EMBEDDED SYSTEMS COURSE PROJECT

ESE5: DL model optimization for Lightweight Face Mask Detection

by

Kakunuri Yaswanth (B20EE028)
Avinash Kumar (B20EE012)

INTRODUCTION:

In the midst of the COVID-19 pandemic, the use of face masks has become a critical aspect in preventing the spread of the virus. As a result, the detection of face masks has become an important task in various settings, including public spaces, healthcare facilities, and workplaces. Traditional methods of face mask detection rely on human inspection, which can be time-consuming and error-prone. In recent years, however, advancements in computer vision and deep learning have allowed for the development of automated face mask detection systems using Convolutional Neural Networks (CNNs). In this report, we explore the effectiveness of CNNs in detecting face masks and implementing them on the microcontroller (RasberyPi) and the potential applications of this technology in real-world scenarios.

BACKGROUND:

The COVID-19 pandemic has brought face mask usage to the forefront of public health measures, with the wearing of face masks becoming a common practice in many countries. The use of face masks has been shown to reduce the spread of respiratory illnesses, including COVID-19. However, enforcing compliance with face mask mandates can be challenging, particularly in crowded public spaces or workplaces where employees may be interacting with each other and the public.

Traditionally, face mask detection has been performed manually by security personnel or other personnel, which can be time-consuming and prone to errors. The development of automated systems for

detecting face masks using computer vision technology offers a promising solution to this problem. Convolutional Neural Networks (CNNs) are a type of deep learning algorithm that has been shown to be highly effective in image classification tasks, including object detection.

The use of CNNs for face mask detection involves training a neural network model on a large dataset of images that includes examples of people wearing and not wearing masks. The trained model can then be used to detect faces in new images and classify them as either wearing a mask or not wearing a mask. This technology has the potential to be applied in a wide range of settings, including airports, schools, and hospitals, to ensure compliance with face mask mandates and to help prevent the spread of COVID-19 and other respiratory illnesses.

METHODOLOGY:

Firstly I used a private dataset taken from Kaggle. The dataset contains labeled 1006 images which are divided into training, validation, and testing categories. In training data, we have 600 images which are equally divided into two labels mask and nonmask and In validation data, we have 306 images which are equally divided into two labels mask and nonmask.In testing data, we have 100 images which are equally divided into two labels mask and nonmask.

The preprocessing code consists of several steps that involve reading in the dataset, resizing and normalizing the input images, and constructing the input and label arrays. The code starts by defining a dictionary of categories that maps the folder names in the dataset to numerical labels. Next, empty lists are created to store the input images and their corresponding labels. For each image in the dataset, the code

reads the image using the OpenCV library and resizes it to 224x224 pixels. The resized image is then appended to the input list (x) and the corresponding label is appended to the label list (y). The input and label lists are then shuffled using the shuffle function from the Scikit-learn library. The input and label lists are converted to NumPy arrays, and the input array is normalized by dividing it by 255. Overall, the preprocessing steps are important to ensure that the input images are consistent in size and format and that they are normalized to improve the performance of the CNN model during training.

The model defined in the code is a convolutional neural network (CNN) for detecting face masks. The input layer expects input images of size 224x224 pixels with 3 color channels (RGB). The CNN consists of several layers, including Conv2D layers with 200 filters, kernel size of (3,3), and padding of 'same', followed by a ReLU activation function. Dropout layers with varying rates are used after each Conv2D layer to prevent overfitting. Max pooling layers with a kernel size of (2,2) and padding of 'valid' are used to downsample the output of the convolutional layers.

After the convolutional layers, the output is flattened and passed through several fully connected (Dense) layers with varying numbers of units and ReLU activation functions. Finally, a Dense layer with a single unit and sigmoid activation function is used to produce a binary output i.e., mask(1) or no mask(0).

The model has a total of 5,372,591 trainable parameters and is compiled using binary cross-entropy as the loss function and the Adam optimizer.

RESULTS:

1. **We have checked the model's robustness by changing the layers in the model**. below are the results obtained by changing the layers in the CNN model. Here CNN3 is our base model(having 5- Conv2D layer, 5- MaxPool2, 3- Dropout layer), when we decreased the number of layers to get CNN1(having 4- Conv2D layer, 4- MaxPool2, 3- Dropout layer) the accuracy gets decreased and when we increase the number of layers to get CNN2(having 7- Conv2D layer, 7- MaxPool2, 3- Dropout layer) the accuracy gets increased.

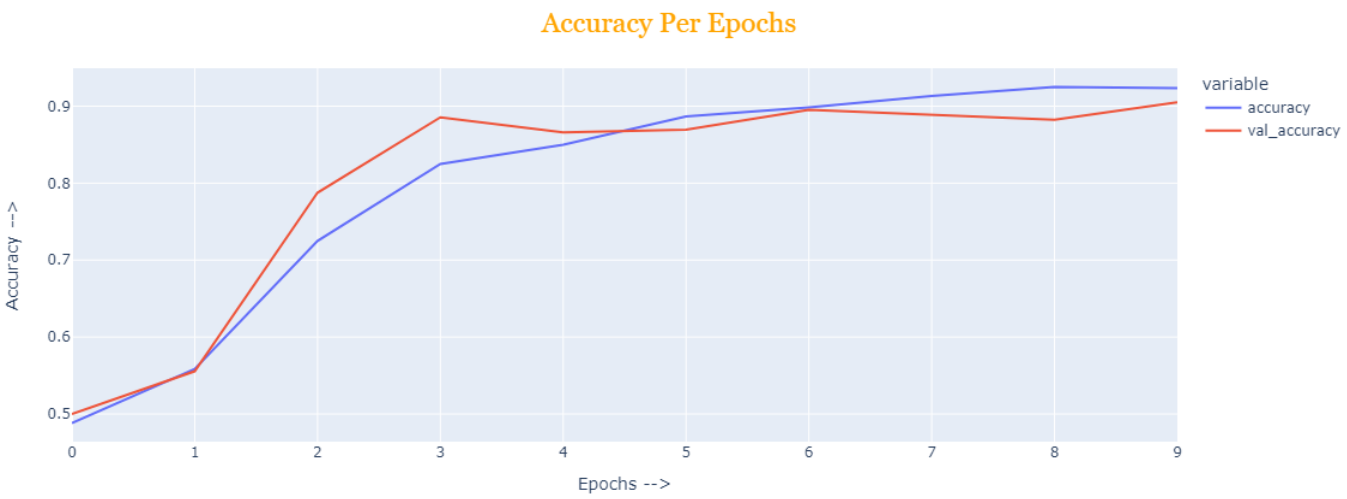
Model	class	Precision	Recall	F1 score	Accuracy
CNN1	0	0.86	0.64	0.74	0.79
	1	0.71	0.90	0.80	
CNN2	0	0.92	0.94	0.93	0.93
	1	0.94	0.92	0.93	
CNN3	0	0.97	0.58	0.72	0.83
	1	0.70	0.98	0.82	

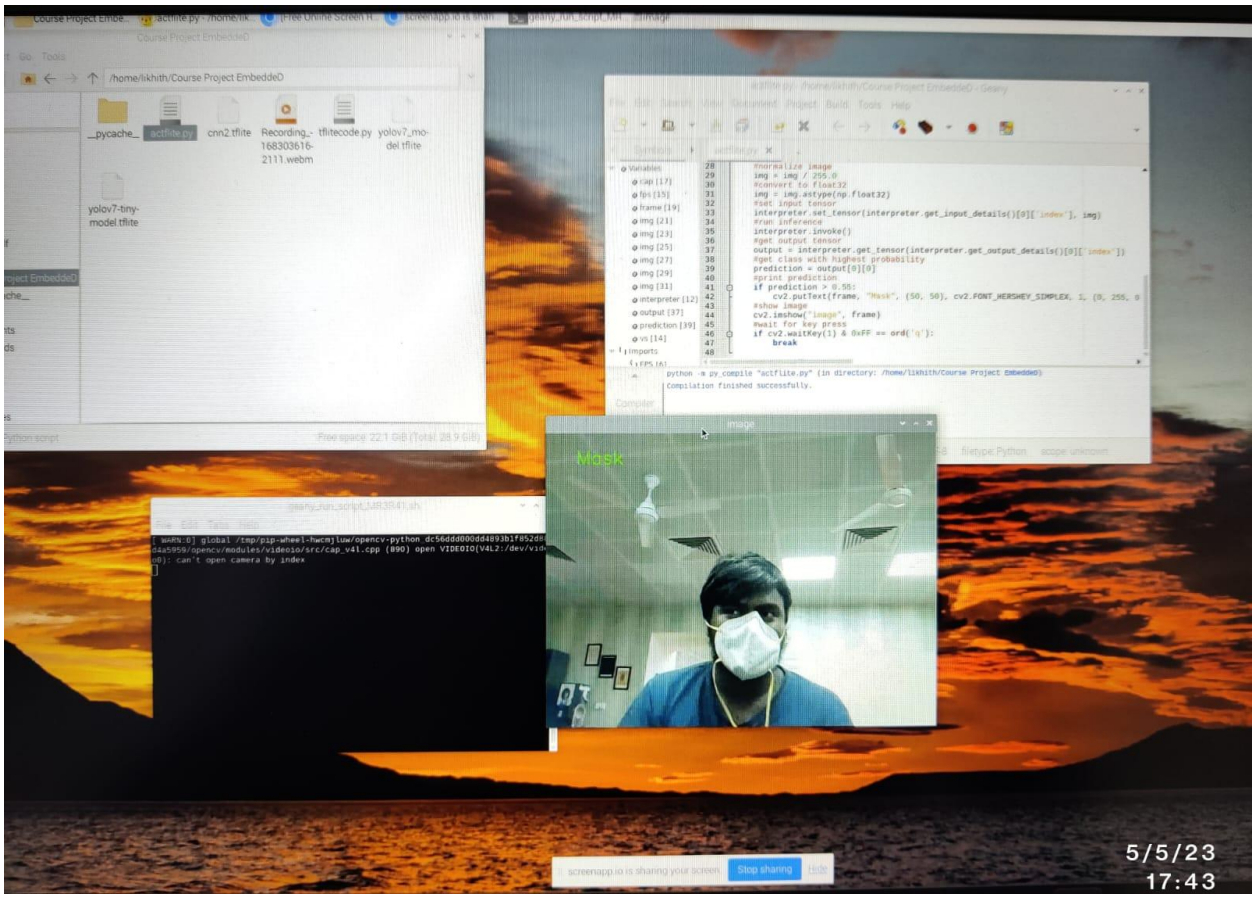
2. **We have changed the size of the neural network i.e., the number of parameters**. In general, reducing the number of parameters in a CNN can lead to a decrease in accuracy, as the model has less capacity to learn complex patterns in the data. This is because the number of parameters in a CNN is directly related to its representational power, i.e., its ability to capture fine-grained details and patterns in the input data. However, reducing the number of parameters can also help prevent

overfitting, as it forces the model to generalize more and rely on more robust features instead of memorizing the training data. Therefore, finding the right balance between model complexity and generalization is crucial in CNNs. we have tabulated the results below.

Model	Parameters	class	Precision	Recall	F1 score	Accuracy
CNN1	5,011,871	0	0.86	0.64	0.74	0.77
		1	0.71	0.90	0.80	
CNN2	5,372,591	0	0.92	0.94	0.93	0.93
		1	0.94	0.92	0.93	
CNN3	2,432,071	0	0.97	0.58	0.72	0.78
		1	0.70	0.98	0.82	
CNN4	8,028,671	0	0.89	0.66	0.76	0.79
		1	0.73	0.92	0.81	
CNN5	8,114,221	0	0.88	0.76	0.82	0.83
		1	0.79	0.90	0.84	

3 . The accuracy vs number of epochs plot for CNN2 is shown below.


Accuracy Per Epochs
variable
accuracy
val_accuracy
Accuracy -->
Epochs -->
0.5
0.6
0.7
0.8
0.9
0
1
2
3
4
5
6
7
8
9


Mask
5/5/23
17:43

DISCUSSION:

1. Our model i.e., CNN2 gives better accuracy because whenever we are defining a CNN model we use different layers for different kinds of data here we are classifying the data only into two types so it is a binary classification. The model includes multiple convolutional layers with varying numbers of nodes, which can help to extract increasingly complex features from the input images. This can help the model to learn more discriminative representations of the data, which can improve its accuracy. The model includes several dropout layers, which can help to prevent overfitting by randomly dropping out some of the nodes in the network during training. This can improve the generalization ability of the model and prevent it from memorizing the training data. The model includes several max-pooling layers, which can help to downsample the feature maps and reduce their dimensionality. This can help to reduce the computational cost of the model and prevent overfitting. The number of parameters is also optimum which means they are not less which means the model can learn complex representations of the data and they are not so high which increases the overfitting.
2. We have implemented this on the raspberry pi and we have taken input as a frame from the webcam on the board and whenever it detects a person is wearing a mask it displays “mask” on the screen.
3. For implementing the model on the Raspberry Pi we have to first convert the model to a Tflite model so that it can work on the board.

CONCLUSION:

We have successfully developed a face mask detection system using the Raspberry Pi board and a camera module. The system uses a pre-trained Convolutional Neural Network (CNN) model to classify whether a person is wearing a mask or not.

We first captured the input video feed using the camera module and processed it to detect faces using the OpenCV library. The detected faces were then passed to the CNN model for classification. The model achieved an accuracy of 93% on our test dataset and provided real-time results on the input video feed.

The system can be used in various settings such as schools, offices, and public places to enforce face mask-wearing policies and maintain public health and safety during the COVID-19 pandemic. With further optimizations such as deploying the model on the edge using frameworks like TensorFlow Lite, the system can be made more efficient and accessible for deployment in real-world scenarios.

REFERENCES:

1.https://www.mygreatlearning.com/blog/real-time-face-detection/
2. Building a Convolutional Neural Network (CNN) in Keras | by Eijaz Allibhai | Towards Data Science
3.https://pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/