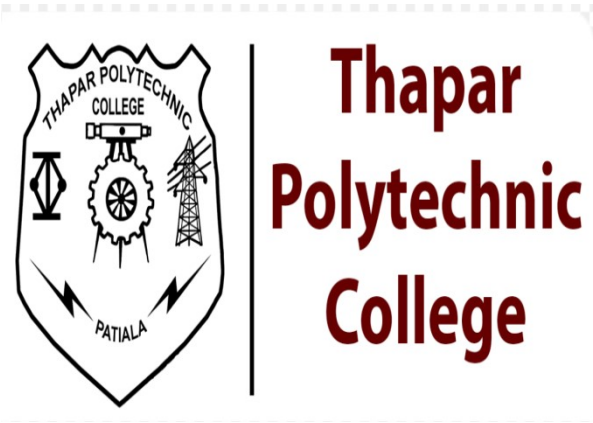
THAPAR POLYTECHNIC COLLEGE

Patiala-147001



Face Mask Detection

Submitted to-
Ms. Tarandeep Kaur
Ms. Aman Singla

Submitted by-
Rishika Gupta (2021/411)
210529505298

ACKNOWLEDGEMENT

This Project work has been the most practical and exciting part of my learning experience, which would be an asset for me and also for my future carrier.

It is my proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this project report. It would not have been possible to prepare this project report in this form without their valuable help, cooperation and guidance.

There were a lot of online resources that I referred to while preparing this project report. I want to thank each one of them for their material, and for the time that they devoted for developing the material. Thanks everyone who has contributed either directly or indirectly in completing Project Report on Minor Project named "**FACE DETECTION USING YOLO7**".

I am grateful to my college Thapar Polytechnic College, Patiala for giving me this opportunity and encouraging us to learn new things and for providing all the required resources and a good working environment. Last but not the least, I wish to thank my parents for financing our studies in this college as well as for constantly encouraging me to learn engineering

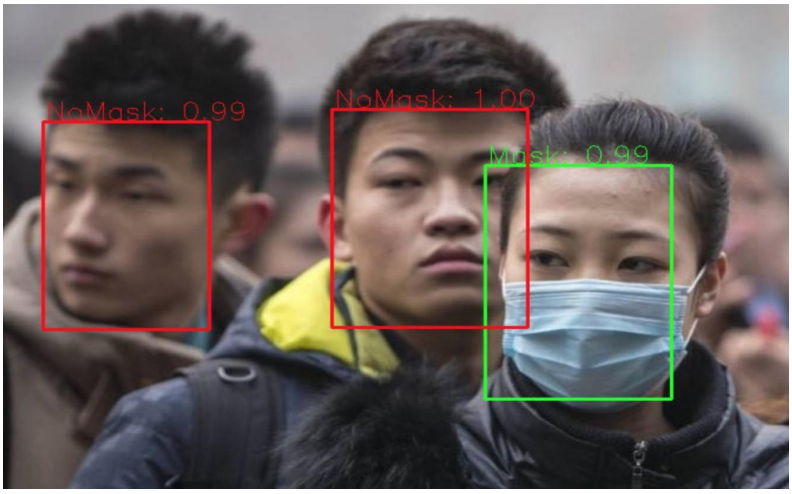
Table of content

Introduction

The "Face Detection using YOLO" project focuses on the implementation and deployment of the YOLO (You Only Look Once) deep learning algorithm for the purpose of real-time face detection in images and video streams. Face detection is a fundamental computer vision task with numerous applications, including security, surveillance, and human-computer interaction. YOLO, known for its speed and accuracy, is a suitable choice for this task.

The objective of this software is to identify the persons who are wearing mask and who are not, to make our work easy to tell them to wear mask if they are not wearing one.

Face mask detection using deep learning is a popular application of computer vision in the context of the COVID-19 pandemic. One such project is detailed in a tutorial by Adrian Rosebrock on his website PyImageSearch. The tutorial explains how to train a COVID-19 face mask detector on a custom dataset with OpenCV, Keras/Tensor Flow, and Deep Learning. The dataset used in the tutorial was created by PyImageSearch reader Prajna Bhandary and consists of 1,376 images belonging to two classes: with_mask and without_mask.



Hardware Requirements

- ARCHITECTURE: All computer operating systems are designed for particular computer architecture.
- PROCESSING POWER: The power of the CPU is a fundamental system requirement for any software.
- MEMORY
- PERIPHERALS
- POWER CONNECTION
- INTERNET CONNECTION

Software Requirements

- **Python**

Python is a popular, high level language that was created by Guido van Rossum in 1991. It is widely used for web development, software development, mathematics, etc. Python is known for simple syntax which allows developers to write programs with fewer lines of code than other programming languages. Python is an interpreted language, python can be created in a procedural way and an object oriented way or a functional way.

- **Kaggle**

Kaggle is an online platform for data science and machine learning enthusiasts. It offers various features, such as:

-Competitions: Users can participate in challenges to solve real-world problems using data and machine learning techniques. Winners can earn prizes and recognition.

-Datasets: Users can find and publish high-quality datasets on various topics, such as sports, health, finance, and more. Users can also explore, analyze, and visualize the data using Kaggle notebooks.

- **Github**

GitHub is a cloud-based platform that provides a Git repository hosting service for software development and version control. It allows users to collaborate on projects, track changes, and manage code using Git, a distributed version control system. GitHub offers various features, such as competitions, datasets, notebooks, courses, and community, to support data science and machine learning enthusiasts. Users can create an account on GitHub for free and access more than 100 million projects and 330 million datasets. GitHub's interface is user-friendly and

provides various tools and libraries, such as Python, Pygame, Kaggle, and more, to implement projects [4]. Users can also share their projects with others and receive feedback and comments [2].

- **NumPy**

NumPy stands for Numerical Python. It is a Python library used for working with arrays, matrices, and higher-dimensional data structures. NumPy provides a high-performance multidimensional array object, and tools for working with these arrays. NumPy was created in 2005 by Travis Oliphant and is an open-source project that you can use freely.

- **Google Colab**

Google colab is a cloud-based platform that allows you to write and execute Python code in your browser without any configuration required. It is a free service provided by Google that provides access to GPUs free of charge. You can use it to analyze and visualize data, import your own data from Google Drive, and even train machine learning models. Colab notebooks are Jupyter notebooks that allow you to combine executable code and rich text in a single document, along with images, HTML, and more. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them.

Methodology

The methodology of this project encompasses several crucial steps:

4.1. YOLO Implementation

The project leverages the YOLO algorithm, a state-of-the-art real-time object detection model. YOLO stands out for its ability to divide images into a grid, predict bounding boxes, and estimate class probabilities for objects within each grid cell. This architecture enables quick and accurate object detection, making it a perfect fit for face detection.

4.2. Data Collection

A diverse and comprehensive dataset of images containing human faces was collected from various sources. It was imperative to ensure that the dataset encompassed a wide range of attributes, including different genders, ages, and ethnicities. This diversity is critical to the model's ability to generalize and accurately detect faces in various real-world scenarios.

4.3. Data Preprocessing

Data preprocessing plays a pivotal role in the success of the project. It involves several key steps:

- Resizing: All collected images are resized to a consistent resolution to create a uniform input format for the model.

- Labeling: Bounding boxes are drawn around faces in the images, creating ground truth labels that the model will be trained to predict.

- Data Augmentation: Data augmentation techniques, such as rotation and flipping, are applied to the images. This augmentation not only increases the dataset's size but also improves the model's generalization by exposing it to various orientations and perspectives.

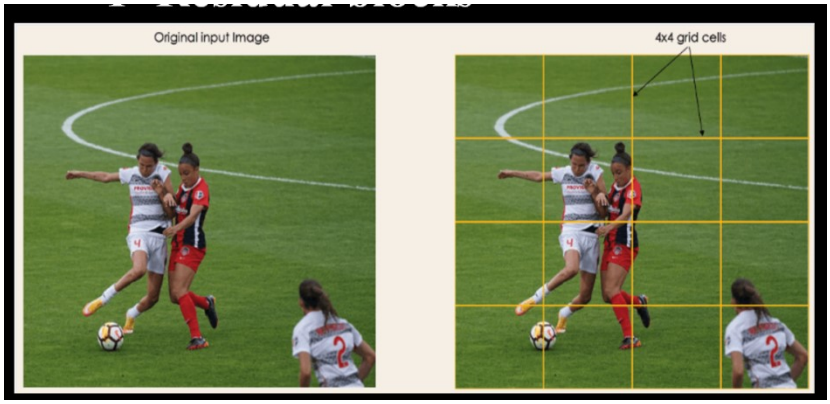
4.4. Training YOLO Model

The heart of the project lies in the training of the YOLO model. This step involves using a suitable deep learning framework, such as TensorFlow or PyTorch, to train the model on the preprocessed dataset. During training, the model's parameters are optimized to minimize detection errors and increase its ability to accurately identify faces.

4.4.1 How YOLO works?

The algorithm works based on the following four approaches:

- Residual blocks = First step starts by dividing the original image (A) into NxN grid cells of equal shape, where N in our case is 4 shown on the image on the right. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.



- Bounding box regression = The next step is to determine the bounding boxes which correspond to rectangles highlighting all the objects in the image. YOLO determines the attributes of these bounding boxes using a single regression module in the following

format, where Y is the final vector representation for each bounding box.

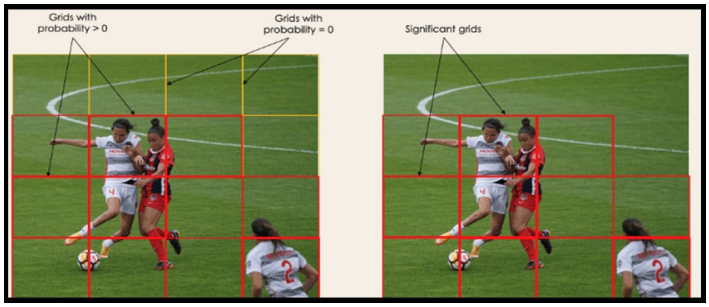
Y = [pc, bx, by, bh, bw, c1, c2] This is especially important during the training phase of the model.

1. ‘pc’ corresponds to the probability score of the grid containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is the simplified version since the probability of each yellow cell is zero (insignificant).

2. ‘bx, by’ are the x and y coordinates of the centre of the bounding box with respect to the enveloping grid cell.

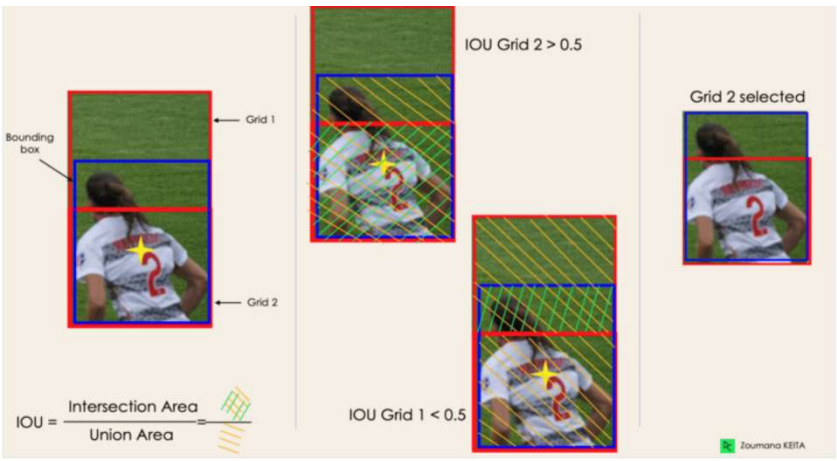
3. ‘bh, bw’ correspond to the height and the width of the bounding box with respect to the enveloping grid cell.

4. ‘c1 and c2’ correspond to the two classes Player and Ball.



- Intersection Over Unions or IOU for short Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. Here is the logic behind it: The user defines its IOU selection threshold, which can be, for instance, 0.5. Then YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area. Finally,

it ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.



- Non-Maximum Suppression. Setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU beyond the threshold, and leaving all those boxes might include noise. Here is where we can use NMS to keep only the boxes with the highest probability score of detection.

4.5. Face Detection

The ultimate goal of the project is to employ the trained YOLO model for real-time face detection. This step entails feeding images and video streams to the model and processing its predictions to obtain precise bounding boxes and class probabilities for detected faces. Post-processing techniques are applied to filter and refine these detections, ensuring high accuracy and minimal false positives.

Results

The success of the project is assessed based on a variety of factors. The key results sought include:

- High Accuracy: The model should achieve high accuracy in detecting faces across diverse images and video streams.

- Real-time Performance: The model should demonstrate real-time detection capabilities, ensuring swift processing for video streams.

- Minimized False Positives and Negatives: The project aims to reduce the number of false positive and false negative detections, which are critical in applications such as security and surveillance.

What makes YOLO popular for object detection?

Some of the reasons why YOLO is leading the competition include its:

- Speed: It can process images at 45 Frames per Second (FPS).
- Detection accuracy: Very few background errors.
- Good generalization: Providing a better generalization for new domains, which makes it great for applications relying on fast and robust object detection.
- Open-source:-Making YOLO open-source led the community to constantly improve the model.

Key characteristics and features of YOLO include:

1. Real-time Object Detection: YOLO is designed for real-time or near-real-time object detection applications, making it suitable for tasks like video analysis and autonomous driving.

2. Single Network Architecture: YOLO uses a single neural network that takes an entire image as input and outputs bounding boxes and class probabilities for detected objects in one go.

3. Bounding Box Predictions: YOLO predicts bounding boxes around objects in the image, along with the class of each detected object. These bounding boxes are represented by (x, y) coordinates for the box's center, width, and height.

4. Multiple Scales: YOLO divides the input image into a grid and makes predictions at multiple scales within the grid. This allows YOLO to detect objects of different sizes.

5. Anchor Boxes: YOLO uses anchor boxes to improve accuracy in predicting bounding boxes for objects. Anchor boxes are predefined shapes that help the model predict bounding boxes more accurately.

6. High Accuracy: YOLOv3 and later versions have achieved competitive accuracy levels while maintaining real-time performance, making them suitable for a wide range of applications.

7. Open Source: YOLO is open-source, and its source code and pre-trained models are freely available, which has contributed to its popularity and adoption in the computer vision community.

YOLO has been widely used in various applications, including object tracking, surveillance, autonomous vehicles, robotics, and more. Its ability to provide real-time object detection with high accuracy has made it a valuable tool in the field of computer vision. Developers often use pre-trained YOLO models and fine-tune them for specific tasks or train custom YOLO models on their datasets to address specific object detection requirements.

5.1 Challenges and Limitations

The "Face Detection using YOLO" project was not without its challenges and limitations. These are vital to understanding the scope and potential for further improvements:

5.1.1. Challenges

- Imbalanced Datasets: Creating a balanced dataset with diverse attributes proved to be challenging, as it often required extensive data collection and curation efforts.

- Overfitting: Overfitting, a common challenge in deep learning, was addressed through techniques like data augmentation and regularization.

- Computational Resources: Training the YOLO model demands significant computational resources, which may pose a challenge for projects with limited access to such resources.

5.1.2. Limitations

- Occlusions: The model may struggle to detect faces that are partially occluded, which is a common real-world scenario.

- Low-Light Conditions: The model's performance may degrade in low-light conditions, highlighting a need for further improvement.

- Extreme Angles: Detecting faces at extreme angles may be challenging, and this limitation should be considered in applications that require robust face detection.

Source code

- Download dataset:-

```
!pip install -q kaggle

!rm -r ~/.kaggle

!mkdir ~/.kaggle

!mv ./kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d andrewmvd/face-mask-detection

!unzip "/content/face-mask-detection.zip" -d "/content/face-mask-detection/"
```

- Importing libraries:-

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
# For preparing
import xml.etree.cElementTree as ET
import glob
import os
import json
import random
import shutil
from PIL import Image, ImageOps

with open('/content/face-mask-detection/annotations/maksssksksss0.xml') as f:
```

```
contents = f.read()
print(contents)

Image.open("/content/face-mask-detection/images/maksssksksss0.png")

Image.open("/content/face-mask-detection/images/maksssksksss1.png")

Image.open("/content/face-mask-
detection/images/maksssksksss100.png")

# creating a image1 object
im1 = Image.open(r"/content/face-mask-
detection/images/maksssksksss100.png")
# applying grayscale method
im2 = ImageOps.grayscale(im1)
im2
```

- Preparing dataset

```
#translate .xml format to .txt format
def xml_to_yolo_bbox(bbox, w, h):
# xmin, ymin, xmax, ymax
x_center = ((bbox[2] + bbox[0]) / 2) / w
y_center = ((bbox[3] + bbox[1]) / 2) / h
width = (bbox[2] - bbox[0]) / w
height = (bbox[3] - bbox[1]) / h
return [x_center, y_center, width, height]

def yolo_to_xml_bbox(bbox, w, h):
# x_center, y_center, width, height
w_half_len = (bbox[2] * w) / 2
h_half_len = (bbox[3] * h) / 2

xmin = int((bbox[0] * w) - w_half_len)
ymin = int((bbox[1] * h) - h_half_len)
```

```
xmax = int((bbox[0] * w) + w_half_len)
ymax = int((bbox[1] * h) + h_half_len)

return [xmin, ymin, xmax, ymax]

classes = []

input_dir = "/content/face-mask-detection/annotations"
output_dir = "/content/labels"
image_dir = "/content/face-mask-detection/images"

os.mkdir(output_dir)

if not os.path.isdir(output_dir):
os.mkdir(output_dir)

import glob

files = glob.glob(os.path.join(input_dir, "*.xml"))
for fil in files:
basename = os.path.basename(fil)
filename = os.path.splitext(basename)[0]
if not os.path.exists(os.path.join(image_dir, f"{filename}.png")):
print(f"{filename} image does not exist!")
continue

result = []

# Parse the content of the xml file
tree = ET.parse(fil)
root = tree.getroot()
width = int(root.find("size").find("width").text)
height = int(root.find("size").find("height").text)

for obj in root.findall("object"):
```

```
label = obj.find("name").text

# check for new classes and append to list
if label not in classes:
classes.append(label)
index = classes.index(label)
pil_bbox = [int(x.text) for x in obj.find("bndbox")]
yolo_bbox = xml_to_yolo_bbox(pil_bbox, width, height)

# convert data to string
bbox_string = " ".join([str(x) for x in yolo_bbox])
result.append(f"{index} {bbox_string}")

if result:
# generate a yolo format text file for each xml file
with open(os.path.join(output_dir, f"{filename}.txt"), "w", encoding = "
utf-8") as f:
f.write("\n".join(result))
# generate the classes file as reference
with open("/content/classes.txt", "w", encoding = "utf-8") as f:
f.write(json.dumps(classes))

#contents like with mask, etc.
with open('/content/classes.txt') as f:
contents = f.read()
print(contents)


Image.open("/content/face-mask-detection/images/maksssksksss0.png")

with open('/content/face-mask-
detection/annotations/maksssksksss0.xml') as f:
contents = f.read()
print(contents)
```

```
with open('/content/labels/maksssksksss0.txt') as f:
contents = f.read()
print(contents)
```

- Created dataset for YOLO7

```
os.mkdir("/content/data/")
os.mkdir('/content/data/train')
os.mkdir('/content/data/val')
os.mkdir('/content/data/test')
os.mkdir('/content/data/train/images')
os.mkdir('/content/data/train/labels')
os.mkdir('/content/data/test/images')
os.mkdir('/content/data/test/labels')
os.mkdir('/content/data/val/images')
os.mkdir('/content/data/val/labels')

metarial = []

for i in os.listdir("/content/face-mask-detection/images"):
srt = i[:-4]
metarial.append(srt)


len(metarial)

# a few image names of in the dataset
metarial[0:10]

def preparinbdata(main_txt_file, main_img_file, train_size, test_size, va
l_size):

for i in range(0,train_size):

source_txt = main_txt_file + "/" + metarial[i] + ".txt"
source_img = main_img_file + "/" + metarial[i] + ".png"
```

```
mstring = metarial[i]
train_destination_txt = "/content/data/train/labels" + "/" + metarial[i] + "
.txt"
train_destination_png = "/content/data/train/images" + "/" + metarial[i]
+ ".png"

shutil.copy(source_txt, train_destination_txt)
shutil.copy(source_img, train_destination_png)

#metarial.remove(file_name[:-4])

for l in range(train_size , train_size + test_size):

source_txt = main_txt_file + "/" + metarial[l] + ".txt"
source_img = main_img_file + "/" + metarial[l] + ".png"

mstring = metarial[l]
test_destination_txt = "/content/data/test/labels" + "/" + metarial[l] + ".tx
t"
test_destination_png = "/content/data/test/images" + "/" + metarial[l] + "
.png"

shutil.copy(source_txt, test_destination_txt)
shutil.copy(source_img, test_destination_png)

#metarial.remove(file_name[:-4])

for n in range(train_size + test_size , train_size + test_size + val_size):

source_txt = main_txt_file + "/" + metarial[n] + ".txt"
source_img = main_img_file + "/" + metarial[n] + ".png"

mstring = metarial[n]
val_destination_txt = "/content/data/val/labels" + "/" + metarial[n] + ".tx
t"
```

```
val_destination_png = "/content/data/val/images" + "/" + metarial[n] + "
.png"

shutil.copy(source_txt, val_destination_txt)
shutil.copy(source_img, val_destination_png)

#metarial.remove(file_name[:-4])

preparinbdata("/content/labels", "/content/face-mask-
detection/images", 603, 150, 100)

# configure .yaml file to guide the model for training
%cd /content/data

yaml_text = """train: /content/data/train/images
val: /content/data/val/images

nc: 3
names: ["without_mask", "with_mask", "mask_weared_incorrect"]"""

with open("/content/data/data.yaml", 'w') as file:
file.write(yaml_text)

%cat data/data.yaml
```

- Downloading YOLO7

```
!# Download YOLOv7 code

%cd /content/

!git clone https://github.com/WongKinYiu/yolov7

%cd yolov7

!ls

!# Download trained weights
```

```
%cd /content/

# Train
!python /content/yolov7/train.py --workers 8 --device 0 --batch-size 16 --epochs 50 --data /content/data/data.yaml  --cfg /content/yolov7/cfg/training/yolov7.yaml --weights '' --name yolov7_1 --hyp /content/yolov7/data/hyp.scratch.p5.yaml
```

- Result Visualization

```
Image.open("/content/runs/train/yolov7_1/results.png")

Image.open("/content/runs/train/yolov7_1/train_batch0.jpg")

Image.open("/content/runs/train/yolov7_1/train_batch9.jpg")

!# Detection
!python /content/yolov7/detect.py --weights /content/runs/train/yolov7_1/weights/best.pt --conf 0.25 --img-size 640 --source /content/data/test/images/

Image.open("/content/runs/detect/exp/maksssksksss1.png")

Image.open("/content/runs/detect/exp/maksssksksss508.png")
```

- Detection

```
# I downloaded one image for use at the detect.
%cd /content
!wget "https://onecms-res.cloudinary.com/image/upload/s--XV7DHKzY--/c_fill,g_auto,h_468,w_830/f_auto,q_auto/people-wearing-mask-at-orchard-road-singapore-feb-3--49-.jpg?itok=GdDk1T6A"

!# Detection
!python //content//yolov7//detect.py --
```

weights /content/model/model_r101_2_class.pt --conf 0.25 --img-size 640 --source //content//12_58_5000.png

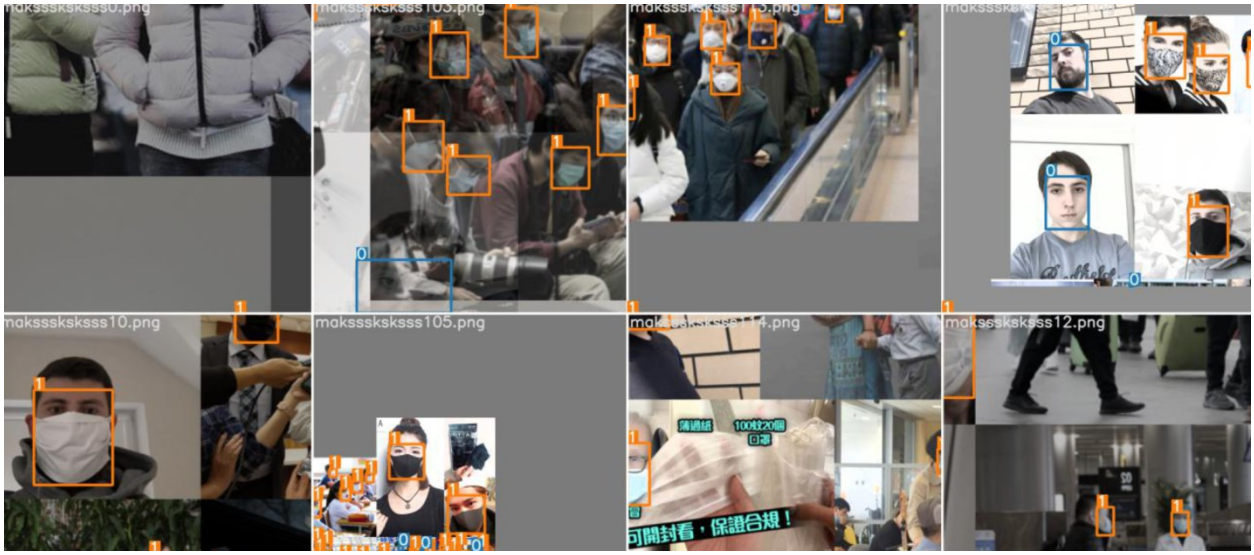
Image.open("/content/runs/detect/exp3/image0.jpg")

Output:

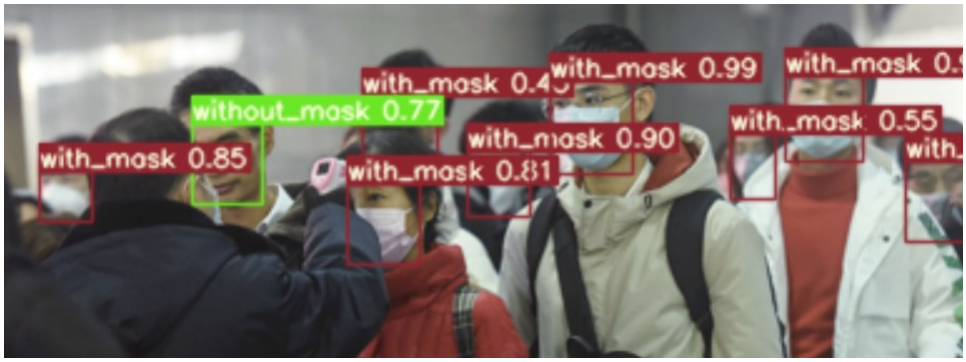
1. Picture from the dataset we uploaded at first:

2. Picture after applying grayscale on it:

3. Training result after YOLO7:

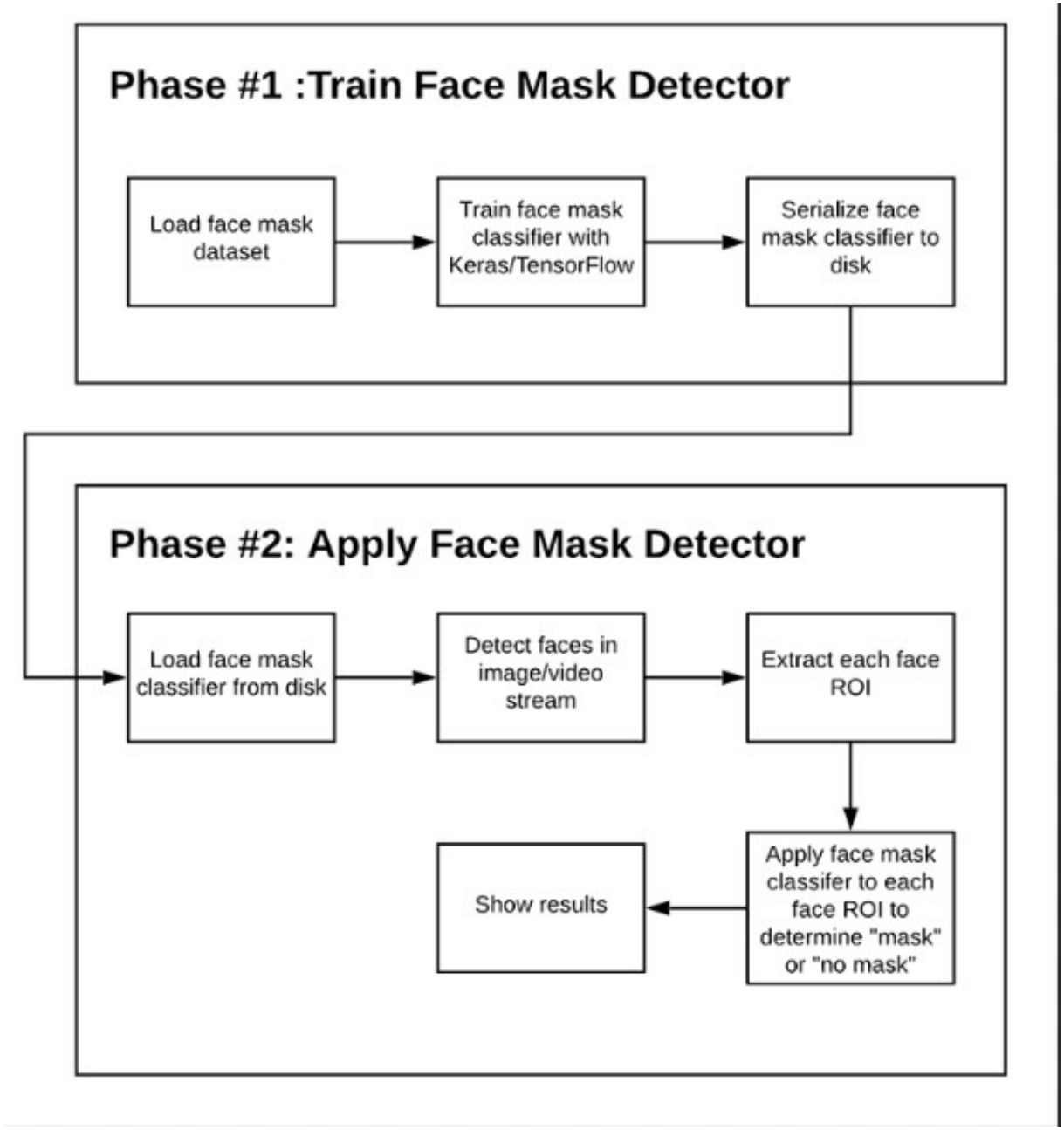


4. After detection result:



5. Final result:

FLOWCHART


Phase #1 :Train Face Mask Detector
Load face mask dataset
Train face mask classifier with Keras/TensorFlow
Serialize face mask classifier to disk
Phase #2: Apply Face Mask Detector
Load face mask classifier from disk
Detect faces in image/video stream
Extract each face ROI
Apply face mask classifer to each face ROI to determine "mask" or "no mask"
Show results

Conclusion

In conclusion, the "Face Detection using YOLO" project harnesses the power of the YOLO algorithm to accomplish accurate and real-time face detection. This capability has far-reaching implications for various domains, including security, surveillance, and human-computer interaction. The project serves as a significant step towards advancing computer vision systems' capabilities.

-The project aimed to create a model that can detect whether people are wearing masks, not wearing masks, or wearing masks incorrectly, using the YOLOv7 algorithm trained on a custom dataset.

-The project used a dataset of 853 images belonging to three classes: with mask, without mask, and mask worn incorrectly. The images were manually annotated using labelimg, a graphical image annotation tool.

-The project followed the steps of data preparation, model training, model testing, and model evaluation. The project used various tools and libraries, such as Python, Pygame, Kaggle, and GitHub, to implement the project.

-The project achieved satisfactory results, with high accuracy and low loss. The model was able to detect faces and masks in various scenarios and conditions, such as different lighting, angles, and backgrounds. The model also demonstrated robustness and generalization to unseen images.

-The project demonstrated the potential and usefulness of applying deep learning and object detection techniques to address real-world problems, such as the COVID-19 pandemic. The project also showed the challenges and limitations of creating a custom object detection model, such as data quality, annotation, and hyperparameter tuning.

References

- Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.

- [YOLO Official Website](https://pjreddie.com/darknet/yolo/)

- [Wider Face Dataset](http://shuoyang1213.me/WIDERFACE/)

-Stack Overflow- Website: Stack Overflow Description: Stack Overflow was an invaluable resource for troubleshooting specific issues, coding challenges, and obtaining solutions to common problems faced during the development of the project. 6. Open Source Art and Sound Resources Various open-source art of sound assets