Face Classification for different Occlusion

Detection

- Detect Faces, masked faces, faces with sunglasses and faces with both mask and sunglasses in Images and Videos.
- Crop them and segregate the images into respective folders (manual process...using python code to do this)

Classification

- Design a four-class classifier (SVM, ReseNet or any) using deep learning models, which classifies the face images as
  - Faces
  - Masked faces
  - Faces with sunglasses
  - Masked faces with sunglasses

Solution and Approach : Dataset

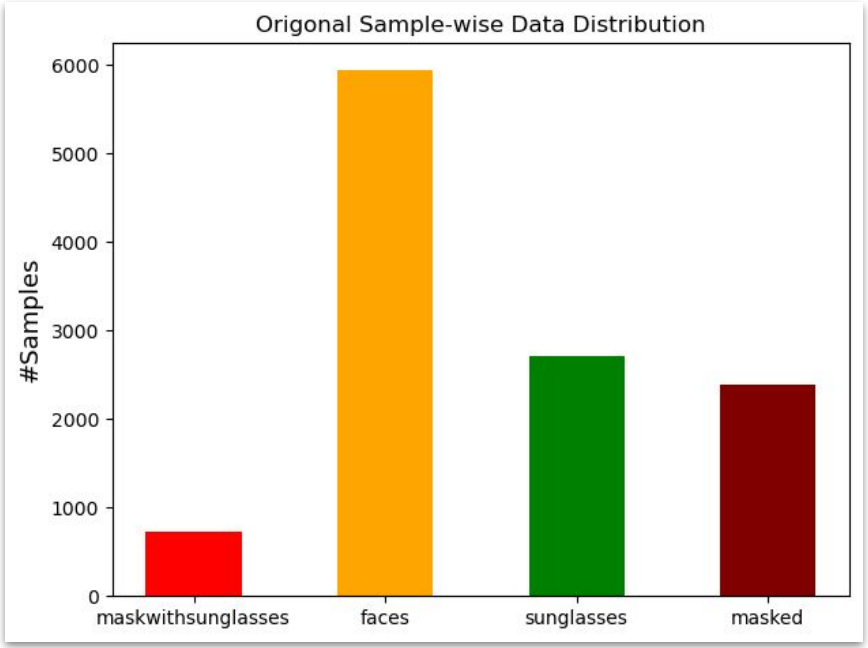
Dataset: ROF data (https://github.com/ekremerakin/RealWorldOccludedFaces) + Web Scraping

Challenges: Uneven number of samples

- Some classes have few samples only
- Web scraping is time taking process
- Images are focused on face

For mask with sunglasses class, we have very few samples and of uneven resolution.

Approach: Added some samples from other dataset into masked class but no data available for mask with sunglasses class.



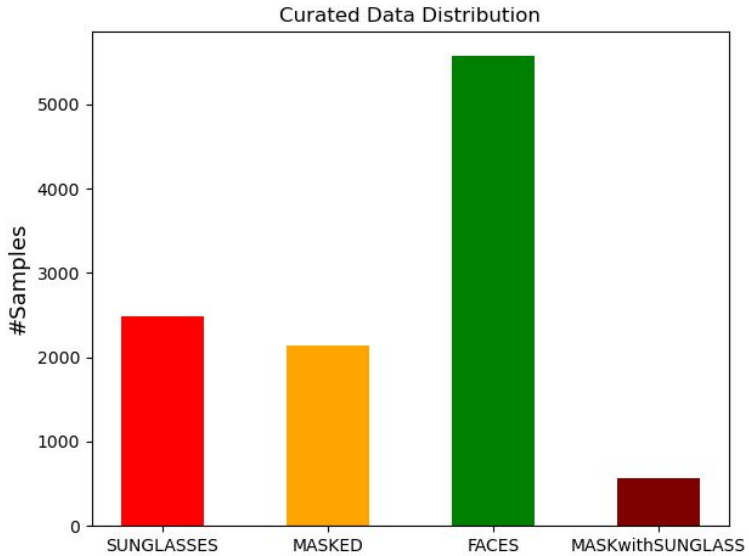
Solution and Approach : Detection Model and Data Curation

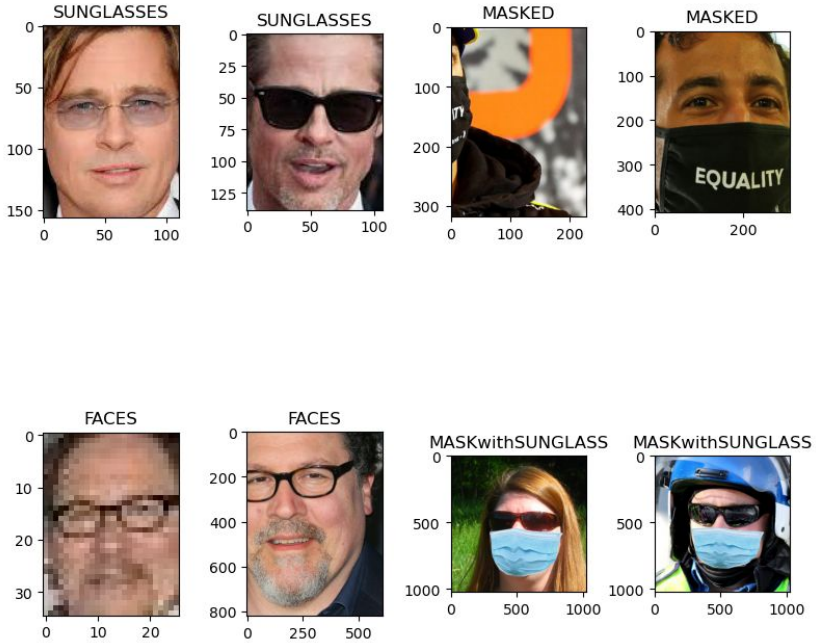
Model Selection: Based on performance over mask with sunglasses class.

Model	**HOG**	**MMOD-CNN**	**MTCNN**	**RetinaFace**	**Harcascade**
Accuracy	104/729	200/729	540/729	100/729	-

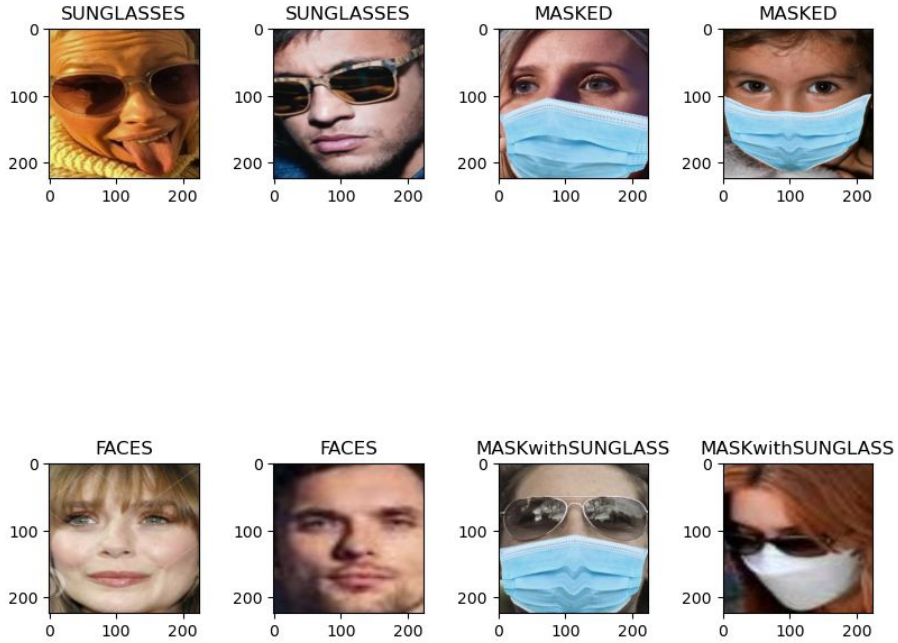
- MTCNN is selected to detect faces from the images and then crop and save the detected face in corresponding folders. Images are resized to same dimension before saving.

- Each folder is then manually curated to contain only one class sample.





Original Data



Curated Data

Solution and Approach : Data Balancing

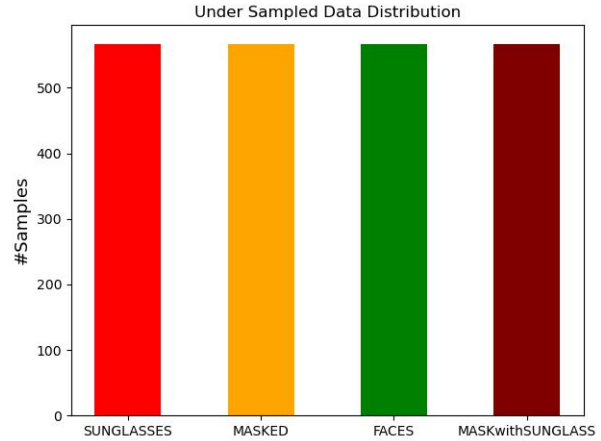
Original Sample size:
|| Sunglasses: 2483 || Masked: 2138 || Faces: 5578 || Mask with Sunglasses: 567 ||

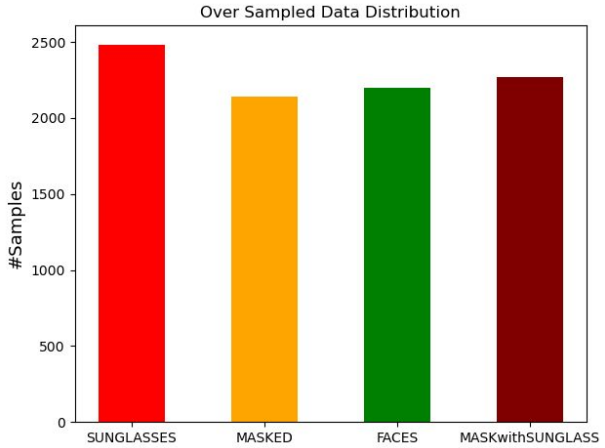
<u>Under-Sampling</u>

- Randomly selected samples equal to the samples present in minority class i.e., Mask with Sunglasses

<u>Over-Sampling</u>

- Kept Sunglasses and Masked samples to their original number of samples
- Randomly selected similar amount of samples from Faces class (e.g. 2200)
- Increased samples in minority class by using data augmentation (1:3)
  - Random flip
  - Random Rotate
  - Adjust Blur, Brightness, Contrast, Sharpness





Solution and Approach : Classification Model

Model: Tried 3 different available Pre-Trained model

- MobileNet (v2_050) : 692,804
- EfficientNet (b0) : 4,012,672
- ResNet18 : 11,178,564

Model Quantization: Applied Post Training Static Quantization using FXGraph mode

- Better than Eager Mode as it automatically place observers
- Implicitly apply module fusing
- Simple and minimal manual steps

Reduction in model size after Quantization:

Model	**MobileNet (v2_050)**	**EfficientNet (b0)**	**ResNet18**
Pre-Quantization Size	3.0 MB	16.4 MB	44.8 MB
Post-Quantization Size	1.3 MB	5.2 MB	11.3 MB

Classification Result (Accuracy/Inference Time)

Model	Original Data Samples		Under-Sampled Data		Over-Sampled Data	
	Pre-Quantization	**Post-Quantization**	**Pre-Quantization**	**Post-Quantization**	**Pre-Quantization**	**Post-Quantization**
MobileNet (v2_050)	0.9882/7.01ms	0.9672/19.56ms	0.9648/5.84ms	0.7683/17.23ms	0.9890/4.01ms	0.8812/17.73ms
EfficientNet (b0)	0.9802/12.78ms	0.5759/31.26ms	0.9648/8.36ms	0.4106/30.38ms	0.9934/10.29ms	0.2529/31.84ms
ResNet18	0.9944/6.09ms	0.9950/9.86ms	0.9677/3.66ms	0.9619/4.29ms	0.9934/3.67ms	0.9934/4.26ms

Observations:

- Overfitting is noticeable in models
- Undersampling reduces overfitting
- ResNet maintains its accuracy after quantization
- Large reduction in accuracy for EfficientNet (~>30%)
- For Mobilenet reduction is about ~20%

Classification Model (Resnet Predictions)

Labels = [0:SUNGLASSES, 1:MASKED, 2:FACES, 3:MASKwithSUNGLASSES]


MASKED


MASKED


MASKwithSUNGLASS


MASKwithSUNGLASS


MASKwithSUNGLASS


MASKED


MASKwithSUNGLASS


MASKwithSUNGLASS


SUNGLASSES


SUNGLASSES

Classification Model (MobileNet Predictions)

Labels = [0:SUNGLASSES, 1:MASKED, 2:FACES, 3:MASKwithSUNGLASSES]


FACES
MASKED
MASKED
MASKED
MASKED
MASKED
MASKwithSUNGLASS
MASKwithSUNGLASS
MASKwithSUNGLASS
MASKwithSUNGLASS
MASKwithSUNGLASS
SUNGLASSES
SUNGLASSES
SUNGLASSES

Classification Model (EfficientNet Predictions)

Labels = [0:SUNGLASSES, 1:MASKED, 2:FACES, 3:MASKwithSUNGLASSES]


FACES

FACES

FACES

MASKED

MASKED

FACES

MASKED

MASKED

MASKwithSUNGLASS

MASKwithSUNGLASS

SUNGLASSES

SUNGLASSES

SUNGLASSES

SUNGLASSES

Solution and Approach : Observation

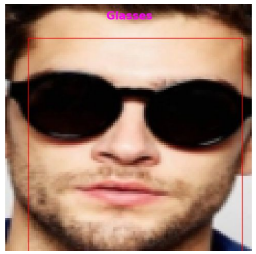
- The Dataset is still not effective. Contains duplicate images diversity is less
- MTCNN sometimes fails to capture face from image an leads to no input for classification.
- Classification models overfits because of less samples and less diversity in the data.
- Most of the time misclassifying faces as the dataset in that class not much accurate
- Performance of lighter model is smooth in the testing phase
- Able to store classified images into separate folders.

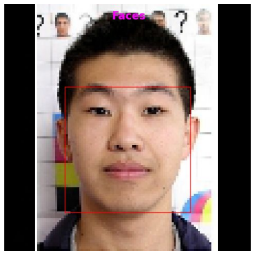
Major Challenges:

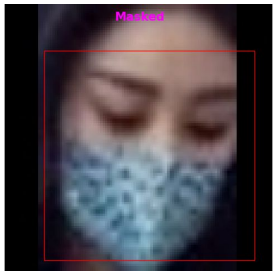
- Quality dataset for all the classes
- Optimization on face detection models

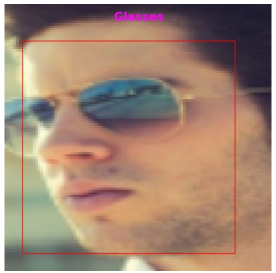
Solution and Approach : Object Detection

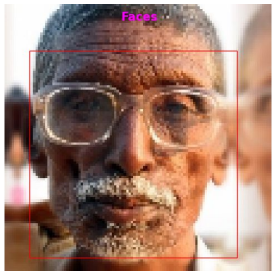
Models: Tried with Faster-RCNN and YOLOv5
Data Samples: Images of shape 112x112 and labels [class_label, center_x, center_y, width, height] of bounding boxes in corresponding .txt file.

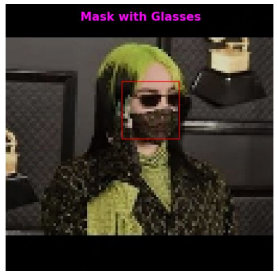










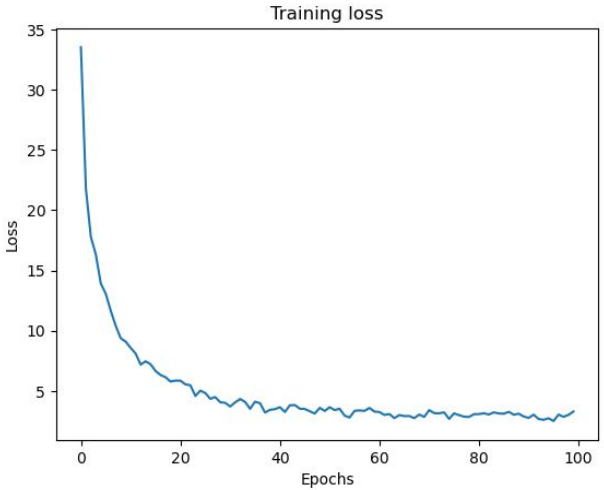


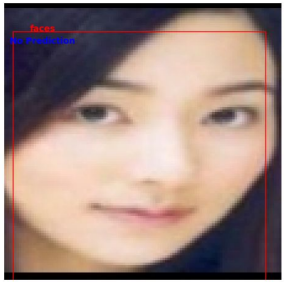
Faster-RCNN Results

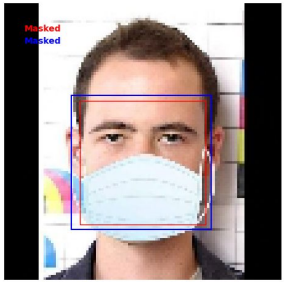
Backbone Architecture: Resnet50 with FPN

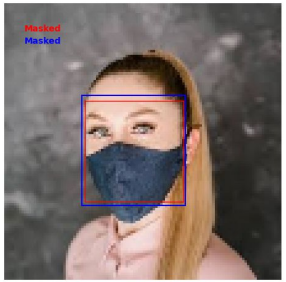
Required bounding box format: COCO format
Dict{“boxes”:[xmin, ymin, xmax, ymax],
“labels”: class_label,
“image_id”: image_id}

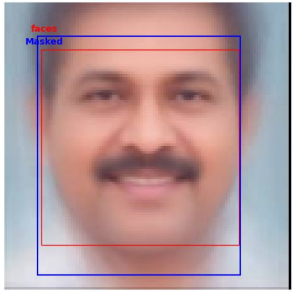
Accuracy over classification: 0.9
(6 no predictions)

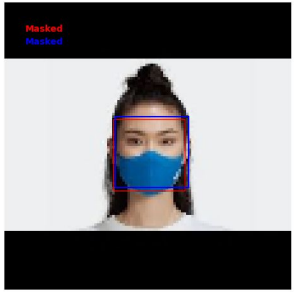


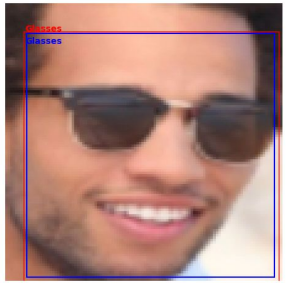


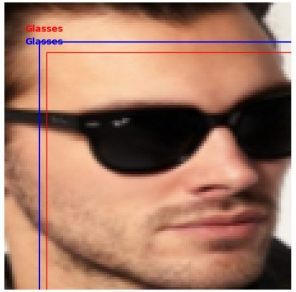












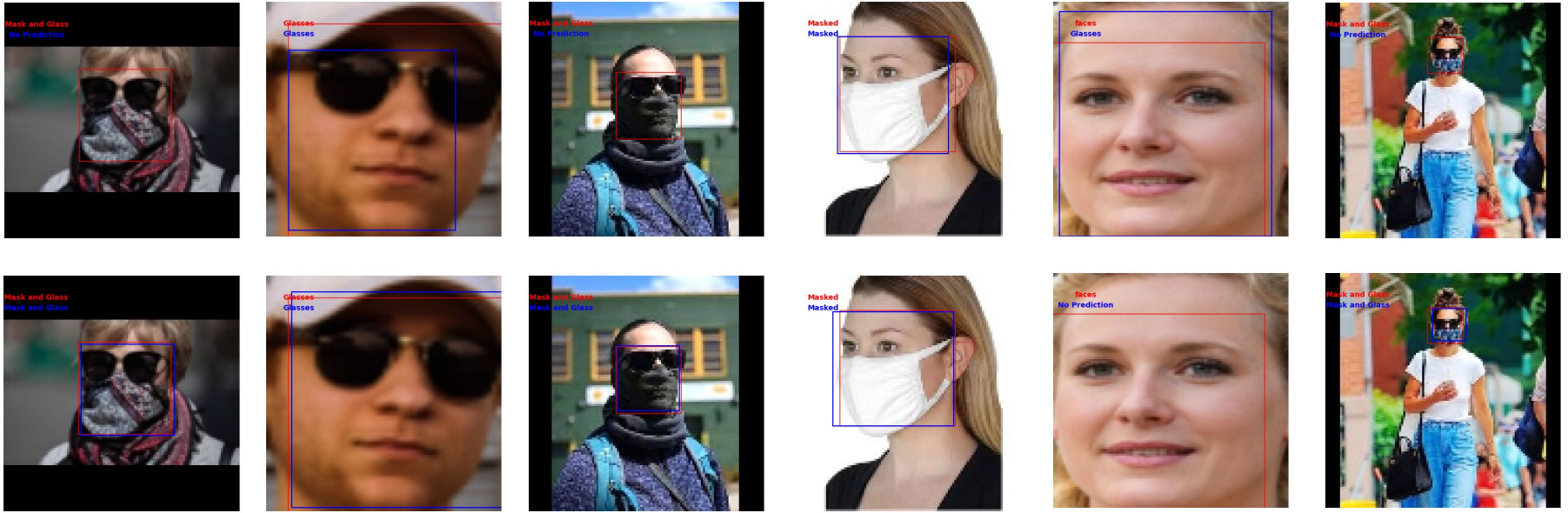
Faster-RCNN Results

Faster-RCNN Quantization

Quantization Method: Used FX-Graph post training Static Quantization (only backbone is quantized)

- Size reduction: for backbone:- 107.7MB to 27.8MB
  - For entire model:- 165.8MB to 86MB
- Inference Time reduction:- 633ms to 530ms
- Classification Accuracy:- 0.5 (no detection ar not included (6))

Some results (top row) and their prediction with non-quantized model (bottom row)

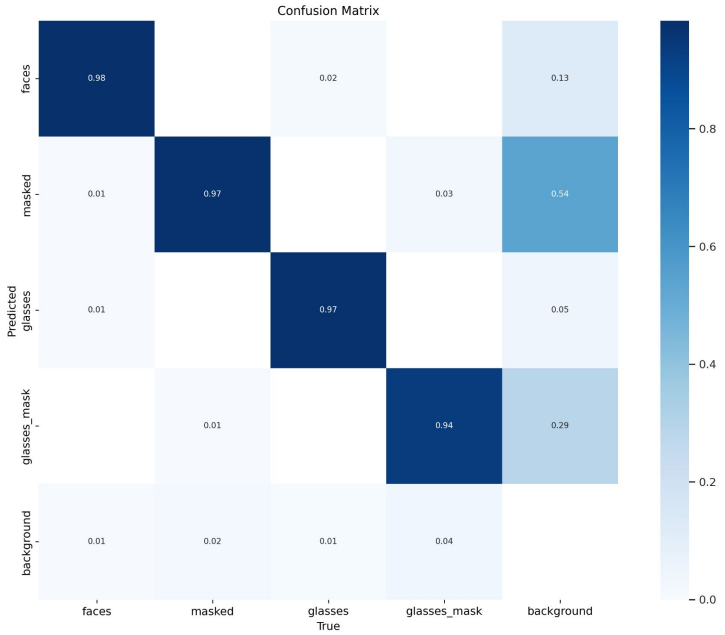


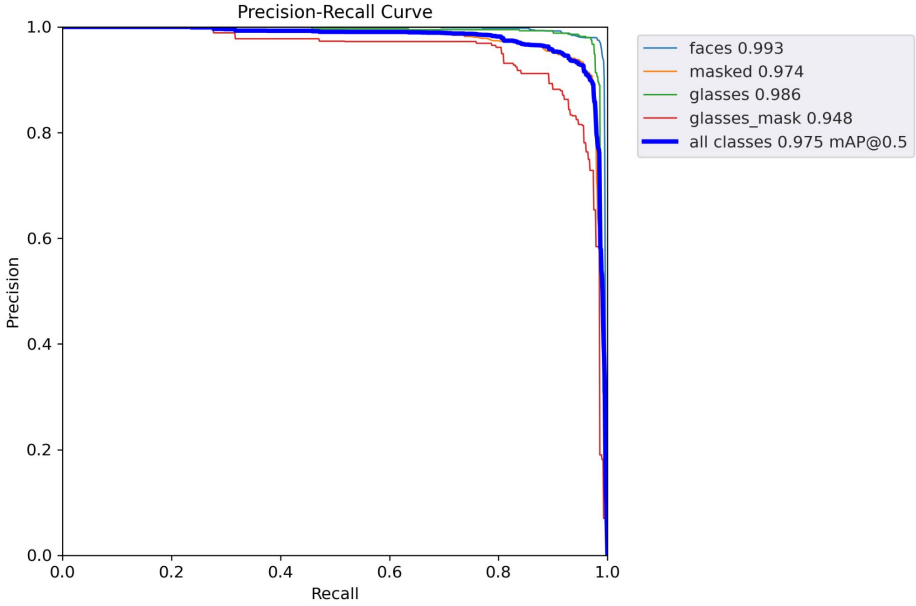
YOLOv5 Results

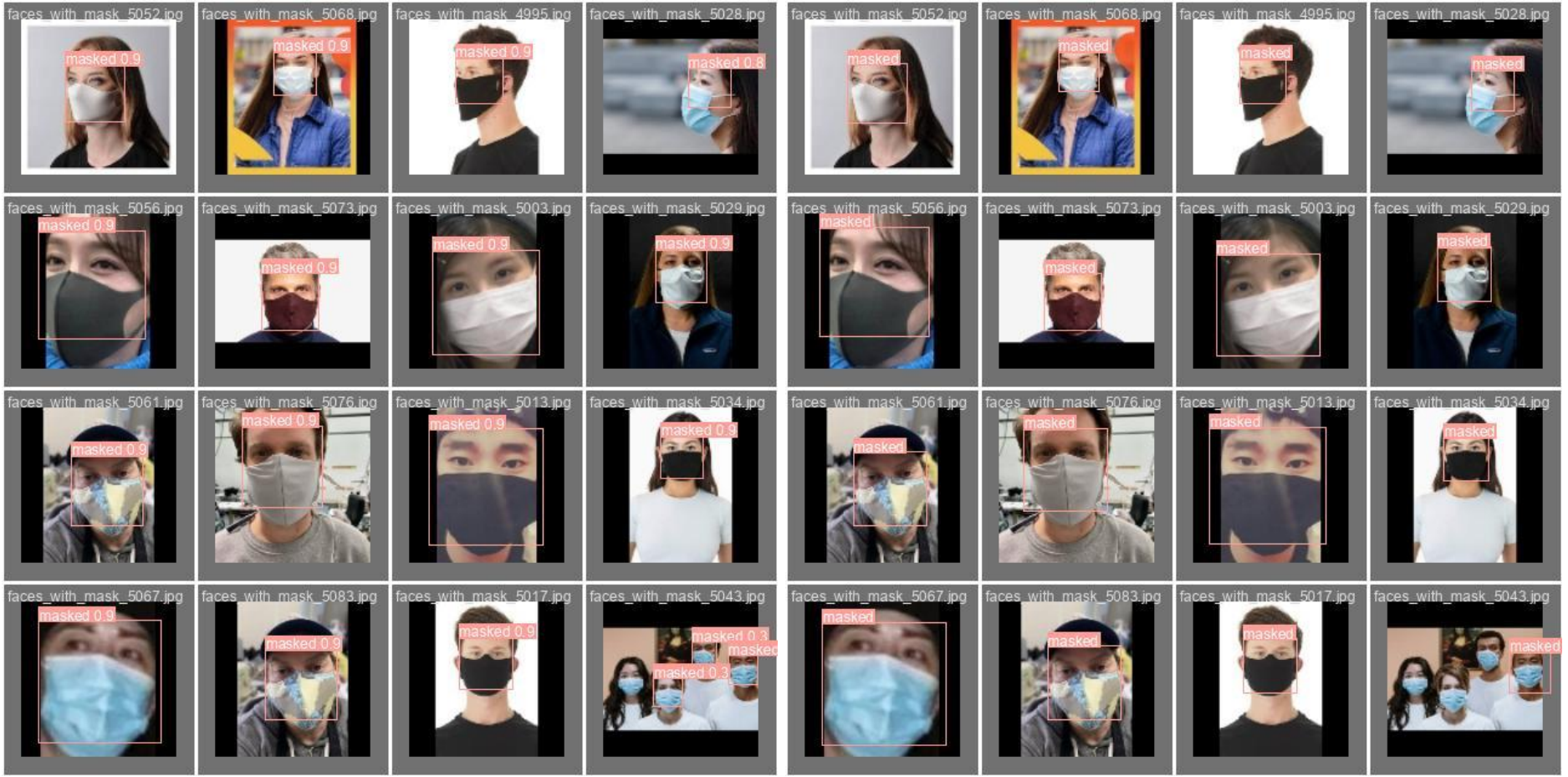
Model Version: v5 small (12MB)

Achieved performance: mAP@0.5 = 0.9791
mAP@0.5:0.95 = 0.7734

Class	Images	Instances	P	R	mAP50	mAP50-95
all	2643	2643	0.942	0.95	0.975	0.78
faces	2643	985	0.975	0.982	0.993	0.844
masked	2643	928	0.934	0.954	0.974	0.741
glasses	2643	452	0.969	0.971	0.986	0.812
glasses_mask	2643	278	0.892	0.893	0.948	0.724

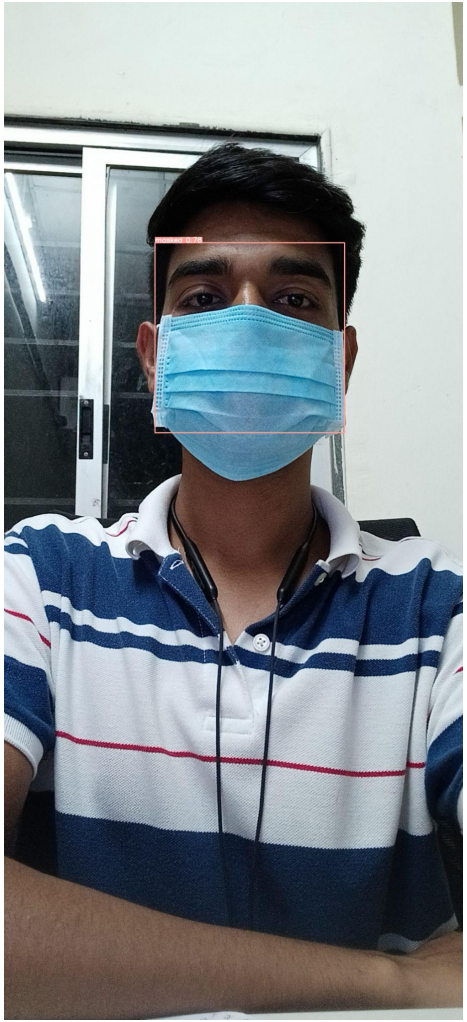
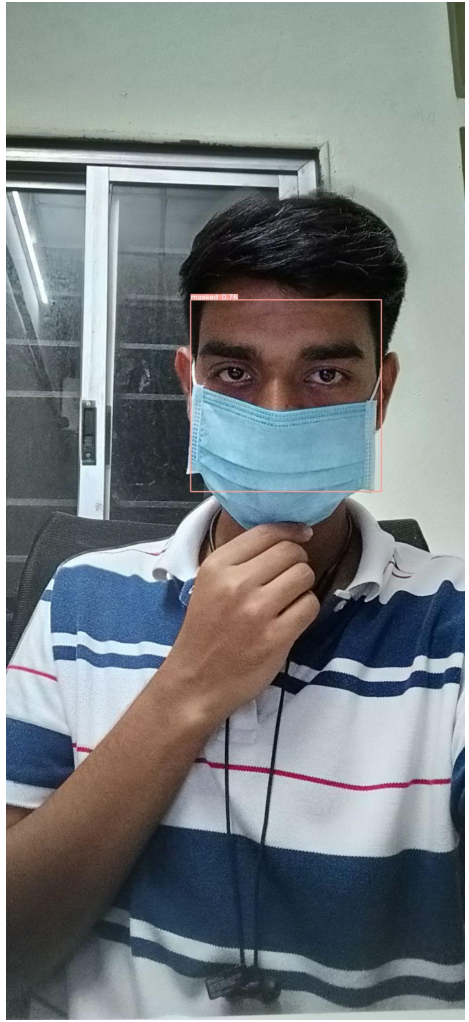
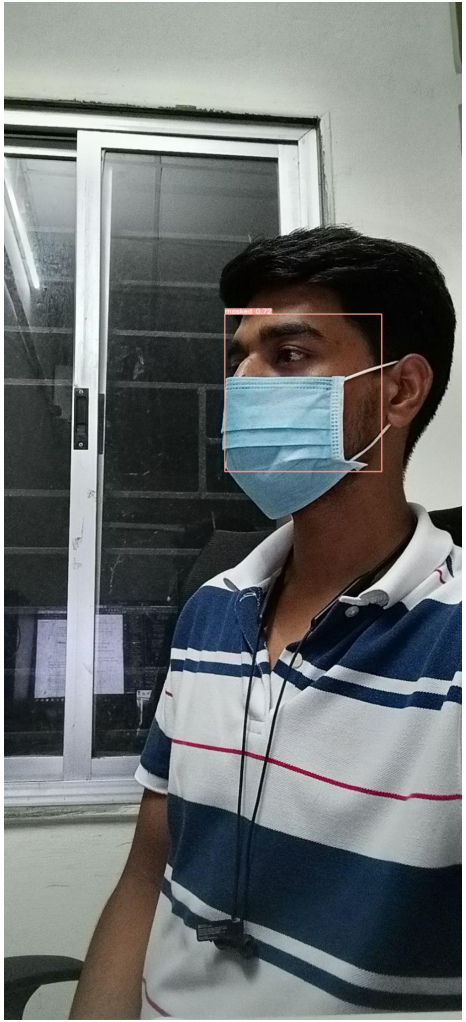
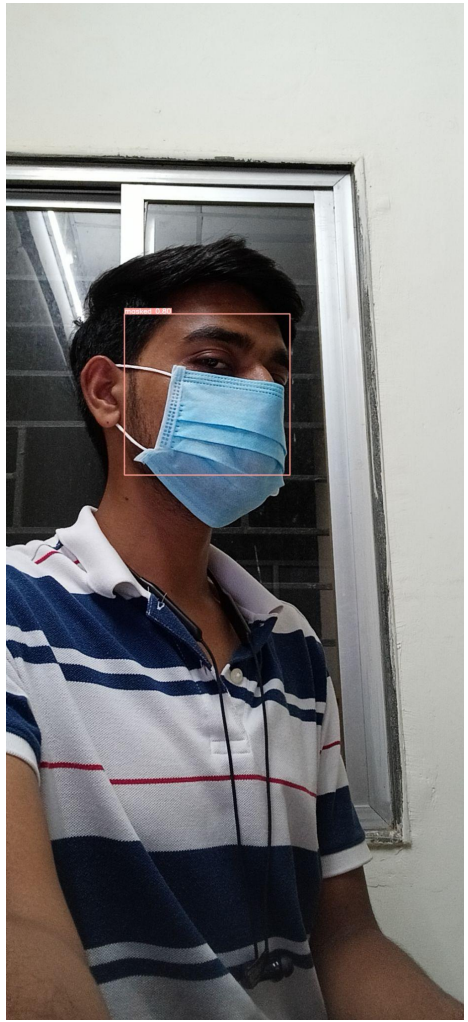


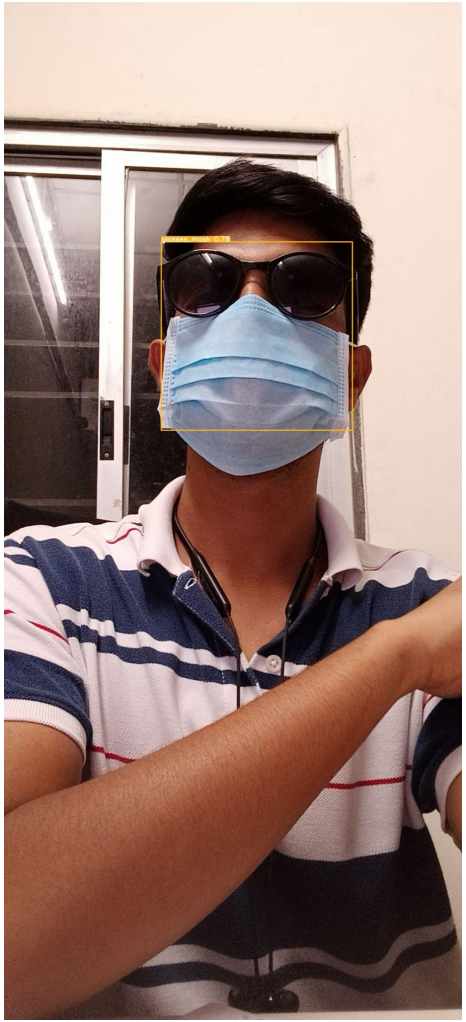
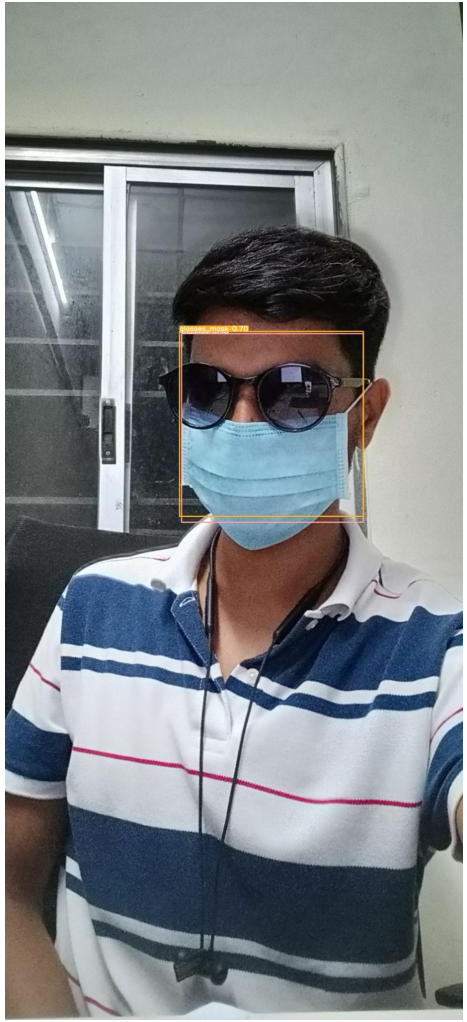
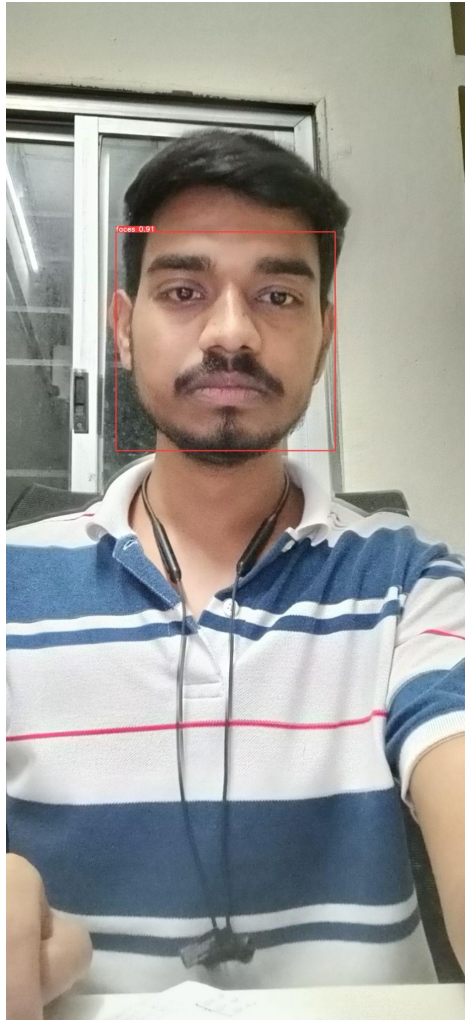
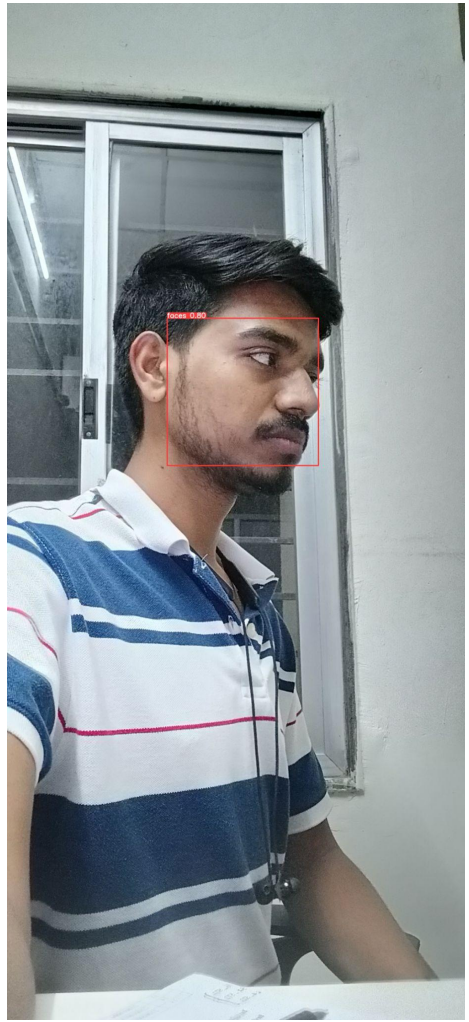


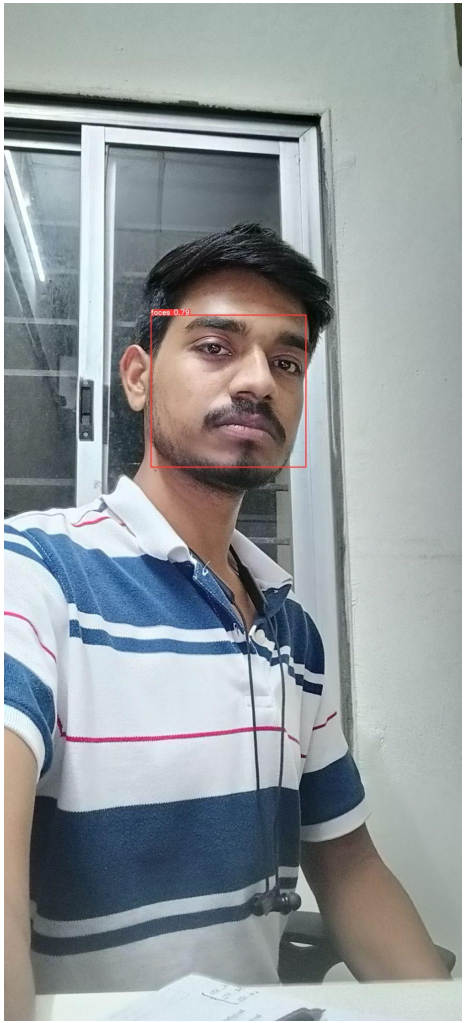
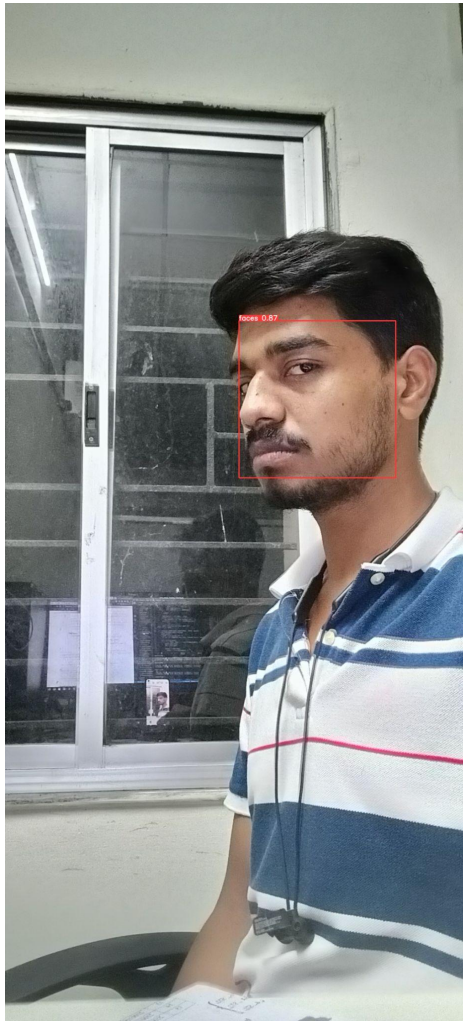
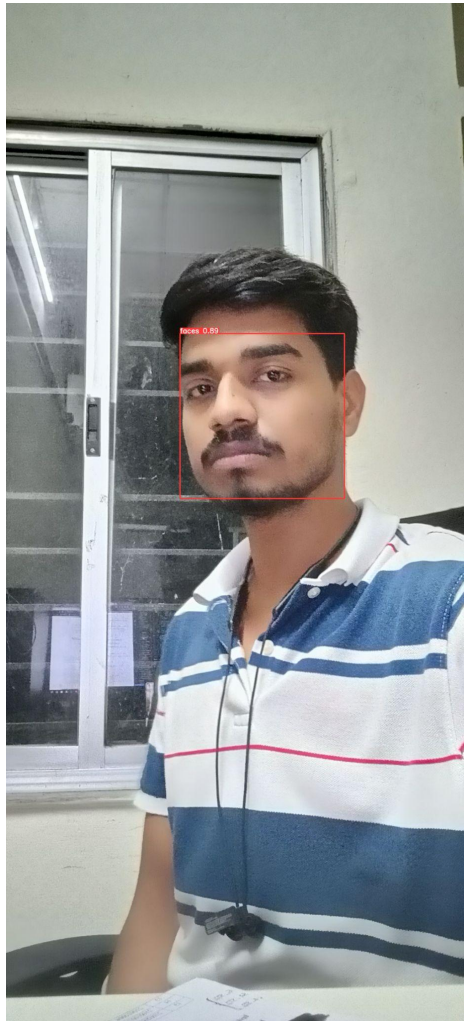
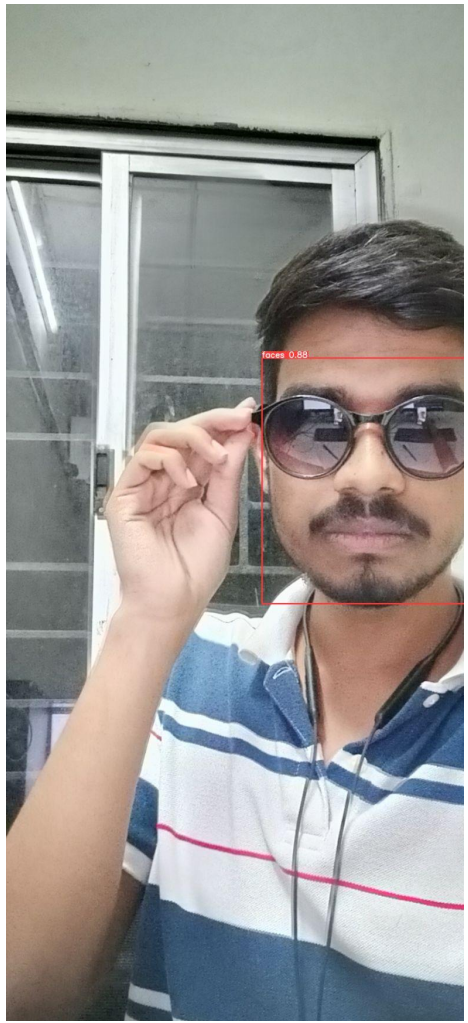


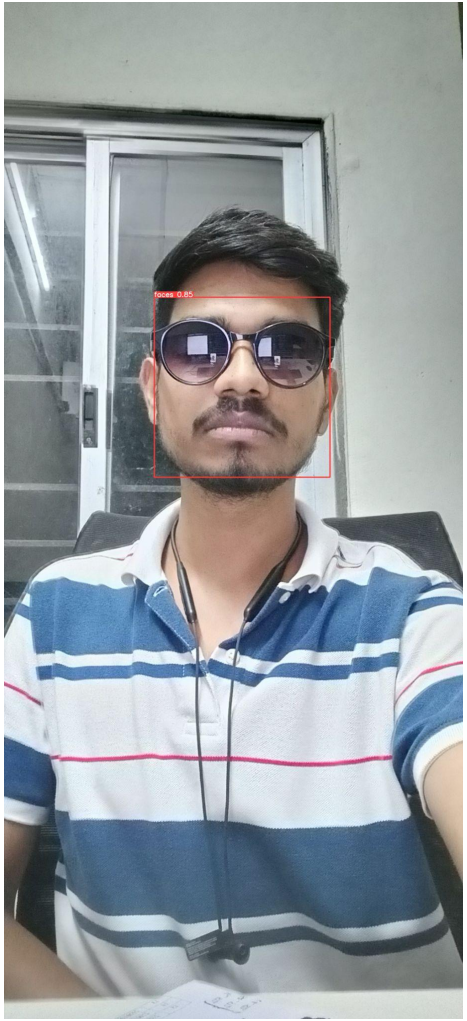
Predictions

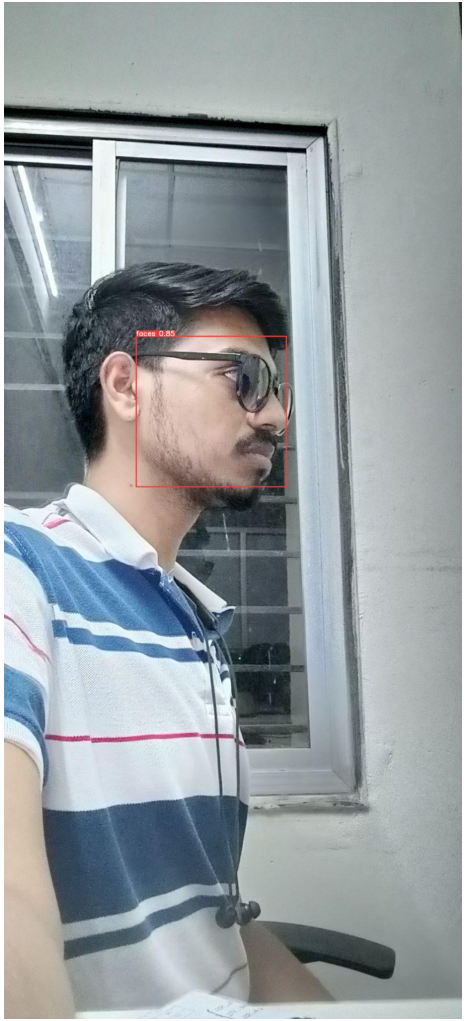
Ground Truth

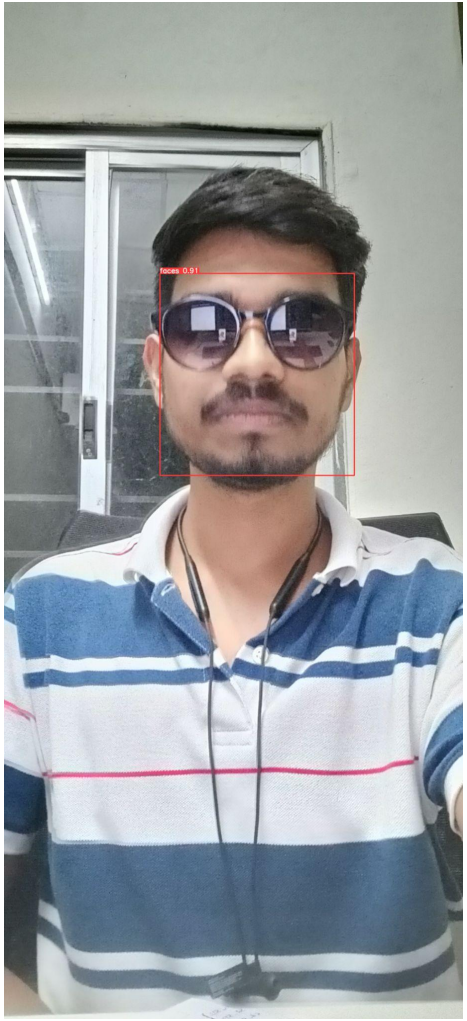






glasses + mask 0.78

glasses + mask 0.70

faces 0.91

faces 0.60


faces 0.79

faces 0.87

faces 0.89

faces 0.88


faces 0.85


faces 0.85


faces 0.91

Observations

- High misclassification in Faster-RCNN
- Sometimes unable to generate predictions
- YOLO is getting confused between faces and faces with glasses
- Faces with sunglasses are detected as faces
- This is because large number of images labeled faces contains glasses in them
- Predicting values for all images
- Mostly misclassified or without prediction class is faces

Solution:

- Data cleaning
- Currently the bounding boxes are generated using face detection method so contains misclassification and even garbage values
- High resolution image

Common Issues

Dataset:

- Limited dataset leading to model overfitting
- Misclassification due to impure data and low resolution

Model Quantization

- Post Training Static Quantization in Eager Mode leads to backend support error in both classification and detection models.
- FX-Graph mode PTSQ leading to higher inference time for classification model
- For detection model, FX-Graph mode Static Quantization can not be used because of non-quantifiable layer such as add and multiplication.
- PTSQ in Eager mode without adding QuantStubs is generating quantized model, but again leads to not supported backend error.
- Adding QuantStubs becomes challenging (in Faster-RCNN) as model returns dictionary values.
- Layer Fusion while quantization also requires thorough description of model.

Conversion in ONNX: Classification Models

Checked Successful conversion by : `onnx.checker.check_model(onnx_model)`

With logits of both pytorch model and onnx model being same

Also performed quantization in onnx format.

Accuracy/Inference Time

Model	Original Data Samples		Under-Sampled Data		Over-Sampled Data	
	Pre-Quantization	**Post-Quantization**	**Pre-Quantization**	**Post-Quantization**	**Pre-Quantization**	**Post-Quantization**
MobileNet (v2_050)	0.9944/4.74ms	0.9931/4.20ms	0.9770/4.75ms	0.9721/4.35ms	0.9969/4.73ms	0.9950/4.02ms
EfficientNet (b0)	0.9869/15.16ms	0.2167/57.63ms	0.9826/17.41ms	0.8643/70.50ms	0.9968/16.23ms	0.4439/55.97ms
ResNet18	0.9981/5.22ms	0.9981/15.12ms	0.9739/4.83ms	0.9708/15.20ms	0.9962/4.69ms	0.9962/11.33ms

Conversion in ONNX: Detection Models

Performed for both Faster-RCNN and YOLOv5s.

Checked Successful conversion by : `onnx.checker.check_model(onnx_model)`
With logits of both pytorch model and onnx model being same

Unable to perform quantization in Faster-RCNN: encountered following error

“onnxruntime::ReshapeHelper::ReshapeHelper(const onnxruntime::TensorShape&, onnxruntime::TensorShapeVector&, bool) input_shape_size == size was false. The input tensor cannot be reshaped to the requested shape. Input shape:{0}, requested shape:{1}”

Possible cause: onnx model gives output as arrays and is not able to divide all input in batch sizes.

In case of YOLOv5, quantized model is not making any inference all metrics are 0.

Assignment Requirement

Model Size: less than 15 MB
Accuracy : > 90%
Inference Time: less than 6ms

Models satisfying requirements:

- MobileNet: All version in onnx format
- YOLOv5 without quantized : Good accuracy with real-time detection (~30fps)

All models are also tested with video inference and YOLOv5 outperforms everyone in terms of accuracy and inference time.

Future Works to Try

- Separate models for each occlusion (classification approach)
- Quality data collection
- Real time detection pipeline for detection methods
- Better accuracy assessment for Faster-RCNN
- Model Quantization and its effect on memory and inference time (More thorough study)