Mask Detection deployment on Low Powered Micro-processor

Risha Dassi
Dept. of Computer Science and Engineering
R.V. College of Engineering
Bengaluru, India
rishadassi.cs18@rvce.edu.in

Vishal M.
Dept. of Computer Science and Engineering
R.V. College of Engineering
Bengaluru, India
vishalm.cs18@rvce.edu.in

Minal Moharir
Dept. of Computer Science and Engineering
R.V. College of Engineering
Bengaluru, India
minalmoharir@rvce.edu.in

Pavan K. R
Dept. of Computer Science and Engineering
R.V. College of Engineering
Bengaluru, India
pavankr.cs18@rvce.edu.in

Shuvam Mitra
Dept. of Computer Science and Engineering
R.V. College of Engineering
Bengaluru, India
shuvammitra.cs18@rvce.edu.in

***Abstract*— Wearing masks has been one of the key methods of preventing the spread of COVID-19 and thus being able to ensure that the people entering the premises of a particular institution are wearing masks thereby reducing the risk of the people within those premises to be affected by the virus. This paper describes the approach of taking Google's pre-trained Inception-V3 architecture and then converting the model to a TFlite version which is deployed on Raspberry Pi 4 making the model portable and affordable.**

***Keywords*—** Convolutional Neural network (CNN), Object Detection, Inception-V3, Raspberry Pi (RPi), Transfer learning.

I. INTRODUCTION

While the world has been battling with the pandemic of Coronavirus disease 2019 (COVID-19), people have been forced to rethink strategies when it comes to work, education and even social greeting norms. The World Health Organization (WHO) has emphasized the importance of wearing masks in order to prevent and limit the spread of COVID-19 since they not only protect healthy individuals when in contact with an infected person but also prevent the spread from an infected individual. For people that are travelling outside and entering premises of their workplaces and institutions, it's important for people to follow the norm of wearing a mask not just for their safety, but also of those around them. Thus, the need for having mask detection systems is now more than ever. It would be difficult to install huge systems for this detection from an economic and space standpoint and deploying this on an embedded system such as Raspberry Pi makes this model a lot more compact. The approach taken by the team was to use a pre-trained Inception-V3 architecture by Google and train it to detect whether the person standing in front of the camera was wearing a mask or not. The model also displayed a bounding box around the detected face with a green box indicating a person wearing a mask and a red one for a person without it. Overall, this implementation of Mask Detection sets itself apart by being adapted for an embedded system that makes it a lot more cost effective and thereby helping enforce preventive measures against a pandemic and potentially saving lives.

II. DATASET

A. Data Collection

The proposed work used datasets from different sources some of which are the Real and Fake face detection dataset from Kaggle [6] and other repositories on open source platforms [7][8]. This was done since one dataset was insufficient in terms of the angle from which the mask is seen and the absolute number of images as well. These datasets were combined for making the model more efficient. These were segregated into directories of masks and without masks images for creating classes that were used for training. The images were pre-processed using the procedure mentioned below.

B. Data Pre-processing and Augmentation

Data pre-processing is a fundamental step to be taken up in training neural networks with the intent of reducing the weight and enhancing the quality of the input without losing any information of interest, thereby making the training computationally less expensive. The preprocess_input(), which is a method under the tensorflow.keras.applications.inception_v3 was used to pre-process the input images. Data augmentation is a technique of increasing the diversity of the training dataset by the application of some random but realistic transformations such as width shift, height shift, shear, rotation, zoom, flipping etc on the images. The dataset was further divided into a train set and validation set in an 8:2 ratio with an intent of tackling overfitting and making the model more efficient. Figure 1 below shows the input images after data augmentation.

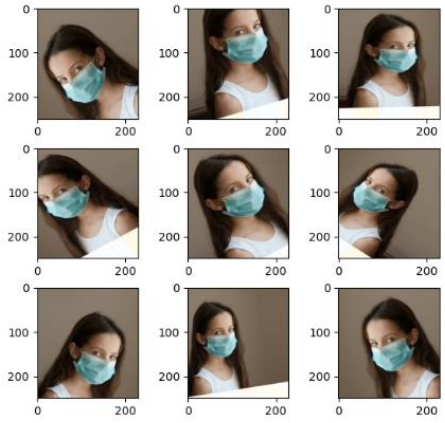

Fig 1: Transformations done to input images using data augmentation.

III. ARCHITECHTURE AND METHODOLOGY

A. *Neural Network Architecture*

Considering the trade-off between accuracy and performance on computationally less powerful edge devices like Raspberry Pi, Google's Inception-V3 architecture [2] was found to be the best fit. An Inception-V3 pre-trained on ImageNet database was used with the motive of adopting the transfer learning approach in training the mask detection model. Transfer learning refers to re-training several layers of a highly efficient pre-trained model with another dataset so as to leverage the performance of this model to fit the needs of the dataset under consideration. This procedure was followed to adapt the model for the expected output and its implementation can be seen in Figure 2 below.

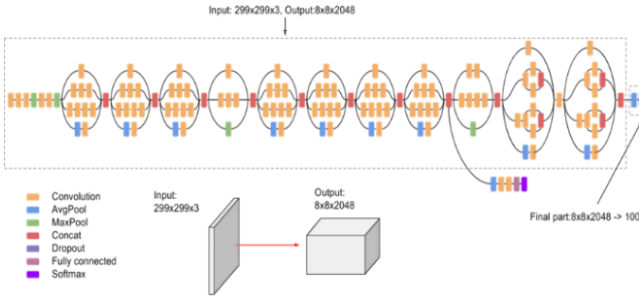

Fig 2: Google Inception-V3 Architecture (Source: https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png)

B. *Model and Training*

The Inception-V3 model used in this work was trained on the ImageNet database and the fully connected output layer had to be explicitly disconnected order to perform transfer learning [1] using the dataset and adapt it for mask detection. The adapted output layer is created by introducing 2-Dimensional average pooling and dropout regularization. The final Dense layer consists of two nodes representing the classes of interest i.e. WITH_MASK and WITHOUT_MASK and this is configured through the SoftMax activation function. The 'RMSPROP' optimizer is used for model compilation. The summary of the created model can be seen in Table 1 below.

Total parameters	21,806,882
Trainable parameters	4,098
Non-trainable parameters	21,802,784

Table 1: Model Summary

It is also observed that the loss and accuracy levels converges significantly after 10 epochs of training as seen in Figure 2 and 3 below:

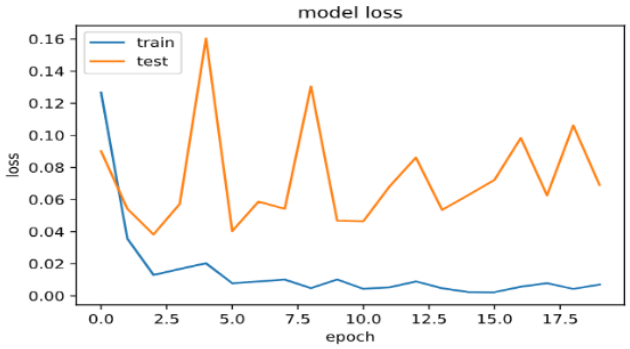

Fig 3: Loss vs Epoch graph

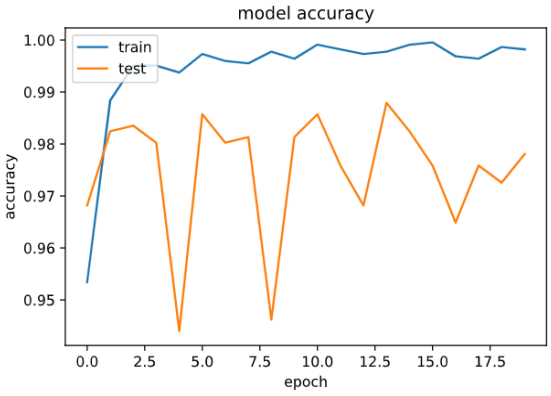

Fig 4: Accuracy vs Epoch graph

C. *Conversion to TensorFlow Lite model*

Deploying a TensorFlow model on an embedded edge device would have an impact on the performance of the deep learning model by slowing it down as the embedded devices cannot render the same performance as that of larger computing systems. By taking into consideration the performance metrics like frame rate and the processing time, the existing model was converted to a TensorFlow Lite model to adjust according to the constraints of RPi.

The TensorFlow Lite converter was used to convert the existing TensorFlow model into a FlatBuffer file(.TFlite) for use by the interpreter. It can introduce optimizations to

improve binary size and performance of the TensorFlow model. The FlatBuffer file is now deployable on an embedded edge device and it can be run locally by using the TensorFlow Lite interpreter. Figure 5 below depicts the changes that take place in the TensorFlow file while undergoing conversion to a TFlite file.

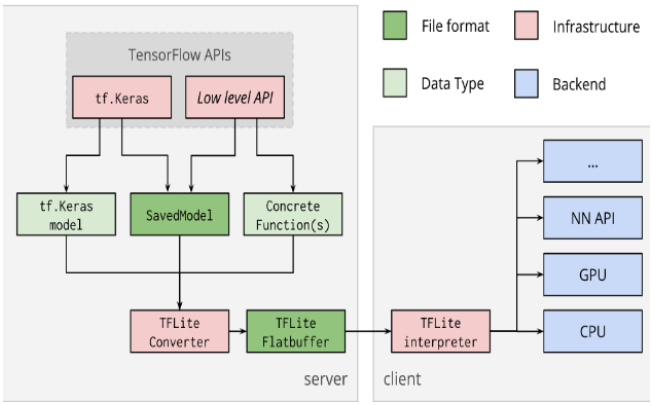

Fig 5: Conversion of a TensorFlow model to TensorFlow Lite model (https://www.tensorflow.org/lite/images/convert/workflow.svg)

IV. Deployment

A. Setup

Python along with other libraries such as imutils, OpenCV etc were installed to feed the video-stream input to the model. TensorFlow Lite interpreter was installed to run the classification model since it is compact enough for running the TFlite models. This is especially suitable for running on embedded devices like microcontrollers, owing to its lightness. For deployment, a separate program had to be developed for loading the TFlite model on the RPi [3].

B. Running the code

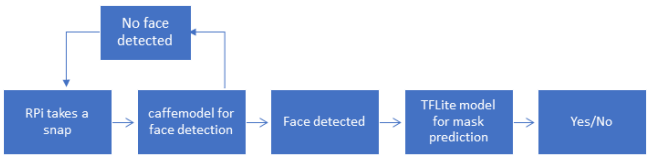

Fig 6: Flowchart of deployment

Figure 6 above details the flow of deployment. In order to detect face masks, pictures of the person's face are captured using a Pi Camera module, of 5mp resolution. The Pi Camera module is an affordable camera with a cable and is compatible with all versions of RPi. It uses CSI (Camera Serial Interface) for interfacing with RPi boards. It is also flexible and is a plug and play camera for RPi.

To be able to obtain input continuously images are taken from a video stream with a frame rate of 5 frames per second. The images obtained from the video stream are processed by first passed through the Caffemodel for detecting the face of the person. Caffe [2] is a deep learning framework made with expression, speed, and modularity in mind. If a face is detected the model returns a bounding box of the face detected and hence the presence of a human face within the picture is confirmed. By using this layer not only the binary decision of mask/no mask becomes easier but also the output is reliable. Then the image is cropped and resized to the input size of this model and then used for prediction of presence of the mask.

With a threshold of 30 frames, each frame is resized to the input size (299*299*3) and passed to TFLite interpreter and a count of positive and negative output is maintained and the class with the higher count is shown to the user as the final output.

V. Result

Considering the population in India that would be using makeshift masks using dupatta's, scarves, handkerchiefs etc. the proposed work tested the model including those scenarios as well. The trade-off while deploying on RPi is the resource constraint due to limited computational power and memory availability.

The proposed work also notes that using the Caffeemodel for face detection improves the accuracy of the algorithm by localizing the region of interest and feeding that to the deep learning model which speeds up the process of classification. 1575 and 1569 images were those of people without and with mask respectively before augmentation that was used for training. The net positive effect on the performance due to the optimizations used by us can be seen in Table 2 where the accuracy of the model has been quantified. Figure 7 and 8 show the bounding box representation results of without mask and with mask scenarios respectively.

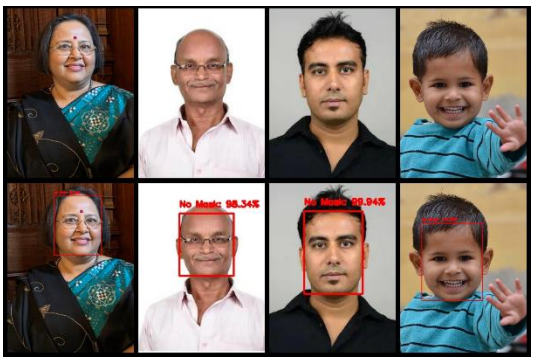

Fig 7: Bounding box for images of people not wearing masks

Fig 8: Bounding box for images of people wearing masks

Input type	No. of inputs	No. of correct predictions	No. of incorrect predictions	Test Accuracy (%)
MASKED	40	37	3	92.50
UNMASKED	40	38	2	95.00

Table 2: Accuracy of the deployed model on real time data

CONCLUSION

The deployment of a deep learning model such as this on a micro-processor increases the portability of such systems which can have much wider applications in healthcare. There is still some scope for improving the efficiency and speed of execution of such models. And further additions such as detecting temperature and oxygen level values along with the mask detection result could help alerts the administration of a particular premises if someone is not following the preventive measures. However, the results achieved despite the limitations of an edge computing device are sufficient to be used in real world applications. Since these can be produced at a cheaper cost as compared to other devices on which deep learning models are deployed, the production can be upscaled, increasing the outreach of such mask detection systems.

REFERENCES

[1] Xiaoling Xia, Cui Xu and Bing Nan, "Inception-v3 for flower classification," 2017 2nd International Conference on Image, Vision and Computing (ICIVC), Chengdu, 2017, pp. 783-787, doi: 10.1109/ICIVC.2017.7984661.

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2818-2826, doi: 10.1109/CVPR.2016.308.

[3] S. S. Walam, S. P. Teli, B. S. Thakur, R. R. Nevarekar and S. M. Patil, "Object Detection and seperation Using Raspberry PI," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2018, pp. 214-217, doi: 10.1109/ICICCT.2018.8473068.

[4] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán and Á. Rodríguez-Vázquez, "Optimum Selection of DNN Model and Framework for Edge Inference," in IEEE Access, vol. 6, pp. 51680-51692, 2018, doi: 10.1109/ACCESS.2018.2869929.

[5] T. Zebin, P. J. Scully, N. Peek, A. J. Casson and K. B. Ozanyan, "Design and Implementation of a Convolutional Neural Network on an Edge Computing Smartphone for Human Activity Recognition," in IEEE Access, vol. 7, pp. 133509-133520, 2019, doi: 10.1109/ACCESS.2019.2941836.

[6] https://www.kaggle.com/ciplab/real-and-fake-face-detection

[7] https://github.com/prajnasb/observations/tree/master/experiements/data

[8] https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset