Embedded System

Course Project: ESE 5

INSTRUCTOR: DR. Binod Kumar

Team	Roll No.
Ayush Gangwar	B20CS008
Harsh Sharma	B20CS017

Project: ***DL model optimization for Lightweight Face Mask Detection***

Objective: Automatic detention of whether a person is wearing a mask or not from either camera input or microphone input of mobile phones or microcontrollers.

Dataset Used: Kaggle , FDDB etc.

Python Libraries and its dependencies :

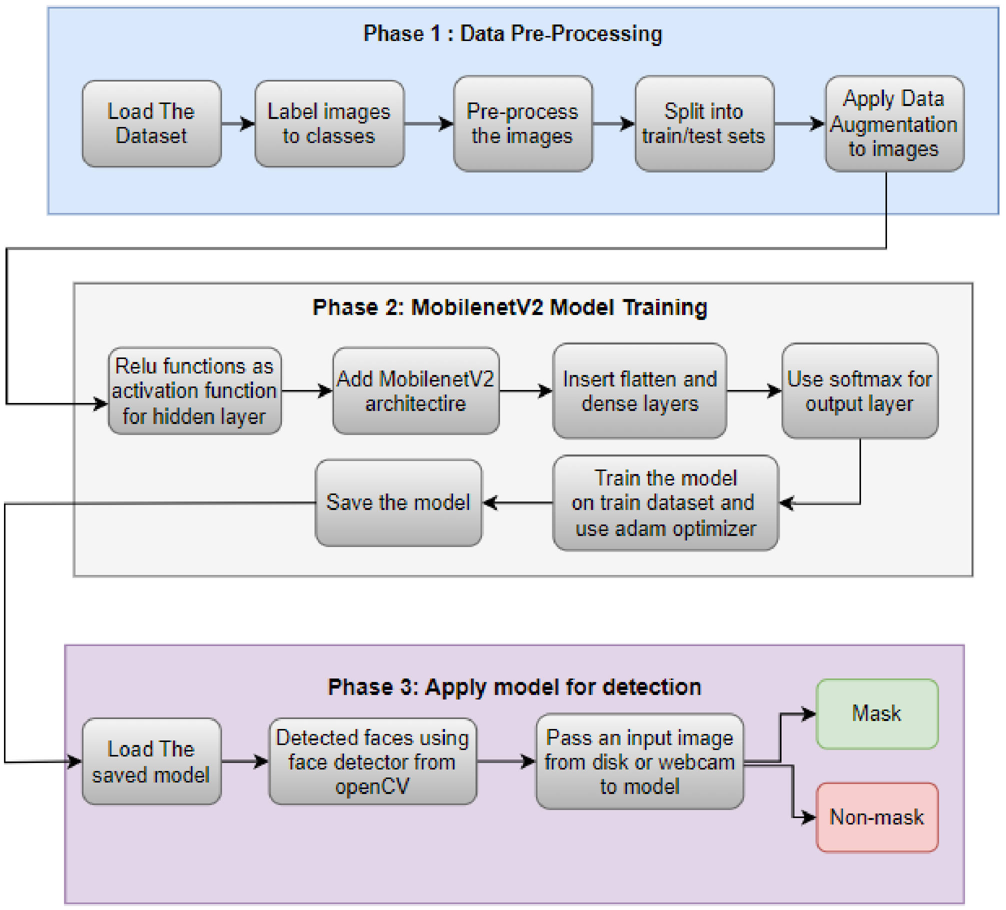
Libraries	**Functions**
Tensorflow/keras	Used to perform image classification Classification with the pre-trained MobilentV2 architecture.
CV2	Operations on images.
Sklearn	Operations on data and metrics operations
Matplotlib	3D Graphs plots etc.
Numpy	Matrix Math etc.
Pandas	Reading the datafiles.

Approach:

Model Architecture:

- We have selected *MobileNet V2* architecture for face mask detection .
    - Why MobileNet V2?-
        - MobileNet V2 achieves its efficiency by using depthwise separable convolutions, which reduce the number of computations and parameters required by standard convolutions. This makes the model faster to train and run, and reduces the risk of overfitting to the training data.
        - And hence,it can run on devices with limited computing power and memory.This makes it a good fit for mobile and embedded devices, as well as for applications where real-time performance is important.

Working of Model:

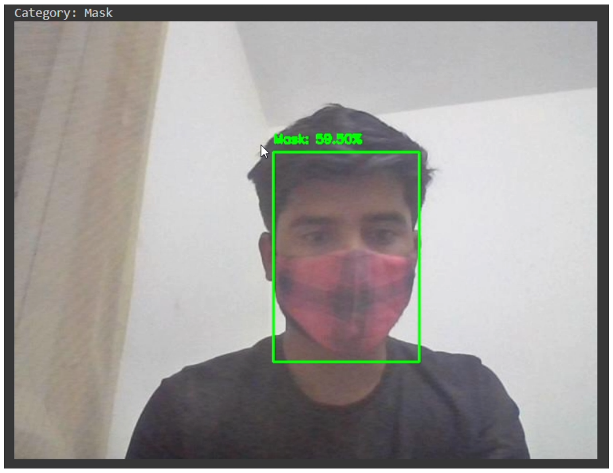

Phase 1 : Data Pre-Processing
Load The Dataset
Label images to classes
Pre-process the images
Split into train/test sets
Apply Data Augmentation to images
Phase 2: MobilenetV2 Model Training
Relu functions as activation function for hidden layer
Add MobilenetV2 architectire
Insert flatten and dense layers
Use softmax for output layer
Train the model on train dataset and use adam optimizer
Save the model
Phase 3: Apply model for detection
Load The saved model
Detected faces using face detector from openCV
Pass an input image from disk or webcam to model
Mask
Non-mask

Our model works in 3 phases viz data pre-processing , building and training the model and lastly detecting an input image into mask and non-mask categories (as explained above).

Functions of layers in Model architecture(MobilenetV2):

- *Input Layer:* This layer accepts the input image and passes it on to the next layer.
- *Convolutional Layers:* Perform feature extraction on the input image. These layers use depthwise separable convolution, which reduces the number of computations and parameters required by standard convolutions.There are 53 conv layers.
- *Bottleneck Layers:* MobileNet V2 uses bottleneck layers to further reduce the number of computations required by the model.There are 28 bottleneck layers.
- *Max Pooling Layers:* MobileNet V2 uses max pooling layers to downsample the feature maps and reduce the spatial dimensions of the input image. There are 13 maxpool layers.
- *Fully Connected Layers:* The 4 fully connected layers use a combination of activation functions, including *ReLU and softmax*, to produce the final output of the model.
    - Why Relu?
        - it is computationally efficient, introduces non-linearity, and helps to prevent the vanishing gradient problem
    - Why only 4 fully connected layers ?
        - The number of fully connected layers varied, and it was found that using 4 fully connected layers resulted in the highest accuracy on the validation set. Therefore, 4 fully connected layers were chosen for the final model architecture.

Demo : Input image is given by webcam.



*** To play the demo video visit the link .*

DL model optimization:

The optimization process can involve various techniques aimed at reducing the model's computational requirements, memory usage, and storage size without significantly impacting its accuracy.

We have adopted a *quantization* technique to optimize our model. It involves reducing the precision of the model's weights and activations, such as from 32-bit floating-point numbers to 8-bit integers or even lower precision. This reduces the storage requirements of the model and makes it more efficient to run on hardware with limited computational resources.

We have written code that loads a pre-trained floating-point model (trained above) using TensorFlow Keras, converts it to a quantized model using TensorFlow Lite, and saves the quantized model as a binary file.

Reducing the size of the neural network design

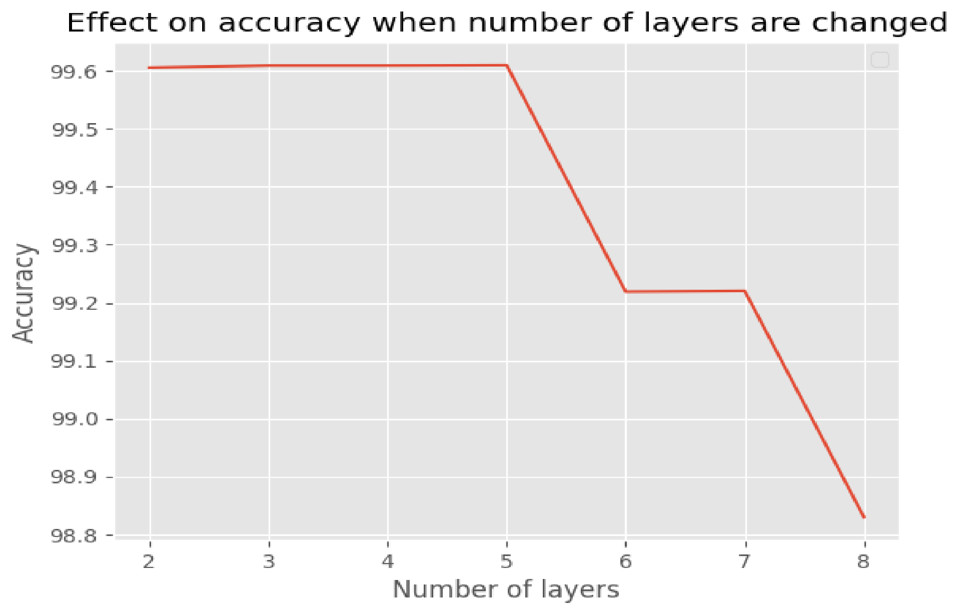
Approach :

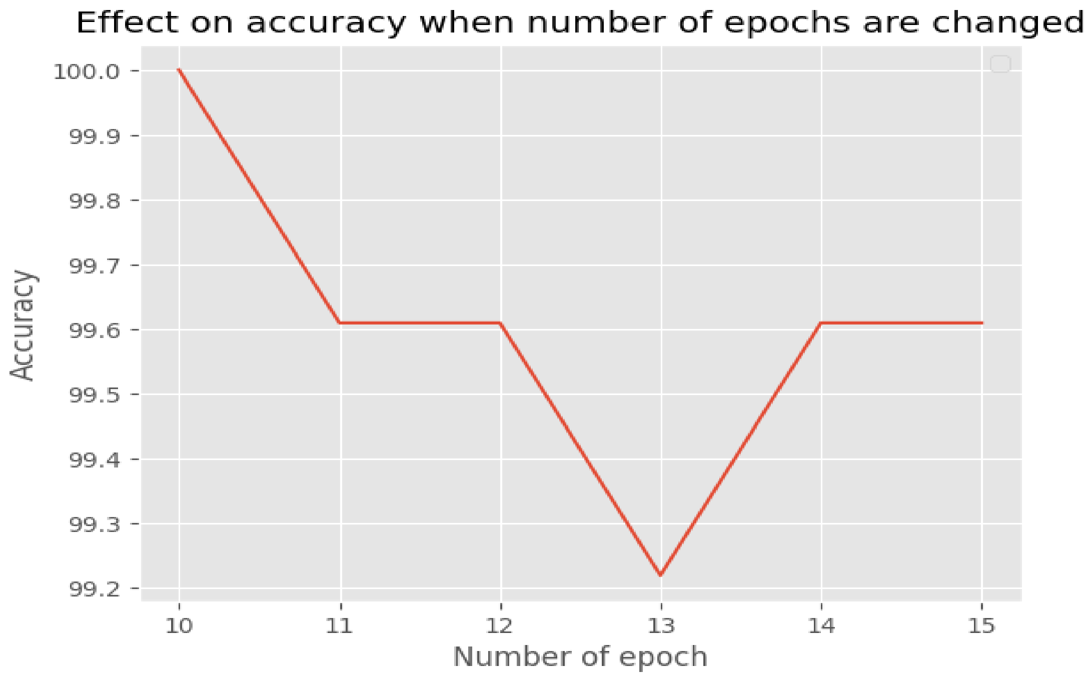
- Observe accuracy changes in neural networks when the number of layers are reduced .
- Observe accuracy changes in neural networks when number of epochs are reduced .

Implementation Explanation:

- **When the number of layers are reduced**
    - In this section we will change the number of layers for a particular number of epochs and observe the accuracy changes .
    - The number of epochs is set to 15 .
- **When the number of epochs are reduced**
    - In this section we will change the number of epochs for a particular number of layers and observe the accuracy changes .
    - The number of layers is set to 4 (4 dense layers in total) .

Results:





Explanation of the result :

- When the number of layers are reduced from 8 to 4 there is an increase in accuracy and from 4 to 2 accuracy decreases slightly . This is because when the number of layers is large there is a chance of slight overfitting or maybe the model is not fine tuned .

- There might be the same reason for the second graph . There is a reduction in accuracy when epochs are increased from 10 to 15 . This might be the case of overfitting or maybe the model is not fine tuned at that number of epochs .

Model robustness check

Approach :

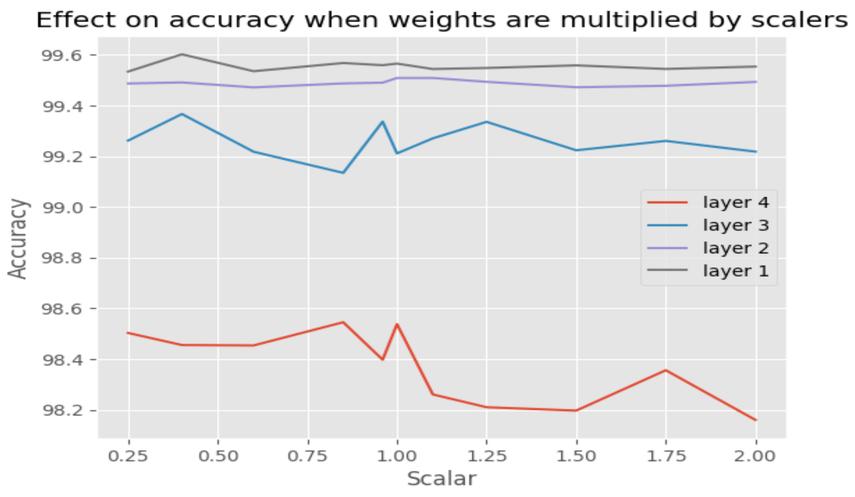
- Observe accuracy changes in neural networks when weights are changed by multiplying with impurities
- Observe accuracy changes in neural networks when weights are changed by adding impurities.

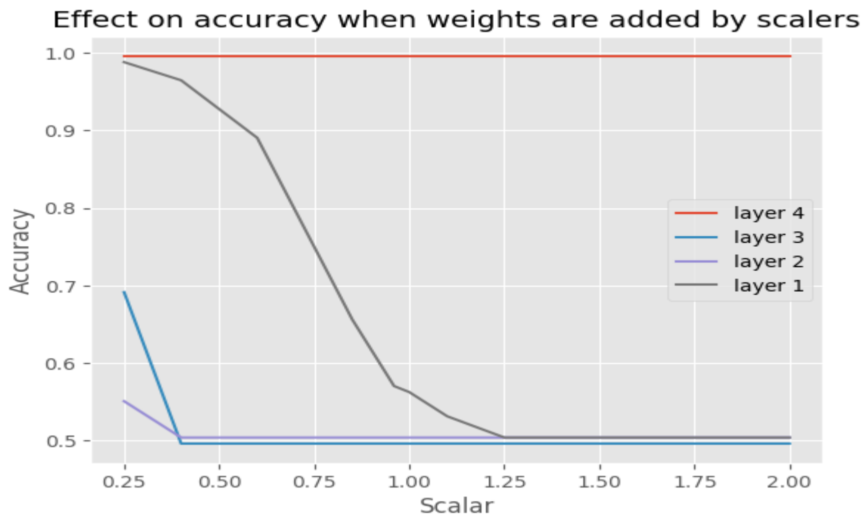
Implementation Explanation:

- **Weights are changed by multiplying with impurities**
    - In this section we will update the weights of one layer keeping all other layers unchanged by multiplying the scalars values and then for every change new accuracy is calculated .
    - The above step is followed for every layer .
    - We can take scalar values from any range.
    - We have taken the following array [.25, 0.4, 0.6, 0.85, 0.96, 1, 1.1,1.25,1.5,1.75,2.]

- **Weights are changed by adding with impurities**
    - In this section we will update the weights of one layer keeping all other layers unchanged by adding the scalars values and then for every change new accuracy is calculated .
    - The above step is followed for every layer .
    - We can take scalar values from any range.
    - We have taken the following array [.25, 0.4, 0.6, 0.85, 0.96, 1, 1.1,1.25,1.5,1.75,2.]

Results:





Some observations from the result :

- The model is highly robust towards weight changes as we can see there is very less change in accuracy when weights are multiplied with scalars .
- There is a huge change in accuracy when weights are changed by adding scalars in the first layer and there is no change when weights are changed by adding scalars in the last layer .
- From the above observation we can say that the least sensitive layer towards weight change is the last layer and the most sensitive layer towards weight change is the first layer .