

Face-Mask-Detection

(A Machine Learning based Application)

Kummari Thrikanth¹, Sandu Pavan Kumar², Kummari Aravind Kumar³, chillapally Yeshwanth⁴, Sheveta⁵

Abstract— In the present situation due to covid-19 there is no efficient face mask detection applications which can be used in densely populated and residential areas, large-scale manufactures and other enterprises to ensure safety. If we could make an efficient system to detect facemasks with an alert system, it will be very much useful to reduce the manpower and increases the security in the society. Metrics tools like “Neptune” and “WandB” are used to find out the system usability and complexity of the application.

Keywords— Mask-Detection-Application; Alert-System; Convolutional Neural Network; MobileNetV2 Model; Metrics-Tools; OpenCV; Usability; Complexity; Neptune; WandB.

I. Introduction

This project is related to the present pandemic issue i.e., detecting whether people are wearing masks properly or not, our input data set contains the set of images of people which will be classified into people with mask and the people without mask images, with Deep learning techniques in Machine learning using python programming language and Jupiter Notebook as a platform, MobileNetV2 Model and Convolutional Neural Network model are developed for training the images in the dataset to see which algorithm gives the best results of accuracy. After training both the models, the accuracy results stated that the MobileNetV2 model is more efficient and accurate than the CNN based model. The accuracy level for the CNN model is 91% and the accuracy level for the MobileNetV2 model is 96% and live streaming OpenCV is used to embed this with webcam. For this application metrics tools like Neptune and WandB are applied to visualize the complexity of the overall application, System usage, an application running time, GPU usage, and plotting the graphs for this application after the execution.

II. Literature Survey

In the literature survey, 34 articles had been referred and had some research on them. Some of the referral links have been declared in the references section. [1] was based on the most appropriate algorithms that are single shot methods such as YOLO or SSD by Alexandr Lorenzo. [2] here, the human face can be detected by the algorithm based on the moving target tracking. On the other hand, it can be divided face-mask into many kinds, such as the frontal face-mask, side face-mask and so on by Fu Yu and

Ningbo Zhu. [3] This pre-processing approach has been tested over the AR face database, using the DCT for features extraction, with MLP and RBF neural network as classifiers and the results compared with those coming from the original image and from the application of a simple black mask by Marco Grassi and Marcos Faundez Zanuy. Finally, from the survey, we came to the conclusion that the model has to be developed using algorithms of Machine-Learning, Deep-Learning, Artificial-Intelligence etc.,

A. Existing System

Applications like face detection are available which are developed using ML, DL, AI algorithms. Even mask detection algorithms have been trained but most of the systems predict the results with moderate accuracy and more data loss, this leads to reduce the optimization of the model and the true prediction rate get decrease. These applications just predict the person is with a mask or without a mask but in more populated areas like traffic and malls their system needs to recognize the mask and if a person is without a mask then the system has to alert the person to put on the mask, there is no alert system in previously developed models. Also, there is no concepts of metrics related tools in the existing systems to see the system usability and complexity of the Face Mask Detection Application.

B. Proposed System

In this paper, to build the application, the MobileNetV2 model have been optimized by decreasing the value of the learning rate to the minimum, and the loss rate while training the system has been minimized so that model will consider every local minimum and the accuracy level is increased. This system also has an additional functionality like an alert system if the detected person is without a mask; it's a sign of indication to the person to put on a mask. Also, metrics tools Neptune and WandB are used to visualize the system usability and complexity components of the application.

III. Methodology

The main steps involved in Machine Learning (ML) model are:

- Importing necessary packages

- Importing data set path
- Data Preprocessing
- Data Augmentation
- Data Split
- Building the model
- Training the model
- Results of the accuracy

Now, let's have a detailed study of each step of the ML Model.

- For the Face Mask Detection application, all the necessary packages are imported from the Tensor Flow and Keras libraries in python.
- The next step is importing the data set. The data set will be stored in the respective directory where the notebook file is stored. The path of the data set directory will be given for importing the data set for the further steps.
- The collected data don't need to be in the desired format. Therefore, data preprocessing techniques are used. Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. For performing data preprocessing using python we need to import some predefined Python libraries. i.e., Pandas.

Data preprocessing techniques include:

-Formatting of data

-Cleaning of data

-Sampling of data

- Data augmentation is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue. There is no need to collect new data, rather it transforms the existing data into the required forms and stores as augmented data.

Data augmentation operations include:

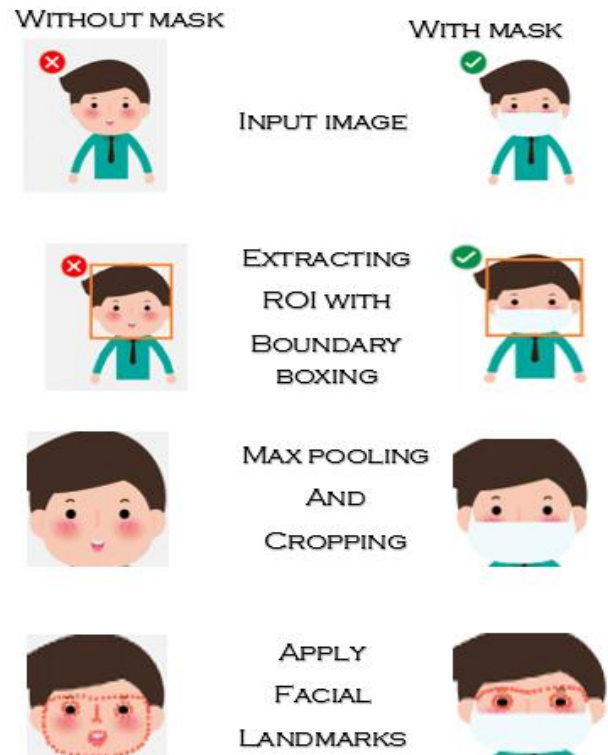
Boundary Boxing

Max Pooling

Cropping

Extracting the Region of Interest (ROI)

Applying the facial landmarks points:



- Whole data will be split for training and testing phases. 80% of the data will be given for the training phase and 20% of the data will be given for the testing phase.

There are two models for the comparative study of Face mask detection application. The two models are deep-learning models.

1. Convolutional Neural Network Model (CNN)

2. MobileNetV2

Both are CNN models, but the actual CNN model is the basic version model whereas the MobileNetV2 model is the updated version model.

Embedding model into live stream:

Deep learning is a branch of artificial intelligence (AI) that includes images and video. Its layering and abstraction give deep learning models almost human-behavior abilities including advanced image recognition techniques.

OpenCV is a widely adopted computer vision software technique used to run previously trained deep learning models on ordinary hardware and generate powerful insights from digital images and video streams. It allows us to see how to view and load images and videos using OpenCV. Also provides how to do classification for both images and videos and leverage MobileNetV2 for custom object detection.

Often, a camera is used to capture live stream of images or videos. This OpenCV library provides a very simple interface to do such things. The trained model is called into the OpenCV model for sequence training of the video streams.

Alert System:

A python library called Play Sound is used in this project. After embedding the trained model with OpenCV, in the video stream if the person is not wearing a mask it will display without mask accuracy for the person identified. This Play Sound library will have an audio file with a message "Please wear a face mask". So, the audio will be played till the person is captured with a face mask.

Metrics Tools:

Neptune:

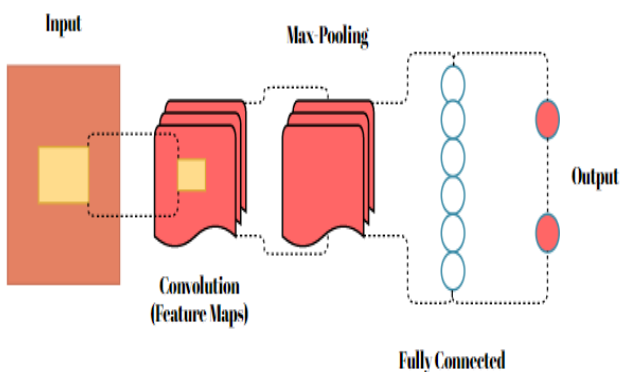
Neptune is a metrics tool used for most of the machine learning based projects. Even it has an option of configuring the jupyter notebbok files into it's directory and configure them together for monitoring the machine learning process running live. It is a lightweight experiment management tool. It helps to keep track of all machine learning based experiments. This tool is used to see the system usability and complexity of a machine learning project.

WandB:

Expansion of WandB is "Weights & Biases". Training modules are monitored in real time using this WandB tool. It can also be integrated with Machine Learning and Deep learning frameworks. It also allows to compare different projects in a same platforms to identify the best performing model.

IV. Implementation

IMPLEMENTATION OF METHODOLOGIES OF THE NEURAL NETWORK MODELS:



Convolution: Convolutions are a type of operation that can be used to learn representations from images. They

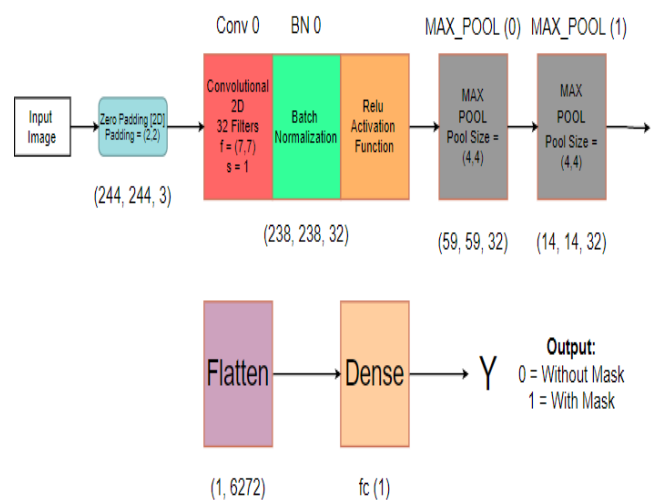
include a learnable kernel slides over the image and performing element wise multiplication with the input. The specification permits for parameter sharing and translation invariance. Below you will be able to find a continuously updating list of convolutions.

Pooling: Max Pooling is a pooling operation that calculates the utmost value for patches of a feature map and uses it to form and create a down sampled (pooled) feature map. It adds a little quantity of translation in-variance – that means translating the image by a small quantity does not considerably have an effect on the values of most pooled outputs.

Convolutional Neural Network Model:

Steps for CNN based Classification:

1. Apply the convolution filter in the first layer
2. The sensitivity of the filter is reduced by smoothing the convolution filter i.e. subsampling.
3. The signal transfers from one layer to another layer are controlled by the activation layer
4. Fasten the training period by using the rectified linear unit (RELU). The neurons in the proceeding layer are connected to every neuron in the subsequent layer
5. During training, the loss layer is added at the end to give feedback to the neural network.



Convolutional Neural Network Architecture

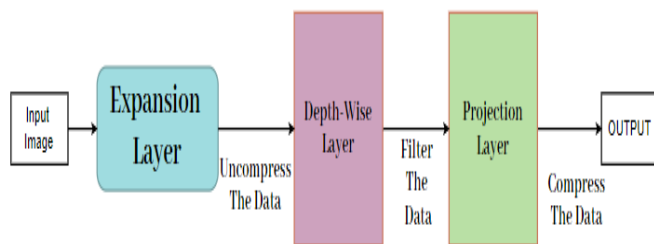
Understanding the architecture:

Each input x (image) features a form of $(244, 244, 3)$ and is fed into the neural network. And, it goes through the subsequent layers:

1. A Zero Padding layer with a pool size of $(2, 2)$.
2. A convolutional layer with thirty-two filters, with a filter size of $(7, 7)$ and a stride equal to 1.
3. A batch normalization layer to normalize pixel values to speed up computation.
4. A Relu activation layer.
5. A Max Pooling layer with $f=4$ and $s=4$.
6. A Max Pooling layer with $f=4$ and $s=4$, same as before.
7. A flatten layer to flatten the 3-D matrix into a one-dimensional vector.

A Dense (output unit) fully connected layer with one neuron with a sigmoid activation (since this is often a binary classification task).

MobileNetV2 Model:



MobileNetV2 Architecture

It is a CNN architecture model for Image classification and Mobile Vision. There are other models as well but MobileNetV2 has less computation power to run or apply transfer learning. This makes a perfect fit for Mobile devices, embedded systems, and computers without GPU or low computational efficiency with compromising significantly with the accuracy of results. It is also best suited for web browsers to have limitations over computation, graphic processing, and storage. This method process very fast and uses fewer parameters.

Open Source Computer Vision Library is used for MobileNetV2 which is mainly aimed at real-time computer vision.

MobileNetV2 for mobile and embedded vision applications are projected, which are based on a streamlined architecture that uses depth wise divisible convolutions to make light-weight deep neural networks.

Two straightforward global hyper-parameters that efficiently trade-off between latency and accuracy are introduced.

The core layer of MobileNetV2 is depth-wise divisible filters, named as Depth wise Separable Convolution. The network structure is another issue to speed up performance. Finally, the width and resolution are often tuned to the trade-off between latency and accuracy.

MobileNetV2 Model has two sub-models.

They are:

->Base Model

->Head Model

Base Model:

This model is mainly used to take inputs, here parameters like

- Imagenet is used which is a pre-trained model for loading the images so that default weights are initialized.

- Input_tensor to set the shape of the image

For example: `shape_image(224,224,3)`

224*224 represents the size of the image.

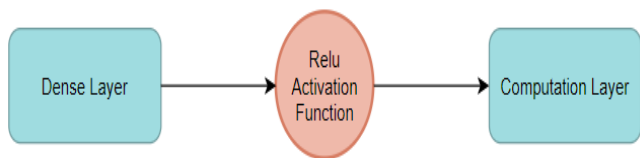
3 represents the channels which are R G B as we are giving the inputs which are colored.

Head Model:

The output of the base model is taken as an input of the head model

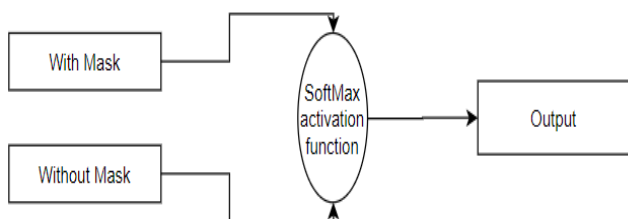
Head model = base model.output

The flow of neural networks is described below



ReLU Activation function is for Non-Linear Usecases.

Computation Layer



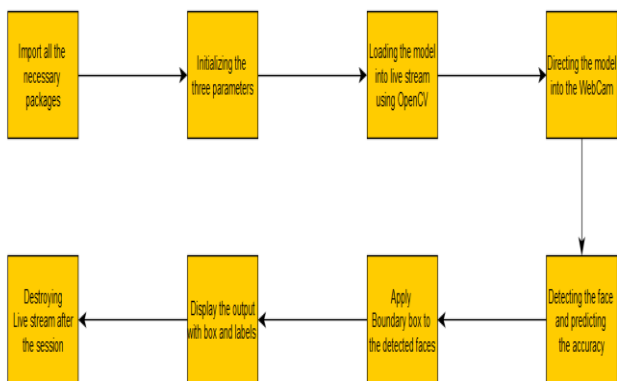
SoftMax Activation function is for finding the probabilities in either '0' or '1'. Non-Linear.

Embedding Face mask Detection Model into live Web Camera for Streaming

The ultimate aim is to check how the trained model is working for the live streaming video. If we embed the Face Mask Detection Model to the nearby CCTVs without more man power, the model can detect whether the people around there are with or without mask.

In order to make an embedded system of our model with the CCTVs, our existing model is further implemented with the applications of OpenCV.

OpenCV Architectural Diagram:



Steps involved in the Architecture:

Applying model in camera:

Step-1:

Initialize three parameters:

- Frame
- Face Net
- Mask Net

Face Net

This parameter is used to for detecting faces in the frames using read Net using CV2 with deep Neural Networks (DNN).

Frame

Frame gets the images of live streaming in the form of frame I.e frame/sec and it is visualized as video in a dialog box.

Mask Net

Here Mobile Net V2 trained model is used for detecting masks.

Step-2

Loading the model into live video streaming, video_stream method is used for it giving the source value as '0', zero indicates the number of cameras are going to use for streaming.

Example: If 2 cameras are going to use then the source value is set to '1'.

Step-3

Detecting the face and predicting the accuracy of safely presence of mask. All initialized parameters are used in this method.

Detect_and_predict_mask (faceNet, MaskNet, frame)

Final outcome of this method is it returns the co-ordinates of the identified face in a rectangle box and predicting the accuracy of the mask

Step-4

Unpacking the bounding box

In the total frame face is identified and need to classify whether a person is wearing mask or not. There labels have been initialized with condition if the person is with mask then it is highlighted in green box with accuracy else

detected without mask then highlighted in red box with accuracy.

Sample:

Colour (b, g, r)- b, g, r represents the colours which are represented in a range (0-255)

(0,255,0)->with mask (green colour is displayed)

(0,0,255)->without mask (red colour is displayed)

Step-5

Displaying the box and Labels

Initialize a rectangle frame using CV2 with parameters frame, and co-ordinates of the detected face, colour, thickness of rectangle.

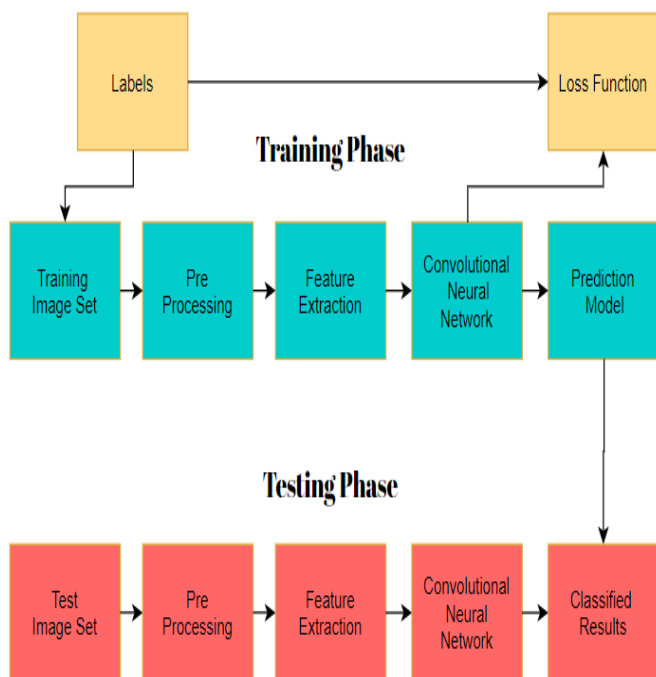
Step-6

Destroying the live stream

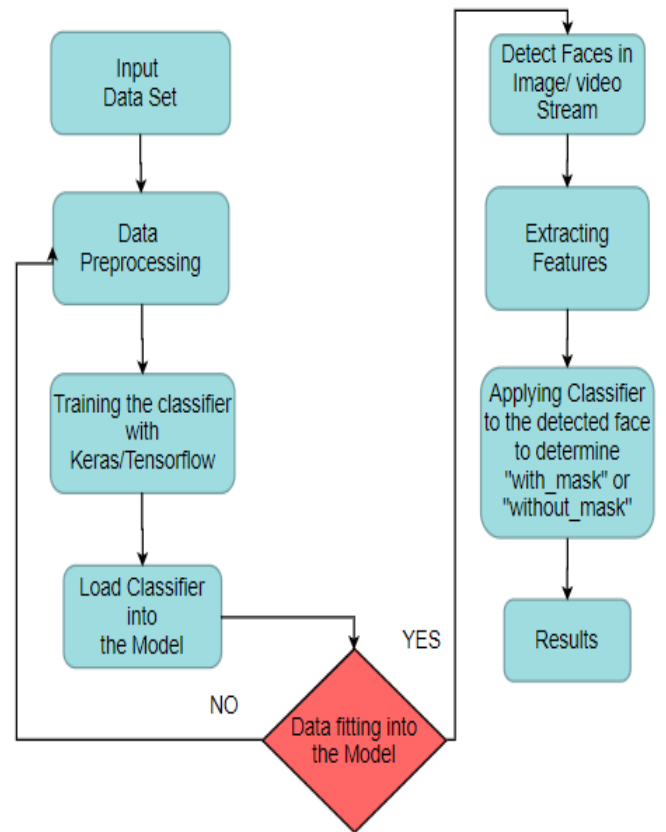
Using Destroy () method- to stop the streaming.

V. ARCHITECTURAL DIAGRAMS

The Training and Testing Phases:



The Overall Model building architecture:



VI. EXPERIMENTAL RESULTS

A. Training and Testing the Models

The two models which are discussed in the implementation section are trained and tested in Jupyter Notebook provided with 20 epochs for both Convolutional neural network model and the MobileNetV2 model. After optimizing the learning rate in the MobileNetV2 model, the accuracy of it has been improved in every epoch when compared with the basic CNN model.

The below figure represents the epochs accuracy of both the neural network models:

CNN MODEL:

```

Epoch 1/20
2743/2743 [=====] - 154s 56ms/step - loss: 0.6588 - accuracy: 0.6819 - val_loss: 0.5372 - val_accuracy: 0.7289
Epoch 2/20
2743/2743 [=====] - 162s 59ms/step - loss: 0.5185 - accuracy: 0.7382 - val_loss: 0.4242 - val_accuracy: 0.8195
Epoch 3/20
2743/2743 [=====] - 153s 56ms/step - loss: 0.4190 - accuracy: 0.8097 - val_loss: 0.4016 - val_accuracy: 0.8222
Epoch 4/20
2743/2743 [=====] - 155s 56ms/step - loss: 0.3383 - accuracy: 0.8491 - val_loss: 0.3315 - val_accuracy: 0.8661
Epoch 5/20
2743/2743 [=====] - 157s 57ms/step - loss: 0.2751 - accuracy: 0.8819 - val_loss: 0.3016 - val_accuracy: 0.8761
Epoch 6/20
2743/2743 [=====] - 191s 70ms/step - loss: 0.2389 - accuracy: 0.8994 - val_loss: 0.3021 - val_accuracy: 0.8688
Epoch 7/20
2743/2743 [=====] - 180s 66ms/step - loss: 0.2150 - accuracy: 0.9114 - val_loss: 0.2879 - val_accuracy: 0.8997
Epoch 8/20
2743/2743 [=====] - 165s 60ms/step - loss: 0.1734 - accuracy: 0.9358 - val_loss: 0.3124 - val_accuracy: 0.8950
Epoch 9/20
2743/2743 [=====] - 165s 60ms/step - loss: 0.1686 - accuracy: 0.9318 - val_loss: 0.3039 - val_accuracy: 0.8863
Epoch 10/20
2743/2743 [=====] - 185s 67ms/step - loss: 0.1320 - accuracy: 0.9471 - val_loss: 0.3060 - val_accuracy: 0.8776
Epoch 11/20
2743/2743 [=====] - 171s 62ms/step - loss: 0.1143 - accuracy: 0.9614 - val_loss: 0.2796 - val_accuracy: 0.8965
Epoch 12/20
2743/2743 [=====] - 153s 56ms/step - loss: 0.0930 - accuracy: 0.9661 - val_loss: 0.2956 - val_accuracy: 0.8980
Epoch 13/20
2743/2743 [=====] - 152s 56ms/step - loss: 0.0848 - accuracy: 0.9727 - val_loss: 0.3168 - val_accuracy: 0.8965
Epoch 14/20
2743/2743 [=====] - 151s 55ms/step - loss: 0.0851 - accuracy: 0.9668 - val_loss: 0.3047 - val_accuracy: 0.8965
Epoch 15/20
2743/2743 [=====] - 802s 292ms/step - loss: 0.0812 - accuracy: 0.9708 - val_loss: 0.2964 - val_accuracy: 0.8819
Epoch 16/20
2743/2743 [=====] - 157s 57ms/step - loss: 0.0606 - accuracy: 0.9792 - val_loss: 0.3064 - val_accuracy: 0.8965
Epoch 17/20
2743/2743 [=====] - 162s 59ms/step - loss: 0.0450 - accuracy: 0.9851 - val_loss: 0.3499 - val_accuracy: 0.8892
Epoch 18/20
2743/2743 [=====] - 170s 62ms/step - loss: 0.0496 - accuracy: 0.9829 - val_loss: 0.3315 - val_accuracy: 0.8892
Epoch 19/20
2743/2743 [=====] - 164s 60ms/step - loss: 0.0369 - accuracy: 0.9883 - val_loss: 0.3497 - val_accuracy: 0.8986
Epoch 20/20
2743/2743 [=====] - 163s 60ms/step - loss: 0.0411 - accuracy: 0.9836 - val_loss: 0.3602 - val_accuracy: 0.8965
    
```

MobileNetV2 MODEL:

```

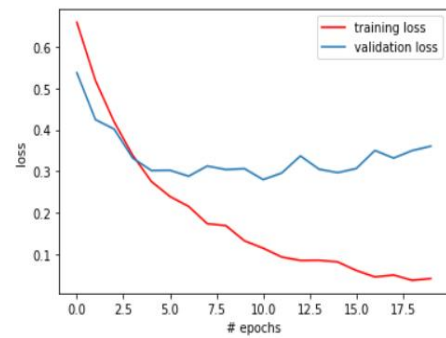
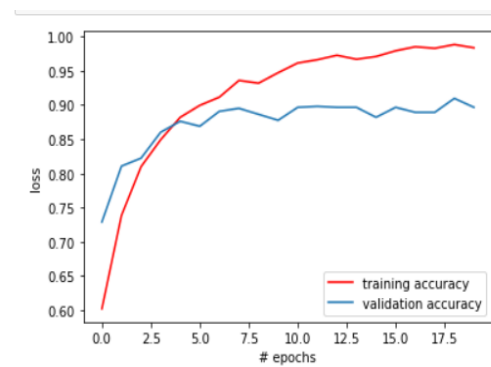
Epoch 1/20
95/95 [=====] - 859s 9s/step - loss: 0.5754 - accuracy: 0.7027 - val_loss: 0.4042 - val_accuracy: 0.8111
Epoch 2/20
95/95 [=====] - 776s 8s/step - loss: 0.3175 - accuracy: 0.8718 - val_loss: 0.3369 - val_accuracy: 0.8533
Epoch 3/20
95/95 [=====] - 745s 8s/step - loss: 0.2425 - accuracy: 0.9028 - val_loss: 0.3443 - val_accuracy: 0.8601
Epoch 4/20
95/95 [=====] - 749s 8s/step - loss: 0.2082 - accuracy: 0.9232 - val_loss: 0.2975 - val_accuracy: 0.8804
Epoch 5/20
95/95 [=====] - 837s 9s/step - loss: 0.1734 - accuracy: 0.9448 - val_loss: 0.4072 - val_accuracy: 0.8519
Epoch 6/20
95/95 [=====] - 1954s 21s/step - loss: 0.1546 - accuracy: 0.9417 - val_loss: 0.3028 - val_accuracy: 0.8859
Epoch 7/20
95/95 [=====] - 937s 10s/step - loss: 0.1434 - accuracy: 0.9430 - val_loss: 0.3911 - val_accuracy: 0.8573
Epoch 8/20
95/95 [=====] - 903s 10s/step - loss: 0.1508 - accuracy: 0.9469 - val_loss: 0.2452 - val_accuracy: 0.9002
Epoch 9/20
95/95 [=====] - 900s 9s/step - loss: 0.1387 - accuracy: 0.9479 - val_loss: 0.2370 - val_accuracy: 0.9076
Epoch 10/20
95/95 [=====] - 853s 9s/step - loss: 0.1210 - accuracy: 0.9604 - val_loss: 0.3965 - val_accuracy: 0.8641
Epoch 11/20
95/95 [=====] - 767s 8s/step - loss: 0.1169 - accuracy: 0.9555 - val_loss: 0.4015 - val_accuracy: 0.8655
Epoch 12/20
95/95 [=====] - 749s 8s/step - loss: 0.1112 - accuracy: 0.9562 - val_loss: 0.2722 - val_accuracy: 0.9088
Epoch 13/20
95/95 [=====] - 2271s 24s/step - loss: 0.1030 - accuracy: 0.9654 - val_loss: 0.2682 - val_accuracy: 0.9080
Epoch 14/20
95/95 [=====] - 852s 9s/step - loss: 0.1041 - accuracy: 0.9624 - val_loss: 0.1774 - val_accuracy: 0.9226
Epoch 15/20
95/95 [=====] - 774s 8s/step - loss: 0.1043 - accuracy: 0.9611 - val_loss: 0.2217 - val_accuracy: 0.9117
Epoch 16/20
95/95 [=====] - 833s 9s/step - loss: 0.0959 - accuracy: 0.9667 - val_loss: 0.2579 - val_accuracy: 0.9040
Epoch 17/20
95/95 [=====] - 898s 9s/step - loss: 0.0832 - accuracy: 0.9703 - val_loss: 0.2533 - val_accuracy: 0.9183
Epoch 18/20
95/95 [=====] - 862s 9s/step - loss: 0.1009 - accuracy: 0.9647 - val_loss: 0.2229 - val_accuracy: 0.9130
Epoch 19/20
95/95 [=====] - 816s 9s/step - loss: 0.0960 - accuracy: 0.9654 - val_loss: 0.2842 - val_accuracy: 0.8954
Epoch 20/20
95/95 [=====] - 857s 9s/step - loss: 0.0930 - accuracy: 0.9687 - val_loss: 0.3258 - val_accuracy: 0.8899
    
```

Classification Report of MobileNetV2 Model:

	precision	recall	f1-score	support
with_mask	0.99	0.79	0.88	383
without_mask	0.83	0.99	0.90	384
accuracy			0.89	767
macro avg	0.91	0.89	0.89	767
weighted avg	0.91	0.89	0.89	767

Below mentioned graphs shows the loss rate vs epochs graph plotted with training loss, validation loss, training accuracy, validation accuracy.

CNN MODEL:



MobileNetV2 MODEL:



B. Mathematical explanation on learning rate

The reason behind Minimizing the learning rate.

Learning Rate value is always range between low-learning rate to the high learning rate.

Low-LR < good-LR < high-LR < Very high-LR

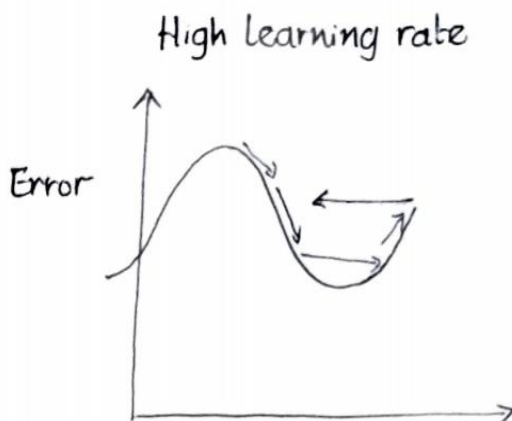
High Learning rate:

When the value of the learning rate is bigger value then the steps to perform the task also ranges higher, which makes the model much faster but in this case while computation is going on some local minimums are missed and the overall model leads to data loss and finally, the accuracy level drops, even model has been built with more image generator model but while training the data set some data to get LOSS.

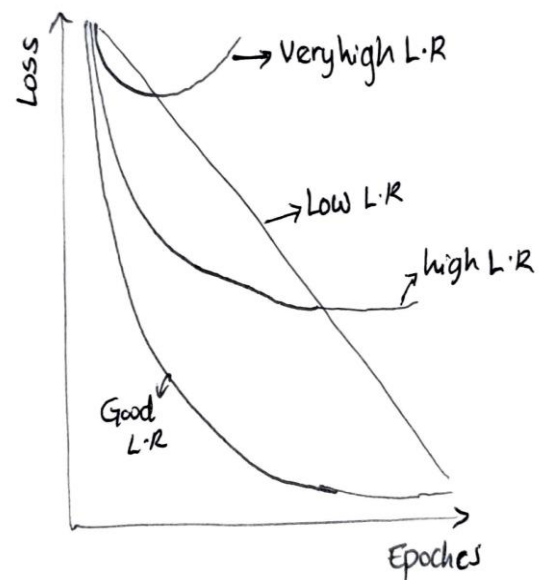
Low Learning rate:

Value of learning rate is smaller so the steps to perform the task also ranges low and every local minimum value has been considered and model gets trained well and more accurate due to minimum loss rate but this takes some time to train the model which is negligible and finally model meets all the criteria to get a perfect model.

Graphs:



Lower the value, slower we travel along the slope, i.e, making sure that model reaches maximum local minima values.



1 Epoch = 1 pass per data

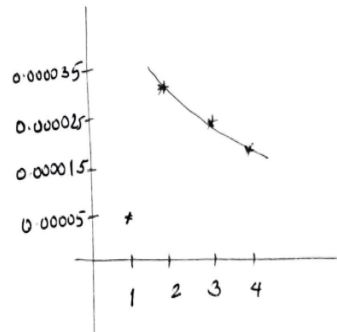
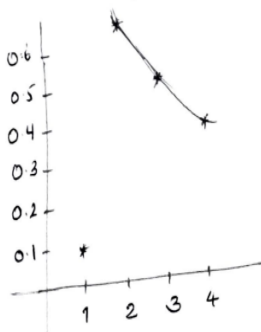
$$\alpha = \frac{1}{1 + \text{decayrate} \times \text{epoch_num}} \times \alpha_0$$

Let $\alpha_0 = 0.2$
decay rate = 1

Let $\alpha_0 = 0.0001$
decay rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4

Epoch	α
1	0.000105
2	0.000033
3	0.000025
4	0.00002

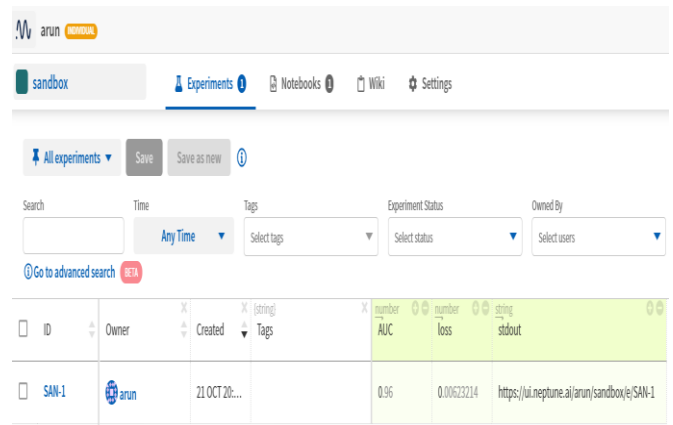


C. Applying Metrics on developed Application

Applications of Neptune Tool:

- Easily keep track of metrics, hyperparameters
- Visualize losses and metrics as your model is training (monitor learning curves)
- Compare learning curves across various models/experiments
- Use interactive comparison table that automatically shows diffs between experiments
- Fetch experiment data and visualize parameters and metrics in notebooks. You can use the HiPlot integration or do custom analysis
- It has other visualization features that are not parameter-metric related

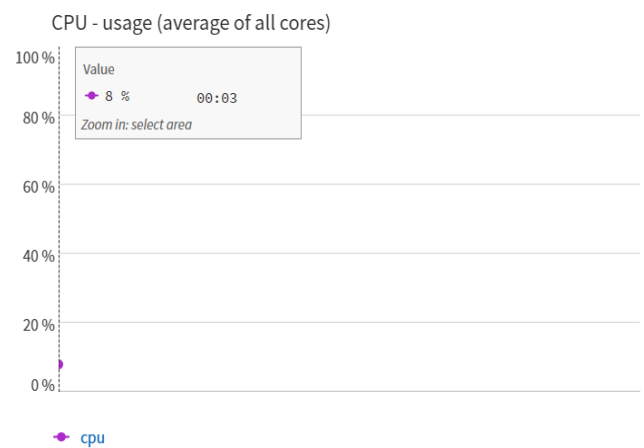
Below figure represents the output after applying Neptune tool for the developed Face mask detection application.



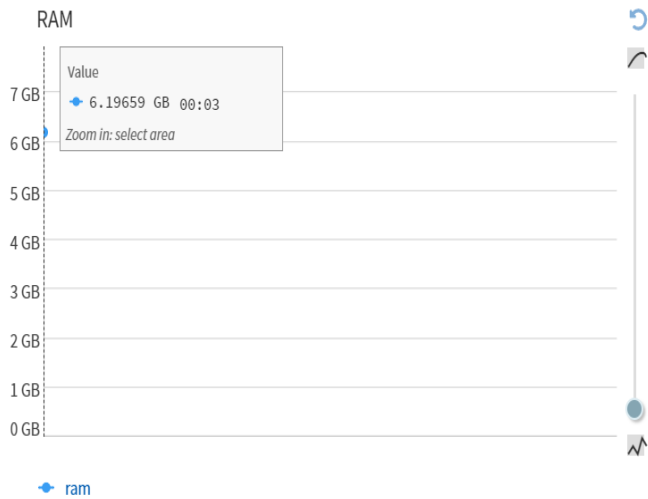
Here, accuracy score and loss rate are model results, these outputs are visualized in Neptune tool, this output URL is generated by the model for visualization.

As mentioned before, this tool helps us to monitor the system usability and complexity of the projects running in real time. The following figures shows the CPU, RAM, GPU consumption and time vs loss plot, logarithmic view etc.,

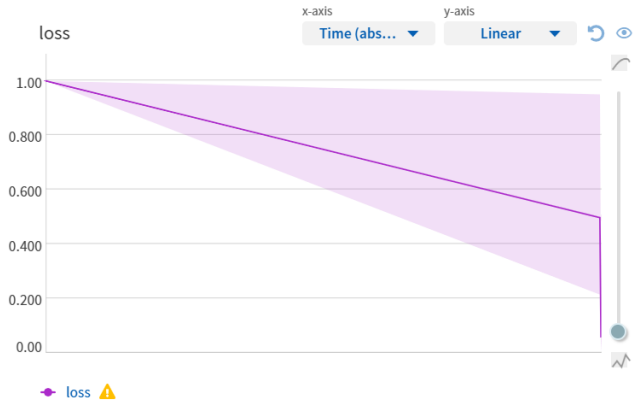
CPU Usage:



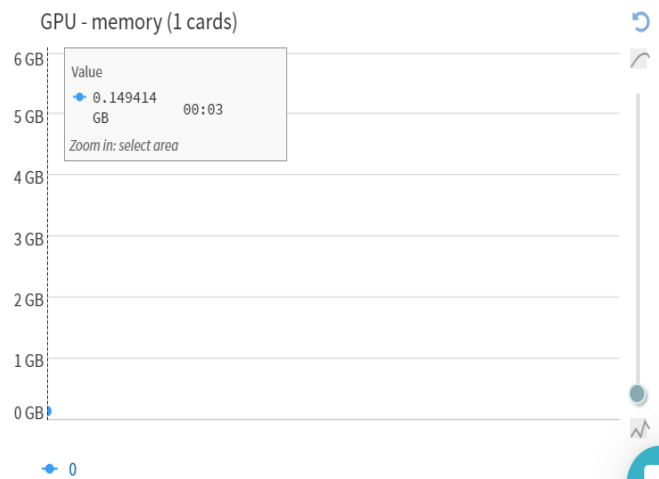
RAM Usage:



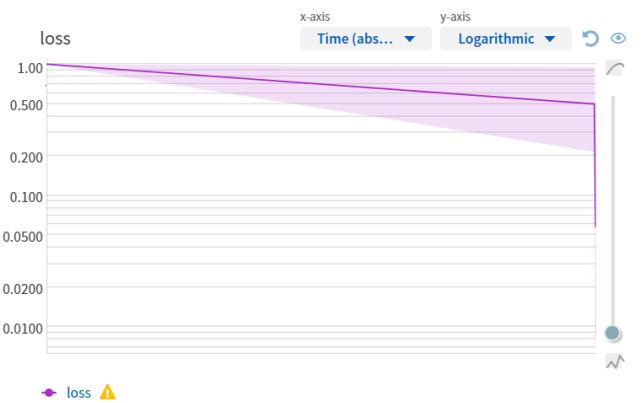
Absolute time vs Loss:



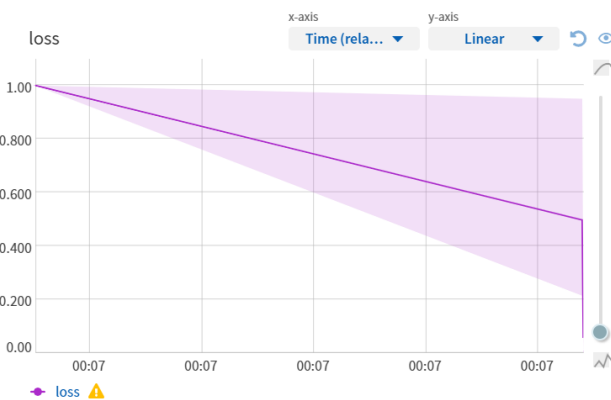
GPU Usage:



Logarithmic view:



Relative time vs Loss:



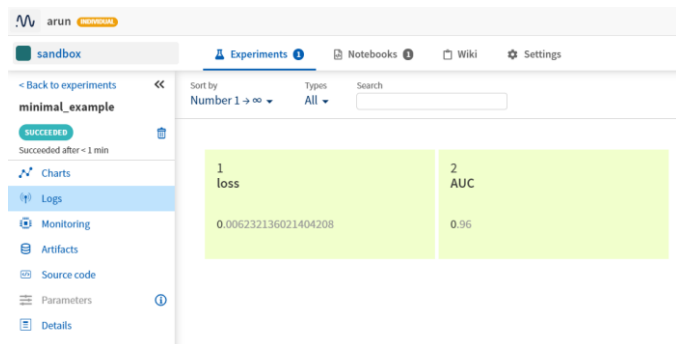
Overall status of the application:

# Metadata		
Experiment name	minimal_example	Status SUCCEEDED
ID	SAN-1	Tags <input type="text" value="Add a tag"/>
Created	2020/10/21 20:28:01	Description <input type="text" value="Add description"/>
Completed	2020/10/21 20:28:10	
Owned by	arun	

Source code generated by Neptune tool:

```
Source code
Size 327 B
Entrypoint main.py
source ↓
Name main.py
1 import neptune
2
3 # The init() function called this way assumes that
4 # NEPTUNE_API_TOKEN environment variable is defined.
5
6 neptune.init('arun/sandbox')
7 neptune.create_experiment(name='minimal_example')
8
9 # log some metrics
10
11 for i in range(100):
12     neptune.log_metric('loss', 0.95*i)
13
14 neptune.log_metric('AUC', 0.96)
```

System Logs:



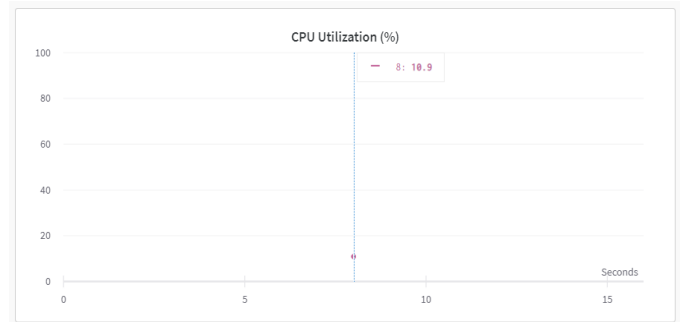
Step	Metric	Value
1	loss	0.006232136021404208
2	AUC	0.96

Applications of WandB Tool:

- Monitor training runs information like loss, accuracy (learning curves)
- Compare runs with a dashboard table showing auto-diffs
- Visualize parameters and metrics via parallel coordinates plot
- Explore how parameters affect metrics with feature (parameter) importance visualization (this I think is experimental)
- It has other visualization features that are not parameter-metric related.
- Results after performing Neptune for face mask detection application

This tool also helps us to monitor the training models of the machine learning in real-time. The below figures represent the system usability using WandB metrics tool.

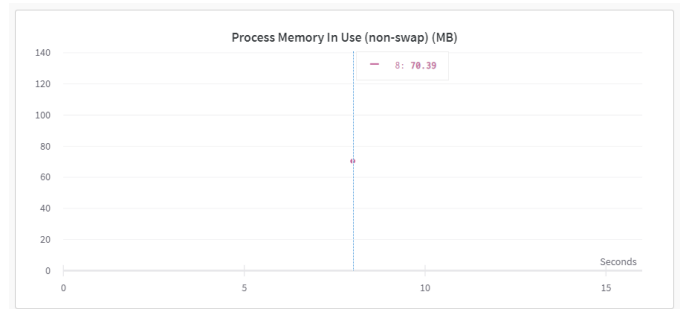
CPU Usage:



System Memory Usage:



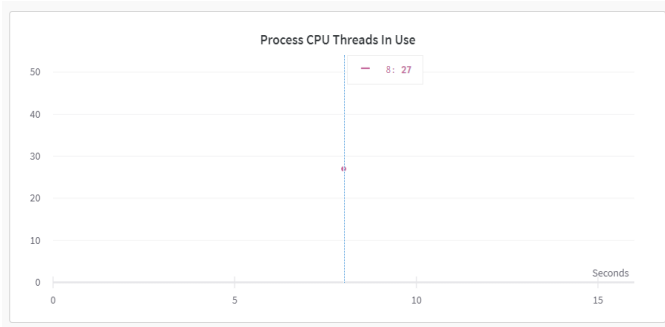
Process Memory Usage:



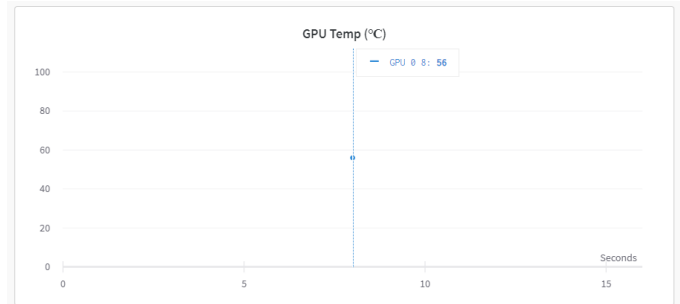
Process Memory Available:



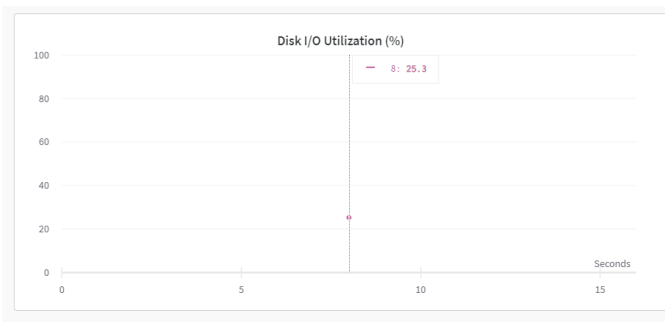
CPU Threads usage:



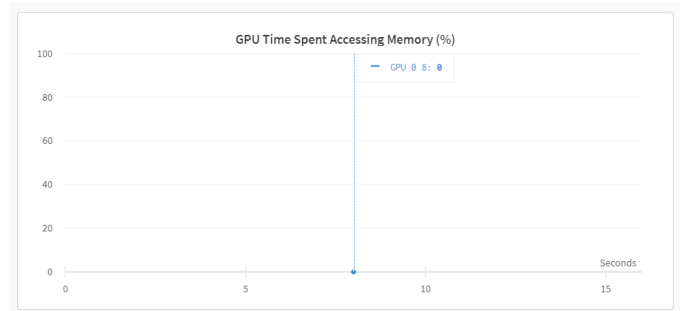
GPU Temperature:



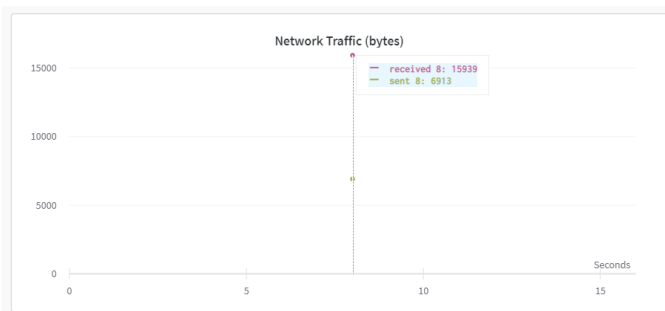
Disk I/O Usage:



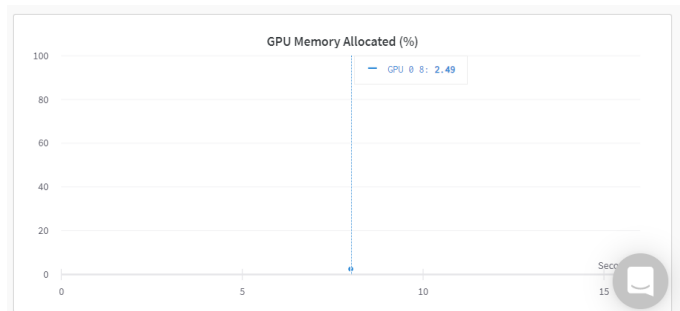
GPU Time Spent Accessing Memory:



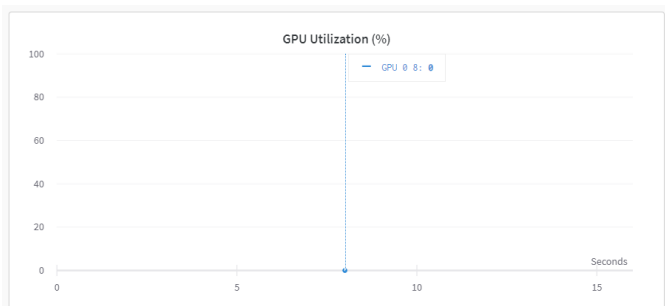
Network Traffic:



GPU Memory Allocated:



GPU Usage:



System utilization and GPU power consumption time and temperature rate are visualized in WandB tool and this link is generated after training the model.

```
Anaconda Prompt (anaconda3)
accuracy          0.99      767
macro avg         0.99      0.99    767
weighted avg      0.99      0.99    767

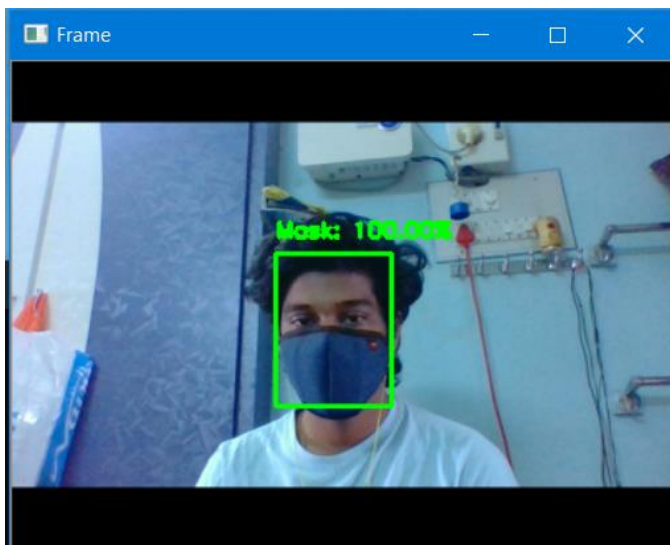
[INFO] saving mask detector model...
wandb: Currently logged in as: arun1 (use "wandb login --relogin" to force relogin)
wandb: wandb version 0.10.8 is available! To upgrade, please run:
wandb: $ pip install wandb --upgrade
wandb: Tracking run with wandb version 0.10.7
wandb: Syncing run ethereal-disco-3
wandb: View project at https://wandb.ai/arun1/face-mask-detection
wandb: View run at https://wandb.ai/arun1/face-mask-detection/runs/2suxrs4w
wandb: Run data is saved locally in wandb\run-20201024_163504_2suxrs4w
wandb: Run "wandb off" to turn off syncing.

Traceback (most recent call last):
  File "train_mask_detector.py", line 142, in <module>
    train_model()
NameError: name 'train_model' is not defined

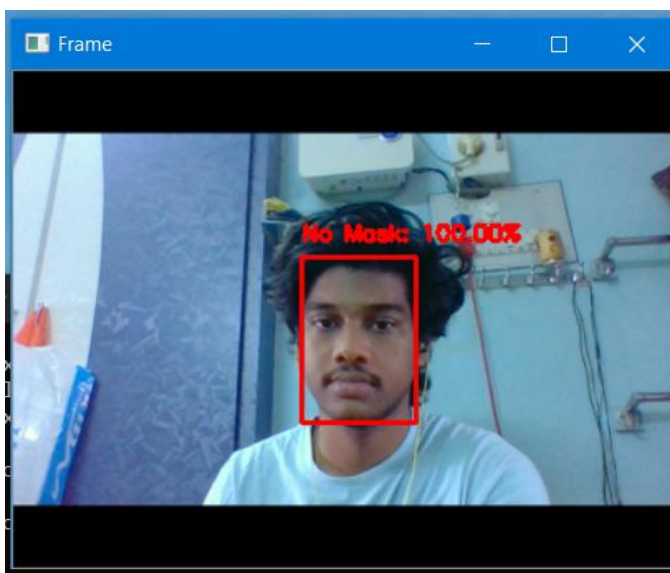
wandb: Waiting for W&B process to finish, PID 6584
wandb: Program failed with code 1. Press ctrl-c to abort syncing.
wandb:
wandb: Find user logs for this run at: wandb\run-20201024_163504_2suxrs4w\logs\debug.log
wandb: Find internal logs for this run at: wandb\run-20201024_163504_2suxrs4w\logs\debug-internal.log
wandb: Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
wandb:
wandb: Synced ethereal-disco-3: https://wandb.ai/arun1/face-mask-detection/runs/2suxrs4w
(base) D:\Face-Mask-Detection>
```

D. Live Streaming Outputs

With Mask:



Without Mask:



E. Applications of usage of Face Mask Detection

Places where we can use this application:

1. Traffic Signals
2. Research centres
3. Shopping malls
4. Hospitals
5. Industrial areas
6. Waste Management

VII. CONCLUSION

In this paper on Face Mask Detection, the project has been implemented in two parts. In the first part, there is a python script using Tensorflow and Keras to train face mask detector model. In the second part, testing the results in a real-time webcam using OpenCV is done. In the training part, two models have been compared one is CNN and the other model is MobileNetV2. Comparing both the models MobileNetV2 has more accuracy i.e., 96% and CNN model has 91% accuracy. By initializing the learning rate in the MobileNetV2 architecture we will get the best accuracy score. Testing the model includes the model embedded with the Webcam. In the training part a dataset containing the set of images with and without masks is used. For the testing part live video streaming using Webcam is used. It has been implemented using OpenCV in python. In addition to this an alert system has been incorporated into the model. Also, the metrics tools like Neptune and WandB are used to monitor the system usability like usage of CPU, RAM, GPU etc., Now, I hereby conclude that our ultimate aim is to check how our model is working for the live streaming video. As per the current pandemic issues, Face Mask is the mandatory thing that everyone should follow. To observe that everywhere security people are being assigned. In order to reduce the man force, we came up with the idea of developing Face Mask Detection Model. Most of the crowded areas will be having CCTVs to see the mischievous activities of the people around. If we embed the Face Mask Detection Model to the nearby CCTVs without more man power, the model can detect whether the people around there are with or without mask. The Face Mask Detection application is going to be used everywhere in this pandemic situation, this application also has an alert system which warns as people to put on the mask even in huge populated areas. This application is not only for this present pandemic situation, also it can be used in wide areas like in chemistry labs and many more; this system allows the person only with the mask and warns the person without mask.

REFERENCES

- [1] <https://medium.com/face-mask-detector-using-deep-learning-yolov3/face-mask-detector-using-deep-learning-yolov3-209b57f77e92>
- [2] <https://scialert.net/fulltext/?doi=itj.2013.1406.1411>
- [3] <https://www.leewayhertz.com/face-mask-detection-system/>
- [4] <https://medium.com/analytics-vidhya/image-classification-using-mobilenet-in-the-browser-b69f2f57abf>

- [5] <https://www.google.com/amp/s/scialert.net/fulltext/amp.php%3fdoi=itj.2013.1406.1411>
- [6] <https://www.securitysystemsnews.com/article/iss-releases-advanced-face-mask-detection-analytics>
- [7] https://link.springer.com/chapter/10.1007/978-3-540-73007-1_85
- [8] https://www.researchgate.net/publication/341095258_Face_Detection_and_Segmentation_Based_on_Improved_Mask_R-CNN
- [9] <https://www.electronicwings.com/users/ptksuraj99/projects/437/face-mask-detection-using-raspberry-pi>
- [10] <https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/>