VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

CSE4014

High perfromance computing

Face mask detection

By

P. Maruthi Sai Saketh(18BCE1024)

S.Lokesh(18BCE1082)

K.G.R.Abhiram(18BCE1051)

Submitted

To

Dr.Anasuya.G (SCSE)

Nov 2020-2021

Index page:

Abstract

During this present pandemic situation its very important to wear the mask when we go out and in all the working places its compulsory to wear mask otherwise there will be no entry. To make the work simpler to check whether people are wearing the mask or not we came up the face mask detecting cameras .Which helps in the office to check people are wearing the mask or not by making buzzer or any other notification based on our convenience.

To detect the face mask we need to train the machine with the masked and mask less face dataset. The problem is in normal CPU it takes a lot of time to train. To reduce the time we are using different ways of high performance computing .To complete the work faster with the help of high performance computing. We are training our datasets on GPU and Spark computational models. As we are also trying to perform the real time video application in CPU it takes a lot of time to convert the video in to frames and then to detect mask and mask less faces from that frames. We are trying to do it on GPU.

Then we are also trying it on the TPU where we can run the same process even faster then the GPU. Finally we are comparing the output with the all the computational models that we did.

Objectives:

- In this present situation its real necessary to wear mask as its not our daily routine from the past days which will be difficult to remember and wear the mask daily whenever we are going outside so to find and notify we are creating this project
- To reduce the time consuming by the CPU to train the dataset and to detect the face mask from an image frame of a video. By using the high performance distributed computing techniques that is likes using GPU, Spark and TPU to make the work simple and faster.
- Our main objective is to create a live mask detecting camera or software where there will be no lag in detecting the number of frames that are coming from the video.
- The main objective of this project is to reduce the work of man power and automate the work with the present technology
- We made the procedure similar for any dataset that is to change the application just we need to change the dataset and train the model.

Introduction:

As the covid condition is really becoming worse we need be secure and follow some precautions when we are going. For detecting the face mask we are using different ways to make the computation fast and accurate. And also in many of the areas people are not wearing the masks if we give the and train the machines of traffic cameras with the dataset and the code so that it will scan and check people in which area they not wearing the mask so that they take action according to that.

As the offices are starting soon it will be more important to wear the mask until the vaccine is released to check that we are verifying the live video cameras weather the workers are wearing the mask are not and informs the security. In some public places like bus stops railway stations temples we are also trying to find how much people are aware of wearing masks in different locations. By comparing how many people are wearing and not wearing the mask.

First we are training our dataset with the help of CPU using keras frame work and finding the time taken for that to reduce the time and improve the performance we are using the GPU. As the GPU is not working the system we are doing the GPU computation the Google colab.

We are also trying with the help of Spark on Google colab with the help of spark packages we are training the dataset with the help of elephes frame work . And we have done through TPU in which there will be multiple cores and as it is not available on the real time we can make use of google colab which can provide the same environment for the application purpose.

And we have compared all the training time difference of different computational models. And we are also done the live video detection which helps to find the real time applications. With the help of this we can be able to find how many people are wearing the mask and how many people are not from this we can give a report on how many people are aware of present situation. And from that we can produce the percent of people aware of covid in particular area.

Software and Hardware requirements

The software that we are using in the project is Google colab, Spark, pytorch, Anaconda.

As we know google colab is open source software which provides virtual GPU and multicore processors for the research and projects works.

Spark

- Spark breaks the application into smaller tasks and assign them to the executors so it executes our application parallely.
- Here after loading the data we need a data structure to hold the data in spark
- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark

GPU

- The graphics processing unit (GPU), also called graphics card or video card, is a specialized electronic circuit that accelerates the creation and rendering of images, video, and animations. It performs fast math calculations while freeing the CPU to perform other tasks.

TPU

- TPU is an architecture used for DL and ML computation made by google.it is not a generic processor(i.e, only tensorflow models will run on it)
- So instead of making a general purpose processor ,google designed it as a matrix processor.
- Generally TPU has more than 30000 ALU's which is comparatively more than   GPU

Hardware

And the hardware is the Nvidia GPU and the CPU with the system

Literature review:

1) Performance and Scalability of GPU-based Convolution Neural Networks

In this paper they given about the implementation of a framework for accelerating training and classification of arbitrary Convolution Neural Networks (CNNs) on the GPU. CNNs are a derivative of standard Multilayer Perceptron (MLP) neural networks. They also describe the basic parts of a CNN and demonstrate the performance and scalability improvement will be get by shifting the computational tasks on the CNN to the GPU. Based on the topology classification and training on GPU is 2 to 24 times faster than CPU.

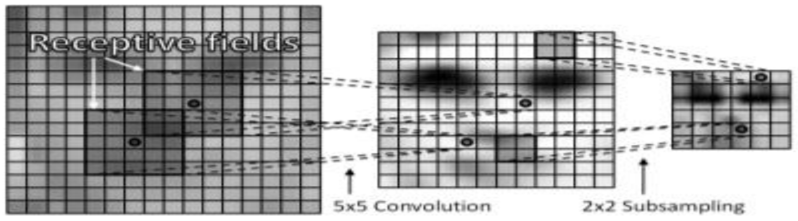


Figure 2. Illustration of the convolution and subsampling process inside a CNN (based on Fig. 3.5 in [13]).

Illustration of CNN

2) Object Detection with Low Capacity GPU Systems Using Faster R-CNN

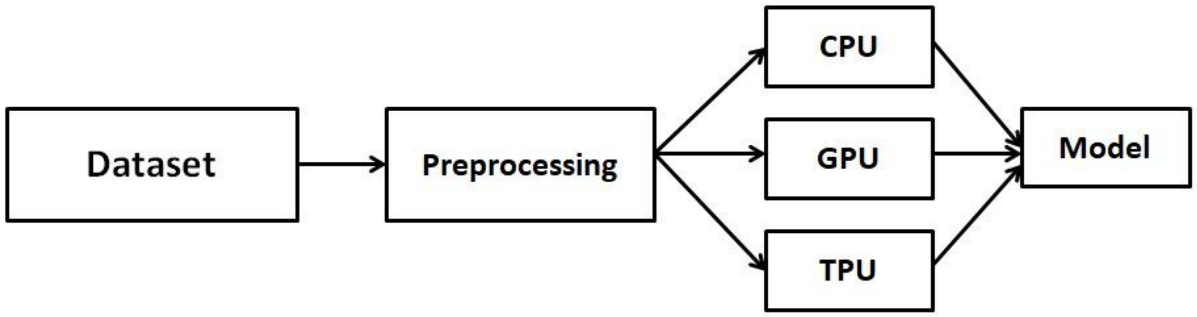
As the object detection places a vital role in the present technology like land planning, city monitoring, traffic monitoring, and agricultural applications etc. And as we know it is essential in the field of aerial and satellite image analysis but it is also a challenge and in this paper they came up with the fast and there are many object detections models. And a multi-scale Faster R-CNN method based on deformable convolution is proposed for single/low graphics processing unit (GPU) systems. Weight standardization (WS) is used instead of batch normalization (BN) to make the proposed model more efficient for a small batch size (1 img/per GPU) on single GPU systems.

3) An Assessment of Gpu and Cpu based Convolutional Neural Network for Classification of White Blood Cells

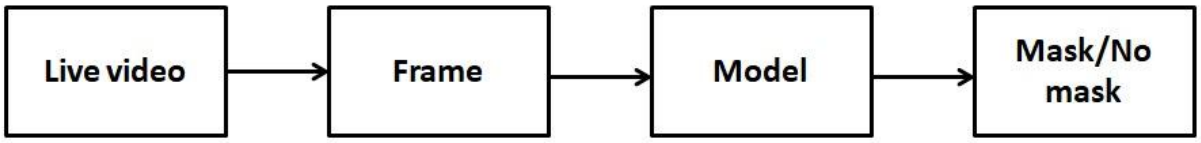
As the task of finding the white blood cells is not easy several efforts are made to address the aforementioned challenges with the use of machine learning and Convolution Neural Network

The Simulation was done using python programming language and python libraries including Keras, pandas, sklearn, numpy, scipy and matplot for potting of graphs of results. The simulation was done on CPU and GPU processor to compare the performance of the processors on CNNs based classification of the data. While CPU has faster clock speed GPU has more cores. Hence the evaluation metrics used which are precision, specificity, sensitivity, training accuracy and validation accuracy revealed that GPU processor outperforms CPU in terms of the stated metrics of comparison. Therefore a high configuration processor (GPU), which handles graphics better is recommended for processing image data that involves the use of machine learning techniques

Modules:



As shown above first we need to have the dataset and then we need to preprocess the dataset here we have done it through 3 ways that is with the help of CPU ,GPU and TPU and then we compare all the three Models.



And we are also trying the live video in which the live video is captured and then it is separated to frames and then the frames compared with the models as created before to find weather the mask is there are not

Preprocessing: The input dataset need to be preprocessed to decrease the complexity and to make the dataset fit in the neural network. Preprocessing steps used are: Converting the image into gray scale image, resizing the image into 50*50 which is the input shape given to the neural network. Converting the image into numpy array and rescaling the array values to 0 to 255 by dividing each pixel by 255.

Training: The preprocessed data is trained in the Convolution neural network under CPU, GPU and TPU. CNN contains many arithmetic operations during forward propagation as well as backward propagation. Our model contains 520,193 parameters which are needed to be trained.

Model: The trained model is saved for further predictions.

Live Video: We performed face mask detection in live video stream.

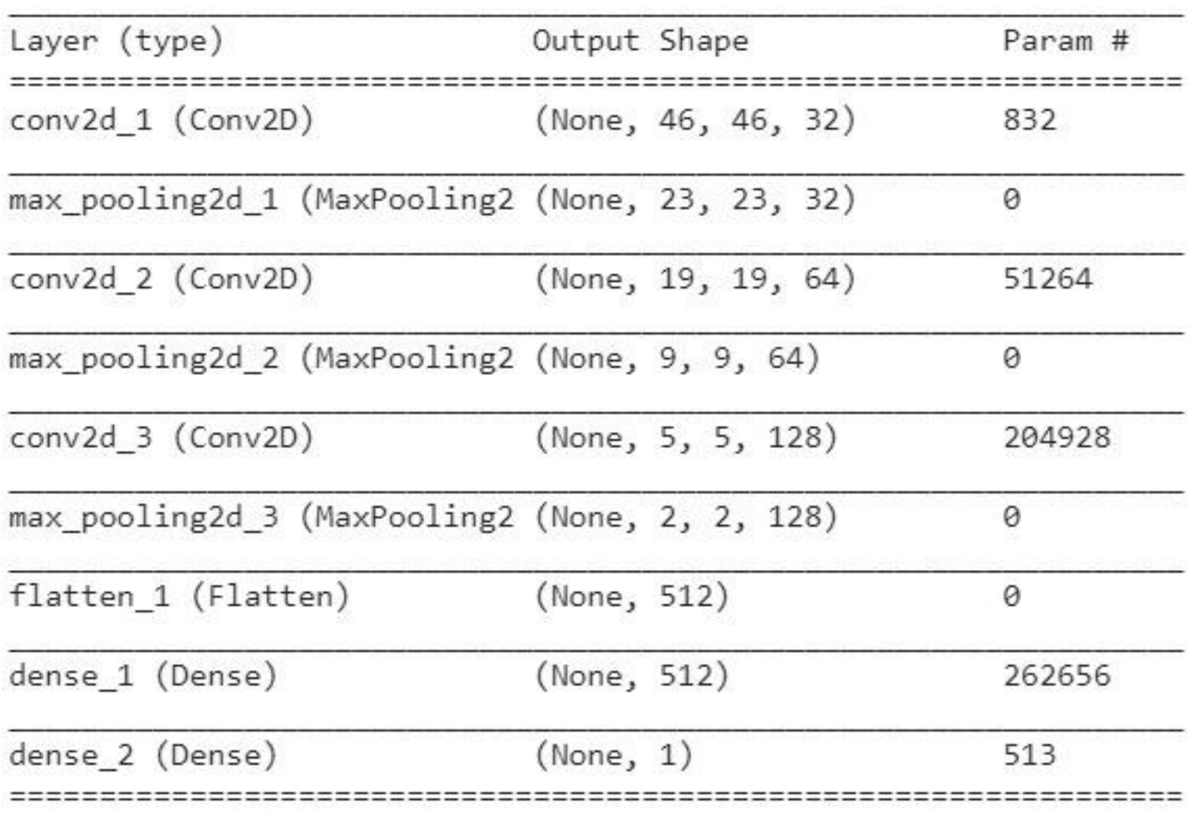
Frames: The video is divided into frames where we preprocess the frame and send it to the saved model to predict with mask/ without mask. A rectangular box is drawn over the face and displays whether it is mask or no mask along with accuracy.

Predict: The saved CNN model predicts whether the input frame is with mask or without mask image.

Implementation:

CNN:

Figure 1 describes our CNN architecture. We used 3 sets of Convolution and max pooling layers followed by a flatten layer, one hidden layer and one output layer. The input shape given to the model is 50*50*1, so the images need to be resized into 50*50 and need to convert them into grayscale images. In the 1st convolution layer we used 32 filters, in the 2nd 64 filters and in the 3rd 128 filters. The hidden layer contains 512 nodes and output contains one node which has value 1/0 (Mask/No Mask). The activation function used in the convolution layers is 'relu' and in the output layer is softmax. A total of 520,193 parameters are present in the neural network.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 46, 46, 32)        832       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 19, 19, 64)        51264     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 9, 64)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 5, 128)         204928    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513       
=================================================================
```

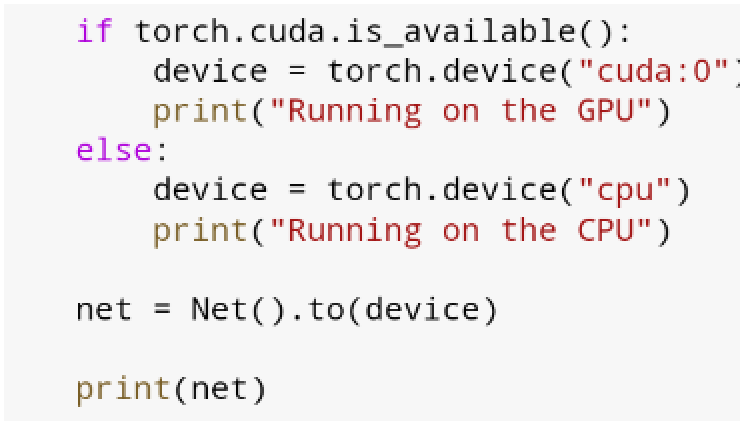
Figure 1-CNN Architecture

Pytorch and CPU:

We used Anaconda to code and train our model. The first step is to read each image from the dataset and convert them into grayscale image and then resize it to 50*50. We used opencv for these preprocessing steps. Then the image is converted into numpy array added to training dataset along with the one hot encode label of it. To train the dataset set using Pytorch framework, first we need to convert it to a tensor. The Pytorch command “torch.Tensor()” helps to do this. Then we need to make a class for CNN architecture. Pytorch contains inbuilt functions which help us to code neural network easily. Above mentioned CNN layers are implemented in this class. We are training for 30 epochs and batch size is set to 16.

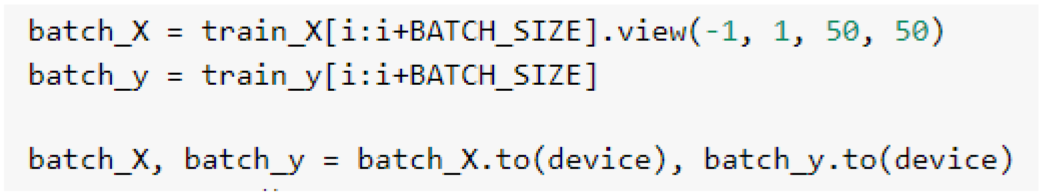
Pytorch and GPU:

We used “Google Colab” to work with GPU. We can convert the CPU program to cuda program just by changing the (.to(cuda)) in the functions that we need to convert and run them in cuda. It runs the code in GPU and distributes the work among all the processors. The preprocessing steps and CNN layer is similar to CPU. Figure 2 set the device as cuda if GPU is available or else it set to cpu. Initially all the variables and classes are set to cpu, so we to give “.to(device)” to the class or variable which we want to run on GPU. We need to take care whether all the dependencies to the neural network class are set to GPU. Because, when we try to operate one device in cpu and other in GPU it throws an error. So both the neural network class and tensors that we are going to train in the neural network should be allocated to GPU. In figure 3, it is shown that batches that we going to train are allocated to GPU.



```
if torch.cuda.is_available():
    device = torch.device("cuda:0"
    print("Running on the GPU")
else:
    device = torch.device("cpu")
    print("Running on the CPU")

net = Net().to(device)

print(net)
```

Figure 2-Allocating CNN to GPU



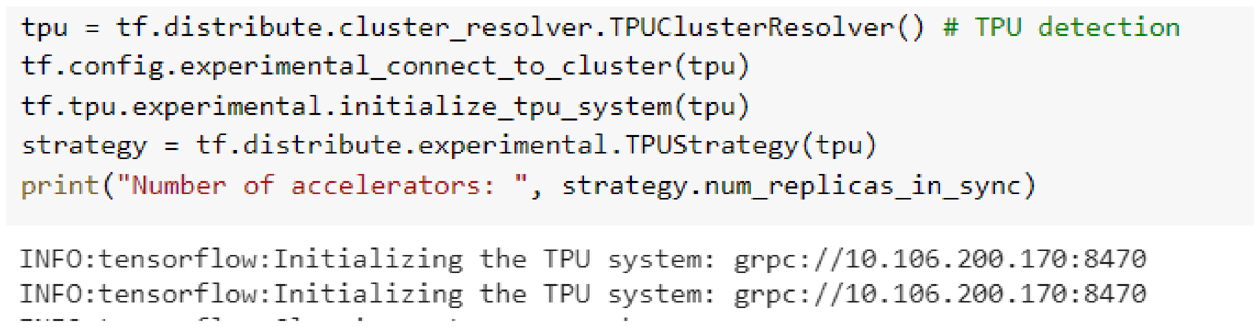
```
batch_X = train_X[i:i+BATCH_SIZE].view(-1, 1, 50, 50)
batch_y = train_y[i:i+BATCH_SIZE]

batch_X, batch_y = batch_X.to(device), batch_y.to(device)
```

Figure 3- Allocating batches to GPU

Tensorflow and TPU:

TPU is an architecture used for DL and ML computation made by Google. It is not a generic processor (i.e, only tensorflow models will run on it) so instead of making a general purpose processor, Google designed it as a matrix processor. TPU sends the parameter from memory into matrix adders and matrix multipliers.TPU loads the data which is from memory. As multiplication is executed, the output in each multiplier moves to the next multiplier. The result would be the summation of all the multiplied results of parameters. During this process of massive calculations and data passing, memory access is not required so that is why high computational throughput on neural network can be achieved by TPU.

We have worked on Google colab so it provides us with TPU. Normal code cannot be runned on TPU so we need to modify the code so that it runs on TPU. To convert, Colab provides us with TPU address and we call this TPU addressing using **gRPC** (gRPCis a modern, open source remote procedure call (RPC) framework that can run anywhere. It enables client and server applications to communicate and makes it to build connected systems). **TPUClusterResolver** does helps to bring the TPU address and creates a cluster to works on resolver is used to creates the initializing system. Even in TPU devices are created but are not converted as it is done GPU. It uses distributed strategy called **TPU strategy** and we pass resolver to TPU strategy and strategy is the final output. Strategy is like devices created in GPU for working in TPU. The variables created within the strategy scope will be replicated all across the replicas while using distributed strategies. **experimental_connect_to_cluster** will make devices on the cluster available to use i.e, calling this more of them will work but will invalidate any tensors which are on old remotes devices. **Initialize_tpu_system(tpu)** helps to initialize the device of TPU. Figure 4 describes how to connect with TPU using above mentioned commands. Colab TPU contains 8 cores, so our training data is distributed among 8 cores which speed up the process.



```
tpu = tf.distribute.cluster_resolver.TPUClusterResolver() # TPU detection
tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)
strategy = tf.distribute.experimental.TPUStrategy(tpu)
print("Number of accelerators: ", strategy.num_replicas_in_sync)
```

```
INFO:tensorflow:Initializing the TPU system: grpc://10.106.200.170:8470
INFO:tensorflow:Initializing the TPU system: grpc://10.106.200.170:8470
```

Figure 4-Initialiing TPU

Spark

Spark breaks the application into smaller tasks and assign them to the executors so it executes our application parallely. Here after loading the data we need a data structure to hold the data in spark. Data frames and datasets are compiled down to RDD (i.e Resilient Distributed Dataset). Spark RDD is resilient, partitioned, distributed and immutable. Resilient – RDDs can recover from failure so they are fault tolerant. Partitioned- Spark breaks the RDD into smaller chunks of data. These pieces are called partitions

Distributed - Instead of keeping those partitions on a single machine, Spark spreads them across the cluster. So they are a distributed collection of data.

Immutable - Once defined, you can't change a RDD. So Spark RDD is a read-only data structure.

There are two main variables to control the degree of parallelism in Apache Spark.

- The number of partitions
- The number of executors

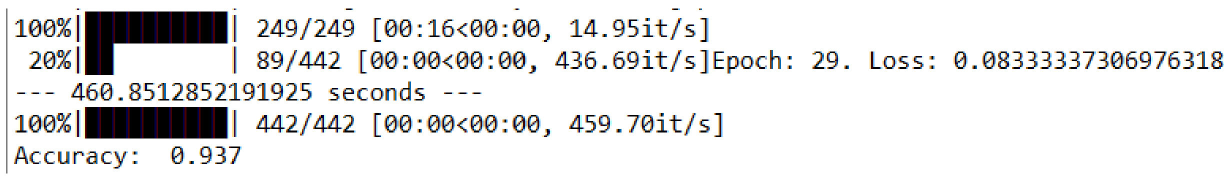
If you have ten partitions, you can achieve ten parallel processes at the most. However, if you have just two executors, all those ten partitions will be queued to those two executors.

Predicting in real time:

Once the model is trained, we can save it and we can use it to detect face mask in real time. We require a face detector to detect faces from the whole image before detecting whether the person is wearing mask or not. We are using "res10_300x300_ssd_iter_140000.caffemodel" to detect the faces from the whole image. Once the live stream starts, we took each frame and preprocess it before sending it to the model. The preprocessed image is sent to res10 caffemodel to detect faces from the image. Then we took each face with the help of coordinates and pass it through our face mask detection model. The detected faces are marked with a rectangle box and above each box we print whether mask or no mask along with accuracy.

Results:

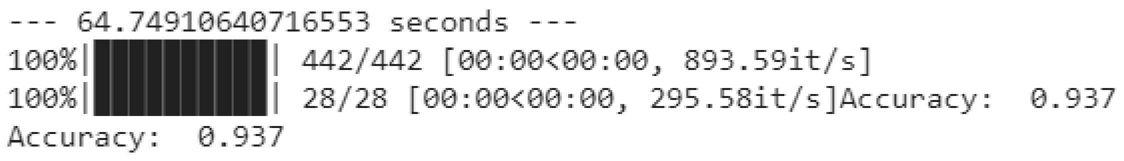
Training on CPU:



```
100%|██████████| 249/249 [00:16<00:00, 14.95it/s]
 20%|██        | 89/442 [00:00<00:00, 436.69it/s]Epoch: 29. Loss: 0.08333337306976318
--- 460.8512852191925 seconds ---
100%|██████████| 442/442 [00:00<00:00, 459.70it/s]
Accuracy:  0.937
```

Training time: 460.85 seconds

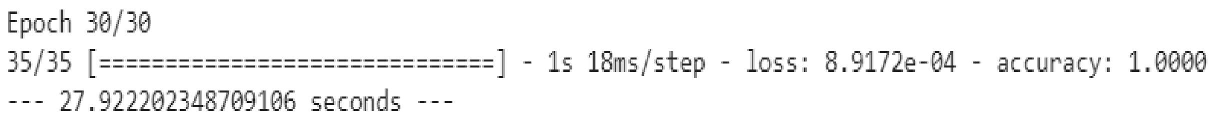
Training on GPU:



```
--- 64.74910640716553 seconds ---
100%|██████████| 442/442 [00:00<00:00, 893.59it/s]
100%|██████████| 28/28 [00:00<00:00, 295.58it/s]Accuracy:  0.937
Accuracy:  0.937
```

Training time: 64.74 seconds

Training on TPU:



```
Epoch 30/30
35/35 [==============================] - 1s 18ms/step - loss: 8.9172e-04 - accuracy: 1.0000
--- 27.922202348709106 seconds ---
```

Training time: 27.92 seconds

With Mask:

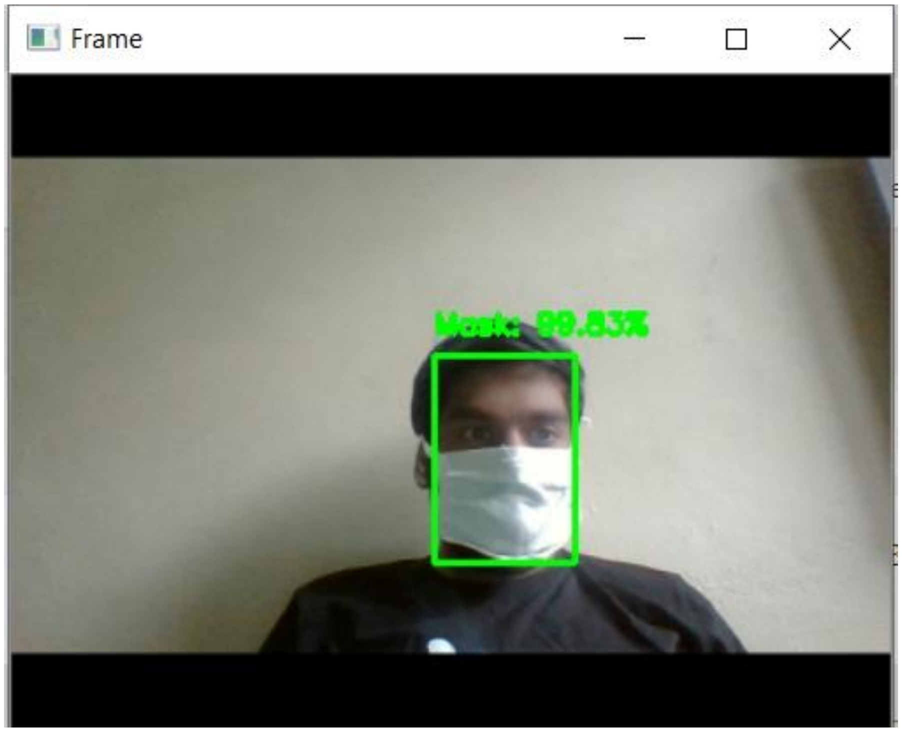


Figure 5-Detecting Mask in real time

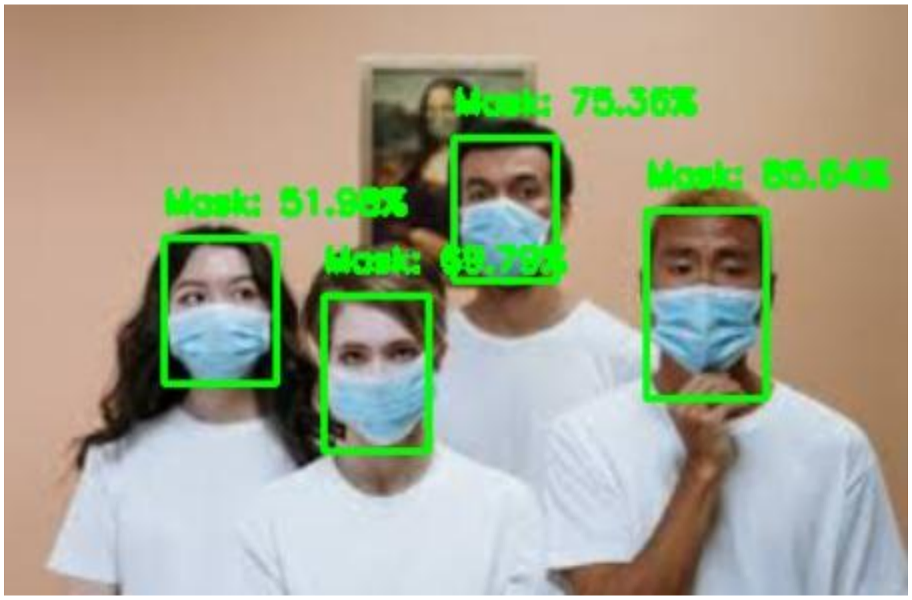


Figure 6-Detecting Mask in image

Without Mask:

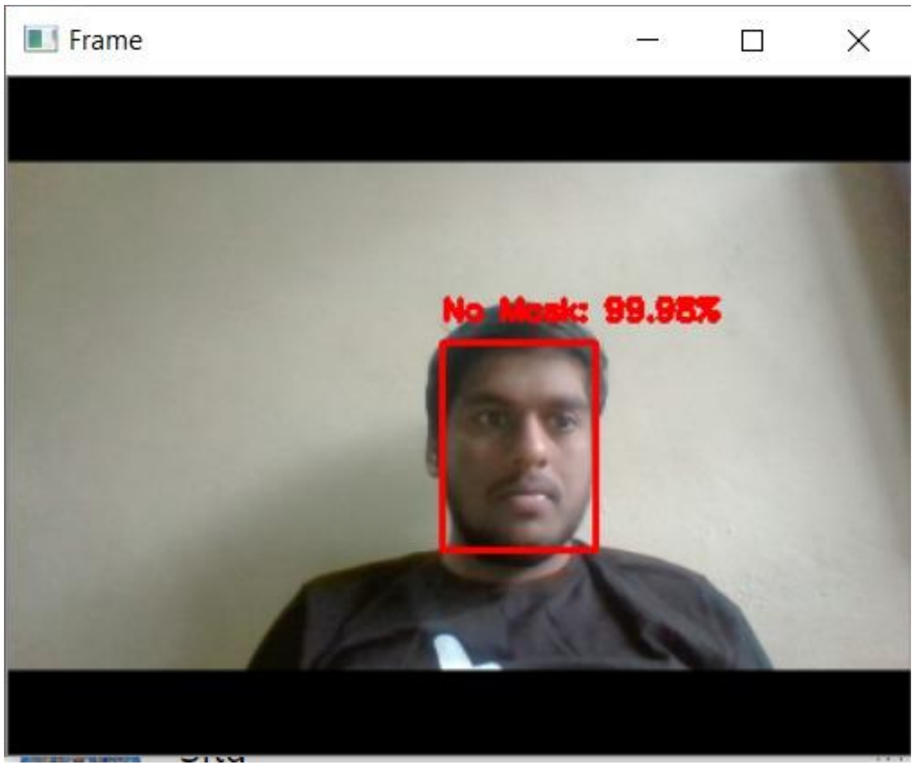


Figure 7-Detecting without Mask in real time

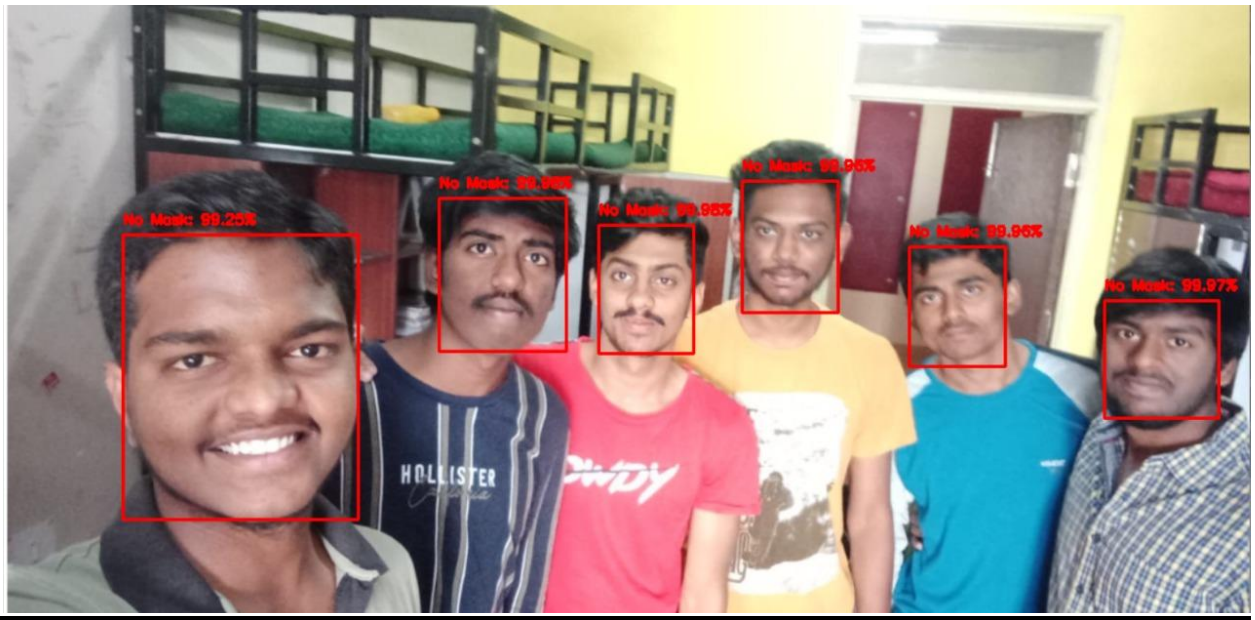


Figure 8-Detecting without Mask in image

Mixed Image:

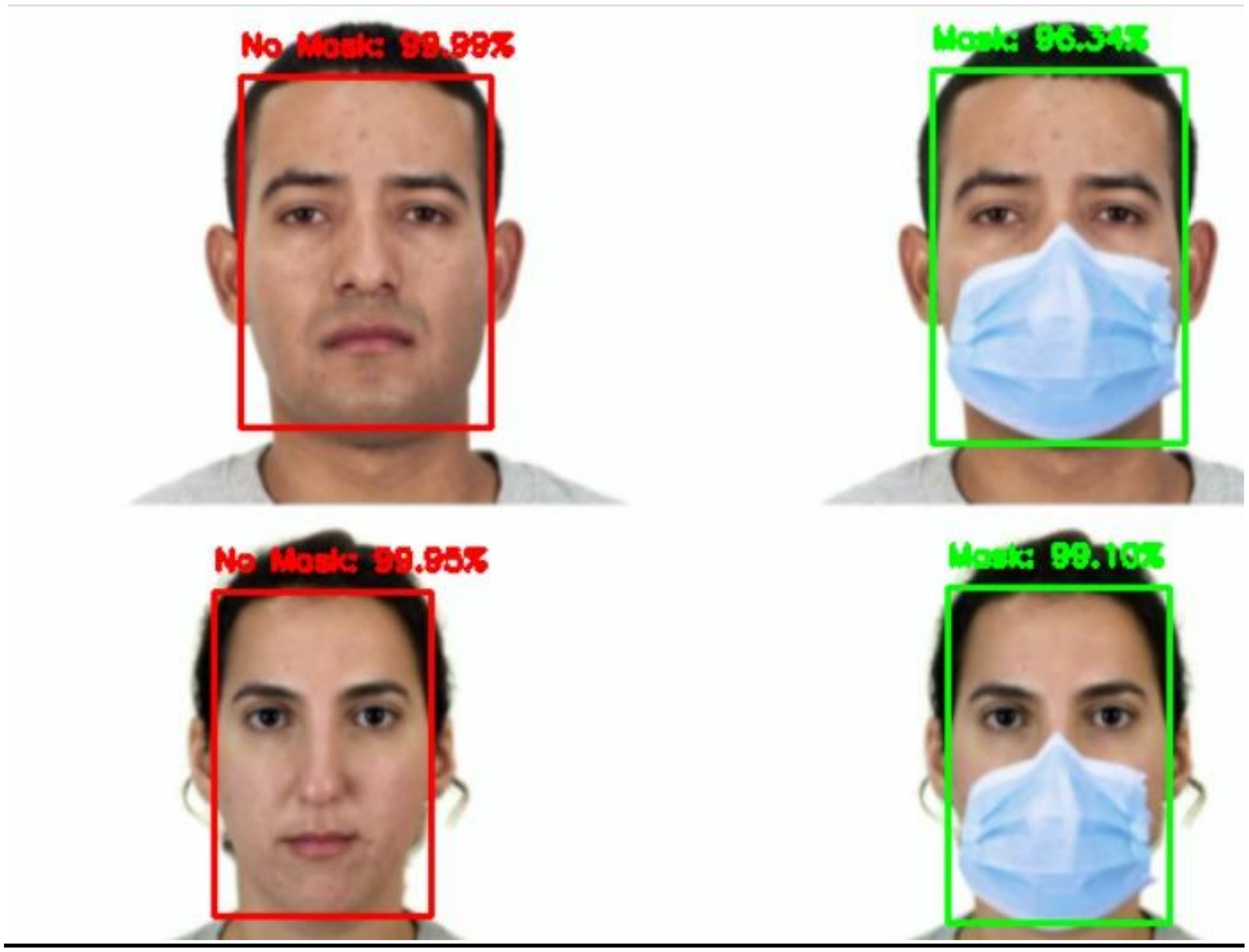


Figure 9-Detecting Mask and without Mask in image

<u>Result Discussion with comparison:</u>

The results show that we got less training time using TPU and GPU as compared to CPU.

Hardware	CPU	GPU	TPU
Training Time	460.85 sec	64.74 sec	27.92 sec

Conclusion:

- With the help of GPU and pytorch frame work and with the help of GPU and spark we detecting the face mask in live video which helps in analysing the awareness of wearing the masks in people. Now there will be no lag in detecting the face mask from the video as we are using the high performance distributed systems.
- Which can be helped in detecting the face masks and allows the person if he/she wears the mask else he will not be allowed in to the office based on the type of application we need we can add to it like creating alarm or glowing the warning lights .
- With the help of this we can also analyse how much people is aware of wearing the mask in different places through that certain actions will be taken by the government on them for increasing the awareness in people
- With the help of this how many people are coming by wearing the mask to an office the necessary precautions can be taken. Which helps in reducing the risk
- As the dataset increases the accuracy also increases.

References:

1) https://www.researchgate.net/publication/221392298_Performance_and_Scalability_of_GPU-Based_Convolutional_Neural_Networks
2) https://www.mdpi.com/2076-3417/10/1/83/pdf
3) https://www.researchgate.net/publication/341358704_An_Assessment_of_Gpu_and_Cpu_based_Convolutional_Neural_Network_for_Classification_of_White_Blood_Cells
4) https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9