Acknowledgments

The successful completion of this project report is due to the culmination of efforts and cooperation of a large number of some incredible people. I would like to express my sincere gratitude to my computer teacher, Mr. Dinesh Kumar, who guided me through each and every step, from research to writing the report, correcting and advising me through the entire process. I would also like to thank the Principal, Mr. Mathew K G, for providing me with this wonderful opportunity of writing a research paper. Lastly, I would like to thank my family, who have always been a source of constant support and encouragement to me.

Abstract

The COVID-19 pandemic is causing a global health crisis. the World Health Organization (WHO) advises people to wear masks in public places. The COVID-19 pandemic has forced governments across the world to impose lockdowns to prevent the spread of the virus. After the breakout of the worldwide pandemic COVID-19, a severe need for protection mechanisms has raised, face masks being the primary one. The basic aim of the project is to detect the presence of a face mask on human faces on live streaming video as well as on images. A hybrid model using deep and classical machine learning for face mask detection will be used for the same. The proposed method detects the face from the image correctly and then identifies if it has a mask on it or not. As a surveillance task performer, it can also detect a face along with a mask in motion. We explore optimized values of parameters using the Sequential Convolutional Neural Network model to detect the presence of masks correctly without causing over-fitting.

Table of Contents

Introduction

The year 2020 has shown mankind some mind-boggling series of events amongst which the COVID-19 pandemic is the most life-changing event which has startled the world since the year began. Affecting the health and lives of masses, COVID-19 has called for strict measures to be followed in order to prevent the spread of disease. From the very basic hygiene standards to the treatments in the hospitals, people are doing all they can for their own and the society's safety; face masks are one of the personal protective equipment. People wear face masks once they step out of their homes and authorities strictly ensure that people are wearing face masks while they are in groups and public places.

To monitor that people are following this basic safety principle, a strategy should be developed. A face mask detector system can be implemented to check this. Face mask detection means the identification of the presence of a mask or not. The first step to recognize the presence of a mask on the face is to detect the face, which makes the strategy divided into two parts: to detect faces and to detect masks on those faces. Face detection is one of the applications of object detection and can be used in many areas like security, biometrics, law enforcement and more. There are many detector systems developed around the world and being implemented. However, all this science needs optimization; a better, more precise detector, because the world cannot afford any more increase in corona cases.

In this project, we will be developing a face mask detector that is able to distinguish between faces with masks and faces with no masks. In this report, the proposed solution will have a detector which employs SSD for face detection and a neural network to detect presence of a face mask. The implementation of the algorithm is on images, videos and live video streams.

Requirements

Software

1. Anaconda
2. Microsoft Visual Studio (or any other Text editor)
3. Python Version 3.9

Libraries

1. Tensorflow version 1.15.2 or higher
2. Keras version 2.3.1
3. Imutils version 0.5.3
4. Numpy version 1.18.2
5. OpenCV version 4.2.0.
6. Matplotlib version 3.2.1
7. Scipy version 1.4.1

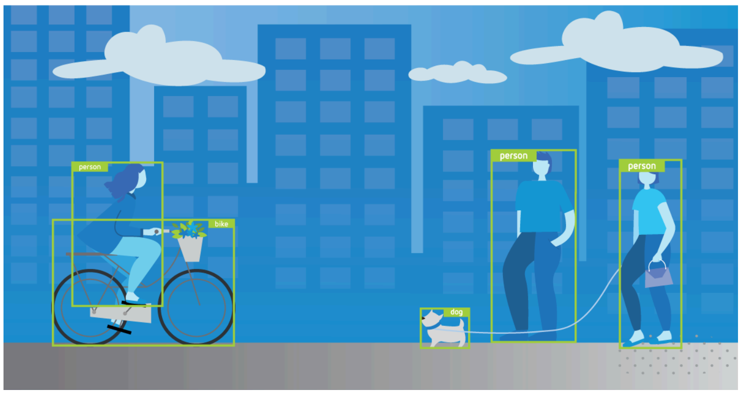
Hardware

1. A GPU (recommended for higher accuracy)
2. Webcam

Object Detection

Object detection is one of the most trending topics in the field of image processing and computer vision. Ranging from small scale personal applications to large scale industrial applications, object detection and recognition is employed in a wide range of industries. Some examples include image retrieval, security and intelligence, OCR, medical imaging and agricultural monitoring.

In object detection, an image is read and one or more objects in that image are categorized. The location of those objects is also specified by a boundary called the bounding box. Traditionally, researchers used pattern recognition to predict faces based on prior face models. A breakthrough face detection technology then was developed named as Viola Jones detector that was an optimized technique of using Haar, digital image features used in object recognition. However, it failed because it did not perform well on faces in dark areas and non-frontal faces. Since then, researchers are eager to develop new algorithms based on deep learning to improve the models. Deep learning allows us to learn features with end-to-end manner and removes the need to use prior knowledge for forming feature extractors. There are various methods of object detection based on deep learning which are divided into two categories: one stage and two stage object detectors



Two stage detectors use two neural networks to detect objects, for instance region-based convolutional neural networks (R-CNN) and faster R-CNN. The first neural network is used to generate region proposals and the second one refines these region proposals; performing a coarse-to-fine detection. This strategy results in high detection performance compromising on speed.

On the other hand, a one stage detector utilizes only a single neural network for region proposals and for detection; some primary ones being SSD (Single Shot Detection) and YOLO (You Only Look Once). To achieve this, the bounding boxes should be predefined. YOLO divides the image into several cells and then matches the bounding boxes to objects for each cell. This, however, is not good for small sized objects. Thus, multi scale detection is introduced in SSD which can detect objects of varying sizes in an image. One-stage detectors have higher speed but trades off the detection performance but then only are preferred over two-stage detectors.

TensorFlow

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind. However, it proved to be very useful for deep learning development as well, and therefore Google open-sourced it.

TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data.

TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.

OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

Keras

Keras is a high-level library that's built on top of TensorFlow. It provides a scikit-learn type API (written in Python) for building Neural Networks. Developers can use Keras to quickly build neural networks without worrying about the mathematical aspects of tensor algebra, numerical techniques, and optimization methods.

The key idea behind the development of Keras is to facilitate experimentations by fast prototyping. The ability to go from an idea to result with the least possible delay is key to good research.

This offers a huge advantage for scientists and beginner developers alike because they can dive right into Deep Learning without getting their hands dirty with low-level computations.

PyTorch

PyTorch is a Python-based scientific computing package that uses the power of graphics processing units. It is also one of the preferred deep learning research platforms built to provide maximum flexibility and speed. It is known for providing two of the most high-level features; namely, tensor computations with strong GPU acceleration support and building deep neural networks on a tape-based auto grad system.

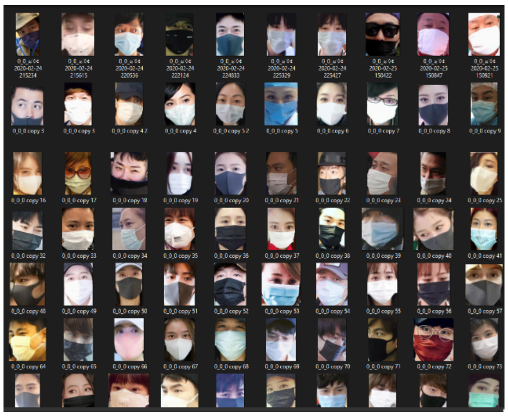
There are many existing Python libraries which have the potential to change how deep learning and artificial intelligence are performed, and this is one such library. One of the key reasons behind PyTorch's success is it is completely Pythonic and one can build neural network models effortlessly. It is still a young player when compared to its other competitors, however, it is gaining momentum fast.

Dataset

The dataset which we have used consists of 3835 total images out of which 1916 are of masked faces and 1919 are of unmasked faces. All the images are actual images extracted from Bing Search API, Kaggle datasets and RMFD dataset. From all the three sources, the proportion of the images is equal. The images cover diverse races i.e., Asian, Caucasian etc. The proportion of masked to unmasked faces determine that the dataset is balanced.

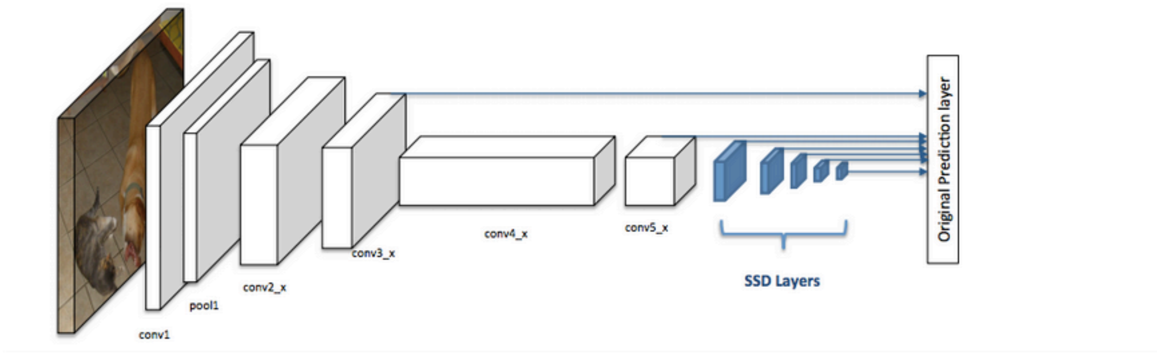
We need to split our dataset into three parts: training dataset, test dataset and validation dataset. The purpose of splitting data is to avoid overfitting which is paying attention to minor details/noise which is not necessary and only optimizes the training dataset accuracy. We need a model that performs well on a dataset that it has never seen (test data), which is called generalization. The training set is the actual subset of the dataset that we use to train the model. The model observes and learns from this data and then optimizes its parameters. The validation dataset is used to select hyperparameters (learning rate, regularization parameters). When the model is performing well enough on our validation dataset, we can stop learning using a training dataset. The test set is the remaining subset of data used to provide an unbiased evaluation of a final model fitted on the training dataset. Data is split as per a split ratio which is highly dependent on the type of model we are building and the dataset itself. If our dataset and model are such that a lot of training is required, then we use a larger chunk of the data just for training which is our case. If the model has a lot of hyperparameters that can be tuned, then we need to take a higher amount of validation dataset. Models with a smaller number of hyperparameters are easy to tune and update, and so we can take a smaller validation dataset.

In our approach, we have dedicated 80% of the dataset as the training data and the remaining 20% as the testing data, which makes the split ratio as 0.8:0.2 of train to test set. Out of the training data, we have used 20% as a validation data set. Overall, 64% of the dataset is used for training, 16% for validation and 20% for testing



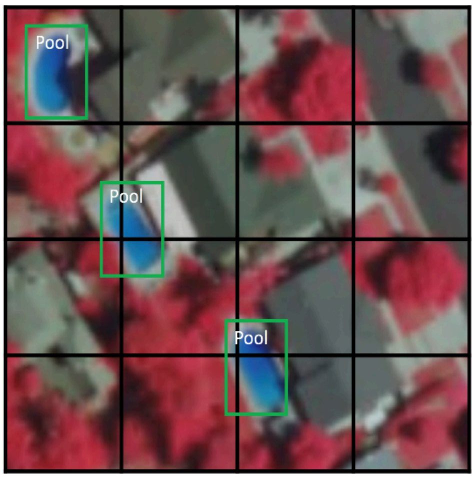
Architecture

Single Shot detector takes only one shot to detect multiple objects present in an image using a multibox**.** SSD has two components: a backbone model and SSD head. Backbone model usually is a pre-trained image classification network as a feature extractor. This is typically a network like ResNet trained on ImageNet from which the final fully connected classification layer has been removed. We are thus left with a deep neural network that is able to extract semantic meaning from the input image while preserving the spatial structure of the image albeit at a lower resolution. The SSD head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the final layer's activations.



Grid cell

SSD divides the image using a grid and has each grid cell be responsible for detecting objects in that region of the image. Detection objects simply means predicting the class and location of an object within that region.



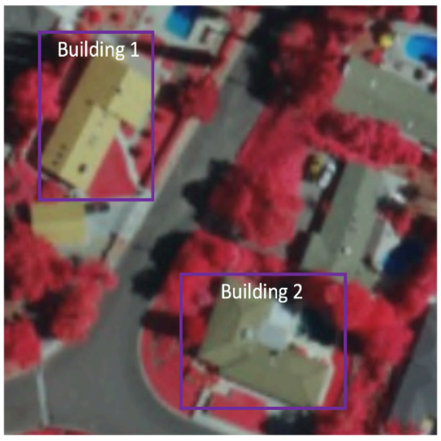
Anchor box

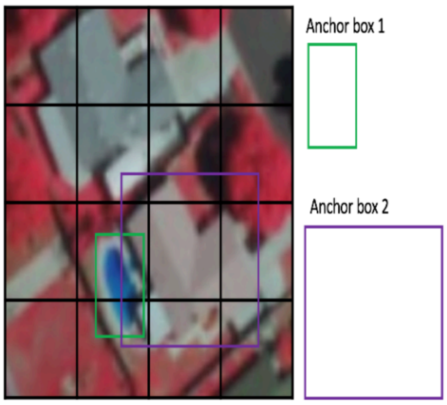
Each grid cell in SSD can be assigned with multiple anchor/prior boxes. These anchor boxes are pre-defined and each one is responsible for a size and shape within a grid cell

SSD uses a matching phase while training, to match the appropriate anchor box with the bounding boxes of each ground truth object within an image. Essentially, the anchor box with the highest degree of overlap with an object is responsible for predicting that object's class and its location.

Aspect ratio

Not all objects are square in shape. Some are longer and some are wider, by varying degrees. The SSD architecture allows pre-defined aspect ratios of the anchor boxes to account for this. The ratios parameter can be used to specify the different aspect ratios of the anchor boxes associates with each grid cell at each zoom/scale level.

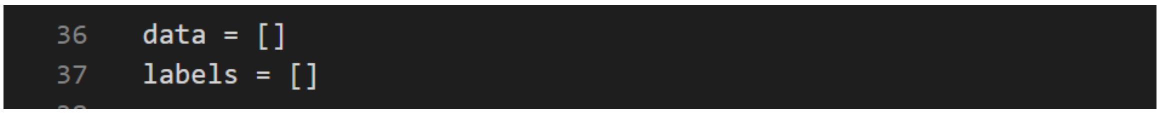




Data Pre-processing

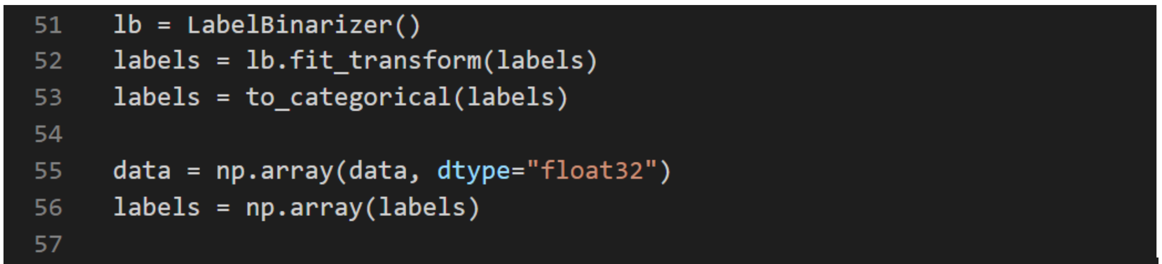
Data Pre-processing

1) Creation of empty list.



```
data = []
labels = []
```

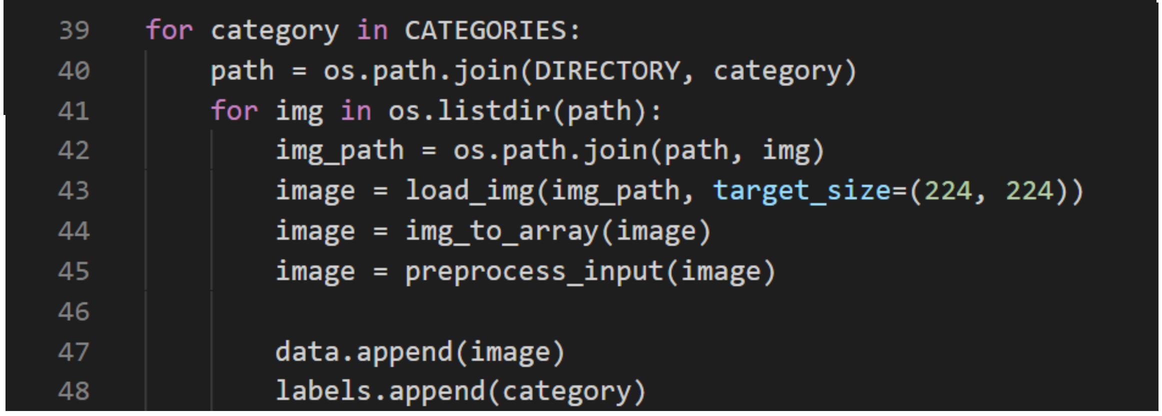
2) Looping through Categories.



```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

```

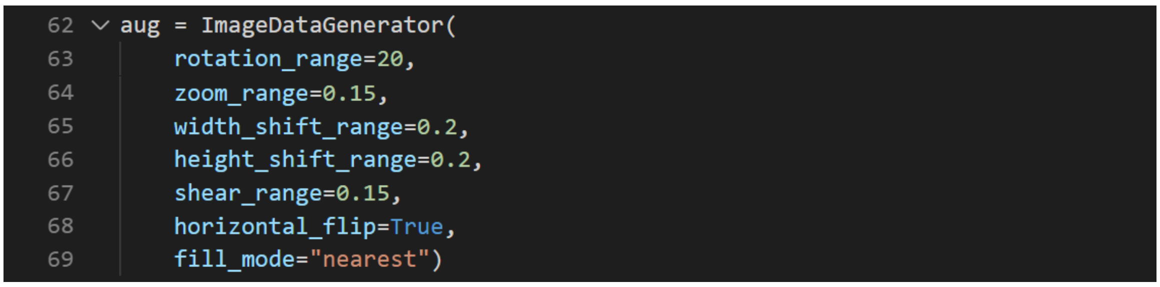


```
for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)
```

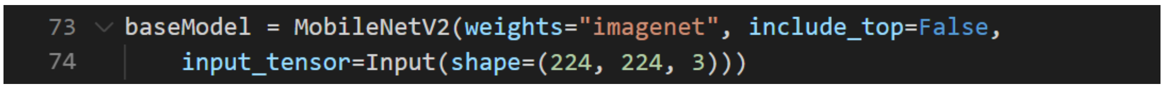
3) Converting to Numerical Value.

4) Training Image Generator for Data Augmentation.



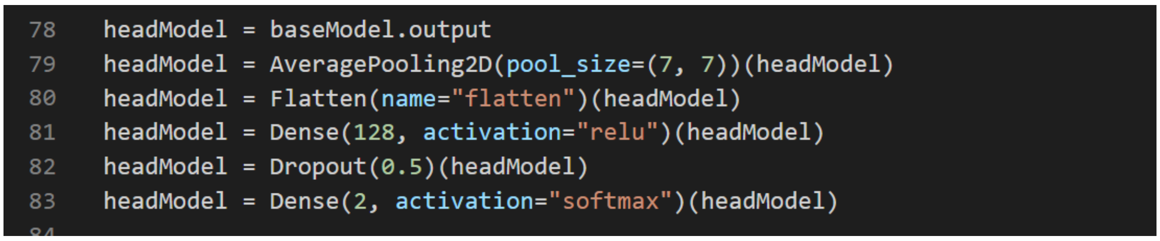
```
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

5) Using MobileNetV2 to create a base model.



```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

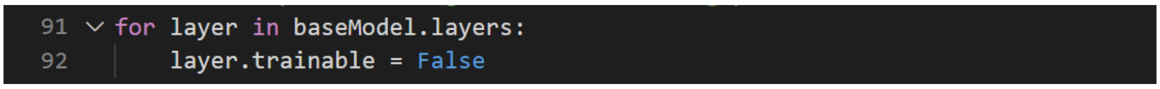
6) Head of the model which will be placed on top.



```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

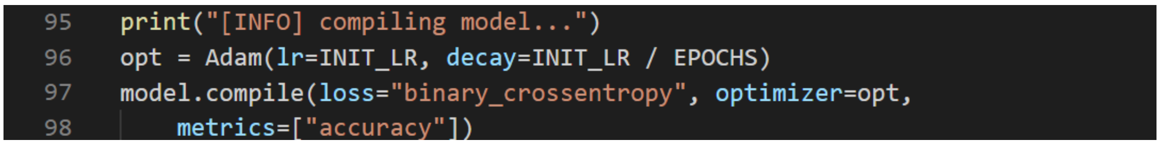
7) Placing of Head of the model on top of the base model. This will be the model trained

8) Looping over all the layers in the base model and freezing them so that they will not be updated during the first training process.



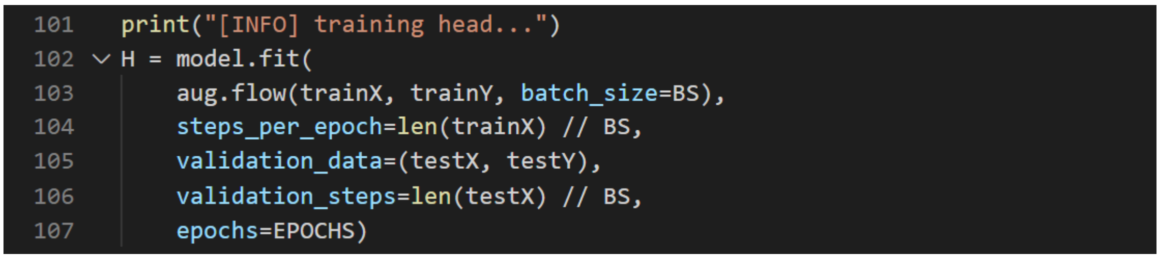
```
for layer in baseModel.layers:
    layer.trainable = False
```

9) Compiling the Model.



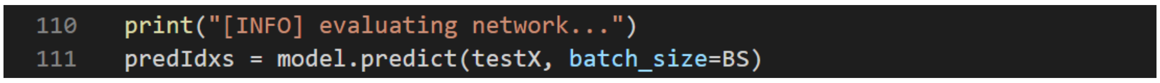
```
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

10) Training the head of the network.



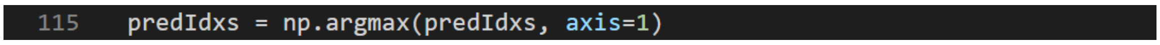
```
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

11) Making predictions on the testing set.



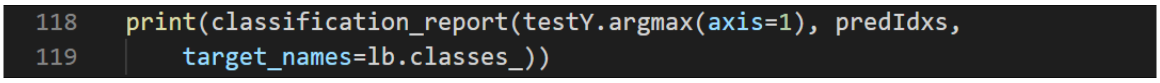
```
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
```

12) Labelling with corresponding largest predicted probability.



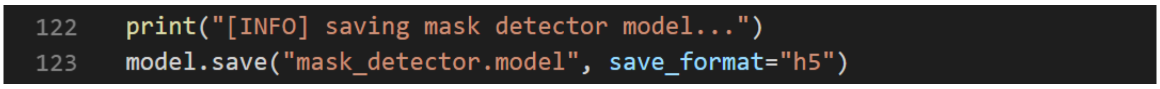
```python
predIdxs = np.argmax(predIdxs, axis=1)
```

13) Presentation of a classification report.



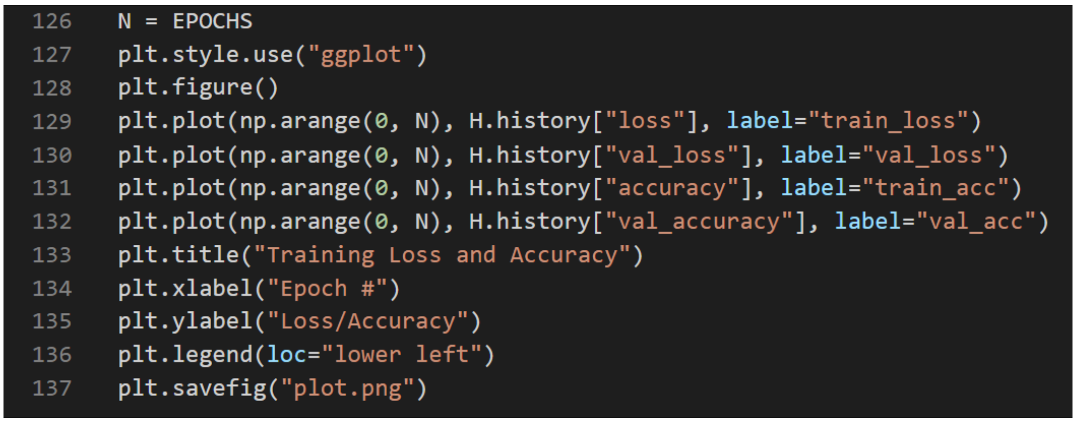
```python
print(classification_report(testY.argmax(axis=1), predIdxs,
	target_names=lb.classes_))
```

14) Serialising the model to disk.



```python
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")
```

15) Plotting the training loss and accuracy.

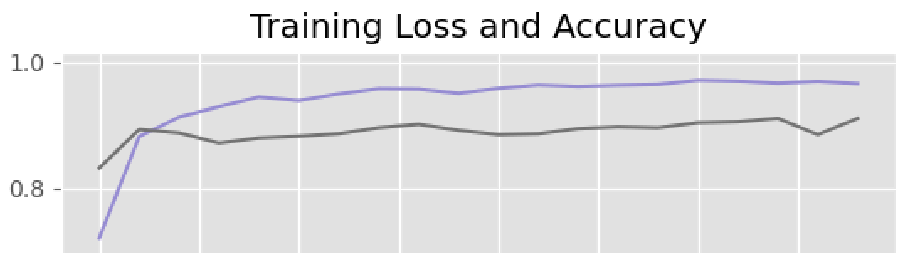


```python
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

Result and Analysis

<u>Use of MobileNetV2</u>

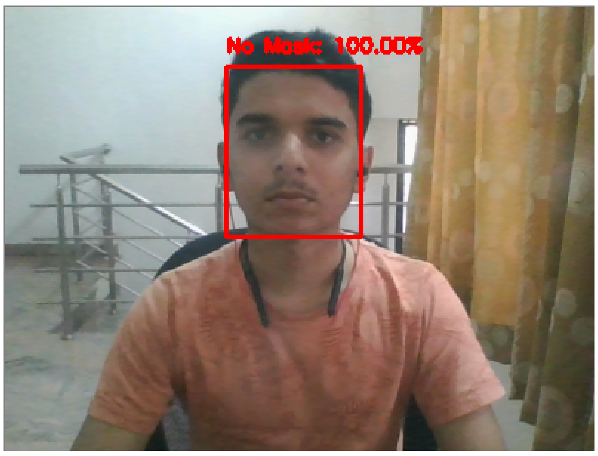
MobileNetV2 gave a very high percentage of accuracy when it came to the detection of unmasked faces. Since the detector can take no risk by falsely concluding that an unmasked face is a masked face, precision is the key metric to decide between these models. The system can efficiently detect partially occluded faces either with a mask or hair or hand. It considers the occlusion degree of four regions – nose, mouth, chin and eye to differentiate between annotated mask or face covered by hand. Therefore, a mask covering the face fully including nose and chin will only be treated as “with mask” by the model.



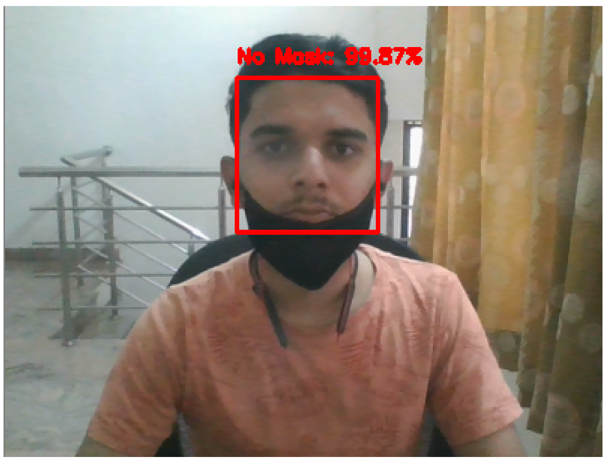
Inference

The model was implemented on images containing one and more faces. It was implemented on videos and live video streams by removing and wearing masks one by one. Some screenshots of the results are shown below

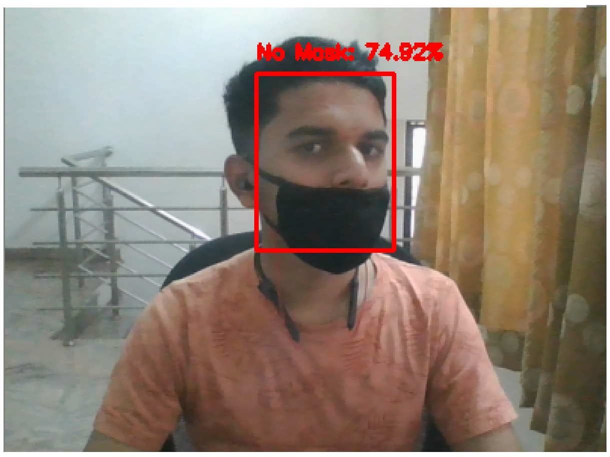
Case 1- Completely without a Face Mask



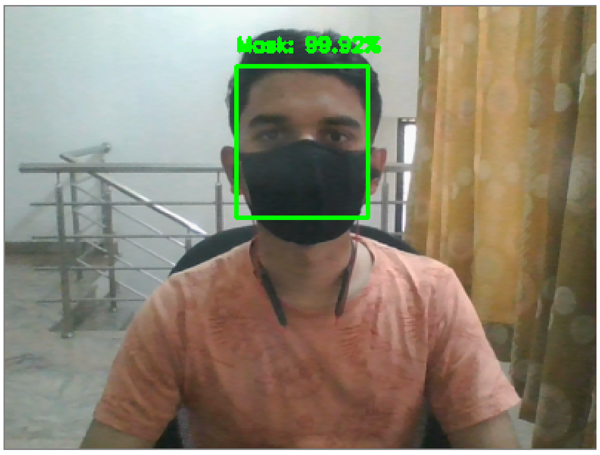
Case 2- Mask not covering mouth and nose



Case 3- Mask not covering nose



Case 4- With Mask

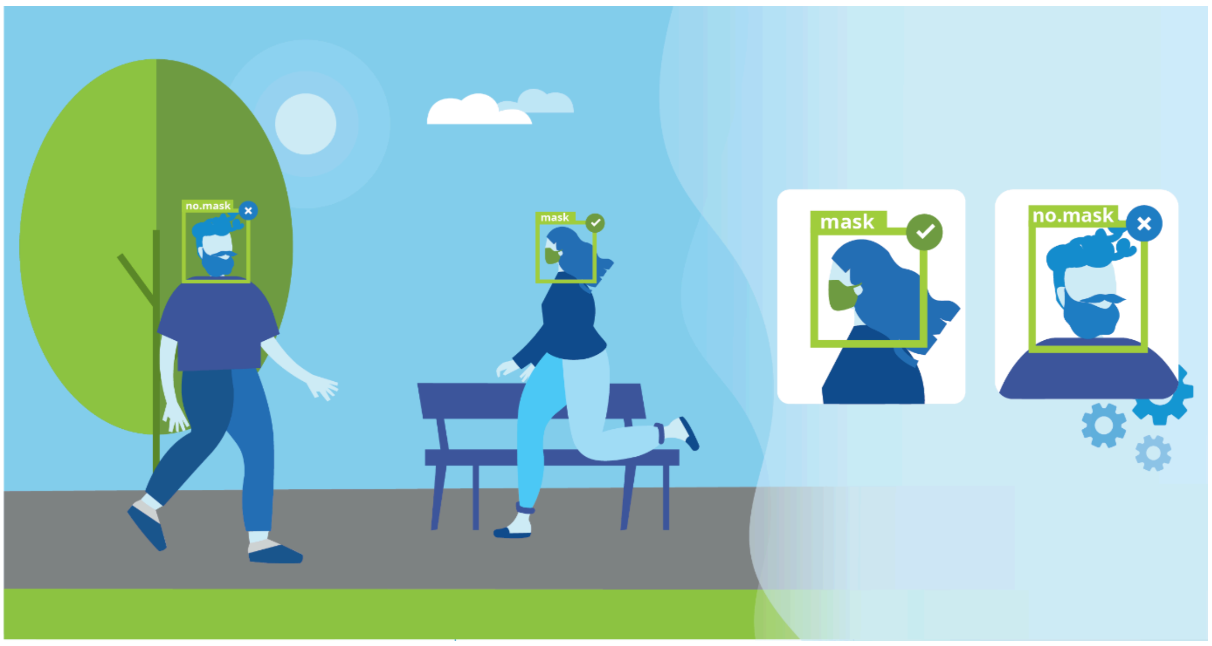


Limitations

The main challenges faced by the method mainly consist of varying angles and lack of clarity due to lack of proper lighting. However, following the trajectories of several frames of the video helps to create a better decision – “with mask” or “without mask.” This can be fixed by using a system with high graphic processing power as it can process data faster and give more accurate outputs.

Future Possibilities

More than fifty countries around the world have recently initiated wearing face masks compulsory. People have to cover their faces in public, supermarkets, public transports, offices, and stores. Retail companies often use software to count the number of people entering their stores. They may also like to measure impressions on digital displays and promotional screens. We are planning to improve our Face Mask Detection tool and release it as an open-source project. This solution can be equated to any existing USB, IP cameras, and CCTV cameras to detect people without a mask. This detection live video feed can be implemented in web and desktop applications so that the operator can see notice messages. Software operators can also get an image in case someone is not wearing a mask. Furthermore, an alarm system can also be implemented to sound a beep when someone without a mask enters the area. This software can also be connected to the entrance gates and only people wearing face masks can come in.



Conclusion

To mitigate the spread of COVID-19 pandemic, measures must be taken. We have modelled a face mask detector using SSD architecture and transfer learning methods in neural networks. To train, validate and test the model, we used the dataset that consisted of 1916 masked faces images and 1919 unmasked faces images. These images were taken from various resources like Kaggle and RMFD datasets. The model was inferred on images and live video streams. To select a base model, we evaluated the metrics like accuracy, precision and recall and selected MobileNetV2 architecture with the best performance having 100% precision and 99% recall. It is also computationally efficient using MobileNetV2 which makes it easier to install the model to embedded systems. This face mask detector can be deployed in many areas like shopping malls, airports and other heavy traffic places to monitor the public and to avoid the spread of the disease by checking who is following basic rules and who is not.

Bibliography

1) *P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, vol.1. IEEE, 2001, pp. I–I.*
2) *R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.*
3) *R. Girshick, "Fast r-cnn," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.*
4) *S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.*
5) *W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.*
6) *J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.*
7) *T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," 2017.*
8) *Haddad, J., 2020. How I Built A Face Mask Detector For COVID-19 Using Pytorch Lightning. Medium. Available at: https://towardsdatascience.com/how-i-built-a-face-mask-detector-for-covid-19-using-pytorch-lightning-67eb3752fd61.*
9) *Rosebrock, A., 2020. COVID-19: Face Mask Detector with Opencv, Keras/Tensorflow, And Deep Learning- Pyimagesearch. PyImageSearch. Available at: https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/ .*
10) *W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.*
11) *Keras code inspiration taken from https://github.com/chandrikadeb7/Face-Mask-Detection*
12) *https://www.upgrad.com/blog/the-whats-what-of-keras-and-tensorflow/*
13) *https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow*
14) *https://developers.arcgis.com/python/guide/how-ssd-works/*