 0.6868 - val_accuracy: 0.5
140
```

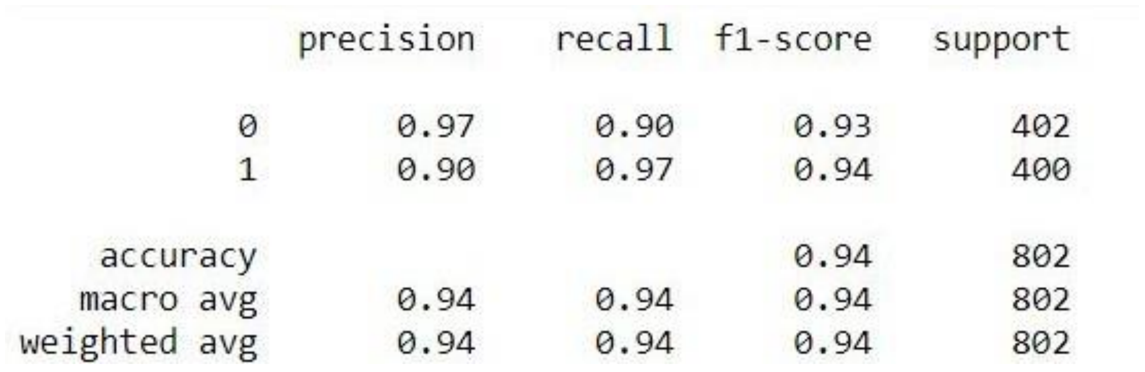
Ended On:



```
Epoch 20/20
60/60 [==============================] - 628s 10s/step - loss: 0.2748 - accuracy: 0.8941 - val_loss: 0.1897 - val_accuracy: 0.9
280
```

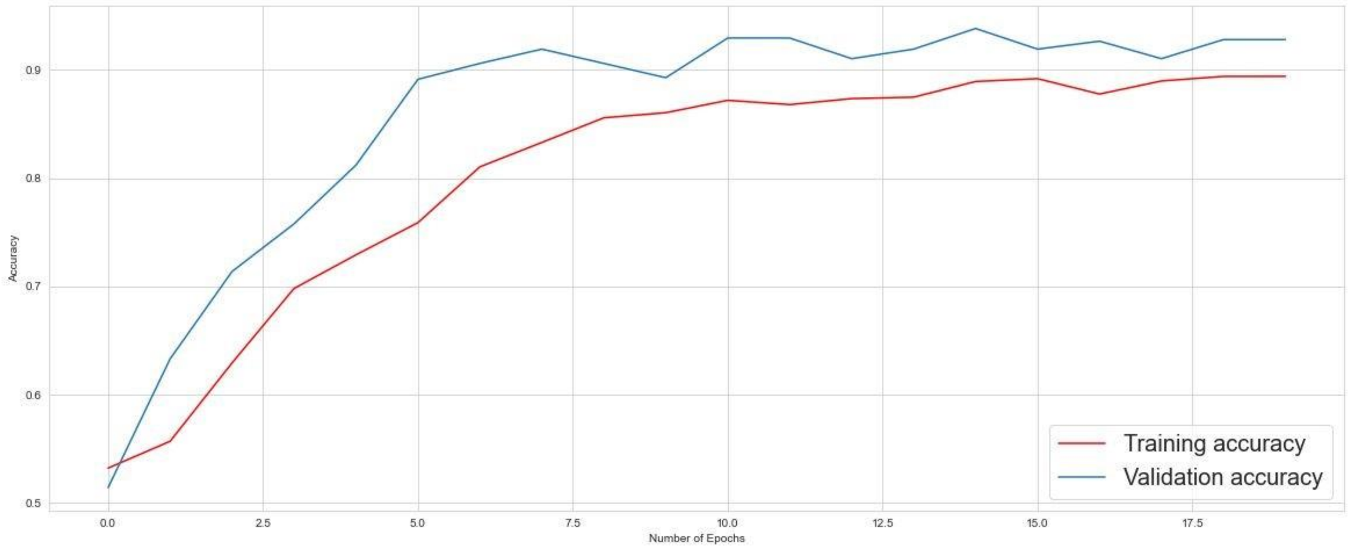
Validation Accuracy: 92.8% (Good)
Validation Loss: 18.97% (VERY HIGH)

Table 3.2: Phase 2 Classification Report



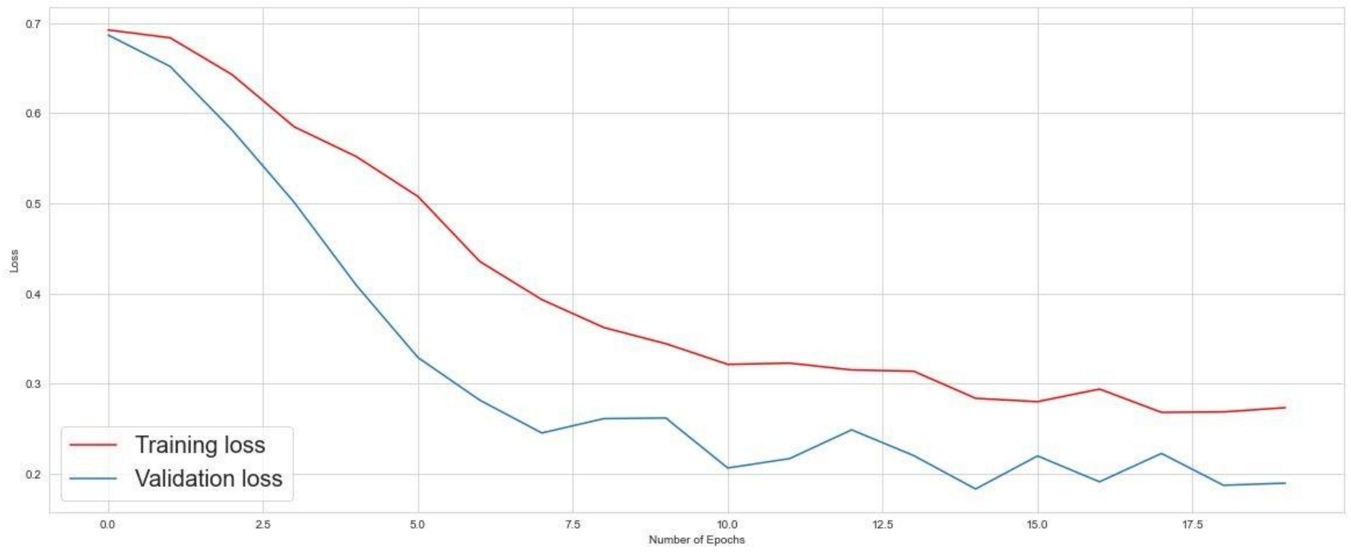
```
              precision    recall  f1-score   support

           0       0.97      0.90      0.93       402
           1       0.90      0.97      0.94       400

    accuracy                           0.94       802
   macro avg       0.94      0.94      0.94       802
weighted avg       0.94      0.94      0.94       802
```

Accuracy Curve:



Graph 3.3: Accuracy Curve

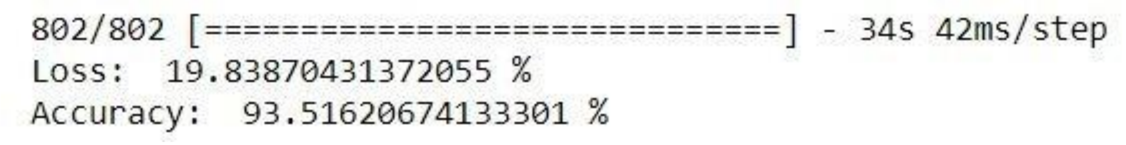
Loss Curve:



Graph 3.4: Loss Curve

On increasing the size of the network, loss is reduced by a small amount, so we find other ways to reduce the bias. High value of loss motivated us to the number of epochs. Also, we tried to get better accuracy by increasing the batch size.

Performance on the test data:



```
802/802 [==============================] - 34s 42ms/step
Loss:  19.83870431372055 %
Accuracy:  93.51620674133301 %
```

Validation Accuracy: 93.5% (Good)
Validation Loss: 19.83% (VERY HIGH)

3.2.3 Third Phase of the designed Network

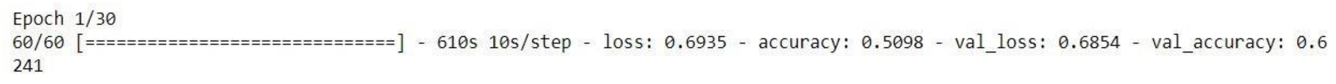
The design and size of the network remained the same, other hyperparameters were altered:

- Decreased learning rate.
- Increased the number of Epochs.
- Increased batch size.
- Expanded dataset.

Number of Epochs: 30
Learning rate: 1e-4
Batch Size: 64

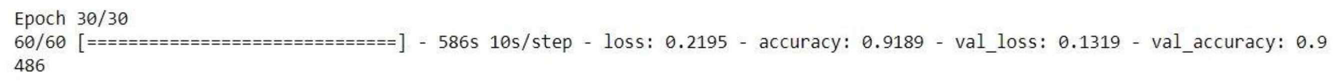
Train-Test Split: 72.5% Training; 12.75% Validation; 15% Test

Started With:



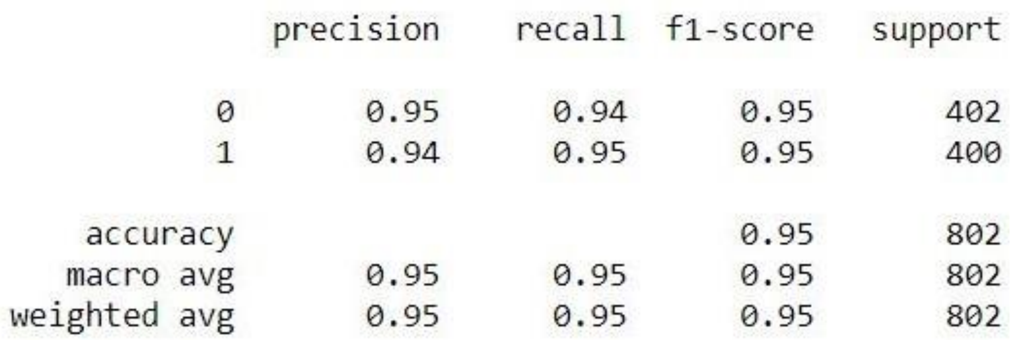
```
Epoch 1/30
60/60 [==============================] - 610s 10s/step - loss: 0.6935 - accuracy: 0.5098 - val_loss: 0.6854 - val_accuracy: 0.6
241
```

Ended On:



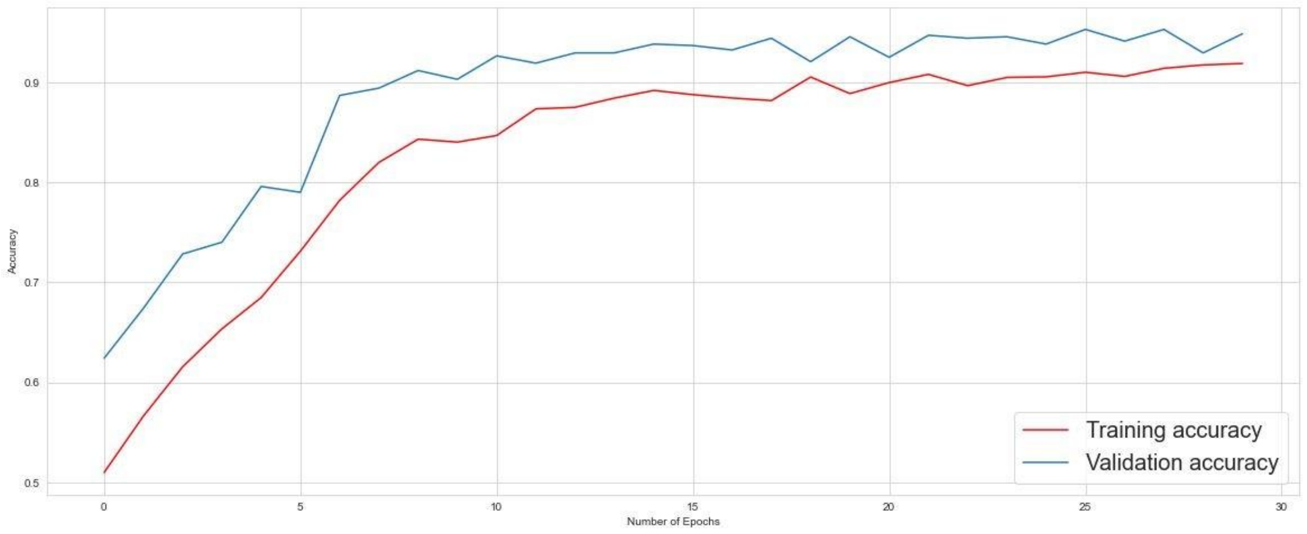
```
Epoch 30/30
60/60 [==============================] - 586s 10s/step - loss: 0.2195 - accuracy: 0.9189 - val_loss: 0.1319 - val_accuracy: 0.9
486
```

Table 3.3: Phase 3 Classification Report



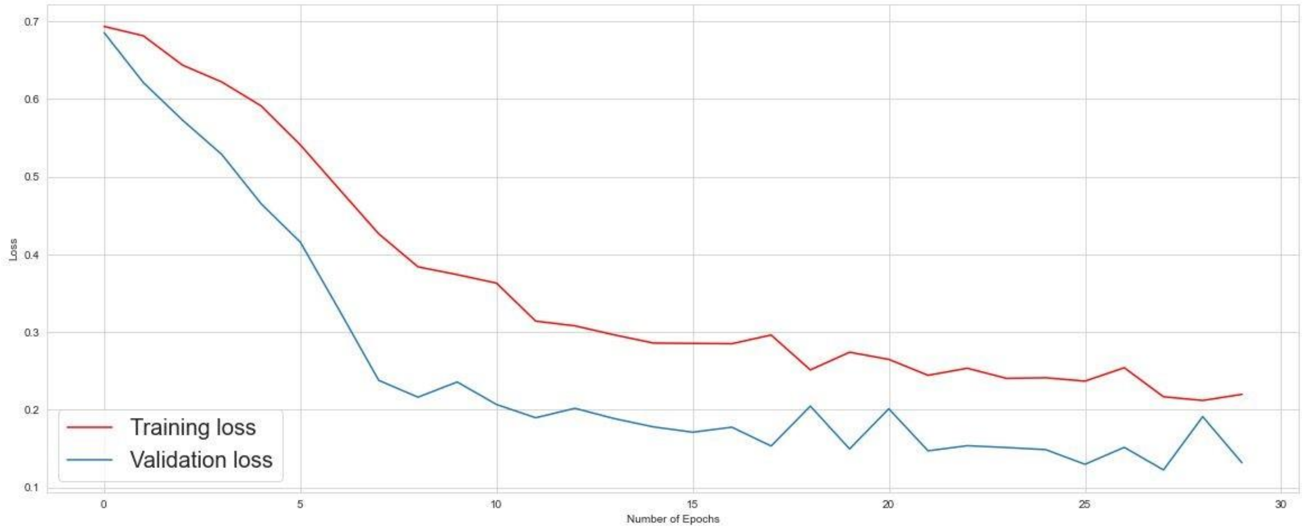
```
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       402
           1       0.94      0.95      0.95       400

    accuracy                           0.95       802
   macro avg       0.95      0.95      0.95       802
weighted avg       0.95      0.95      0.95       802
```

Accuracy Curve:



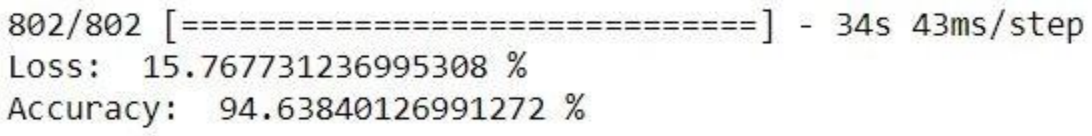
Graph 3.5: Accuracy Curve

Loss Curve:



Graph 3.6: Loss Curve

Performance on the test data:



```
802/802 [==============================] - 34s 43ms/step
Loss:  15.767731236995308 %
Accuracy:  94.63840126991272 %
```

Loss is significantly high.

Probably the learning rate is very low, which is why the gradient descent does not converge.

Also, the number of epochs need to be increased to converge GD to a minimum.

3.2.4 Final Model

Final changes in the model architecture included an additional convolutional and max pooling layers. With increased number of epochs, learning rate and batch size, the model showed significantly better results.

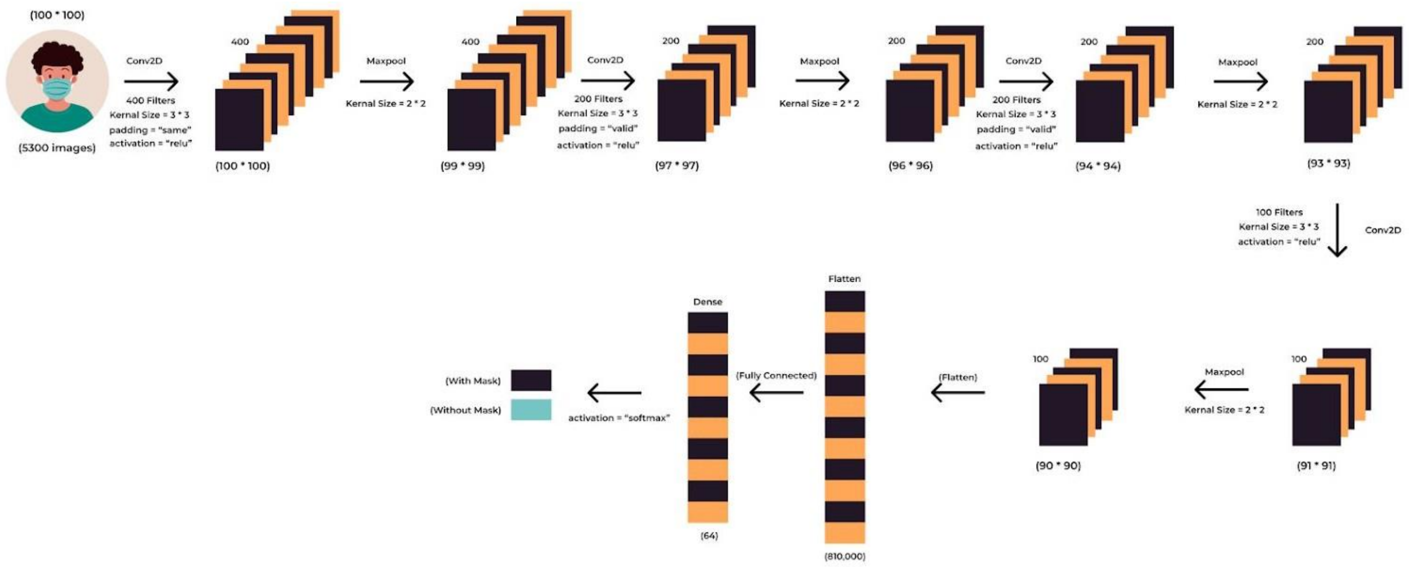
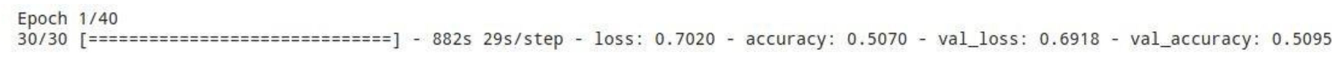


Fig 3.3: Network Architecture of the Final Model

Number of Epochs: 40
Learning rate: 1e-3
Batch Size: 128

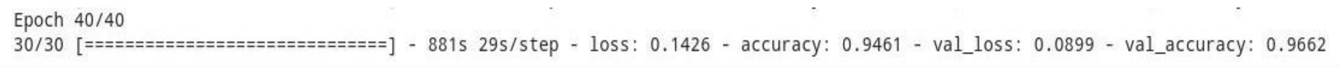
Train-Test Split: 72.5% Training; 12.75% Validation; 15% Test

Started With:



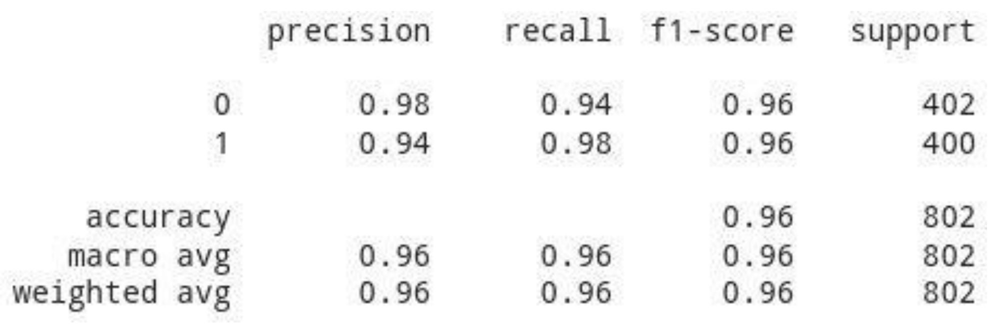
```
Epoch 1/40
30/30 [==============================] - 882s 29s/step - loss: 0.7020 - accuracy: 0.5070 - val_loss: 0.6918 - val_accuracy: 0.5095
```

Ended On:



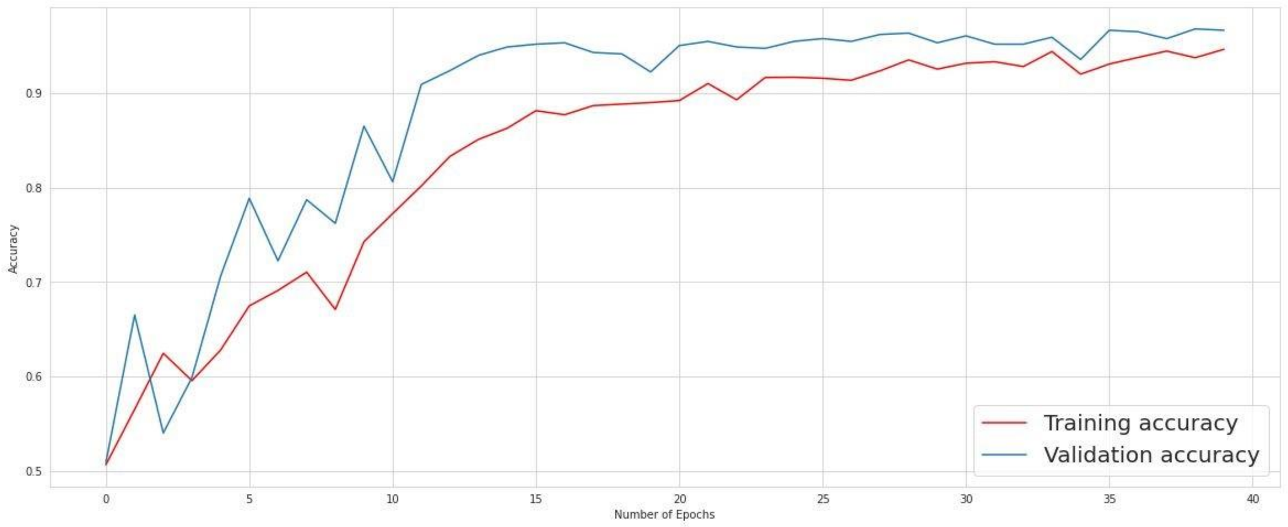
```
Epoch 40/40
30/30 [==============================] - 881s 29s/step - loss: 0.1426 - accuracy: 0.9461 - val_loss: 0.0899 - val_accuracy: 0.9662
```

Table 3.4: Phase 4 Classification Report



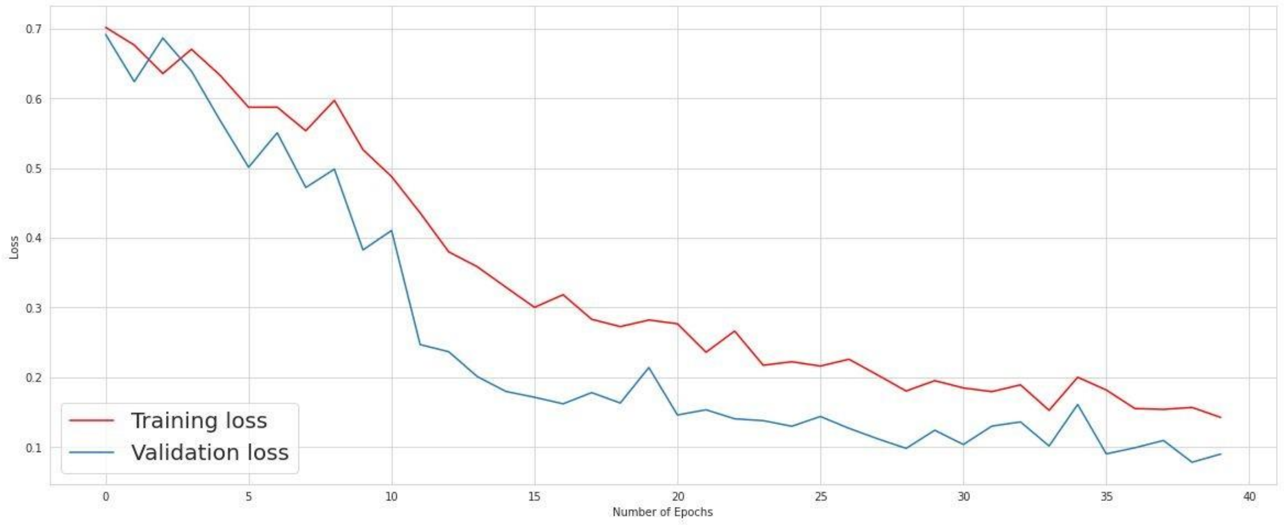
```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96       402
           1       0.94      0.98      0.96       400

    accuracy                           0.96       802
   macro avg       0.96      0.96      0.96       802
weighted avg       0.96      0.96      0.96       802
```

Accuracy Curve:



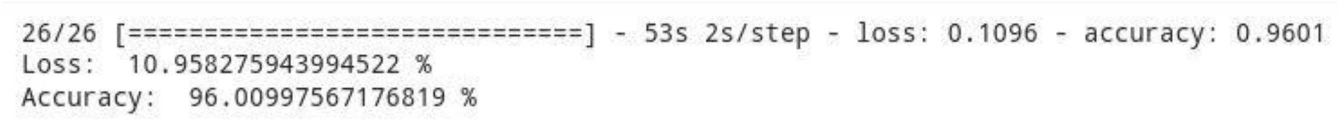
Graph 3.7: Accuracy Curve

Loss Curve:



Graph 3.8: Loss Curve

Performance on the test data:



```
26/26 [==============================] - 53s 2s/step - loss: 0.1096 - accuracy: 0.9601
Loss:  10.958275943994522 %
Accuracy:  96.00997567176819 %
```

Loss is significantly reduced to about 11% on TEST DATA.
Accuracy improved to around 96% on TEST DATA.

3.3 Face Mask Detection

In this project, we have used a prebuilt cascade classifier from Open Source Computer Vision Library, that detects faces from the input image and identifies the region of interest. The cascade classifier falls a little short when it comes to accuracy, however it works well in real-time because of it's excellent frame rate of 15fps, which is pretty quick for real-time applications. Also, since it is lightweight, it is easily deployable into modules of embedded systems. The region of interest identified by the cascade classifier is then rescaled to 100*100 size, which is then fed as input to the CNN. The CNN detects whether the person is wearing the mask or not.

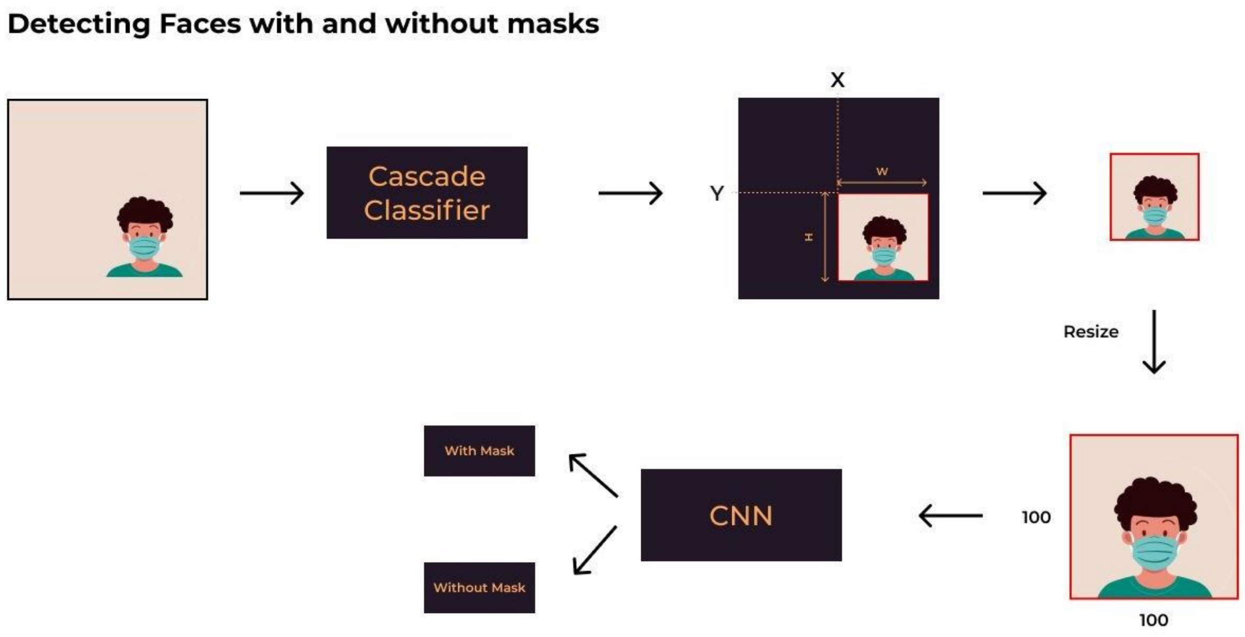


Fig 3.4: Flow Diagram of the Project

3.4 Flow graph of the Major Project Problem

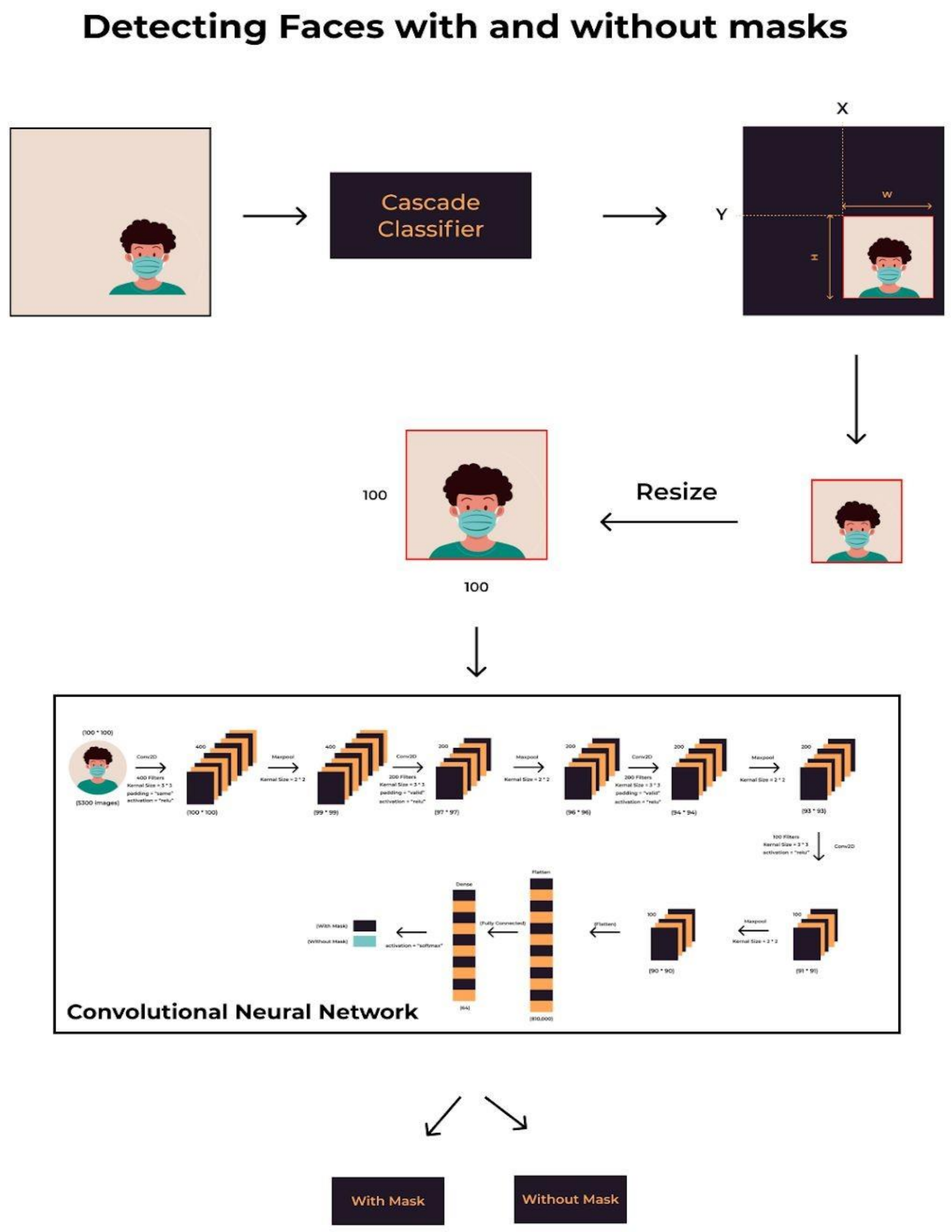
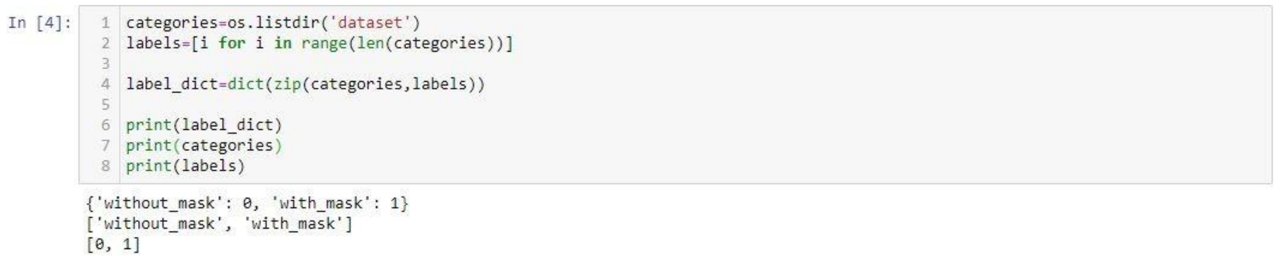


Fig 3.5: Complete Flow Diagram of the Project

3.5 Code Snippets

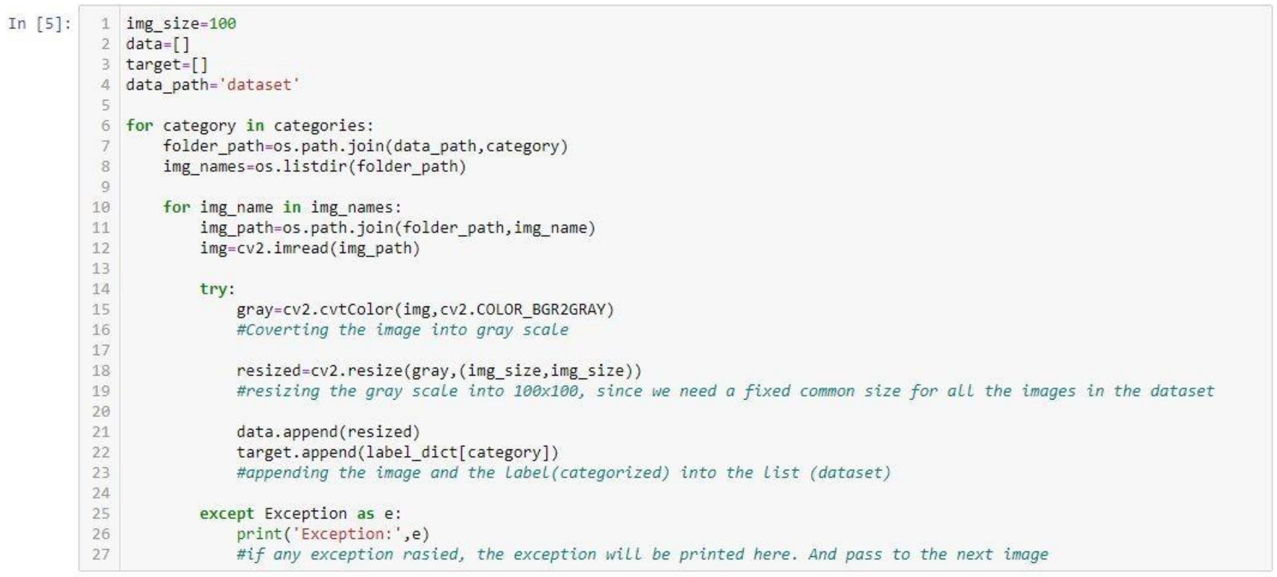
3.5.1 Data Preprocessing



In [4]:

```python
categories=os.listdir('dataset')
labels=[i for i in range(len(categories))]

label_dict=dict(zip(categories,labels))

print(label_dict)
print(categories)
print(labels)
```

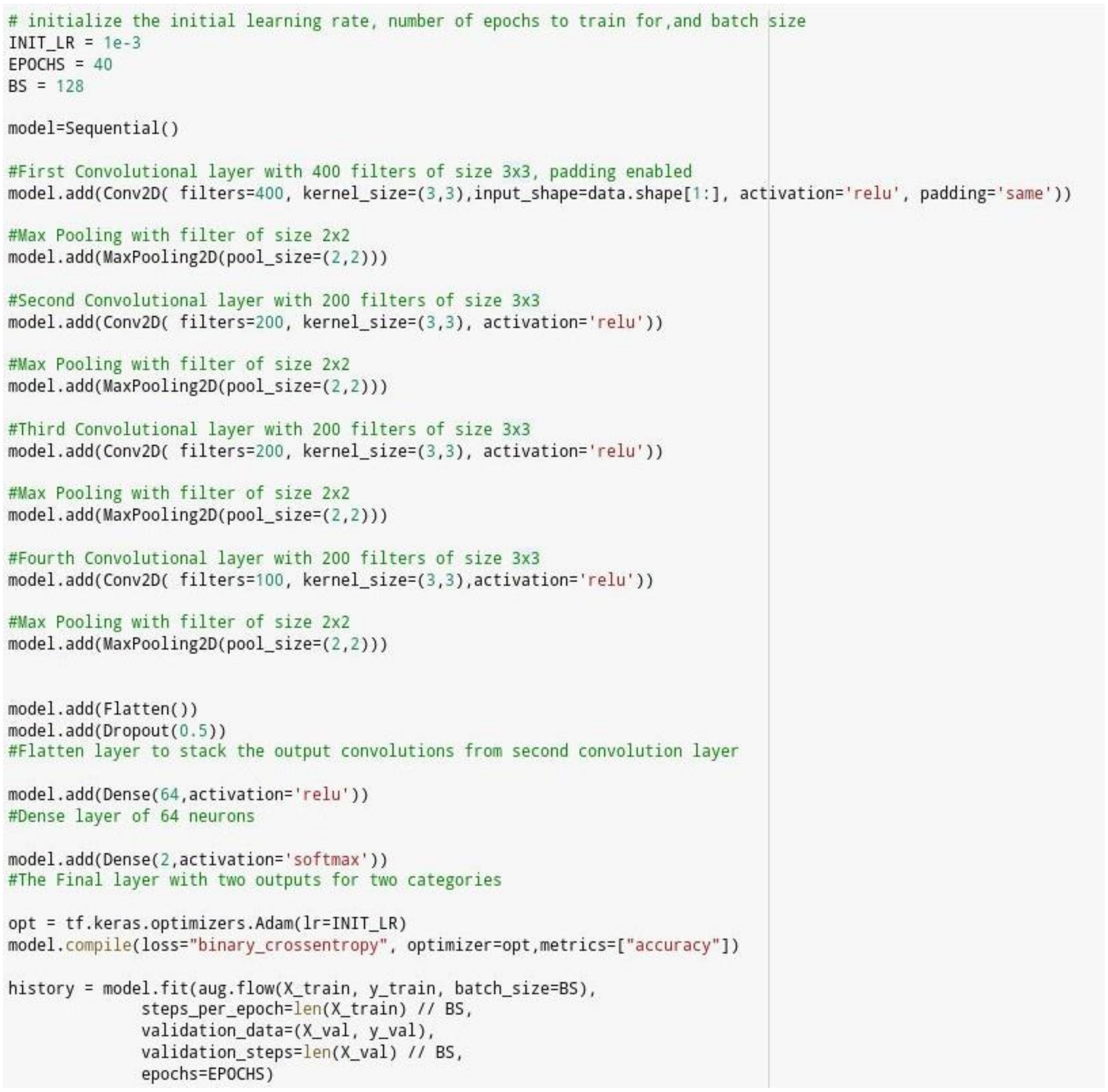
```
{'without_mask': 0, 'with_mask': 1}
['without_mask', 'with_mask']
[0, 1]
```



In [5]:

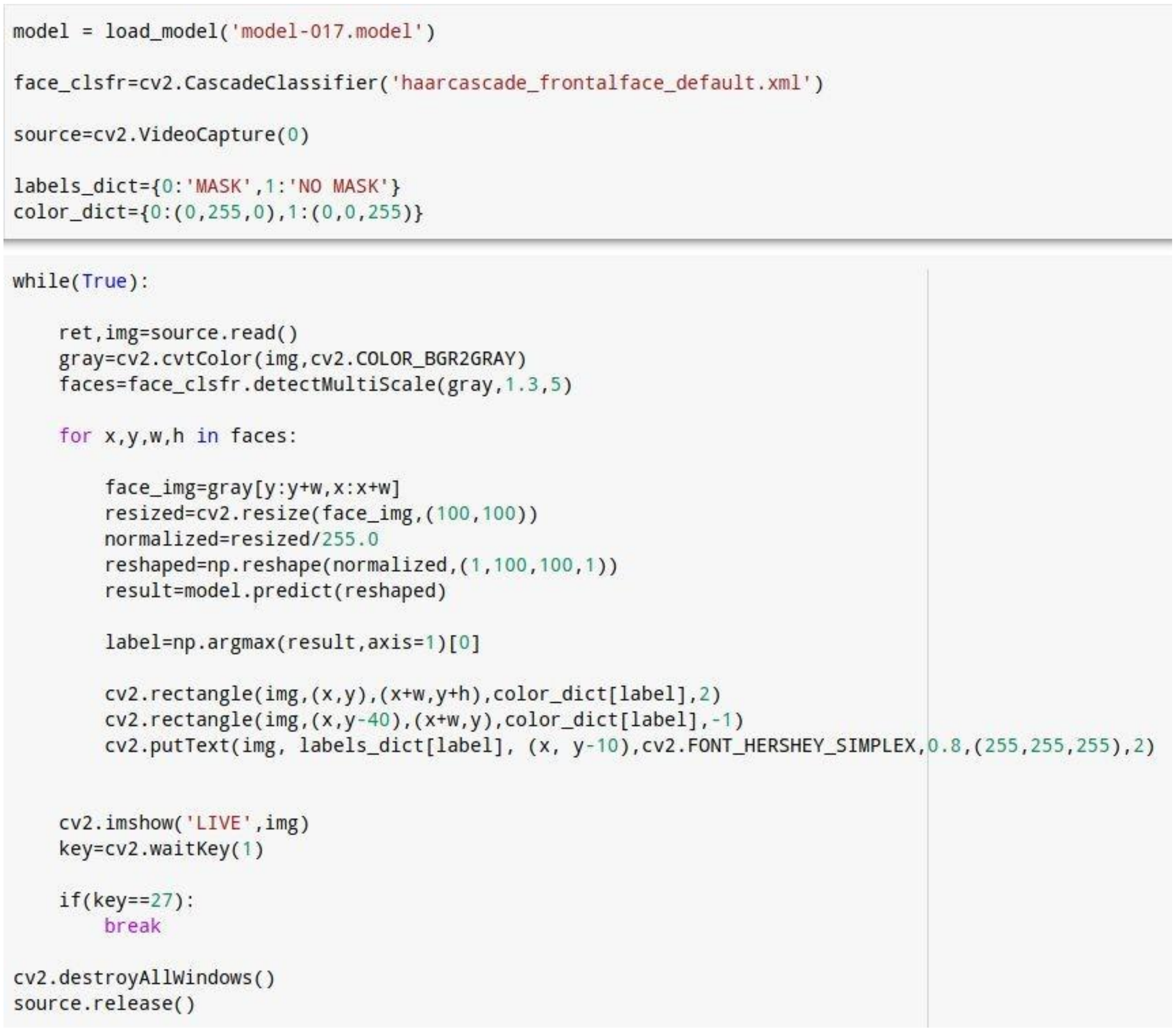
```python
img_size=100
data=[]
target=[]
data_path='dataset'

for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #Coverting the image into gray scale

            resized=cv2.resize(gray,(img_size,img_size))
            #resizing the gray scale into 100x100, since we need a fixed common size for all the images in the dataset

            data.append(resized)
            target.append(label_dict[category])
            #appending the image and the label(categorized) into the list (dataset)

        except Exception as e:
            print('Exception:',e)
            #if any exception rasied, the exception will be printed here. And pass to the next image
```

3.5.2 Training the Network



```
# initialize the initial learning rate, number of epochs to train for,and batch size
INIT_LR = 1e-3
EPOCHS = 40
BS = 128

model=Sequential()

#First Convolutional layer with 400 filters of size 3x3, padding enabled
model.add(Conv2D( filters=400, kernel_size=(3,3),input_shape=data.shape[1:], activation='relu', padding='same'))

#Max Pooling with filter of size 2x2
model.add(MaxPooling2D(pool_size=(2,2)))

#Second Convolutional layer with 200 filters of size 3x3
model.add(Conv2D( filters=200, kernel_size=(3,3), activation='relu'))

#Max Pooling with filter of size 2x2
model.add(MaxPooling2D(pool_size=(2,2)))

#Third Convolutional layer with 200 filters of size 3x3
model.add(Conv2D( filters=200, kernel_size=(3,3), activation='relu'))

#Max Pooling with filter of size 2x2
model.add(MaxPooling2D(pool_size=(2,2)))

#Fourth Convolutional layer with 200 filters of size 3x3
model.add(Conv2D( filters=100, kernel_size=(3,3),activation='relu'))

#Max Pooling with filter of size 2x2
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Flatten())
model.add(Dropout(0.5))
#Flatten layer to stack the output convolutions from second convolution layer

model.add(Dense(64,activation='relu'))
#Dense layer of 64 neurons

model.add(Dense(2,activation='softmax'))
#The Final layer with two outputs for two categories

opt = tf.keras.optimizers.Adam(lr=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])

history = model.fit(aug.flow(X_train, y_train, batch_size=BS),
              steps_per_epoch=len(X_train) // BS,
              validation_data=(X_val, y_val),
              validation_steps=len(X_val) // BS,
              epochs=EPOCHS)
```

3.5.3 Real Time Face Mask Detection



```
model = load_model('model-017.model')

face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

source=cv2.VideoCapture(0)

labels_dict={0:'MASK',1:'NO MASK'}
color_dict={0:(0,255,0),1:(0,0,255)}
```

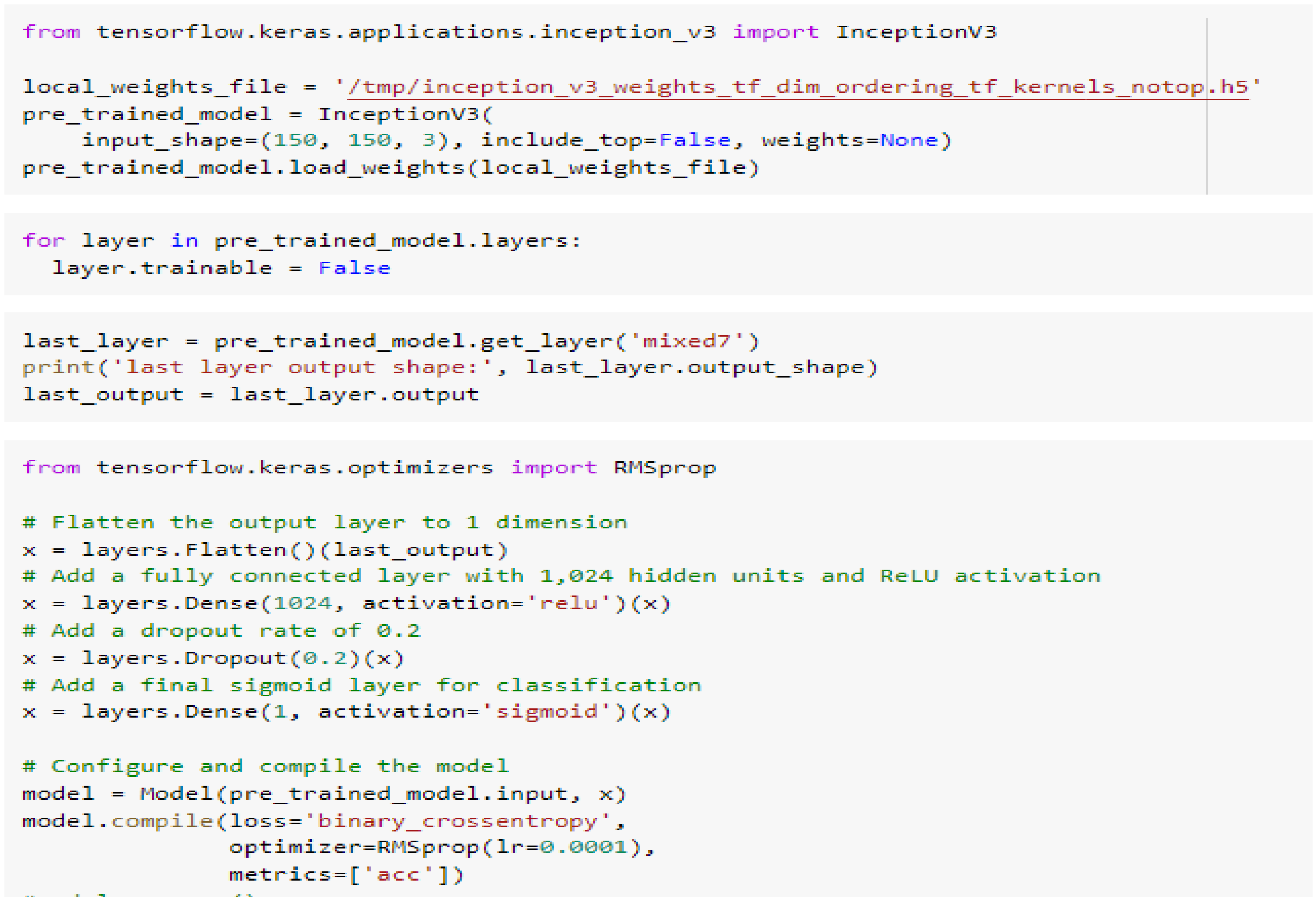
```
while(True):

    ret,img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=face_clsfr.detectMultiScale(gray,1.3,5)

    for x,y,w,h in faces:

        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(100,100))
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,100,100,1))
        result=model.predict(reshaped)

        label=np.argmax(result,axis=1)[0]

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```

3.6 Transfer Learning :

- In Transfer Learning, we make use of the knowledge gained while solving one problem and applying it to a different but related problem.
- When we train the network on a large dataset (for example: ImageNet) , we train all the parameters of the neural network and therefore the model is learned. It may take hours on your GPU.
- We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the original dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset.

◦ If the new dataset is very small, it’s better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers . Retrain only the new layers.

3.6.1 Transfer Learning



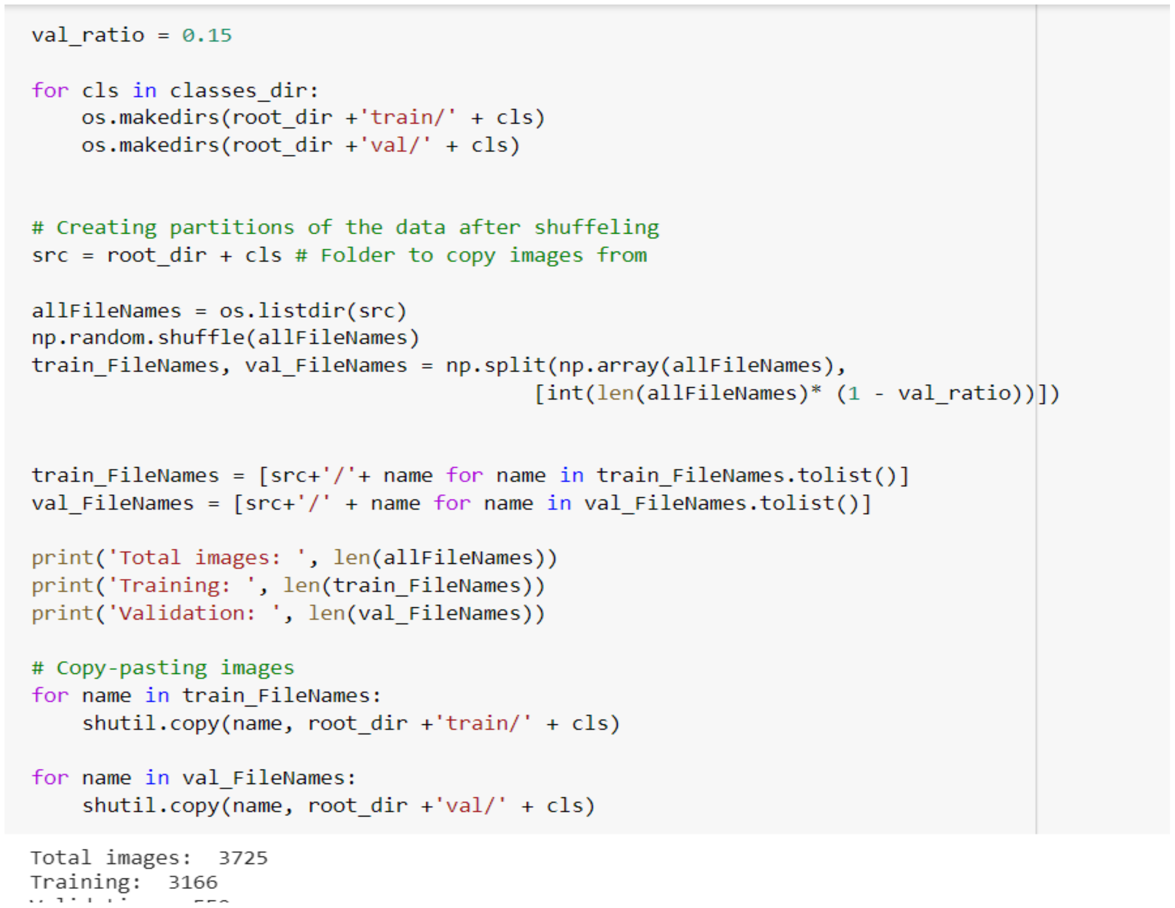
```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pre_trained_model = InceptionV3(
    input_shape=(150, 150, 3), include_top=False, weights=None)
pre_trained_model.load_weights(local_weights_file)
```

```
for layer in pre_trained_model.layers:
  layer.trainable = False
```

```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape:', last_layer.output_shape)
last_output = last_layer.output
```

```
from tensorflow.keras.optimizers import RMSprop

# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model = Model(pre_trained_model.input, x)
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.0001),
              metrics=['acc'])
```

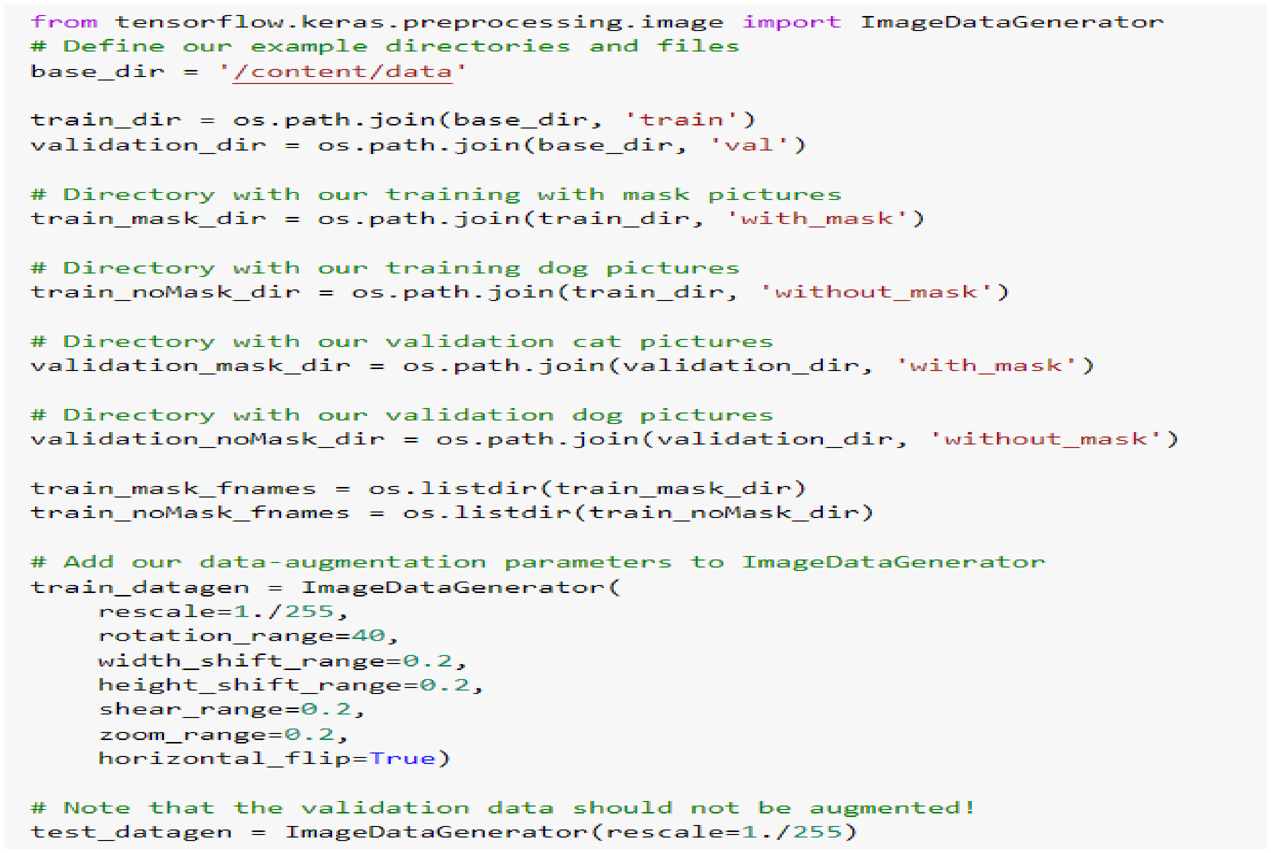
3.6.2 Creating Training and Validation directory



```
val_ratio = 0.15

for cls in classes_dir:
    os.makedirs(root_dir +'train/' + cls)
    os.makedirs(root_dir +'val/' + cls)


# Creating partitions of the data after shuffeling
src = root_dir + cls # Folder to copy images from

allFileNames = os.listdir(src)
np.random.shuffle(allFileNames)
train_FileNames, val_FileNames = np.split(np.array(allFileNames),
                                          [int(len(allFileNames)* (1 - val_ratio))])


train_FileNames = [src+'/'+ name for name in train_FileNames.tolist()]
val_FileNames = [src+'/' + name for name in val_FileNames.tolist()]

print('Total images: ', len(allFileNames))
print('Training: ', len(train_FileNames))
print('Validation: ', len(val_FileNames))

# Copy-pasting images
for name in train_FileNames:
    shutil.copy(name, root_dir +'train/' + cls)

for name in val_FileNames:
    shutil.copy(name, root_dir +'val/' + cls)
```

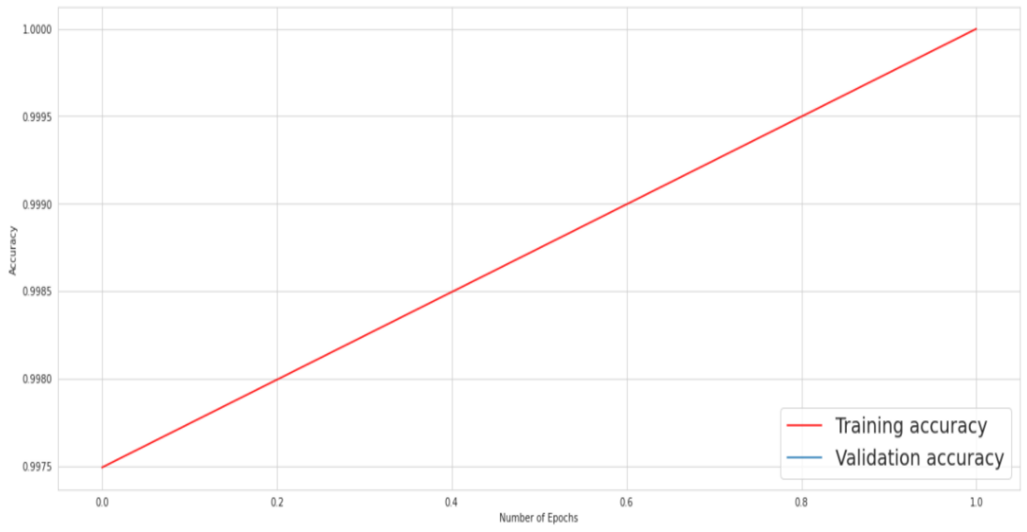
```
Total images:  3725
Training:  3166
```

3.6.3 Setting directory path and creating ImageDataGenerator

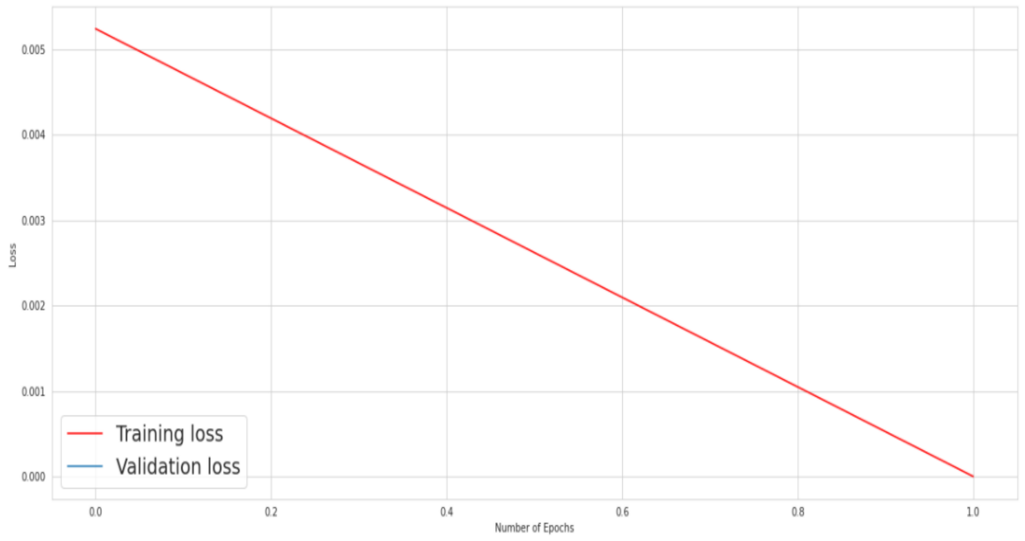


```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define our example directories and files
base_dir = '/content/data'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'val')

# Directory with our training with mask pictures
train_mask_dir = os.path.join(train_dir, 'with_mask')

# Directory with our training dog pictures
train_noMask_dir = os.path.join(train_dir, 'without_mask')

# Directory with our validation cat pictures
validation_mask_dir = os.path.join(validation_dir, 'with_mask')

# Directory with our validation dog pictures
validation_noMask_dir = os.path.join(validation_dir, 'without_mask')

train_mask_fnames = os.listdir(train_mask_dir)
train_noMask_fnames = os.listdir(train_noMask_dir)

# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)
```

Accuracy & Loss Graph



Graph 3.9 Accuracy graph



Graph 3.10 Loss Graph

Chapter 04
CONCLUSIONS

4.1 Discussion on the Results Achieved

The following results were achieved upon the implementation of this project:

- Real-time monitoring was achieved.
- 100% accuracy , 0% Loss on Test set.
- Face covered with hands were not classified as masked.
- Side facing positions were classified appropriately.
- The model could not detect masks at larger distances.

4.2 Application of the Project

The goal of this Major Project was to design and develop a system capable of detecting face masks in support of taking appropriate precautions in this pandemic situation. It focuses on achieving good accuracy without using heavily-designed complex networks having extensive hardware requirements which are not feasible in practical situations.

4.3 Limitations of the Project

If under any circumstance, the images taken by the camera module aren't clear enough to classify the system fails. Therefore, the proposed system has the following limitations:

- If the camera module is placed at a distance from the crowd, the model may not be able to give accurate results.
- The model has been designed in a simple fashion, it has no way to classify whether the person in front of the camera is wearing a mask properly or not.
- The model has not been trained by adversarial examples and is hence susceptible to bayesian error.

4.4 Future Work

The system at this stage is a "Proof of Concept" for a much substantial endeavor. This will serve as a first step towards a distinguished technology that can bring about an evolution aimed at ace development. The developed system has special emphasis on real-time monitoring with flexibility, adaptability and enhancements as the foremost requirements.

Future enhancements are always meant to be items that require more planning, budget and staffing to have them implemented. There following are couple of recommended areas for future enhancements:

- **Use of object detection Algorithms:** Object detection algorithms can be used to trace humans in the camera. After this, a face detection classifier can be used to detect faces, and this model can be used to detect masks.
- **Use of lighter Networks:** Traditional CNNs are heavy which might pose a problem in real-time deployment of the project. Instead, alternatives such as MobileNetV2, etc can be used so that its hardware requirements meet the feasibility studies in the SDLC.

REFERENCES

[1] X. Liu, S. Zhang, COVID-19: Face masks and human-to-human transmission, Influenza Other Respiratory. Viruses, vol. n/a, no. n/a, doi: 10.1111/irv.12740.

[2] S. Feng, C. Shen, N. Xia, W. Song, M. Fan, B.J. Cowling Rational use of face masks in the COVID-19 pandemic Lancet Respirate. Med., 8 (5) (2020), pp. 434-436, 10.1016/S2213-2600(20)30134-X

[3] "WHO Coronavirus Disease (COVID-19) Dashboard." https://covid19.who.int/ (accessed October 21, 2020).

[4] D.S.W. Ting, L. Carin, V. Dzau, T.Y. Wong Digital technology and COVID-19 Nat. Med., 26 (4) (2020), pp. 459-461, 10.1038/s41591-020-0824-5

[5] D.M. Altmann, D.C. Douek, R.J. Boyton What policy makers need to know about COVID-19 protective immunity Lancet, 395 (10236) (2020), pp. 1527-1529, 10.1016/S0140-6736(20)30985-5

[6] O. Abdel-hamid, L. Deng, and D. Yu, "Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition," no. August, pp. 3366–3370, 2013.

[7] Wei Xiong , Bo Du, Lefei Zhang, Ruimin Hu, Dacheng Tao "Regularizing Deep Convolutional Neural Networks with a Structured Decorrelation Constraint " IEEE 16th International Conference on Data Mining (ICDM) , pp. 3366–3370, 2016.

[8] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient- based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

[9] D. Stutz and L. Beyer, "Understanding Convolutional Neural Networks," 2014.

[10] I. Kokkinos, E. C. Paris, and G. Group, "Introduction to Deep Learning Convolutional Networks, Dropout, Maxout 1," pp. 1–70.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, C. V Jan, J. Krause, and S. Ma, "ImageNet Large Scale Visual Recognition Challenge.".

t

ORIGINALITY REPORT

11%	6%	1%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.geeksforgeeks.org Internet Source	2%
2	Submitted to Sogang University Student Paper	2%
3	Submitted to Jaypee University of Information Technology Student Paper	1%
4	Submitted to Higher Education Commission Pakistan Student Paper	1%
5	Submitted to Utah Education Network Student Paper	1%
6	www.edureka.co Internet Source	1%
7	ethesis.nitrkl.ac.in Internet Source	<1%
8	madhukamisetty.blogspot.com Internet Source	<1%
9	www.ir.juit.ac.in:8080	

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ……………………….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: PRATIKSHA **Department:** CSE/IT **Enrolment No** 171326

Contact No. ____________ **E-mail.** 171326@juitsolan.in

Name of the Supervisor: MR. PRATEEK THAKRAL

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters):

FACE MASK DETECTION USING CONVOLUTIONAL NEURAL NETWORK AND TRANSFER LEARNING

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

– Total No. of Pages = 39
– Total No. of Preliminary pages = 37
– Total No. of pages accommodate bibliography/references = 1

[illegible]

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……11……….(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

[illegible]

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

……………………………………………………………………………………………

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com