VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

FACE MASK CLASSIFICATION AND DETECTION BASED ON STATE-OF-THE-ART DEEP LEARNING APPROACH

By

Nguyen Quynh Huong

A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of Bachelor of Data Science

Ho Chi Minh City, Vietnam

2022

FACE MASK CLASSIFICATION AND DETECTION BASED ON STATE-OF-THE-ART DEEP LEARNING APPROACH

APPROVED BY:

_____ ,

Mai Hoang Bao An, Ph.D.

_____

_____

_____

_____

THESIS COMMITTEE

ACKNOWLEDGMENTS

The first and foremost thing I'd want to do is express my thank to Dr. Mai Hoang Bao An, who serves as my supervisor during this thesis topic, for his mentoring and assistance during the process. The direction and support I got during my studies were really enthusiastic, and he has directed and motivated me to find new things about machine learning and accomplish this tough but enjoyable experience. I'd want to use this moment to convey my gratefulness for him.

I must express my very profound gratitude to the International University-Vietnam National University HCMC, especially those lecturers at the School of Computer Science and Engineering, for providing me with a strong foundation in technical knowledge and practical skills that I was able to use in my thesis. I also would like to express my thanks to the authors of the articles and books that I have consulted for the great knowledge they have shared with the community. This thesis would not have been possible without them.

In summary, this thesis has certainly piqued my interest in machine learning, which I already know from school, and it was a fantastic chance to put that knowledge and skill to use in a real-world application. While working on this face mask detection thesis, I gained a great deal of good experience and helpful advice from my supervisor. I place on record, my sense of gratitude to who directly or indirectly, have lent me a hand during the last 4 years of school.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. Google Colab : Colaboratory by Google
2. ML: Machine learning
3. DL: Deep learning
4. CNN: Convolutional Neural Networks
5. AI: Artificial intelligence
6. ANNs: Artificial Neural Networks
7. SVM: Support Virtual Machine
8. VOC: Visual Object Classes
9. COCO: Common-Objects in Context
10. CSPNet: Cross-Stage Partial network
11. FLOPS: Floating-point-operations / second
12. PANet: Path Aggregation Network
13. FPN: Feature pyramid network
14. SPP: Spatial pyramid pooling
15. IoU: Intersection over Union
16. YOLO: You Only Look Once
17. GUI: Graphic user interface

ABSTRACT

Corona is one of the most destructive viruses that has ever produced a pandemic in human life, not only in terms of direct victims but also in terms of the socio-economic consequences of the virus' transmission. The 2nd anniversary of the global coronavirus pandemic passed away in 2021. However, it's still impossible to say how long the epidemic will last.After reviewing a study by the World Health Organization on COVID-19, the country's national government urged residents to use facemask in order to reduce the incidence of COVID-19 transmission. As a result of COVID-19, there are presently no facemask detection app that are in great demand for ensuring safety in public area.

In the context of the outbreak of COVID-19, A facemask detection model based on deep learning and YOLOv5 may be used in real-time applications, such as surveillance cameras, that need face-mask detection for safety reasons. We used OpenCV to recognize faces in real-time, which we found really useful. Thanks to computer vision and deep learning techniques, we aim to be able to identify whether or not the individual in the video is wearing facemask. In order to guarantee that public safety requirements are followed, this program may be combined with embedded systems for use in airports, train stations, offices, and other public areas. It can also be used in the military and companies...

CHAPTER 1: INTRODUCTION

1.1. Background

1.1.1 Off-guard humanity in the midst of COVID-19

It has been more than two years since the COVID-19 virus outbreak; therefore, humanity has yet to overcome its influence. The pandemic has resulted in a significant loss of human life throughout the world, and even though that number is declining, the impact that COVID-19 has on public health is no doubt a threat to all lives. As governments strive to establish a new normal, some begin to lift mobility and social-distancing restrictions. There is nothing wrong with adapting to the new normal. However, many people tend to mistake the new normal state for the end of the pandemic. Unfortunately, a decline in the death rate does not imply an end to the fatal pandemic.

It is now common knowledge that the virus that rose to fame overnight is spread from one person to another through respiratory droplets. Contrary to what was said, countless people started to lower their guard as they lower their masks. When that happens, COVID-19 perceives their action as a welcoming invitation. With the amount of negligence it has received, COVID-19 has become more dangerous now than ever.

1.1.2 Face mask as a shield from flu

Face masks, in conjunction with other safety precautions such as vaccination, frequent hand-washing, and physical distancing, are unquestionably the shield that defends humanity from the worldwide adversary. Back when COVID-19 vaccination was only a pipe dream, people had to make do with face masks to protect themselves. Even before the COVID-19 virus breakout, masking had been proven to be a critical public health technique. It is apparent that understanding the role of such a measure in infection reduction is of utmost importance.

From surgical to non-surgical procedure masks, as long as they are well-fitted to the contours of one's face to prevent air leakage around the edges, these are masks that offer flu protection; protection from all flu viruses, not only COVID-19. Large particles in the air, such as respiratory droplets or aerosols, are filtered out when a person breathes in through a mask, and what these particles convey is nothing new to humanity: flu viruses. When an infected person coughs, sneezes, talks, or just breathes deeply, the virus may spread in micro liquid particles from their mouth or nose. That is when masks come into play. Wearing a medical mask can reduce the passage of infectious droplets from a symptomatic patient to another person, as well as potential environmental contamination by these droplets, according to a World Health Organization study

of influenza, influenza-like illnesses, and human Coronaviruses (excluding COVID-19). The study also provides concrete evidence that medical masks are highly effective in terms of source control, as when an infected individual wears one to avoid continuous transmission. That being said, masks should be used for both protection and source control purposes.

To put it in a nutshell, there are numerous studies on the benefits of wearing masks to improve public health from not just COVID-19 but all flu viruses. This critical strategy should not be taken for granted to flatten the epidemiologic curve, not even when vaccination is available.

1.1.3 Policy on wearing face masks

The devastating impact of the COVID-19 pandemic has not been ignored in Vietnam, which borders China and is the source of the epidemic. When the outbreak was well under control, Vietnam faced a second wave and suffered many months of insignificant community transmission. According to the website Worldometer, the World Health Organization and the United States have applauded Vietnam, which has a population of more than 96 million people, for its prompt response and efforts in limiting the virus.[1]

During the coronavirus outbreak, state leaders and the media have constantly emphasized the importance of wearing masks. The media landscape was littered with images of government officials donning masks at meetings. It was not just a motivating message, but also a rule that all citizens had to follow. Although the World Health Organization advises that only individuals who have been positively confirmed with COVID-19 use medical masks to avoid transmission, mask adoption appears to be linked to a slower rate of COVID-19 transmission in some Asian nations. South Korea, China, and Vietnam, to name a few, are all excellent examples. The study by Huynh [2] found that Vietnamese people wear masks because they believe they are under danger. According to announcement No. 98, the Vietnamese government has requested that in public places like airports, bus stations, supermarkets, and public transportation, all Vietnamese and foreign nationals be required to wear face masks since March 16, 2020 [3] . Face masks have also been required for all passengers on local and international flights to and from Vietnam, as well as during their time at airport terminals. Free masks were distributed to people who did not have them at airports and train stations. Vietnamese authorities have been putting stiffer fines on anyone who refuses to wear face mask. Those who do not follow the A person disobeying the government's instructions could face a punishment of up to 130 dollars, whereas according to CEIC Data, the per capita income in Vietnam was just 2,779 dollars in December 2020.

1.2. Problem Statement

Every ounce of technological innovation and creativity used in the fight against the COVID-19 outbreak brings us closer to defeat. Artificial intelligence and machine learning are essential tools for better understanding and dealing to the COVID-19 epidemic. The science of machine learning allows computers to imitate human intellect and swiftly detect patterns and insights from vast amounts of data. In the battle against COVID-19, enterprises have used their machine learning capabilities in a number of areas, consist of expanding customer communications, determining how COVID-19 spreads, and accelerating research and treatment [4].Recent research has shown that AI may be used to identify the presence of face masks during the Covid-19 epidemic Since wearing a mask is a matter of utmost importance for people to curb the rate of infection by the Covid-19 virus, the outcome of the study will definitely produce a positive impact on the global movement of reinforcing wearing a mask in public [5] . Artificial Intelligence (AI) is used in this thesis, as well as deep learning and machine learning.

In summary, machine learning algorithms are used to analyze a large quantity of data related to COVID-19 patients and attempt to identify the probable causes of susceptible scenarios that arise with COVID. Using deep learning models, researchers are able to determine the effectiveness and precision of COVID-19 prevention measures. According to strict rules and regulations, several countries now demand the use of face masks while going out in public. The law enforcement in charge of enforcing these rules and regulations checks the public to see whether anybody is wearing a face mask. It is, however, very time-consuming to manually check each and every member of a group for face masks. It was because of this that I came up with the present application, in which I use numerous machine learning and deep learning models to teach the general public how to recognize those wearing facemask or not. When a face mask is detected in real time video, the model can automatically and reliably label the image with some indication of who is wearing it or not.

1.3. Scope and Objectives

COVID-19 has obliged individuals worldwide to wear face masks and has also slowed the country's economic development [6] . Nowadays, people are required to wear face masks in public locations under a series of enforced protocols. Even experts have shown that wearing facemask may help prevent the virus from spreading. The research department determined the efficacy of

N95 and surgical masks to be 91% and 68%, respectively [7]. However, effectiveness in preventing the spread of this disease has waned as a result of incorrect use of face masks in public spaces. It is critical to automate the process of detecting individuals using facemasks in public locations since this will safeguard individuals and avert a local epidemic.

Machine Learning is a fascinating subset of AI in which computers are educated autonomously for new dataset via iterations. It is trained repeatedly until the required output is obtained. Typically, ML is utilized in image processing, real-time advertising, and spam filtering methods, among other applications. DL is a subset of machine learning approaches based on ANNs [8] that uses hierarchical architecture to learn abstractions from data. DL is composed of several neural networks which are managed by the processor's cores. It is a relatively new method that is frequently employed in artificial intelligence applications, including computer vision.

It has been a decade since Convolutional Neural Networks, first developed in the late 1980s, have been widely used. The COVID-19 pandemic's effect on computer vision research has led to new research areas being explored in this context by researchers. CNN is a key deep learning technique that is capable of performing robust training. Classification and object detection are the two approaches that will be used in this thesis to solve the issue of face mask detection. When it came to the Classification approach, we built the model using Resnet50 and MobileNetv2 and then compared both of these models. In building the model for object detection, we make use of YOLOv5. These are Convolutional Neural Networks, which are most widely used because of their robust feature learning and classification capabilities. The outline of the report is that we will present the limitations of the classification approach to the problem of mask detection, and explain the question of why we decided to apply object detection as a solution to the face mask detection problem. After that, we proceed to present in further depth the network architecture, the steps involved in the model development process, and the evaluation metrics for each approach. Finally, the results of this study.

CHAPTER 2: RELATED WORK AND APPLIED TECHNOLOGIES

2.1 Related work

In 2018, Boyko, Basystiuk, and Shakhovska published a research article[9] using the Dlib and OpenCV libraries. The purpose of this study is to evaluate the performance of these two most popular CV libraries while searching for and detecting objects using the hog technique. The OpenFace library was used to get the coordinates of the face boundary. To have better accuracy, they used 128 facial feature extractions and divided the facial features into groups.

In 2020, Militante & Dionisio (presented a study[11] in which a dataset of 25,000 pictures with a pixel resolution of 224x224 are used to achieve a 96% accuracy. They used ANNs to simulate human brain activation simulations. They are utilizing Raspberry Pi in their research to identify when someone enters a public space without wearing a mask.

In the research article in 2020, Das, Ansari, and Basak presented their research [12] in which they compared the accuracy and loss rates of two datasets. The outcome of this study is based on the OpenCV, TensorFlow, and Keras libraries. Das et al. created their own dataset, consisting of 1376 photographs, 690 with white masks, and 686 without; the second dataset comes from Kaggle, consisting of 853 images categorized into two categories, with and without masks. They trained their model using 20 epochs with a test split of 90 percent training and 10 percent validation data, which resulted in an accuracy rate of 95.77 percent for dataset1 and 94.58 percent for dataset2.

Loey, Manogaran, Taha, and Khalifa in 2021 showed their study [13] proposing the use of three datasets, arguing that the same algorithms should be used to differentiate the accuracy of the datasets. The three datasets utilized are RMFD, SMFD, and LFW . This study employed both deep learning (ResNet50) and classical machine learning (SVM), and the researchers believe ResNet-50 performs better as a feature extraction method. ResNet-50 is used as a feature extractor, whereas the SVM (Support Virtual Machine) is used for training, validating, and testing. Using the technologies employed to perform the research, they archived 99.64 percent in RMFD, 99.99 percent in SMFD, and 100 percent in LFW.

A study report on the detection of medical face masks was provided by Loey, Manogaran, Taha, and Khalifa in 2021 [14] . In this research, they compared the accuracy of two datasets. They calculated the average precision of both datasets and believe that integrating YOLOv2 with the ResNet50 model might help them achieve a high average accuracy. The validation accuracy and loss for each epoch, iteration, and variable learning rate are the experimental results of this study.

Guillermo et al. (2020) presented their research paper which has been used for face mask detection. In this research paper, they have implemented this model to create an artificial dataset on their own. They have used around 600 raw without mask images and have obtained an artificial dataset by putting the mask on the face by coordinates gathered using machine learning algorithms. To fight COVID-19's spread, the object detection technique may be used to build an effective face mask detection system.

YOLO is the state-of-the-art object detection algorithm with a proven track record YOLO. Susanto, et al. employed the YOLO deep learning algorithm to identify the white ball and goal that are incorporated into humanoid robot soccer, as seen in [15]. The NVIDIA JETSON TX1 controller board was used to carry out this method. Liu, etal. [16] presented the other work that used the YOLO. They used classic image processing in this project to shoot noise, blurring, and rotating filters in the actual environment. The YOLO technique was then used to build a robust model to enhance traffic sign identification. Yang, et al., on the other hand, employed the YOLO technique to accurately and quickly recognize the face in a real-time application. In contrast to [17] they enhanced the YOLO algorithm for recognizing the face in a video sequence and compared the detection accuracy to the old technique in [18] .They utilized the FDDB dataset to train and test the model as well. Zhao, et al [19] have also worked on improving the YOLO model. They developed the YOLO model to identify pedestrians, addressing two issues: leveraging real-time saliency via security cameras and extracting detailed information on distinguishing features.

After a few years, the YOLO system was upgraded to YOLOv2, which could recognize over 9000 different item types. A unique multi-scale training approach was devised in this version [20]. The YOLOv2 for image recognition and additional testbenches for a CNN accelerator were then built by Kim et al. The YOLOv2 approach was used in this study via simulation and FPGA experiment [21]. Harisankar, et al. [22] on the other hand, added the Model H architecture to the YOLOv2 to identify and locate pedestrians. They employed a ZED camera to produce a depth map in order to accurately identify the pedestrian.

Redmon et al. within 2 years released the YOLOv3, which, while having a slightly larger architecture than the previous version, is as accurate as the SSD method but three times quicker [23]. As a result, Hu, et al. [24] used this version to recognize employees wearing or not wearing helmets in films. The initial step of the YOLOv3 model was to locate and intercept the worker video, after which a sample model of wearing and not wearing a helmet was created. Theoretical analysis and actual findings show that the suggested method is capable of accurately detecting the

helmet. Because the YOLO technique is open source, anybody may enhance the YOLO method's algorithm. Alexey, et al. introduced the YOLOv4 with ideal speed and accuracy, as shown in [25]. The YOLOv4 was utilized by Firas Amer and Mohammed S. H. Al-Tamimi. In the paper show that the YOLOv4 is two times faster than the other deep neural network technique, as an object detection method. This new version's performance may boost YOLOv3's AP by 10% and its FPS by about 12%.[26]. In this thesis, we use YOlOv5 which is the latest version of YOLO family, trains quicker on the sample task, and batch inference generates real-time results. YOLOv5 has seen considerable performance improvements, hitting a maximum frame rate of 140 frames per second. The weight file of YOLOv5 is about 90% less than that of YOLOv4, enabling it to be used on embedded devices. YOLOv5 outperforms YOLOv4 in terms of accuracy and sensitivity to small objects. Despite the fact that YOLOv4 trains more slowly, its performance may be improved to increase FPS. Due to the fact that YOLOv5 is implemented in PyTorch and YOLOv4 is implemented in Darknet, YOLOv5 may be simpler to get into production, although YOLOv4 is where top-accuracy research may continue to advance. Hence, we applied YOLOv5 for the face mask detection to detect face mask in real-time.

2.2 Applied technologies

2.2.1 Colab

Google Colab is a Google Research project that offers a free Jupyter-based environment for Python development and execution. A programming notebook is a shell or kernel that looks like a word processor and enables us to write and execute code. Data for Google Colab processing may be mounted on Google Drive or imported from any online source. Google Colab does not need installation and provides free access to GPUs. The ability to share live code, data processing (cleaning and transformation), mathematical equations, numerical simulations, data visualizations, machine learning models, and a range of other projects with others is one of Google Colab's key advantages. GPUs are currently dominating in machine learning and deep learning due to enhanced capabilities of more compute-intensive workloads and streaming memory models. GPUs give exceptional performance via parallelism and may generate millions of threads in a single call. Even though GPUs have slower clock rates and lack tools for handling multiple cores, they perform much better than CPUs.

Besides, Google Colab allows you to combine executable code and rich text in a single document, together with images, HTML, LaTex, and other forms. It comes pre-loaded with a crucial machine learning library, such as TensorFlow, making it suitable for building ML and DL models. Colab is a fantastic tool for building neural networks. Utilizing a CPU-based hardware accelerator, we may accomplish parallelism and the execution of multiple threads. Google Colab notebooks may be publicly shared as instructional notebooks. The combination of HTML components and text style makes an attractive and relevant lesson notebook, and the mixing of text with code is very effective for explaining code flow and logic. Data scientists and machine learners may analyze and visualize data using the full power of Python modules. Google Colab can upload code from GitHub and import data from Kaggle.

2.2.2 Streamlit

After training the model, there are several options for deploying a web-based project, such as Flask, Django, and Streamlit.Flask and Django are rather complex, while Streamlit enables us to develop apps for your machine learning project using fundamental programming. It also supports hot-reloading, which enables you to update your application in real-time as you edit and save material. Using Streamlit, creating an application is as simple as defining a variable, and adding a widget is as simple as declaring a variable.There is no need to build a backend, configure numerous routes, or handle HTTP requests. Without utilizing HTML, CSS, or JS, you can design user interfaces for our models using Streamlit. The majority of models are unattractive and expire within a Jupyter notebook. Streamlit, on the other hand, may generate a nice UI for our model and show it off to others. Creating a user interface allows users to interact with your model in a friendly format.

The development of a machine learning model is worthless if it is not made available to the general public or a specific client. Generally speaking, if you are working for a client, you should deploy the model in the client's environment. However, if you are working on a public-facing project, you should use web deployment methods. Streamlit is the most efficient method for deploying lightweight websites.

2.2.3 Roboflow:

Roboflow is a developer framework for CV that improves data collection, preprocessing, and model training techniques. Roboflow provides us with access to both public datasets and the ability

to contribute our custom datasets. Roboflow supports many formats for annotations. Data preprocessing includes operations such as image orientations, resizing, contrast adjustments, and data augmentation.

Within the framework, the complete process may be coordinated with teams. Existing model libraries for model training include EfficientNet, MobileNet, YOLO, TensorFlow, PyTorch, and others. Additionally, model deployment and visualization options are provided, thereby spanning the whole state of the art.

CHAPTER 3: METHODOLOGY

1. Limitation of classification method

Techniques for image classofication and object identification are important for computer vision. Using digital photos as inputs, these technologies enable robots to detect and understand objects in real time. Computer vision technology has been used in several industries throughout the years, including healthcare, manufacturing, and retailing.

Image classification requires passing the whole picture through a classifier (such as a deep neural network) to get a label. Classifiers evaluate the whole picture but do not provide the exact location of the label. Object detection is more complex since it combines two tasks (object localization and image classification) and constructs a bounding-box around each object's interest in image and labels them with their class label. Slightly more advanced object detection generates a bounding box around the identified object. On the other hand, object detection systems will almost always outperform classification networks when it comes to detecting real items.

The performance of an image classification model is measured by the mean classification error over all predicted class labels. Whereas, the precision and recall of each of the best-matching bounding boxes for the identified objects in the image measured the performance of an object detection model.

Hence, the classification method can also detect whether someone is wearing a face mask or not, but when detecting in the image there are many people, it will not be accurate. So, we apply object detection specifically using YOLOv5. In the next few chapters, we will elaborate more on the network architecture of each method and why we chose YOLOv5 for the Object Detection method.

2. Classification with Resnet50 vs MobileNetv2

2.1 Classification:

Classification is a kind of supervised machine learning in which the algorithm learns from the data and then applies what it has learnt to categorize fresh observations. To put it another way, the training dataset is used to get better boundary conditions that may be used to identify each target class; once these boundary requirements have been identified, the target class is predicted. Binary classifiers work with just two possible outputs or classifications (for example, positive or negative sentiment, etc.). We provide a binary neural network classifier for correctly wearing a face mask in this paper**.**

2.2 Network architecture:

2.2.1 Network architecture of Resnet50:

ResNet [27]is another exemplary deep neural networks. Prior to ResNet, the most sophisticated DNN (deep neural network) grew in complexity. Deep neural networks, on the other hand, are famously hard to train owing to issue of vanishing gradients: when backpropagating to earlier layers, repeated multiplication may cause the gradient to become endlessly small. In ResNet, you could have up to 152 levels of depth by making use of a skip connection (also known as a shortcut connection). In the ResNet implementation, there are two sorts of shortcut modules. At the shortcut layer, the initial block is an identity block with no convolution layer. The input and output dimensions are the same in this instance. The second is a convolution block with a shortcut for the convolution layer. The input dimensions are less than the output dimensions in this scenario. In both blocks, 1 x 1 convolution layers are added to the network at the start and finish. This technique, known as bottleneck design, reduces the amount of network parameters while maintaining network performance. One of the ResNet versions is ResNet-50, a 50-layer convolutional neural network. A total of 48 Convolution layers, one MaxPool layer, and one Average Pool layer are present. The ResNet-50 architecture is shown in detail in the figure below. ResNet is a deep residual learning framework. Even with very complex neural networks, the issue of disappearing gradients is resolved. Resnet-50 contains over 23 million trainable parameters, which is much less than previous designs despite having 50 layers. Consider a neural network block with x input for which we want to discover the true distribution H (x). The difference (or residual) will be denoted as follows:

$$R(x) = \text{Output} - \text{Input} = H(x) - x$$

Rearranging it, we get

$$H(x) = R(x) + x$$

The layers are learning the residual, R, since we have an identity relationship owing to x. (x). A typical network's layers learn the real output (H(x)), but a residual network's layers learn the residual (R(x)). Learning the residual of the output and input rather than just the input has also been discovered. Because they are bypassed and add no complications to the architecture, activation functions from previous layers may be reused in the identity residual model.

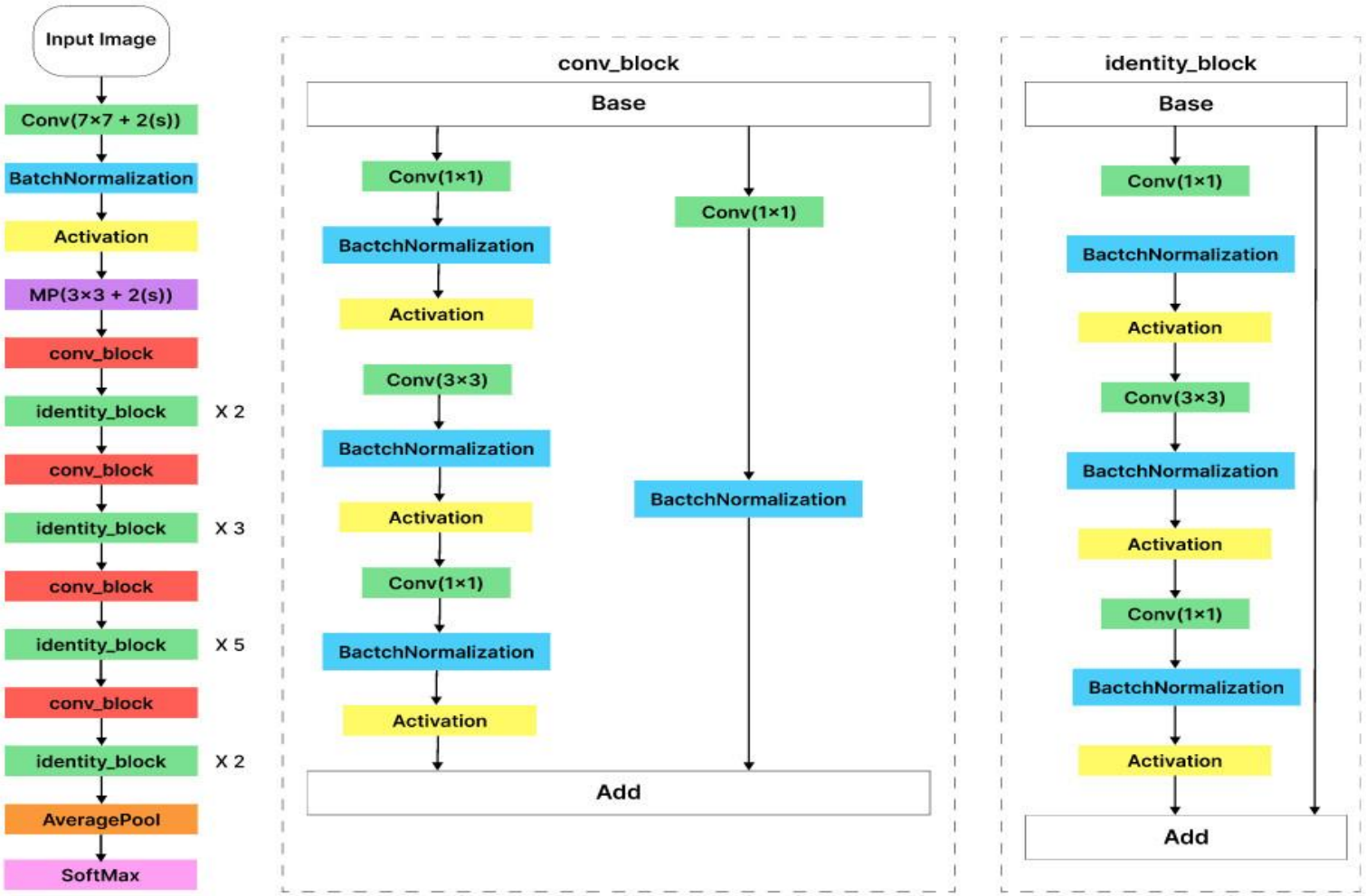
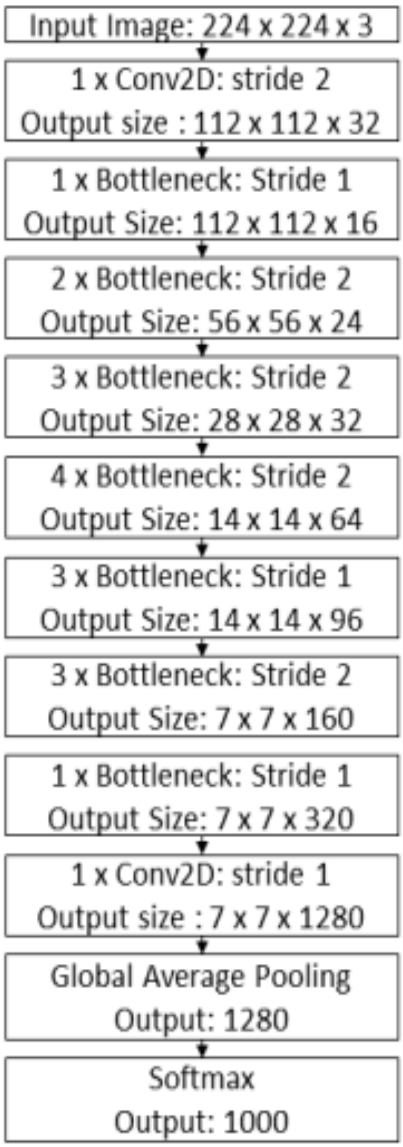


Figure 1: ResNet 50 architecture

2.2.2 Network architecture of MobileNetV2:

MobileNetV2 [28] is a depth-separable convolutional architecture for basic blocks with residuals that is bottleneck-tolerant. As seen in Figure 2, there are two types of blocks. Each block has three tiers. The first is 1x1 convolutions using "ReLU6." The second layer employs depth-wise "convolution," whereas the 3rd layer employs non-linear 1x1 "convolution." The first stratum is a one-stride residual block. As illustrated in Table 1, the 2nd layer is a residual block with stride two that is used for shrinking. Classification is used to determine the input class, while regression is used to update the bounding-box. The majority of detection backbone networks are networks for classification, with the exception of the last fully connected layers. In order to perform object recognition tasks, the backbone network receives images as input and produces feature maps for each.



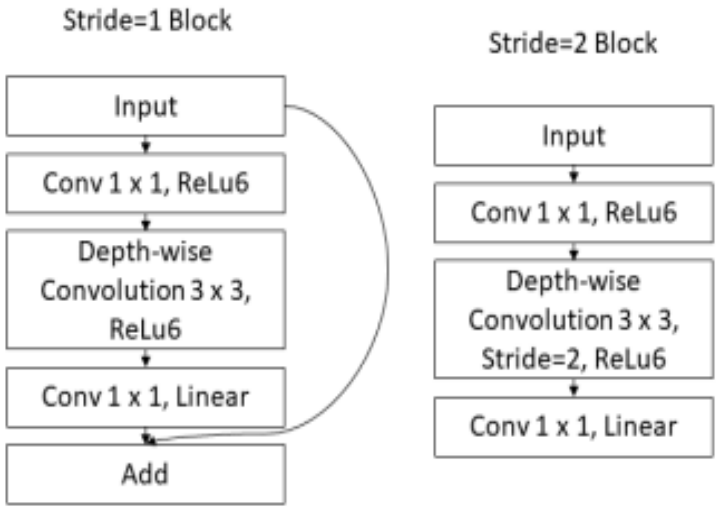


Figure 2 MobileNet-V2 Architecture and Bottleneck Stride Blocks

Input	Operator	Output
"h x w x k"	1 x 1 "conv2d", "ReLU6"	"h x w x tk"
"h x w x tk"	3 x 3 "dwise" S=s, "ReLU6"	$\frac{h}{s}$ x $\frac{w}{s}$ x tk
$\frac{h}{s}$ x $\frac{w}{s}$ x tk	Linear 1 x 1 conv2d	$\frac{h}{s}$ x $\frac{w}{s}$ x k'

Table 1 Residual block in the k-> k' channel

Typically, to extract feature maps with high-quality classification problems, well-established training approaches are used. The basic model refers to this component of the model. The "image net" weights of the MobileNetV2 network are used as the foundation for this model. Hundreds of thousands of images have been used to train ImageNet, an image database., making it incredibly effective for categorizing. images. During training, the evaluated "bounding boxes" are compared to the "ground truth boxes," and the trainable parameters are adjusted accordingly during backpropagation. There are kernels that are utilized in each feature space to produce results that include matching scores for each pixel and the appropriate bounding-box dimensions for each object. The classifier and the basic model make up MobileNet. The basic model is reused, the head is trimmed, and there are two layers that are entirely linked.

2.3 Steps in Building the Model

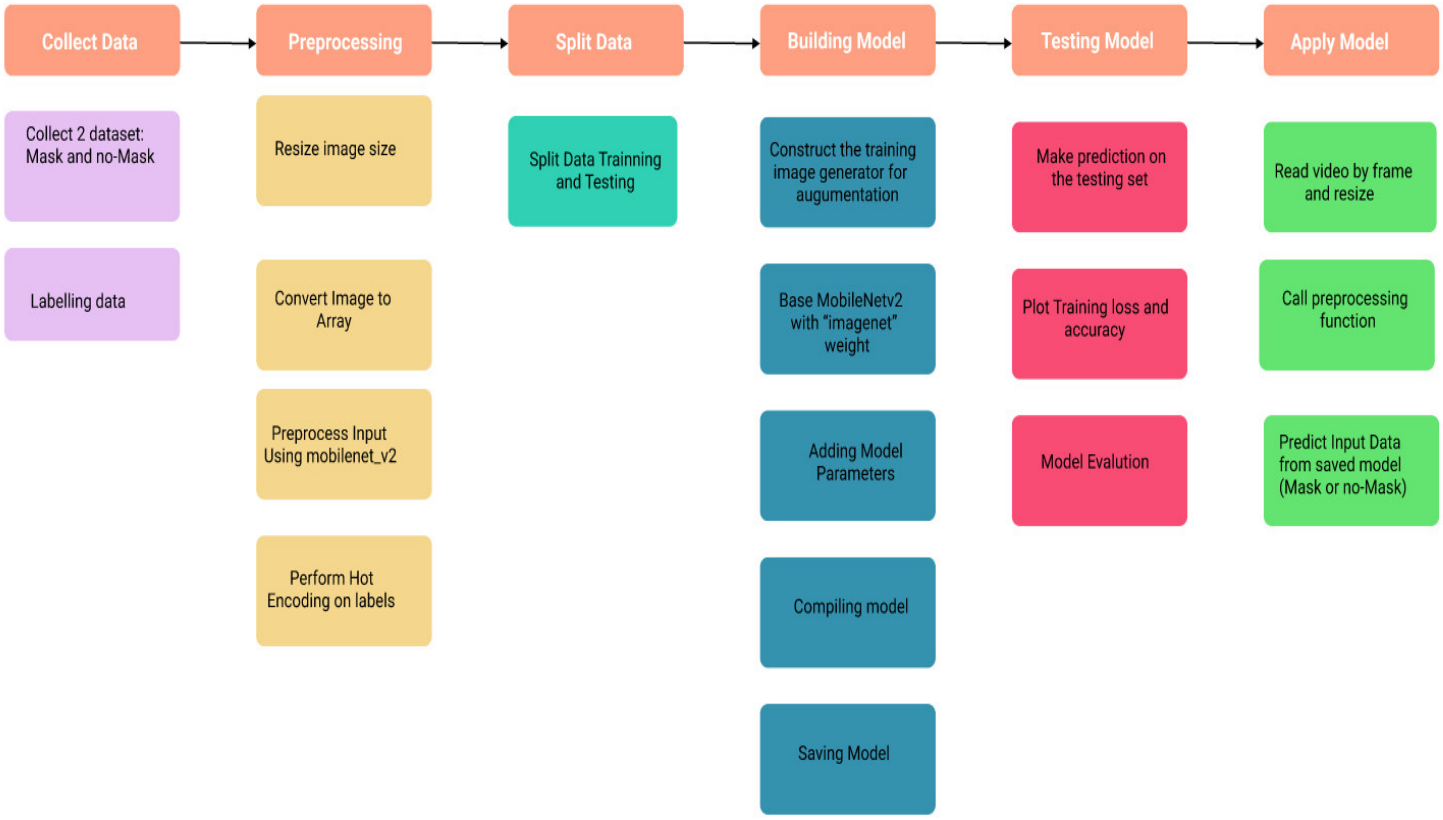


Figure 3 Steps in building model using Resnet50 and MobileNetv2

2.3.1 Data Collecting:

This Face Mask Detector model is based on data from Kaggle. People with masks and those without mask are both included in the dataset. The model will distinguish between persons wearing masks and those who aren't. 2.165 data with-mask and 1.930 data without-mask are used to build the model. The image is cropped at this point until the only viewable object is the object's face. With and without a mask image, the collected data are classified.. The following is an example of data:

Figure 4: Data with-mask (Data using for Resnet50 and MobileNetv2)

Figure 5 Data without-mask (Data using for Resnet50 and MobileNetv2)

2.3.2 Preprocessing Preprocessing:

Preprocessing data is required before it can be used for training and testing.. Scaling the image size, converting the image to an array, preprocessing input using MobileNetV2 (or Resnet50), and using hot encoding on labels are the four processes in the preprocessing. Picture scaling is an important preprocessing step in CV because of the efficacy of training models. We employed Convolutional Neural Networks, hence the dataset pictures were shrunk to 224224 pixels. The next step is to generate an array from the whole dataset's photos. A loop function is used to turn the picture into an array. The picture will then be used to preprocess input using MobileNetV2 (or Resnet50). Because many ML algorithms can't operate directly on data labeling, this phase's last step is to employ hot encoding on labels. All of the input variables and output variables, including this algorithm, must be numeric. So that the algorithm can interpret and analyse the data, the labeled data will be transformed to a numerical label.

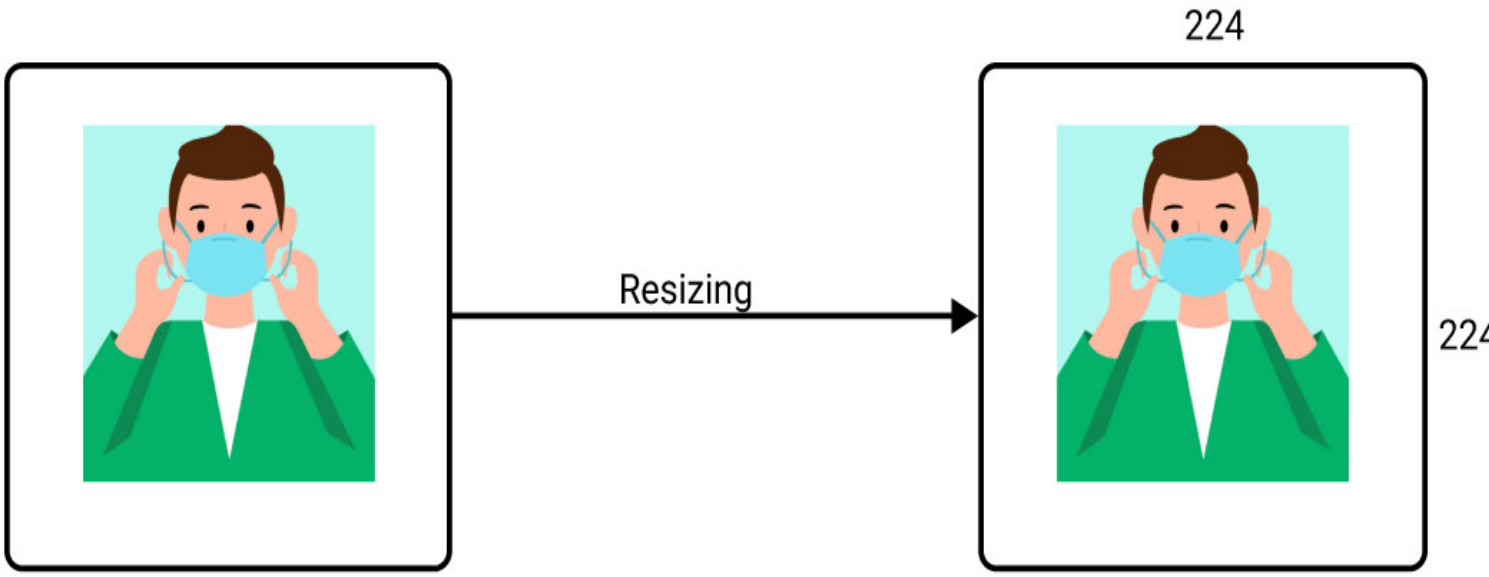


Figure 6: A diagram to show the resizing of an image

2.3.3 Split the Data:

Following the preprocessing step, the data is divided into two files: training (80%), and testing data (the remaining 20%). Images with and without masks are included in each batch

2.3.4 Building model:

The model is created in six steps: building the augmented training image generator, constructing the base model using MobileNetV2 (or Resnet50), adding model parameters, compiling, training the model, and lastly storing the model for future processes.

2.3.5 Testing model:

There are procedures in testing the model to ensure that it can predict effectively. Making predictions about the testing set is the first stage.

2.3.6 Applying face mask detection model:

A deep learning framework - Caffe that Berkeley AI Research (BAIR) and other collaborators built and maintain for the community as a quicker, more powerful, and more effective alternative to current object identification algorithms. The network architecture and the res10 300x300 SSD iter 140000 are described in the deploy.prototxt file. Caffemodel is the model that contains the layer weights. Two parameters are accepted by the cv2.dnn.readNet method ("path/to/prototxtfile" and "path/to/caffemodelweights"). After the model has been trained, it may be used to recognize face masks in both static photos and real-time video streams. After reading the movie frame by frame, the face detection algorithm is applied. The procedure continues if a face is discovered. On detected frames with faces, reprocessing will be conducted, including shrinking the picture size, converting to an array, and preprocessing the input using MobileNetV2 (or Resnet50). The next step is to use the stored model to predict input data. With the aid of a previously established model, predict the input picture that has been processed. In addition, whether or not the individual is wearing a mask, as well as the prediction %, will be indicated on the video frame.

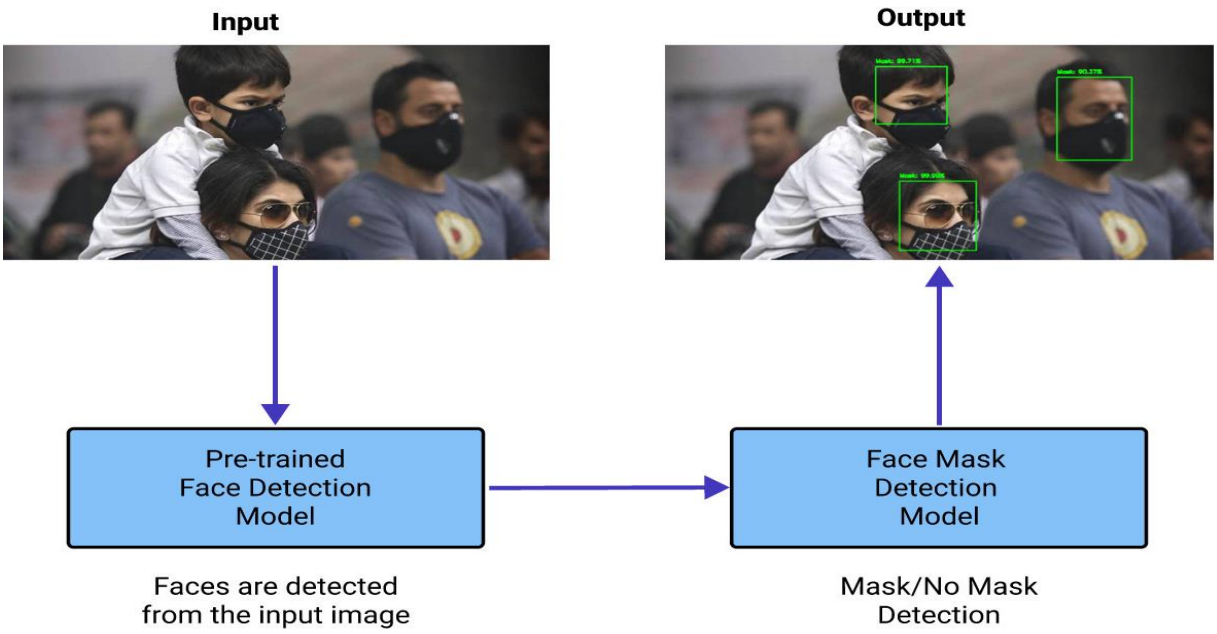


Figure 7: Diagram showing the implemented face mask detector model

2.4 Performance metrics:

Classification models are used to predict the target class of a data sample in classification problems. The classification model predicts the proportion that each event falls into one of two categories. Before these models can be used to solve real-world problems, they must be evaluated for performance. ML classification models' performance measures are used to evaluate how well they function in a given context. These performance metrics include accuracy, precision, recall, and F1-score.

2.4.1 Accuracy:

The ratio of true positives and true negatives to all positive and negative data is the model accuracy. It's a metric that measures how well machine learning classification models perform. In other words, accuracy refers to the likelihood that our machine learning model would accurately predict a result based on the total number of predictions it has made. Although the accuracy rate is impressive, it informs us nothing about the errors our machine learning models make when working with new data.

- Accuracy = $\frac{Tp + Tn}{Tp + Tn + Fp + Fn}$

Tp stands for true positive, Tn for true negative, Fp for false positive, and Fn for false negative in the example above.

2.4.2 Precision:

To find out how many of the model's correctly predicted labels are in fact correct, look at the precision score. Positive predictive value (PPV) is another name for precision. False positives and false negatives are traded off using precision and recall. The distribution of classes affects precision. The precision will be reduced if there are more samples in the minority class.

- Precision = $\frac{Tp}{Tp + Fp}$

2.4.3 Recall:

The model recall score reveals how well the model can distinguish true positives from false positives. It assesses how effectively our ML model correctly recognizes all true positives in a dataset. The higher the recall score, the better the ML model can distinguish between positive and negative samples. Recall is also known as sensitivities or the percentage of true positives. A high recall score suggests that the model is effective at detecting positive instances. In contrast, a low recall score suggests that the model is bad in identifying positive examples. Recall is often used in combination with other performance indicators, like as precision and accuracy, to provide a comprehensive understanding of the model's performance. It represents, mathematically, the proportion of true positives to the total of true positives and false negatives.

- Recall = $\frac{Tp}{Tp + Fn}$

2.4.4 F1-score:

Accuracy and recall are represented by the model F1 score, which is the model's overall score. ML model performance statistic F1-score weighs Precision and Recall equally, giving it a viable alternative to accuracy measurements. To convey high-level information about the model's output quality, it's often expressed as a single numerical value.

- F1 = $\frac{2 * (Precision * Recall)}{Precision + Recall}$

3. Object detection with YOLOv5

3.1 Object detection:

Object detection is one of the most fundamental challenges in computer vision research. The primary goal is to find interesting objects in the image. It is necessary to know the specific categories of each item as well as the bounding box of the object. Thanks to the ongoing development of deep learning methods in recent years, object identification algorithms have

become one of the research hotspots in the area of deep learning. They're employed in a variety of real-world settings, including intelligent video surveillance, car automated driving, and robot environmental perception.

Based on the head in different object detectors, object detection models may be separated into two categories: one-stage object detectors and two-stage object detectors. The most common two-stage object detector is the R-CNN series. The most common one-stage object detection models are YOLO and SSD. We used a YOLOv5 model to train a YOLOv5 model to determine whether persons are wearing masks or not in this thesis.

3.2 Network architecture of YOLOv5

YOLO is a real-time object detector that is state-of-the-art, and YOLOv5 is built on YOLOv1 and YOLOv4. It has attained top-tier performance on two official object identification datasets-Pascal VOC and Microsoft COCO -as a consequence of continual improvements. The YOLOv5 system's network architecture is shown in the diagram below. It was decided that YOLOv5 would be first learner for a variety of reasons. Through the use of CSPNet, YOLOv5 built CSPDarknet, which serves as the backbone of Darknet. CSPNet reduces the model's parameters and FLOPS (floating-point operations per second) to enhance inference speed and accuracy while simultaneously decreasing the model's size by embedding gradient changes into the feature map. It is essential that object be detected quickly and accurately, and a model's compact size impacts its inference efficiency on resource-constrained edge devices. A route aggregation network (PANet) is used as the YOLOv5's neck to improve data flow. An innovative feature pyramid network (FPN) design with an expanded bottom-up route is used by PANet.

YOLOv5 stands out from the others. PyTorch is used instead of Darknet. The backbone of the network is CSPDarknet53. Repetitive gradient information in large backbones is tackled by including gradient change into the feature map, decreasing inference time, enhancing accuracy and reducing model size by reducing the number of parameters in the backbone model. Route aggregation networks (PANet) are used as a bottleneck to improve the flow of data. There are several levels in the PANet's new feature pyramid network (FPN), including bottom-up and top-down. As a consequence, low-level model attributes propagation is enhanced. PANet improves the accuracy of object localization by enhancing lower-level localization. Furthermore, the YOLOv5 head is similar to the YOLOv4 and YOLOv3 heads, resulting in three separate feature map outputs enabling multi-scale prediction. It also contributes to the efficient prediction of tiny to big items in

the model. The image is given to CSPDarknet53 for feature extraction before being passed again to PANet for feature fusion. The YOLO layer produces the final results. The architecture of the YOLOv5l algorithm is seen in the Figure below. The focusing layer is derived from the YOLOv3 structure. It replaces the first three YOLOv3 layers with a single YOLOv5 layer. In addition, Conv indicates a convolution layer. C3 is comprised of 3 convolution layers and a module whose bottlenecks are cascaded. Spatial pyramid pooling (SPP) is a pooling layer designed to eliminate the network's fixed size limitation. For layer fusion, upsampling is utilized to upsample the preceding layer at the closest node. As the name implies, Concat acts as a cutting layer for the prior one. Conv2d's network head detection modules round out the set.

The fundamental distinction between the architectures of YOLOv3, YOLOv4, and YOLOv5 is that YOLOv3 employs the Darknet53 backbone. The backbone of the YOLOv4 architecture is CSPdarknet53, whereas the backbone of the YOLOv5 architecture is Focus. The benefits of adopting a Focus layer are lower CUDA memory requirements, fewer layers, and improved forward and backpropagation.

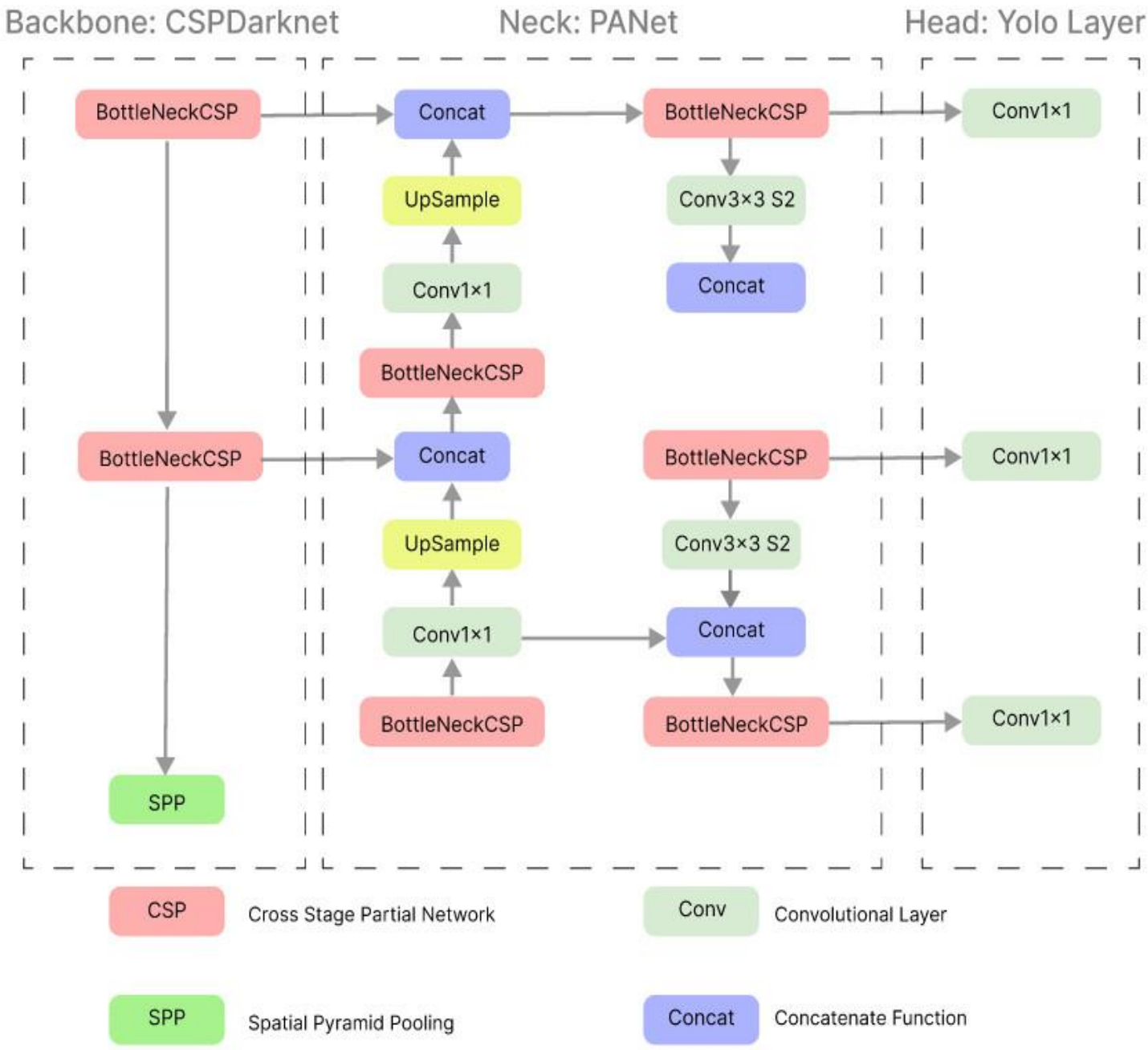


Figure 8: The Architecture of YOLOv5. It is divided into 3 parts: The backbone is CSPDarknet, the neck is PANet, and the head is YOLO Layer. Before being fed into PANet for feature fusion, the data is sent to CSPDarknet for feature extraction. Lastly, Yolo Layer then produces detection findings (class, score, location, size)

3.3 Steps in building the model:

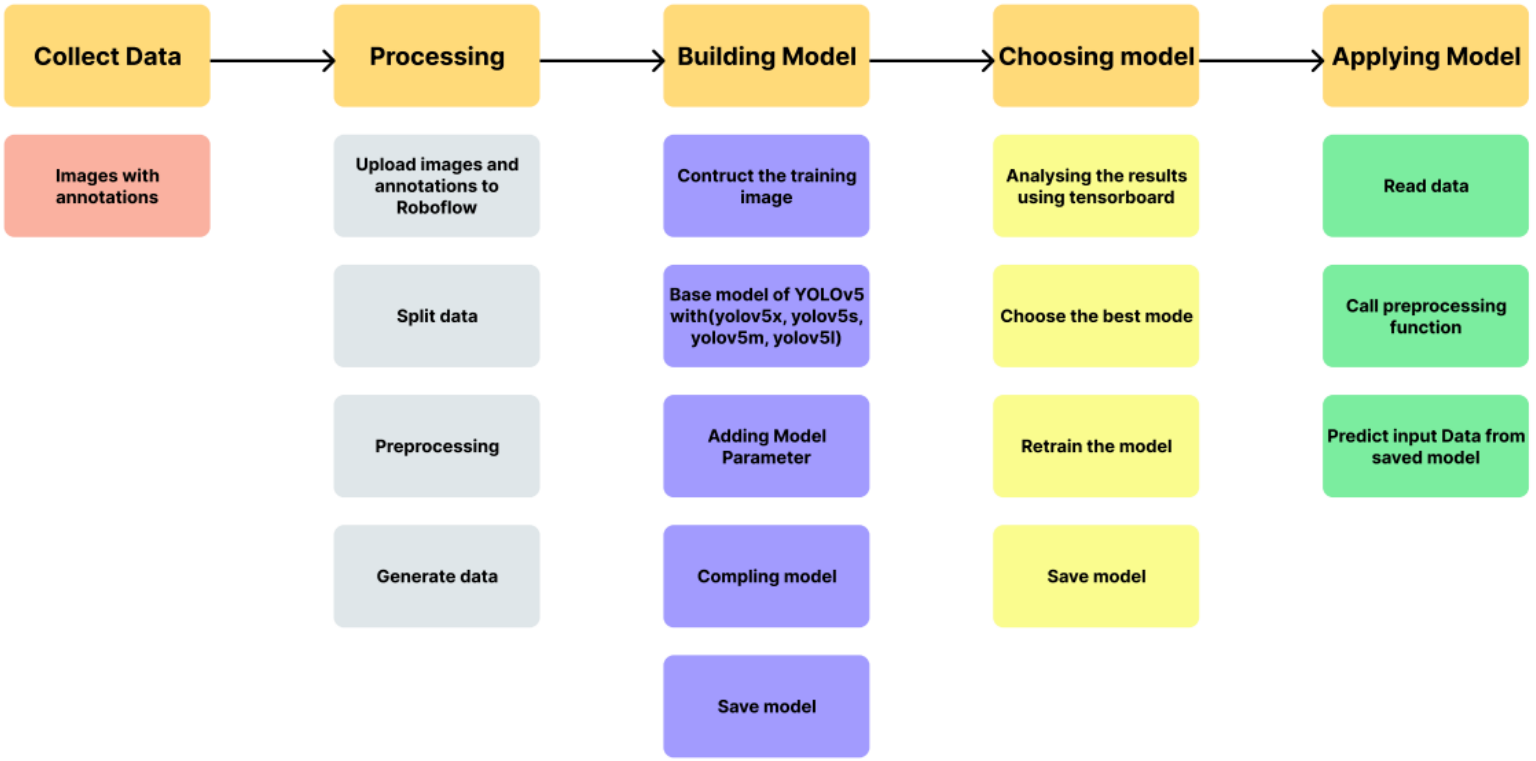


Figure 9: Steps in building model using YOLOv5

3.3.1Collecting data: dataset (with 2 folder images, annotations)

In the absence of immunization, masks are one of the only feasible defenses against COVID-19, and they play a critical role in protecting human health from respiratory illnesses. It is possible to create a model that can detect people who are wearing masks, not wearing masks, or wearing masks improperly using this dataset. This collection contains 853 pictures from the three courses, together with their bounding-box in PASCAL VOC format. There are three types of masks: with -mask, without-mask, and incorrectly- mask. However, we only utilize two of the three classes in this thesis: with mask and without mask.

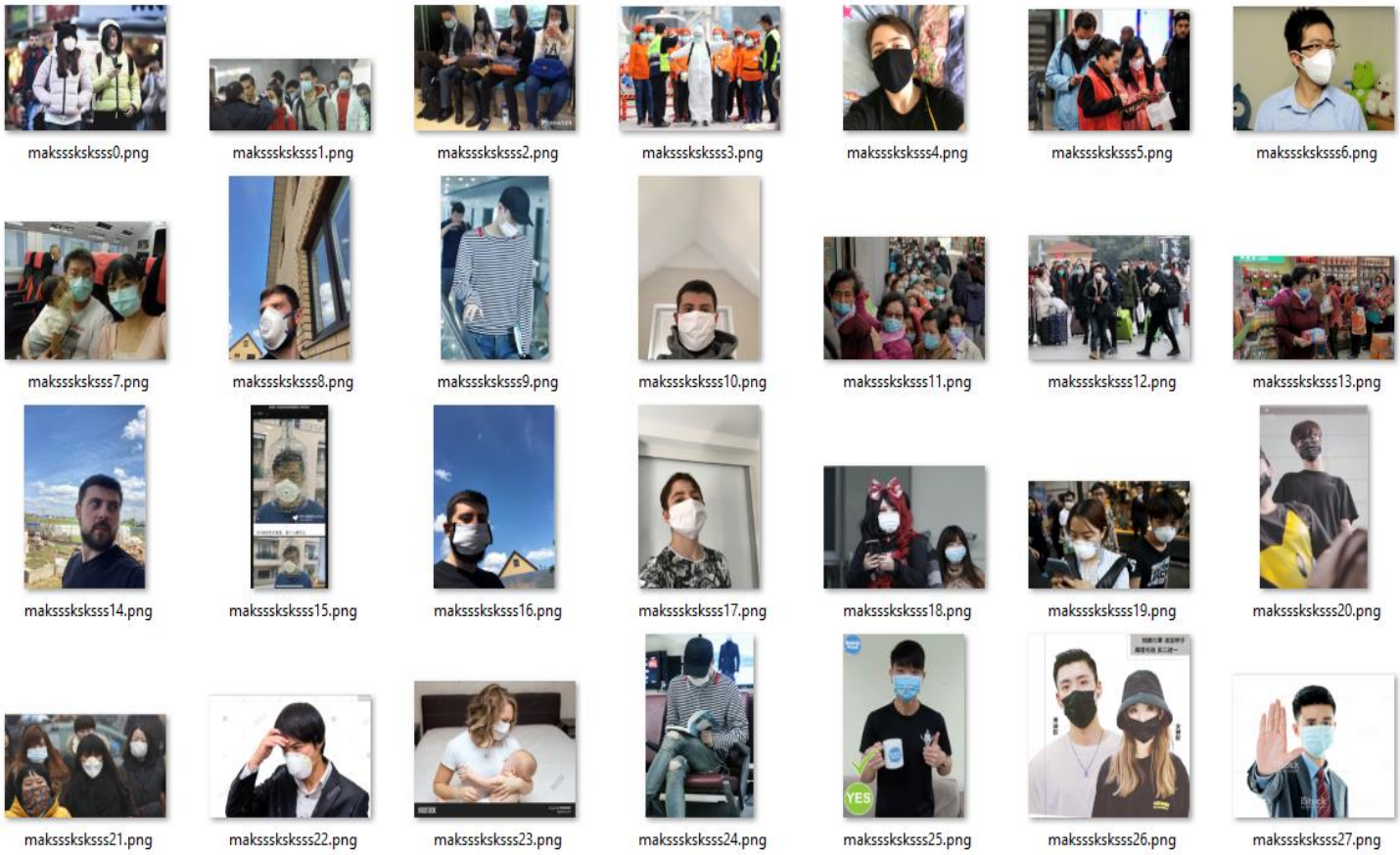


Figure 10: Data of YOLOv5 (Images)

maksssksksss0.xml	5/22/2020 7:18 AM	XML Document	2 KB
maksssksksss1.xml	5/22/2020 7:18 AM	XML Document	4 KB
maksssksksss2.xml	5/22/2020 7:18 AM	XML Document	2 KB
maksssksksss3.xml	5/22/2020 7:18 AM	XML Document	4 KB
maksssksksss4.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss5.xml	5/22/2020 7:18 AM	XML Document	2 KB
maksssksksss6.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss7.xml	5/22/2020 7:18 AM	XML Document	2 KB
maksssksksss8.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss9.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss10.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss11.xml	5/22/2020 7:18 AM	XML Document	6 KB
maksssksksss12.xml	5/22/2020 7:18 AM	XML Document	5 KB
maksssksksss13.xml	5/22/2020 7:18 AM	XML Document	3 KB
maksssksksss14.xml	5/22/2020 7:18 AM	XML Document	1 KB
maksssksksss15.xml	5/22/2020 7:18 AM	XML Document	1 KB

Figure 11: Data of YOLOv5 (Annotations)

The data of Resnet50 or MobileNetv2 described above is a dataset consisting of only faces divided into 2 folders (with_mask and without_mask). Meanwhile, YOLOv5's dataset also includes 2 folders, 1 folder for images and 1 folder for annotations corresponding to images. In YOLOv5's Image Dataset, an image may include one or more people, including both masked and non-masked people.

3.3.2 Processing:

Upload images and annotation to Roboflow:

We drag and drop both the images folder and annotations folder at the same time to Roboflow

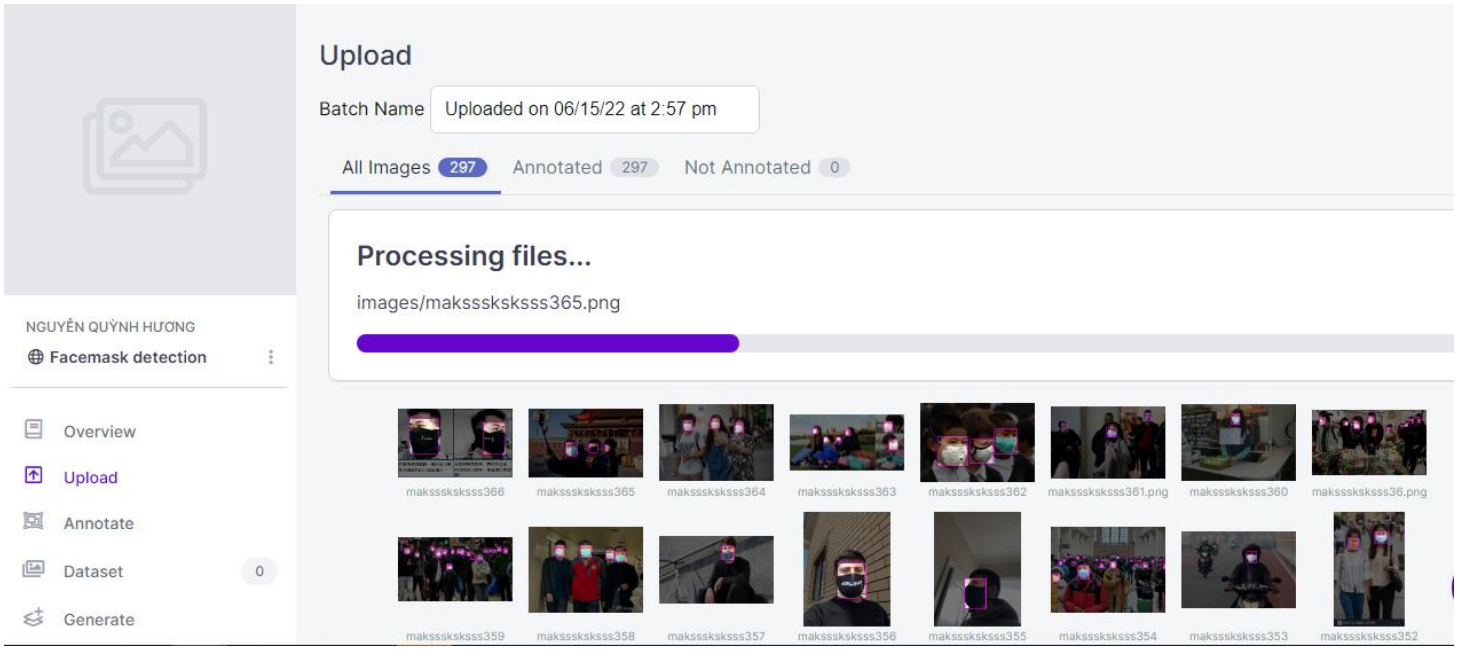


Figure 12: Upload images and annotation to Roboflow

Split Data: After we upload data, we will split dataset into Train/Valid/Test

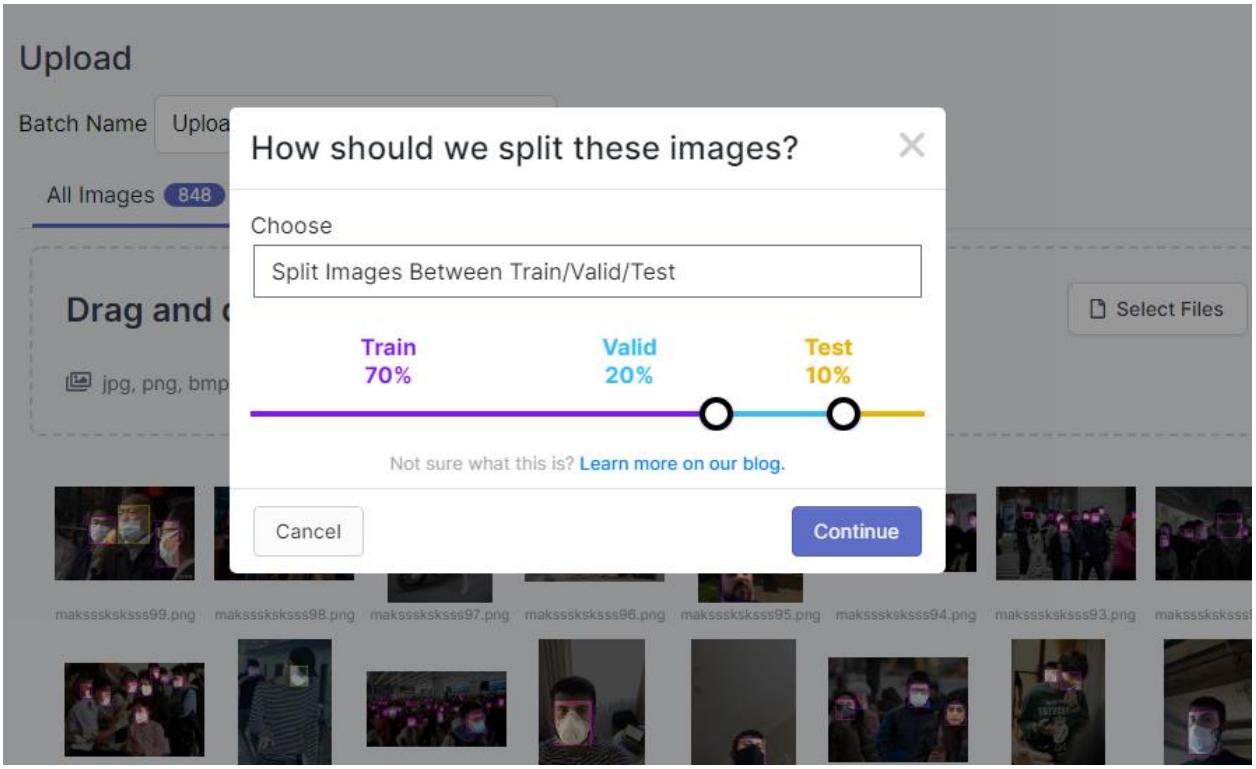


Figure 13: Split Data step with Roboflow

3.3.3 Preprocessing:

In this step, because we just used 2 classes, with_mask and without_mask, I chose to modify class options to drop 1 class, mask_weared_incorrect. We also resized the image to 416x416 pixels.

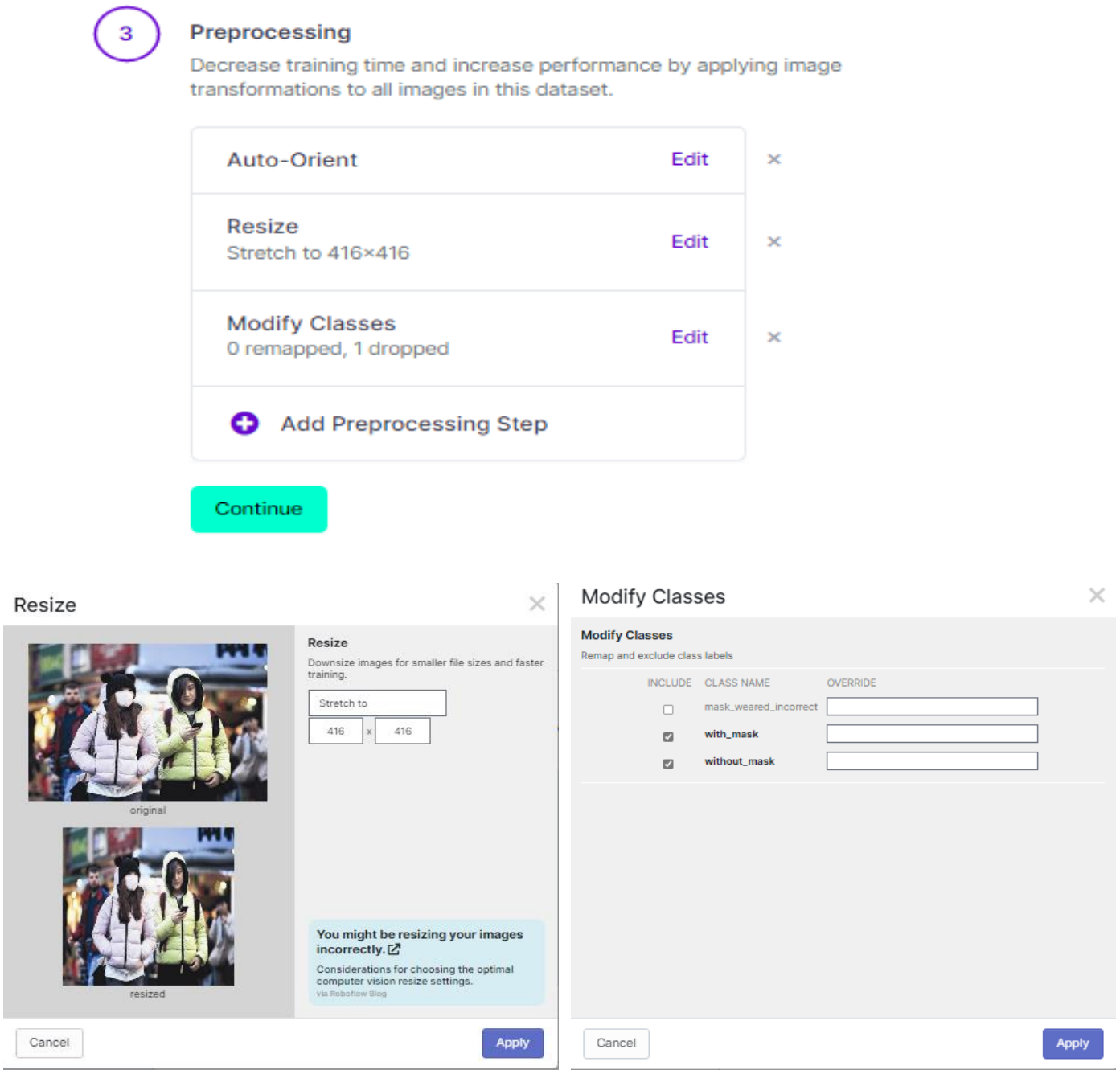


Figure 14: Preprocessing step

3.3.4 Generating data:

Because the initial dataset is images with PASCAL VOC-formatted bounding boxes. Therefore, for using this dataset for YOLOv5 generated new data with YOLOv5 Pytorch format as figure beblow:

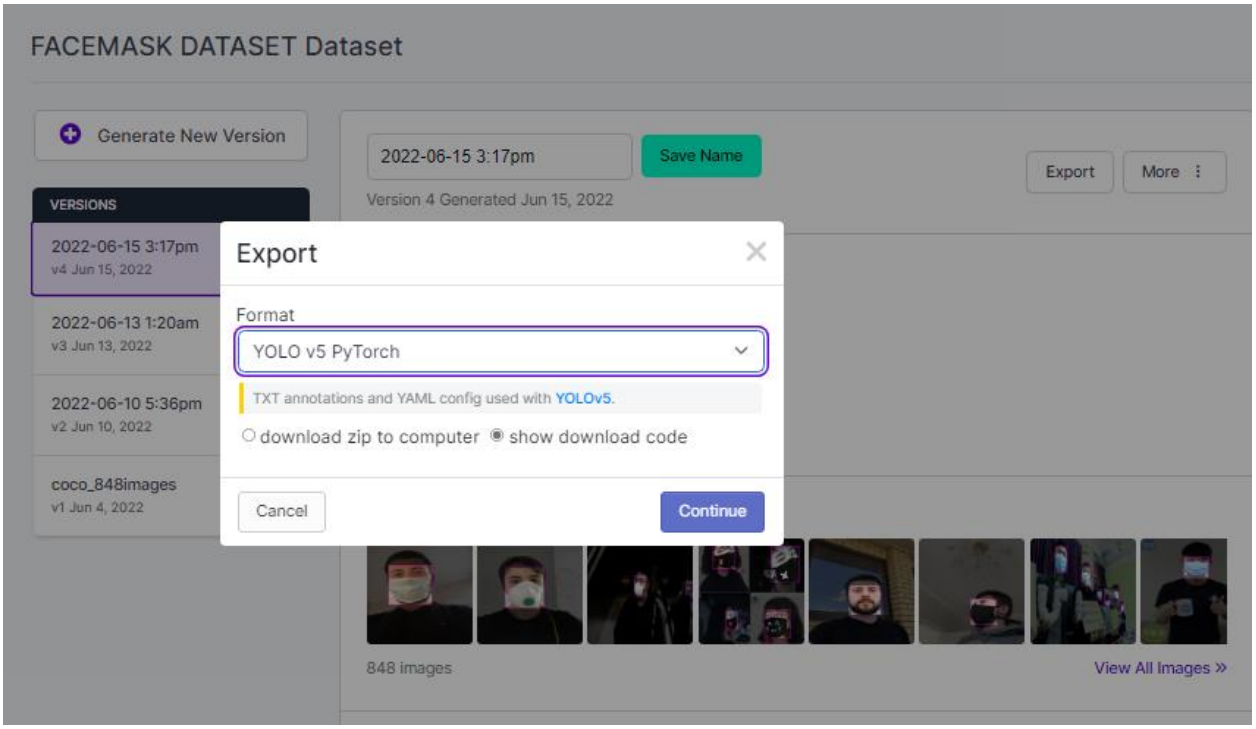


Figure 15: Generating data step

3.3.5 Building Model:

The model is built in six steps: producing the training image, using YOLOv5 to create the base model, adding model parameters, compiling the model, training the model, and lastly storing the model for use in the next process.. In this step, we have trained the model with four weights: YOLOv5s, YOLOv5m, YOLOv5l,YOLOv5x.

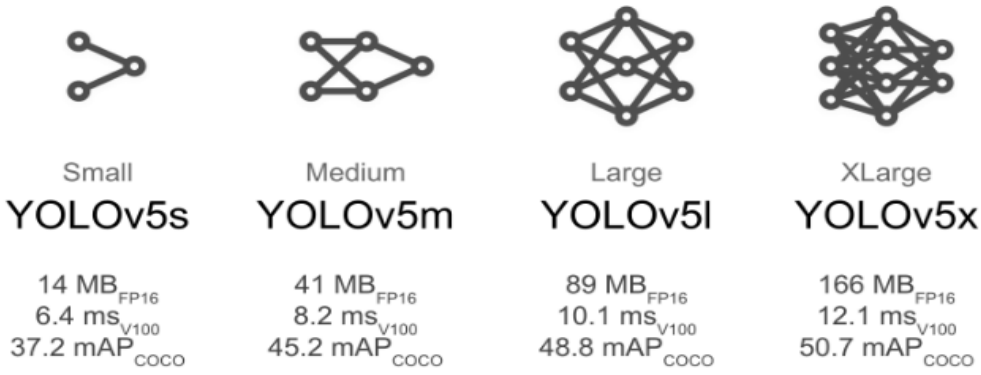


Figure 16: Four weights of YOLO: YOLOv5s, YOLOv5m, YOLOv5l,YOLOv5x

.

YOLOv5s: It is the small model in the family, with about 7.2 million parameters, and is suited for CPU-based inference.

YOLOv5m: With 21.2 million parameters, this is a medium-sized model. It's probably the best model for a wide range of datasets and training since it attempts to reach a good balance between precision and quickness.YOLOv5l: The YOLOv5 family's large model, with 46.5 million parameters. It's useful for datasets requiring the detection of tiny things.

YOLOv5x: It is the biggest of these models, as well as having the greatest mAP. Although it is slower than the others and includes 86.7 million parameters.

3.3.5Choosing model:

After the previous step, we used tensorboard to analyze the results and chose the model with the best performance. Then, we trained the chosen best model with the same weight as in the previous step, but with a higher number of epochs. Lastly, saving the model after retraining it for use in future prediction processes.

3.3.6 Applying the model:

After training, the model may be used to detect face masks in both still images and streaming video. In order to identify faces, the video is read frame-by-frame. The process continues if a face is discovered. The stored model will be used to predict the new input data. Predict the output picture of a previously stored model that has been applied to the input image. On the video frame, the prediction % and whether or not the object is wearing mask will be labeled.

3.4 Performance metrics

We predicted class labels using classification, where the model returns a single label that is either correct or incorrect. This kind of binary classification simplifies calculation accuracy; nevertheless, object detection is not that easy. A perfect match between predicted and ground-truth bounding boxes is unachievable due to the changeable model parameters. Object detection systems like YOLO are evaluated using the mean average precision (mAP). By comparing the ground-truth bounding box to the detected box, the mAP generates a score. The model's detection accuracy improves as the score increases. We looked at the confusion matrix, accuracy, precision, and recall score in detail in 2.4(chater 3). We'll next continue look at how the mAP is calculated using accuracy and recall.

3.4.1 Precision - Recall curve

For sake of illustrating the tradeoff between precision and recall values at certain thresholds, a precision-recall curve exists. As a result of this curve, the best threshold for optimizing both measurements may be determined. Keep in mind that as recollection increases, accuracy decreases.

The accuracy of accurately recognizing each sample decreases as the number of positive samples rises . This is to be expected, as the model is more likely to fail when there are a lot of samples.

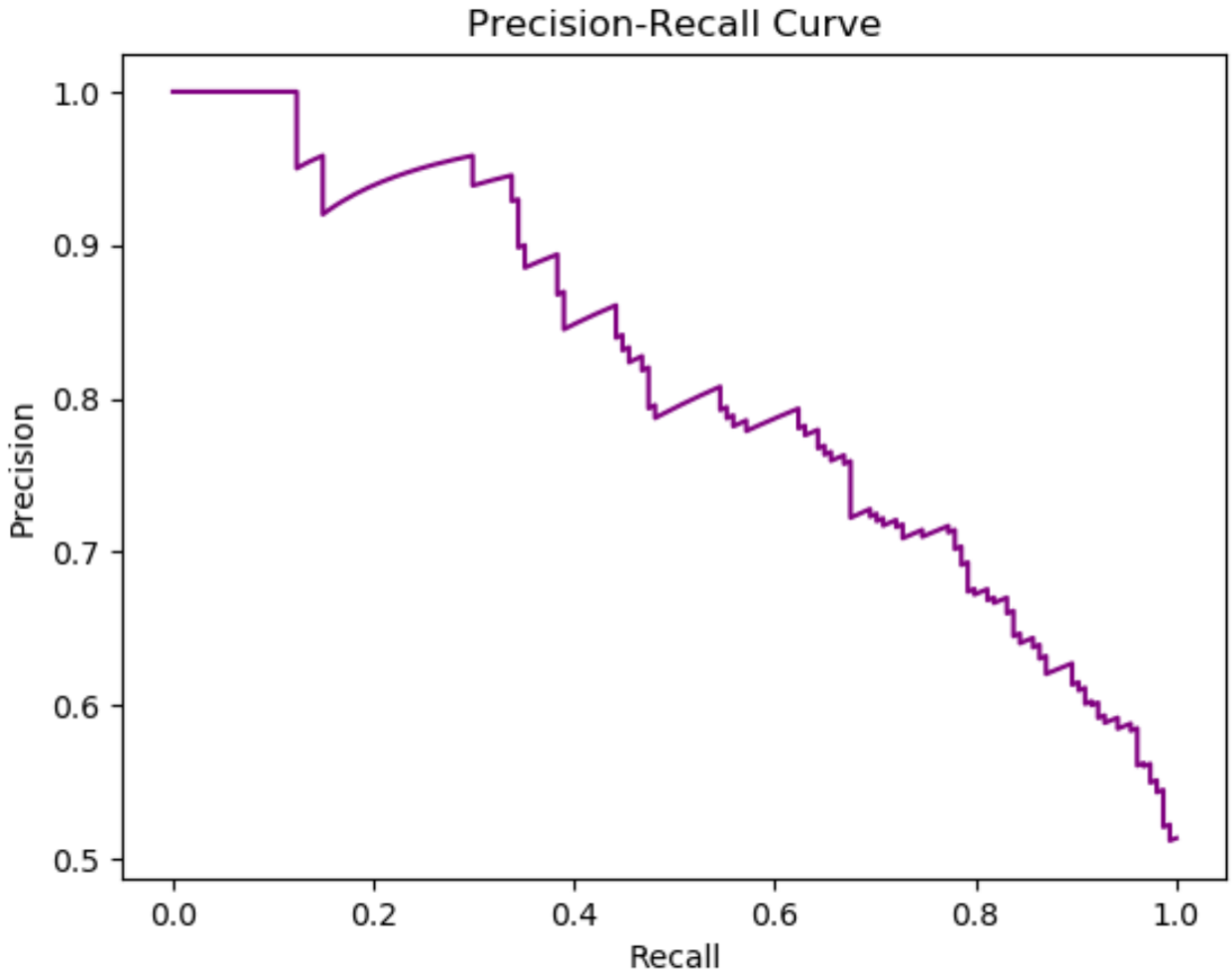


Figure 17: Example of Precision-Recall Curve plot (PR plot)

The precision curve makes determining the point where precision and recall are both high easily.

3.4.2 F1-score

Due to the simplicity of the curve, it may be possible to utilize the previous graph to identify the optimum precision and recall values. Utilizing a measure known as the f1 score is a more efficient method. The f1 metric evaluates the stability between precision and recall. When f1 is high, accuracy and recall are also high. A lower f1 score means a larger imbalance between precision and recall.

3.4.3 Intersection over Union (IoU)

mAP is computed using IoU. A number between 0 and 1 that represents the degree of overlap between the expected and ground-truth bounding boxes. IoU is an essential accuracy metrics to monitor while gathering human annotations.

This is the definition of IoU:This is how IoU is defined:

$$IoU = \frac{area(P_b \cap G_b)}{area(P_b \cup G_b)}$$

Pb and Gb stand for predicted and ground-truth boxes, respectively. Also, the intersection and union areas of the two boxes are denoted by area(Pb ∩ Gb) and area(Pb ∪ Gb), respectively.. In this situation, if IoU >= 0.5, it is a match; otherwise, it is not. The next section presents how to apply IoUs to determine the mean average precision (mAP) for an object detection model

3.4.4 Average precision and Mean average precision :

To sum up the accuracy-recall curve into a single metric, we may use the average precision The difference between the current recall and subsequent recalls is computed and then multiplied by the current precision using an iterative loop that iterates through all precisions/recalls. It is also possible to formulate the AP as the summing weighted precisions at each threshold, where weight represents an increase in recall. AP is calculated using the following formula.:

$$AP = \sum_{k=0}^{k=n-1} [Recall(k) - Recalls(k+1)] * Precisions(k)$$

$Recalls(n) = 0, Precisions(n) = 1$
$n = Number\ of\ thresholds.$

The average precision is represented by the mAP measure. As a result, the average AP score for each class is calculated. However, they might have the same meaning in some contexts. The mean average precision may be calculated by combining predicted object detections with ground-truth object annotations .Regarding each ground truth box in the image, IoU is computed for each prediction. These are the thresholds (often 0.5 to 0.95) and a greedy method is used to match predictions to ground-truth boxes. mAP is defined:

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

$AP_k = the\ AP\ of\ class\ k$
$n = the\ number\ of\ classes$

CHAPTER 4: IMPLEMENTATION AND RESULTS

1. Implementation and results of MobileNetV2 and Resnet50_v2 model

1.1 Training result of MobileNetV2 model and Resnet50 model

1.1.1 Training result of MobileNetV2 model and Resnet50 model

Epoch	Loss	Accuracy	Val_loss	Val_acc
1/20	0.3776	0.8513	0.1326	0.9805
2/20	0.1383	0.9633	0.0794	0.9841
3/20	0.0892	0.9772	0.0653	0.9829
4/20	0.0800	0.9772	0.0564	0.9841
5/20	0.0641	0.9818	0.0548	0.9853
6/20	0.0612	0.9787	0.0565	0.9853
7/20	0.0512	0.9824	0.0468	0.9853
8/20	0.0509	0.9824	0.0461	0.9853
9/20	0.0470	0.9867	0.0480	0.9841
10/20	0.0420	0.9883	0.0466	0.9853
11/20	0.0457	0.9870	0.0480	0.9878
12/20	0.0374	0.9873	0.0452	0.9902
13/20	0.0398	0.9889	0.0420	0.9878
14/20	0.0315	0.9917	0.0426	0.9878
15/20	0.0397	0.9883	0.0405	0.9878
16/20	0.0398	0.9886	0.0391	0.9866
17/20	0.0333	0.9880	0.0394	0.9890
18/20	0.0300	0.9907	0.0441	0.9878
19/20	0.0309	0.9895	0.0381	0.9890
20/20	0.0343	0.9880	0.4330	0.9915

Table 2: Training iterations of MobileNetv2

We can observe from the table that beginning with the 2nd epoch, accuracy increased and loss decreased. The table represented by the plot in Figure below.

Figure 18: Training Accuracy/Loss Plot Graph of MobileNetV2

The model's accuracy may be increased with fewer iterations if the accuracy line stays steady. Model assessment is the next phase, as shown in the following table:

	precision	**recall**	**f1-score**	**support**
with_mask	0.99	1.00	0.99	433
without_mask	1.00	0.98	0.99	386
accuracy			0.99	819
macro avg	0.99	0.99	0.99	819
weighted avg	0.99	0.99	0.99	819

Table 3:Model Evaluation of MobileNet V2

1.1.2 Training result of Resnet50 model

Epoch	Loss	Accuracy	Val_loss	Val_acc
1/20	0.3772	0.8893	0.0461	0.9878
2/20	0.0905	0.9794	0.0368	0.9939
3/20	0.067	0.9831	0.0297	0.9951
4/20	0.0497	0.9871	0.0252	0.9939
5/20	0.0492	0.9864	0.0228	0.9939
6/20	0.0488	0.9877	0.0218	0.9951
7/20	0.0312	0.992	0.0211	0.9963
8/20	0.0327	0.9922	0.0207	0.9963
9/20	0.0283	0.9927	0.0202	0.9963
10/20	0.0281	0.9944	0.0194	0.9963
11/20	0.0321	0.993	0.0187	0.9963
12/20	0.0348	0.991	0.0176	0.9963
13/20	0.0218	0.9948	0.018	0.9963
14/20	0.0212	0.944	0.0182	0.9963
15/20	0.0201	0.995	0.0181	0.9963
16/20	0.0255	0.9934	0.0183	0.9963
17/20	0.0202	0.9947	0.0178	0.9963
18/20	0.0189	0.996	0.0185	0.9963
19/20	0.0155	0.9954	0.0182	0.9963
20/20	0.0201	0.9954	0.0187	0.9963

Table 4: Training iterations of Resnet50

Figure 19: Training Accuracy/Loss Plot Graph of Resnet50

	precision	**recall**	**f1-score**	**support**
with_mask	1.00	1.00	1.00	433
without_mask	1.00	0.99	1.00	386
accuracy			1.00	819
macro avg	1.00	1.00	1.00	819
weighted avg	1.00	1.00	1.00	819

Table 5

Table 5: Model Evaluation of Resnet50:

1.2 Facemask detection web application using Streamlit

1.2.1 GUI:

This is the GUI's facemask detection using Streamlit:

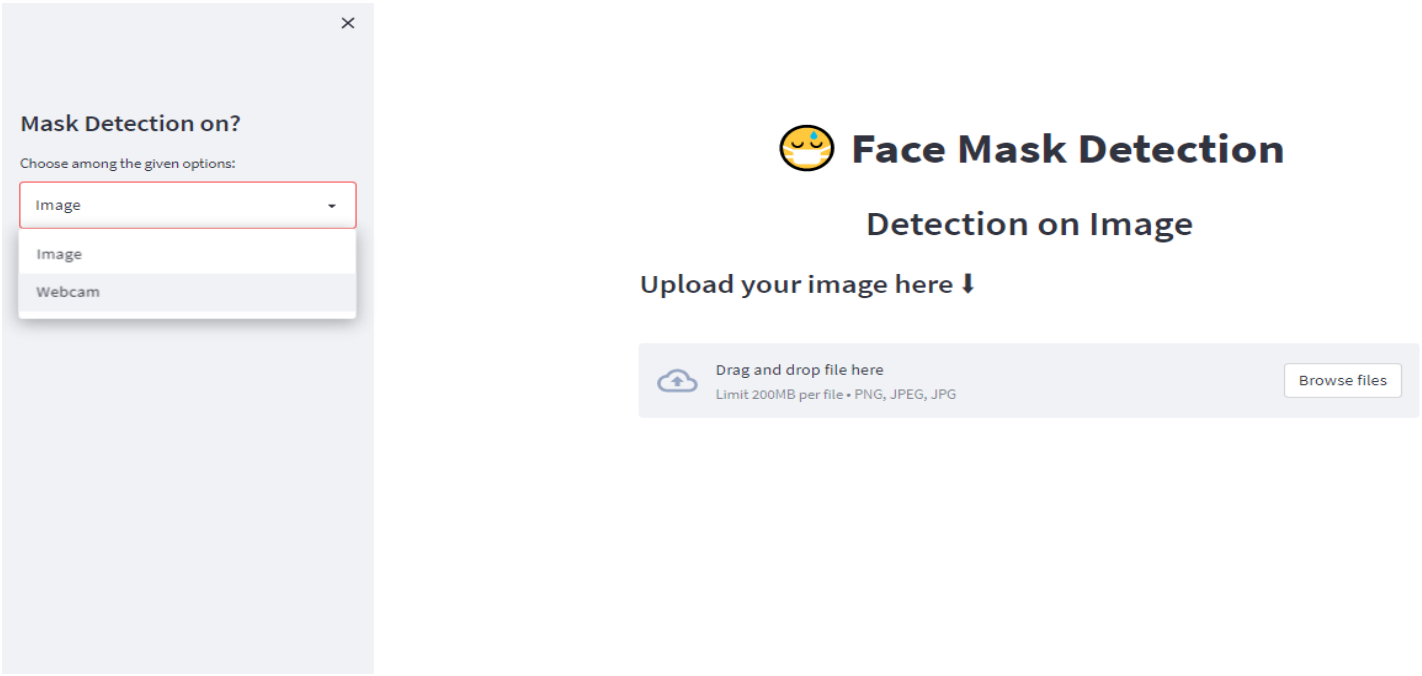


Figure 20: GUI's Face mask detection using Resnet50/ MobbileNetv2

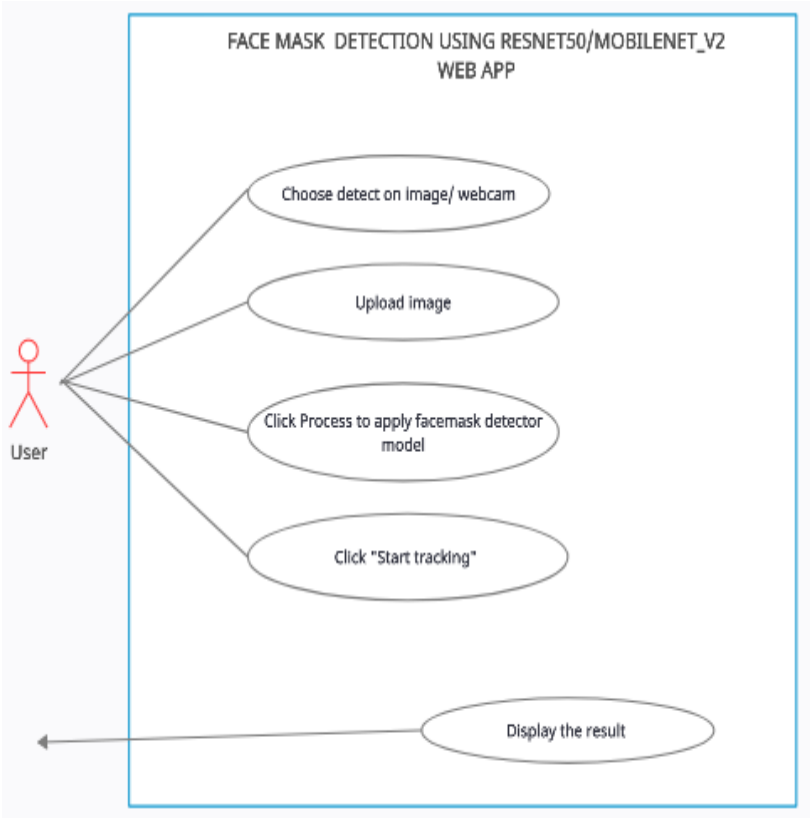


Figure 21: Use Case Diagram of Facemask Detection web application

1.2.2. Detect on image:

Step 1: Choose dectection on image

Step 2: Upload Image

Step 3: Click Process

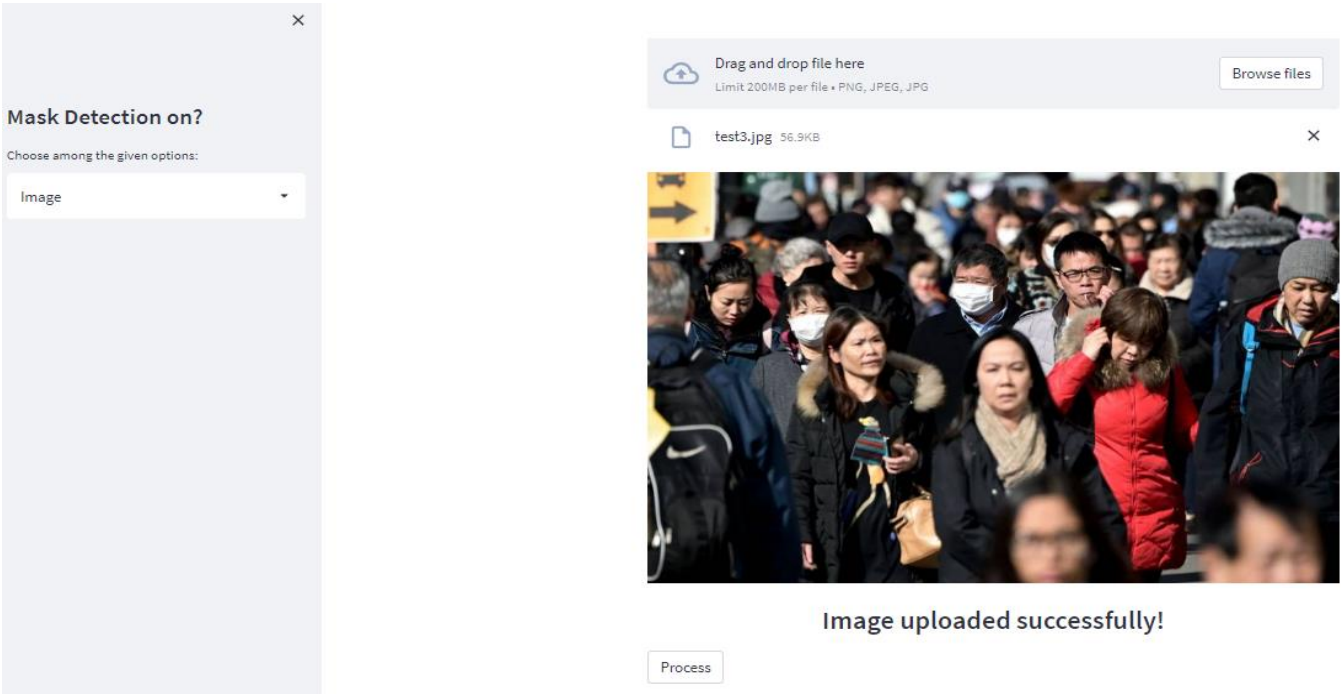


Figure 22: The user interface at detection on image mode

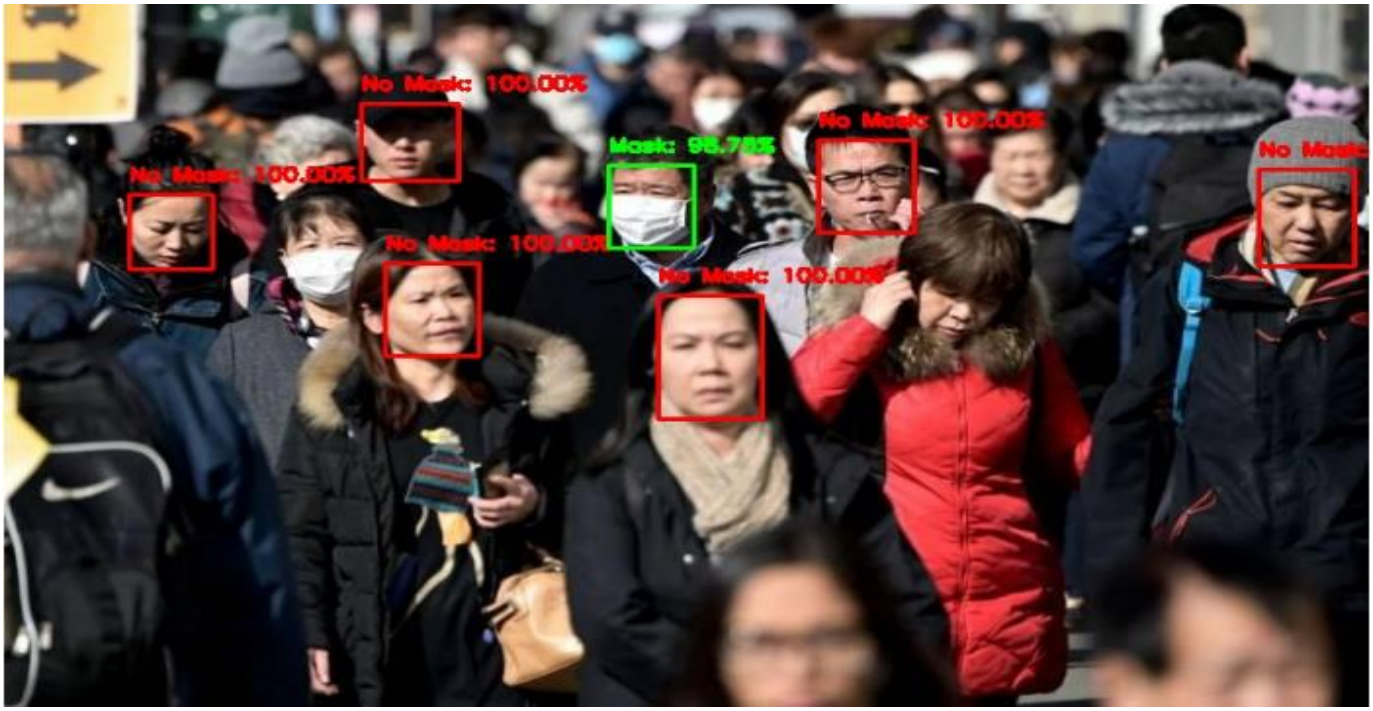


Figure 23: Facemask detection with MobileNetV2 (a)

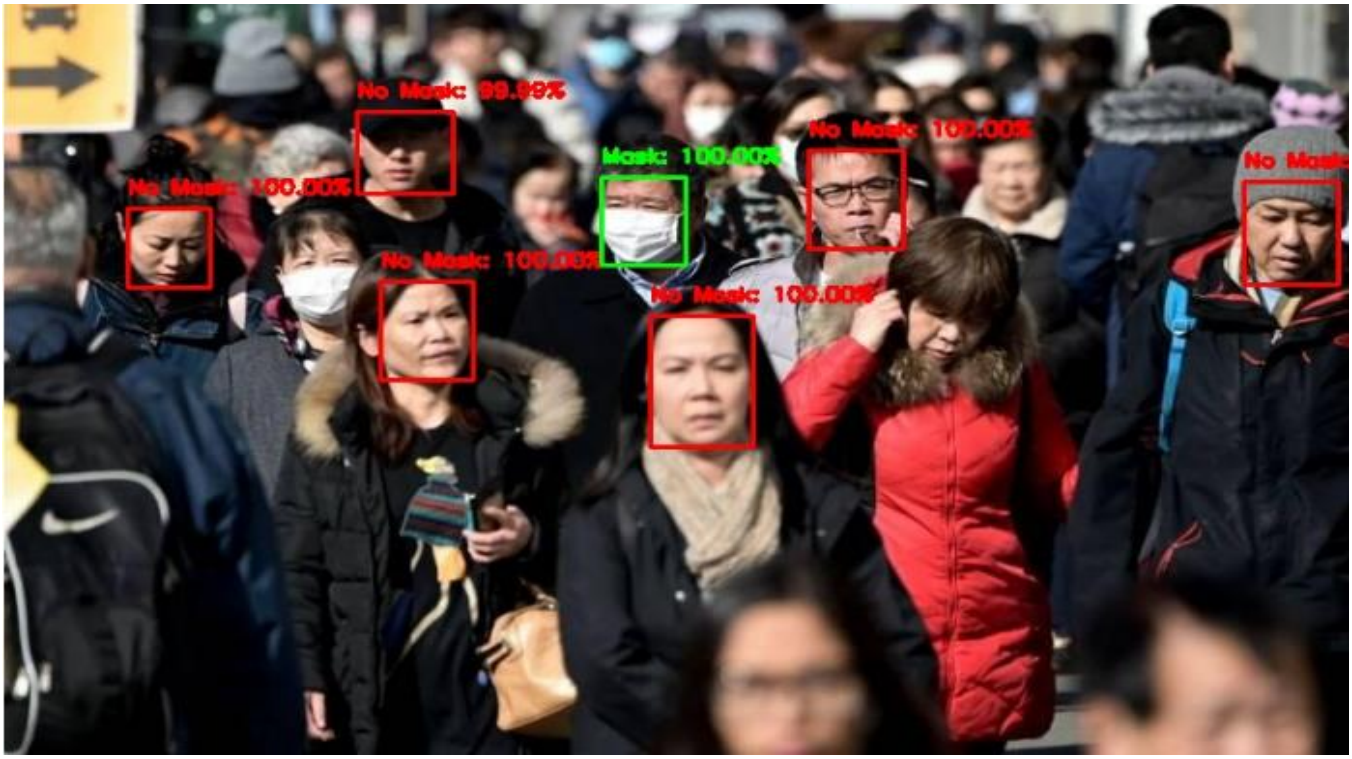


Figure 24: Facemask detection with Resnet50 (a)

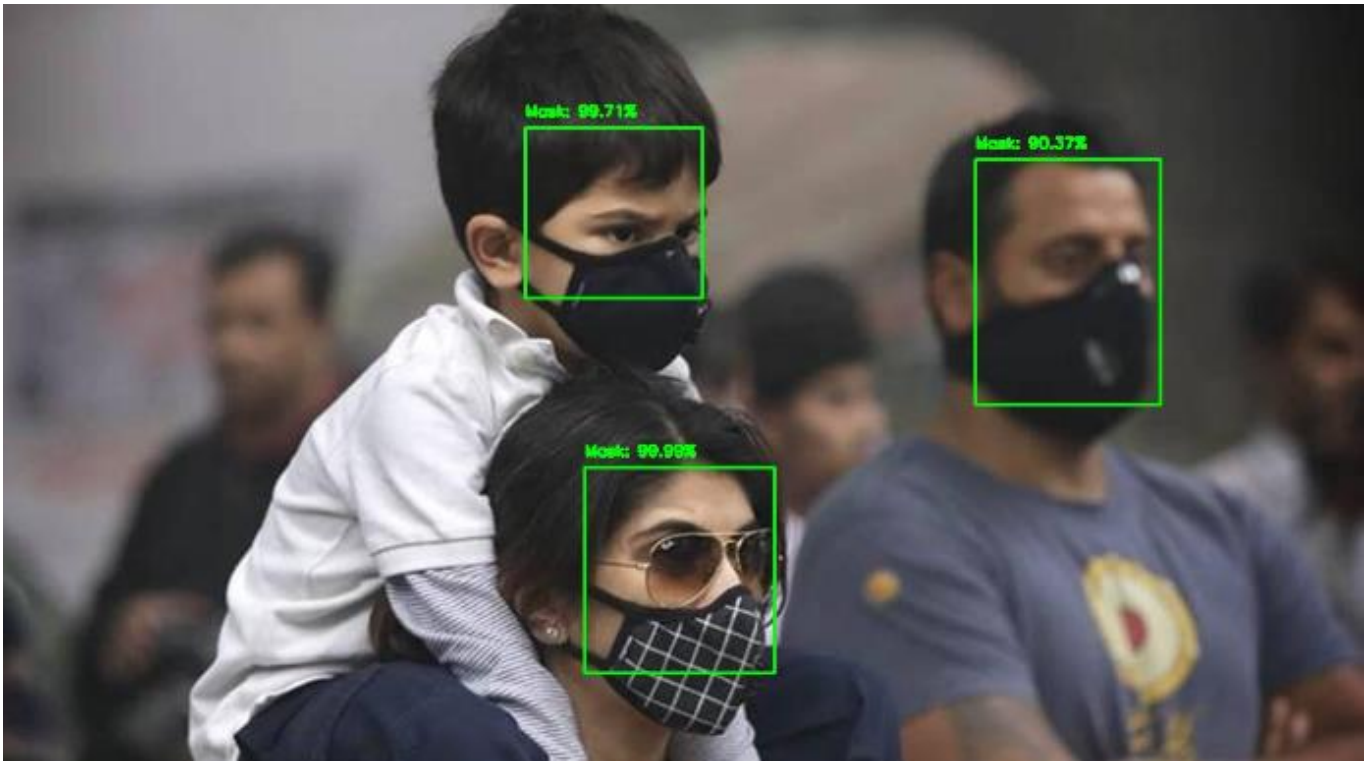


Figure 25: Facemask detection with MobileNetV2 (b)

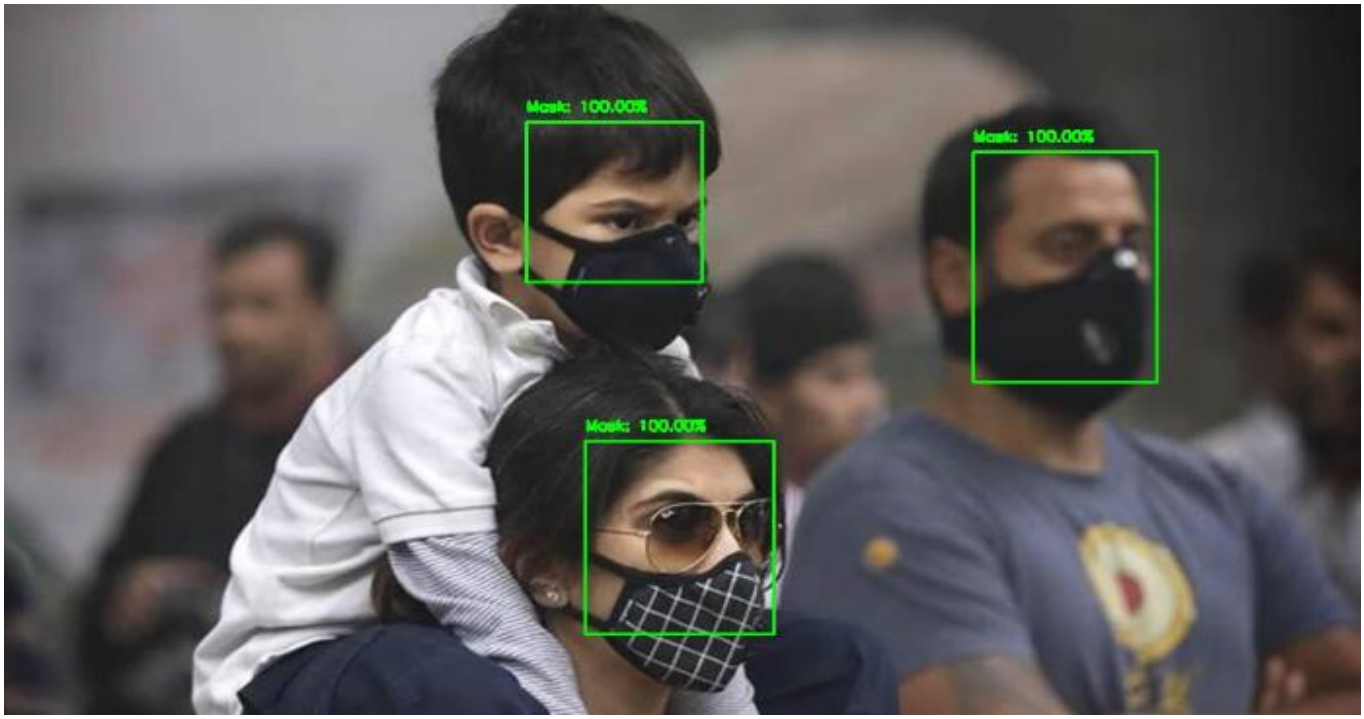
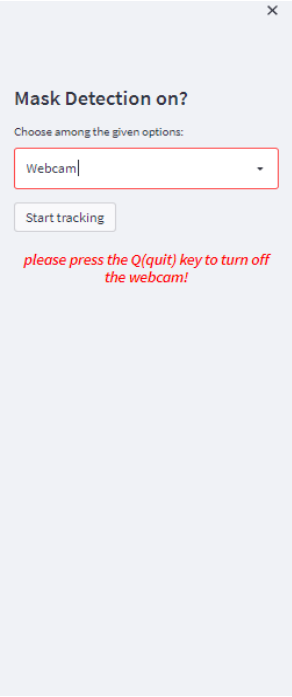


Figure 26: Facemask detection with Resnet50 (b)

1.2.3 Detect on webcam:

Step 1: Choose dectection on image

Step 2: Click "Start Tracking" and wait for the window of the face mask detection app opens.



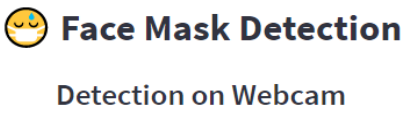


Figure 27: User interface at detection on webcam mode

Figure 28: Facemask detection on webcam with MobileNetV2

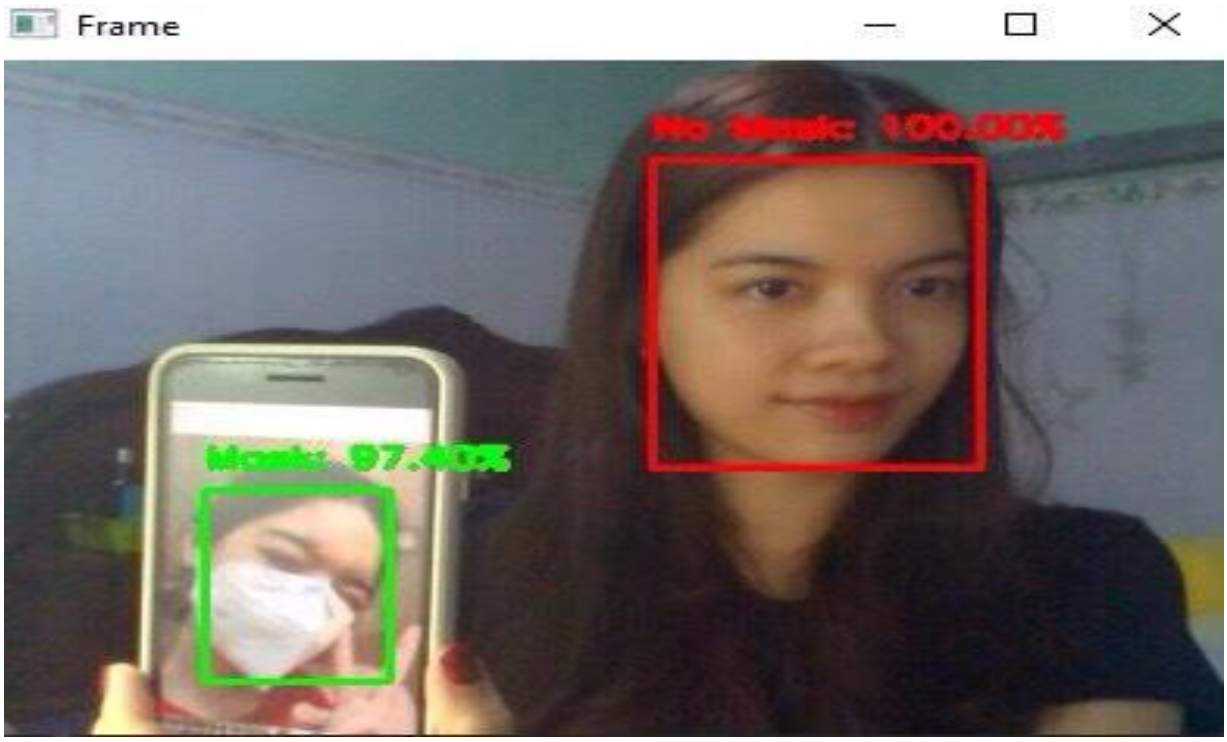


Figure 29: Facemask detection on webcam with Resnet50

2. Implementation and results of YOLOv5 model:

2.1 Training result of YOLOv5 model over 50 epochs:

2.1.1 Overall performance of YOLOv5 model

2.1.1 YOLOv5s

Yolov5s							
	Class	**Images**	**Labels**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
	all	170	652	0.887	0.832	0.865	0.57
	with_mask	170	553	0.912	0.906	0.942	0.651
	without_mask	170	99	0.863	0.758	0.788	0.488

Table 6: Model Evaluation of YOLOv5s

2.1.2 YOLOv5m

Yolov5m							
	Class	**Images**	**Labels**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
	all	170	652	0.887	0.82	0.874	0.587
	with_mask	170	553	0.925	0.913	0.953	0.669
	without_mask	170	99	0.848	0.727	0.796	0.504

Table 7: Model Evaluation of YOLOv5m

2.1.3 YOLOv5l

Yolov5l							
	Class	**Images**	**Labels**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
	all	170	652	0.873	0.818	0.876	0.595
	with_mask	170	553	0.918	0.913	0.949	0.666
	without_mask	170	99	0.827	0.723	0.803	0.524

Table 8: Model Evaluation of YOLOv5l

2.1.4 YOLOv5x

Yolov5x							
	Class	**Images**	**Labels**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
	all	170	652	0.92	0.783	0.87	0.593
	with_mask	170	553	0.936	0.898	0.949	0.678
	without_mask	170	99	0.904	0.667	0.79	0.509

Table 9: Model Evaluation of YOLOv5x

2.1.2 Plots of precision curve, recall curve, Precision-Recall curve, F1-score of yolov5

2.1.2.1 Precision curve by confidence

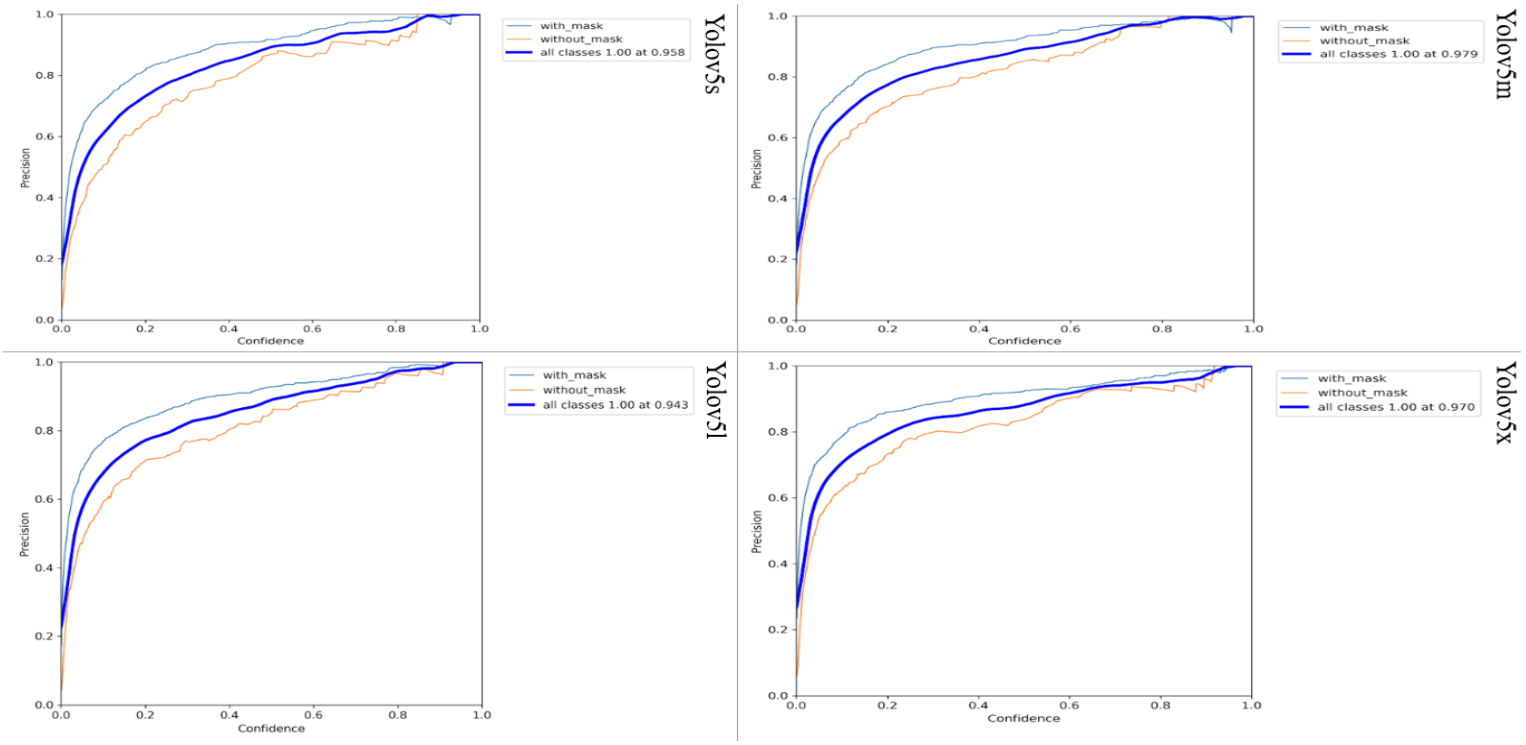


Figure 30: Precision curve by confidence threshold plot

2.1.2.2 Recall curve by confidence

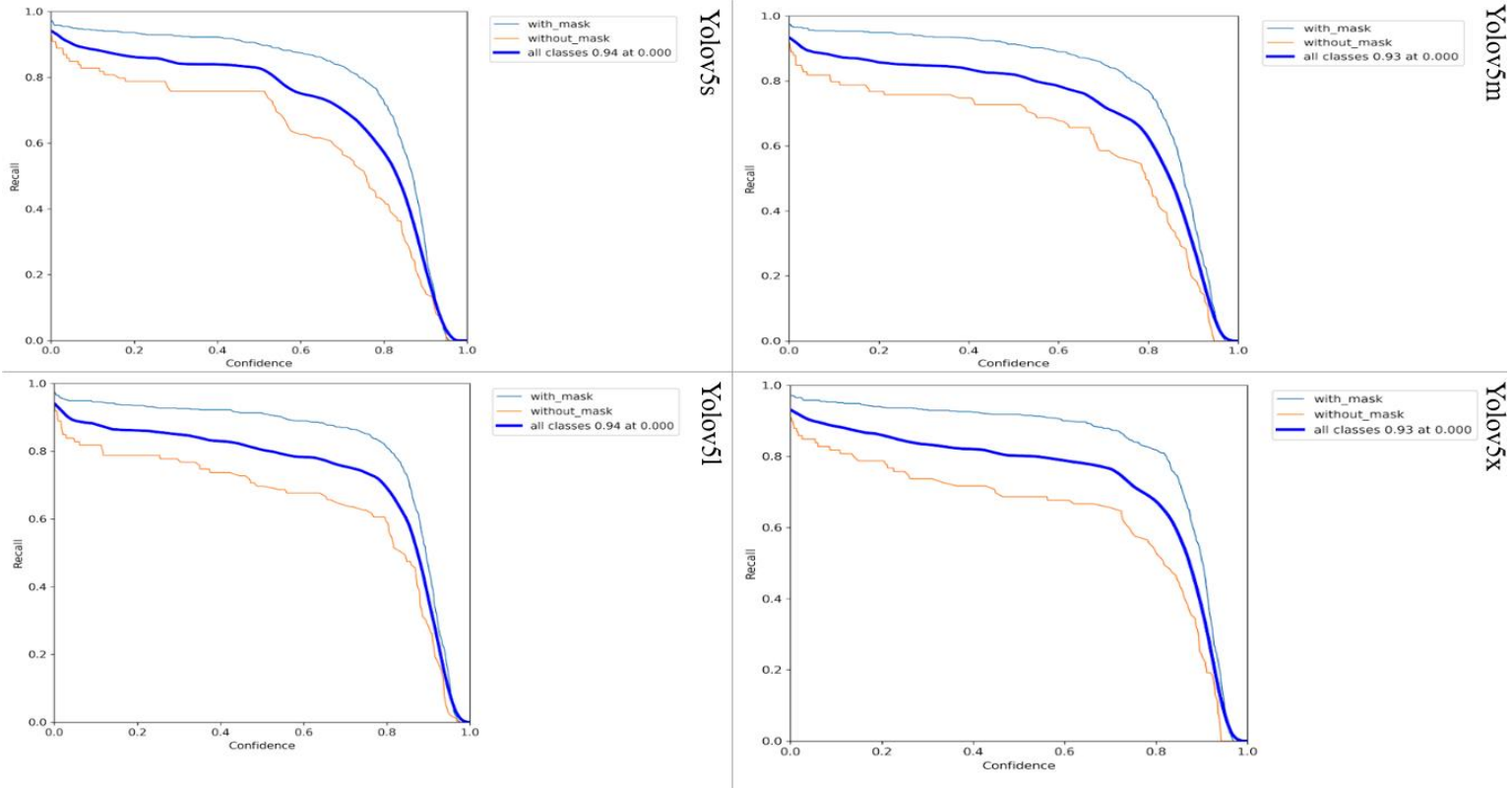


Figure 31: Recall curve by confidence threshold plot

2.1.2.3 Precision-Recall curve

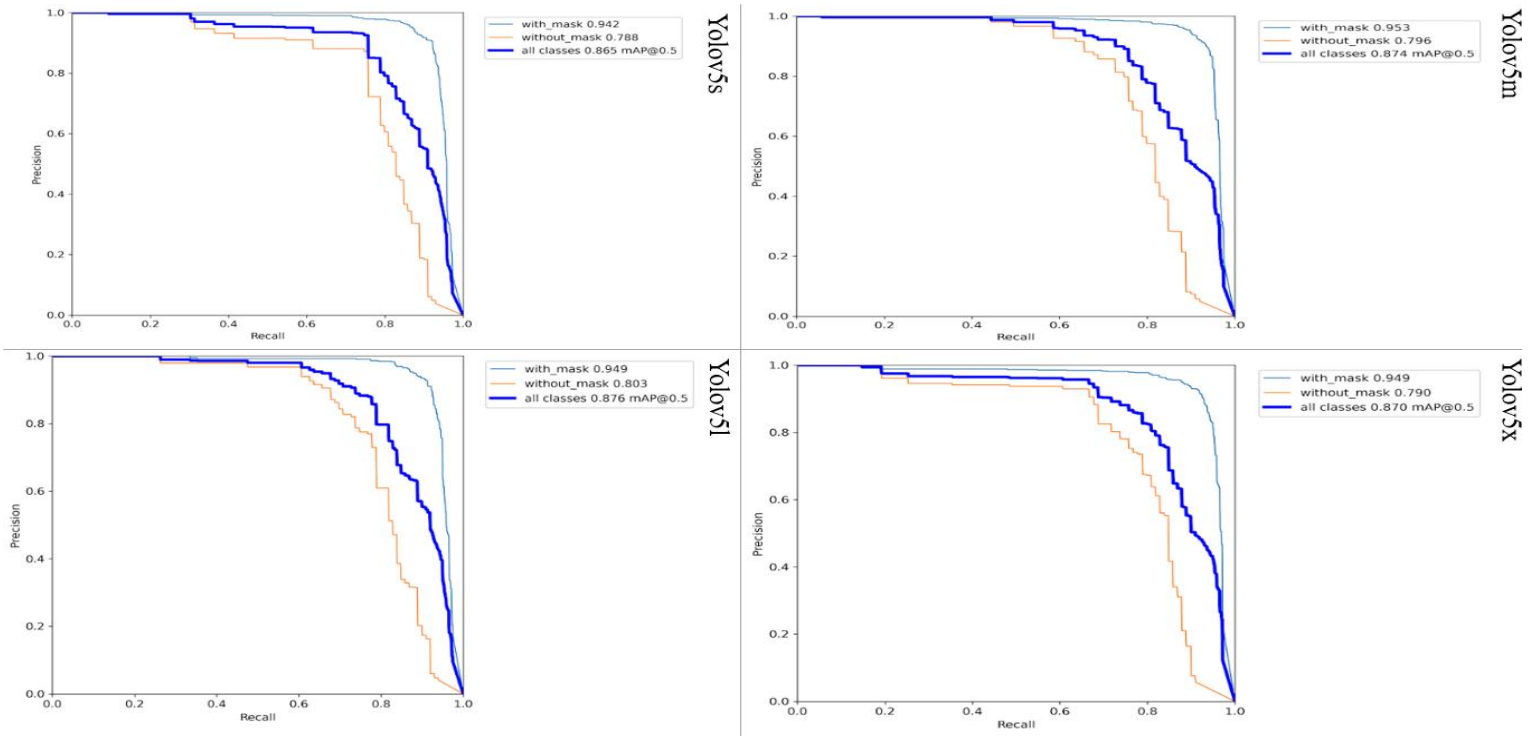


Figure 32: Precision-Recall plot

2.1.2.4 F1-score by confidence

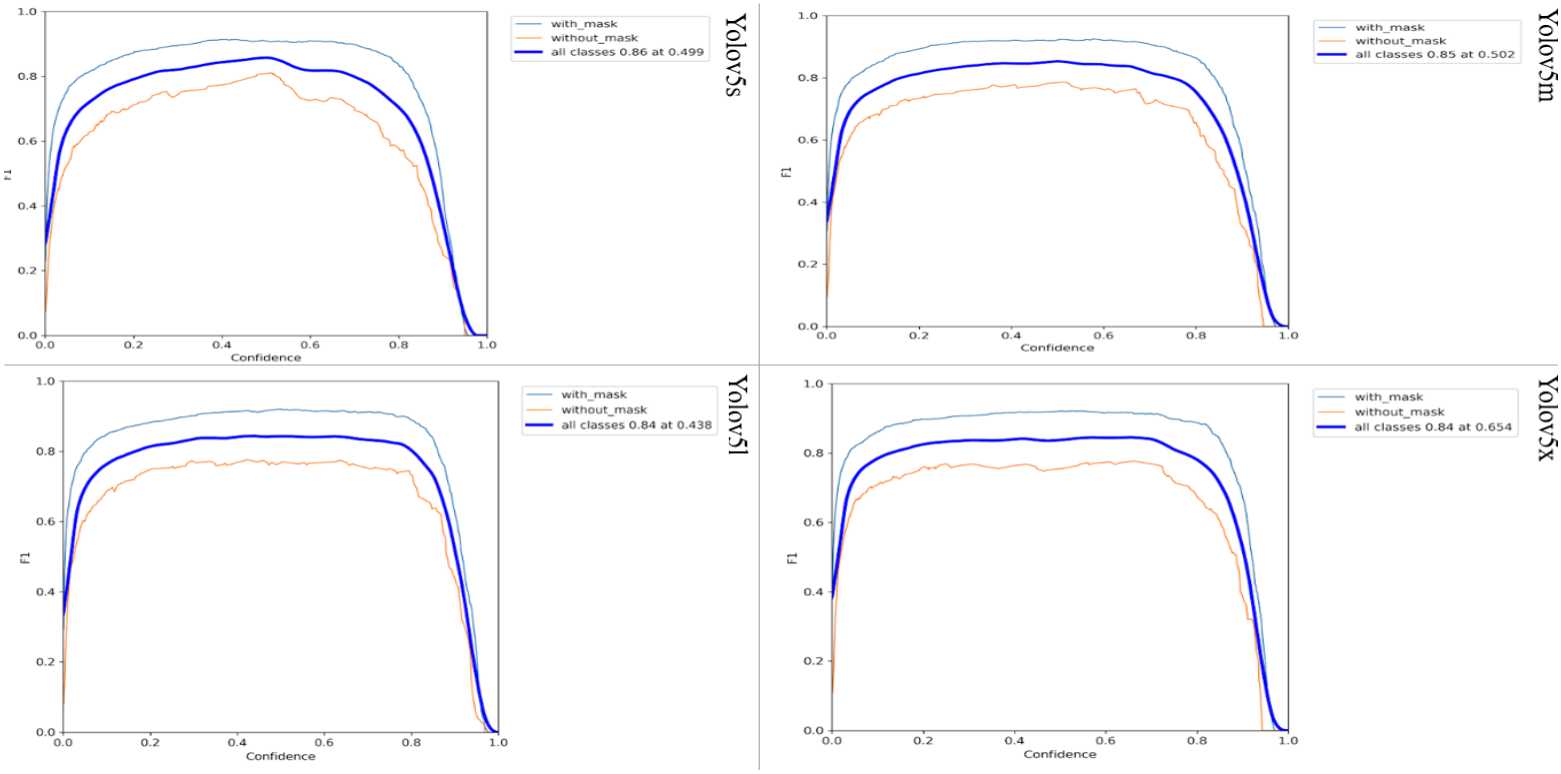


Figure 33: F1-score plot

2.1.3 Multiple evaluation parameters observed of trainning YOLOv5 model over 50 epochs:

2.1.3.1 YOLOv5s

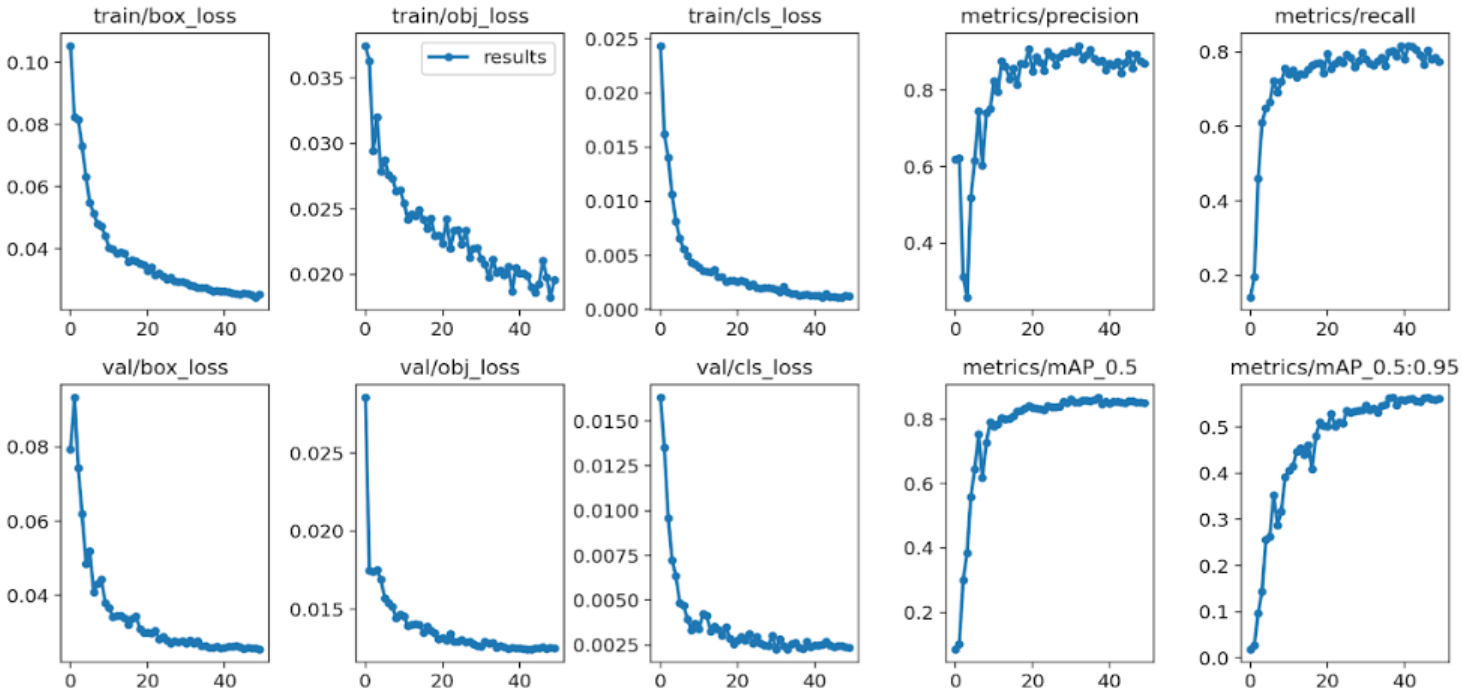


Figure 34: Multiple evaluation parameters observed of training YOLOv5s model over 50 epochs

2.1.3.2 YOLOv5m

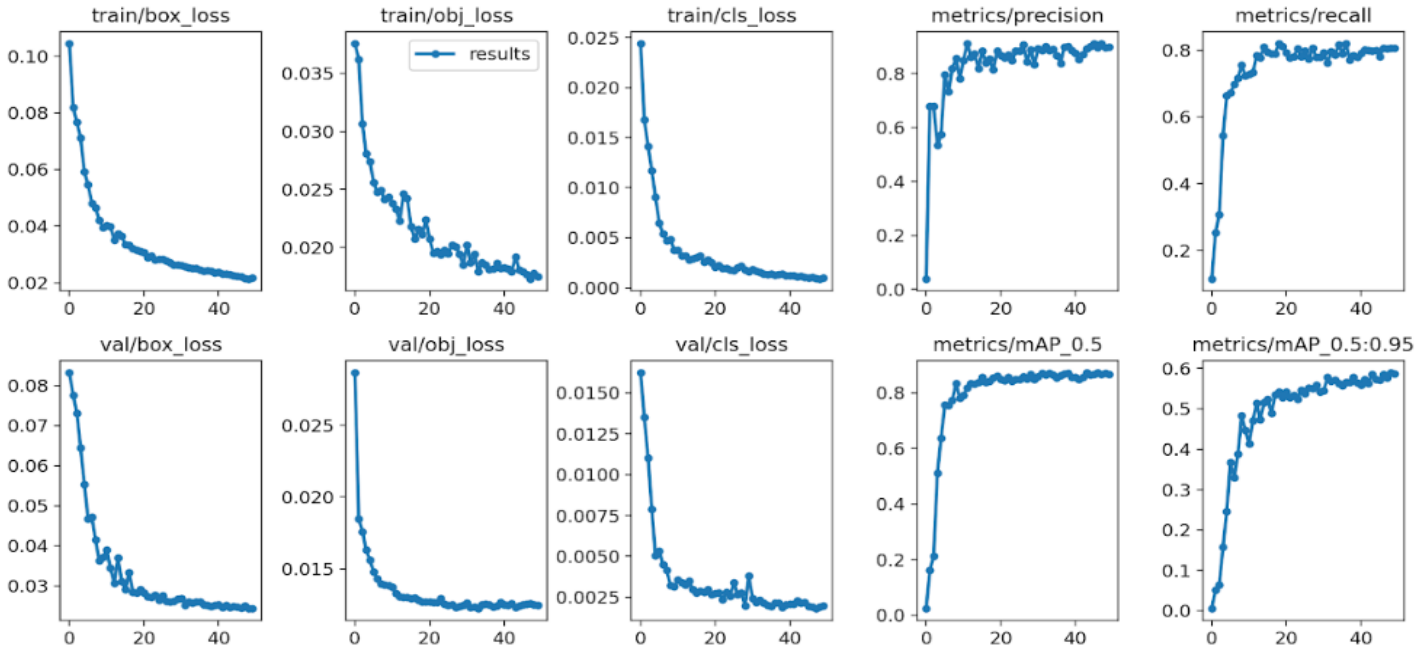


Figure 35: Multiple evaluation parameters observed of training YOLOv5m model over 50 epochs

2.1.3.3 YOLOv5l

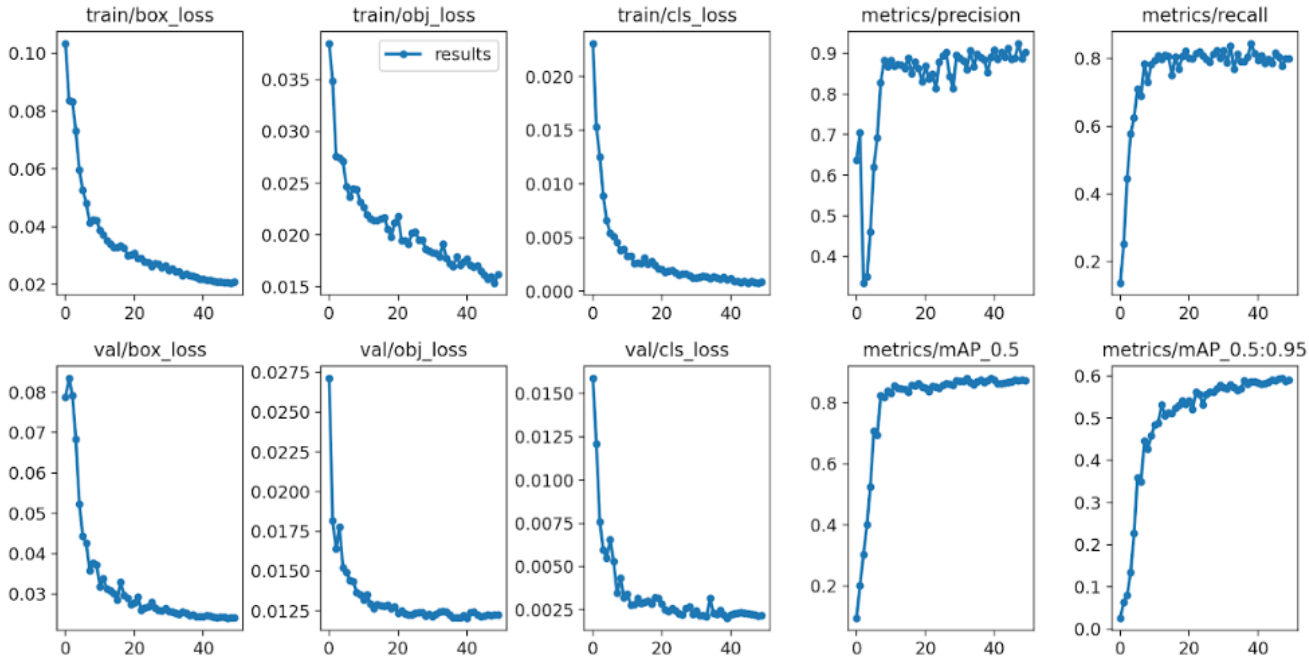


Figure 36: Multiple evaluation parameters observed of training YOLOv5l model over 50 epochs

2.1.3.4 YOLOv5x

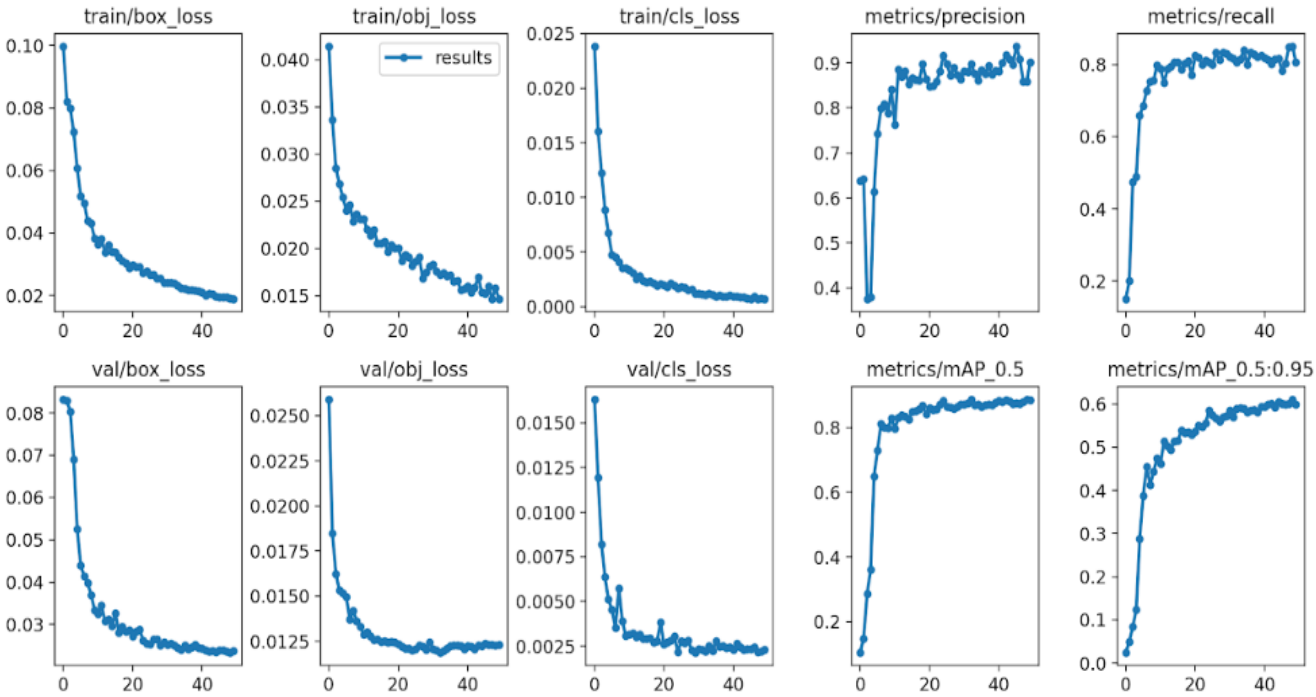


Figure 37: Multiple evaluation parameters observed of tranning YOLOv5x model over 50 epochs

2.1.4 Analysing the training results with four weights using tensorboard and re-train the model:

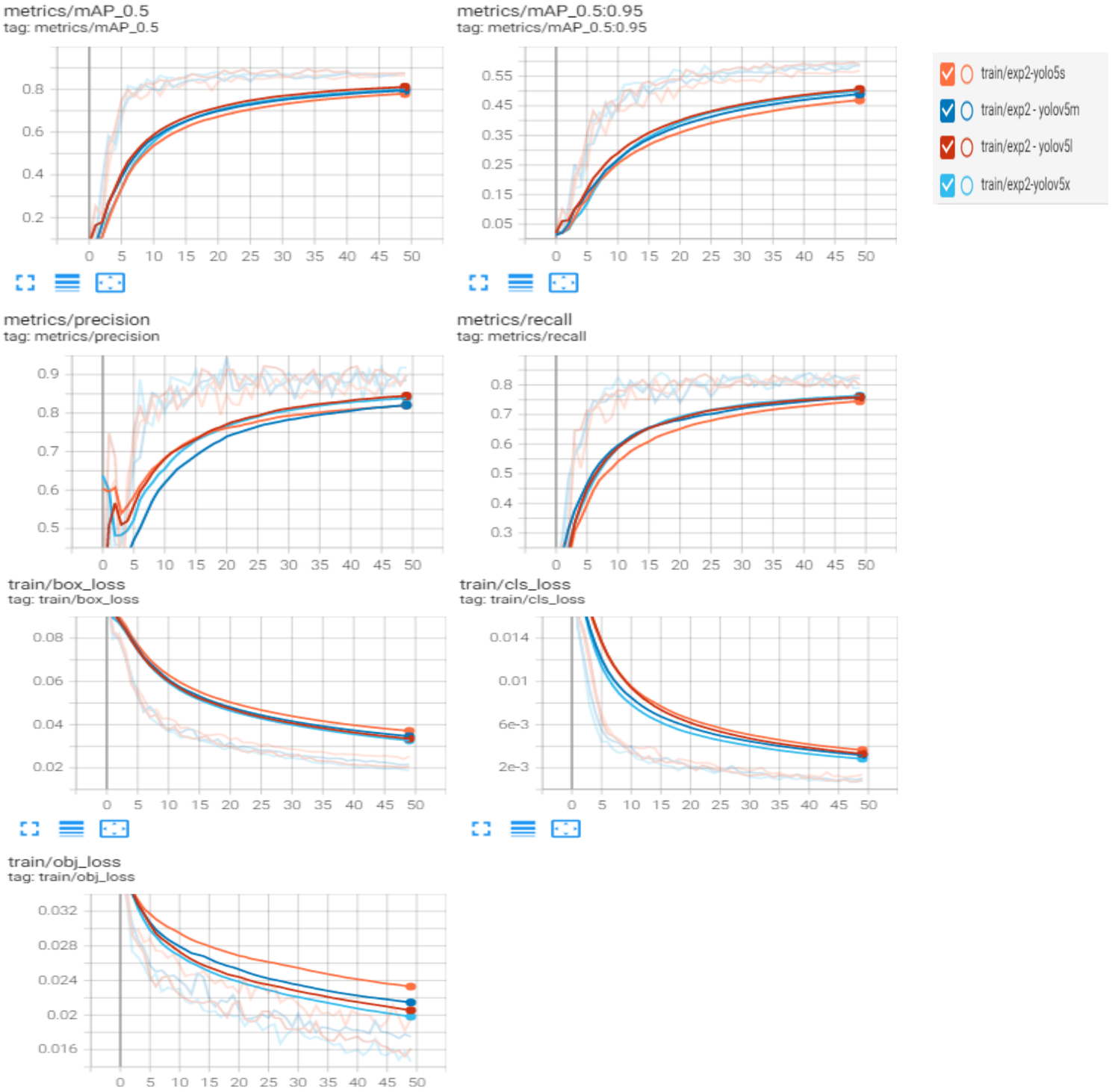


Figure 38: Training results of YOLOv5 with tensorboard

From the training plot and result in section 2.1- section 2.2 of each model and the tensorboard here, we can see that the model trained with YOLOv5l.pt (large) is the best model. Therefore, we chose this model to re-train with the image size is 416x416 pixels, and the batch equals 8. Over 150 epochs, we have the result:

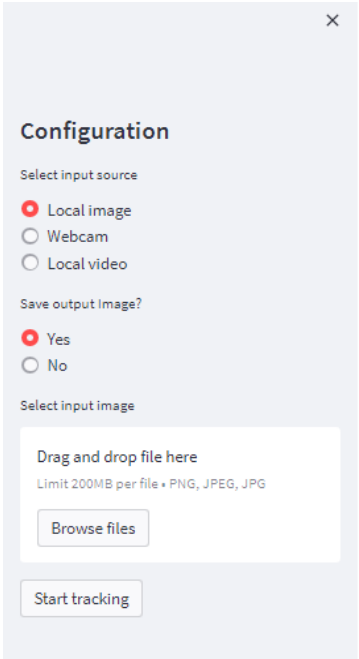
Class	Images	Labels	P	R	mAP@.0.5	mAP@.0.5:0.95
all	170	652	0.942	0.766	0.863	0.588
with_mask	170	553	0.942	0.89	0.948	0.667
without_mask	170	99	0.941	0.643	0.779	0.508

Table 10: Model evaluation of YOLOv5 after re-training

After re-training the model, we demonstrate the result of applying this model for face mask detection by building a facemask detection web application supported by Streamlit as in the next part.

2.2 Facemask detection web application using Streamlit

2.2.1 GUI:



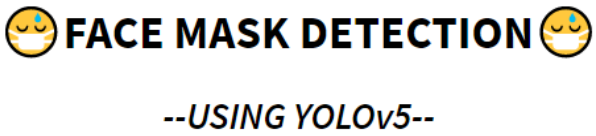


Figure 39: GUI's facemask detection using YOLOv5

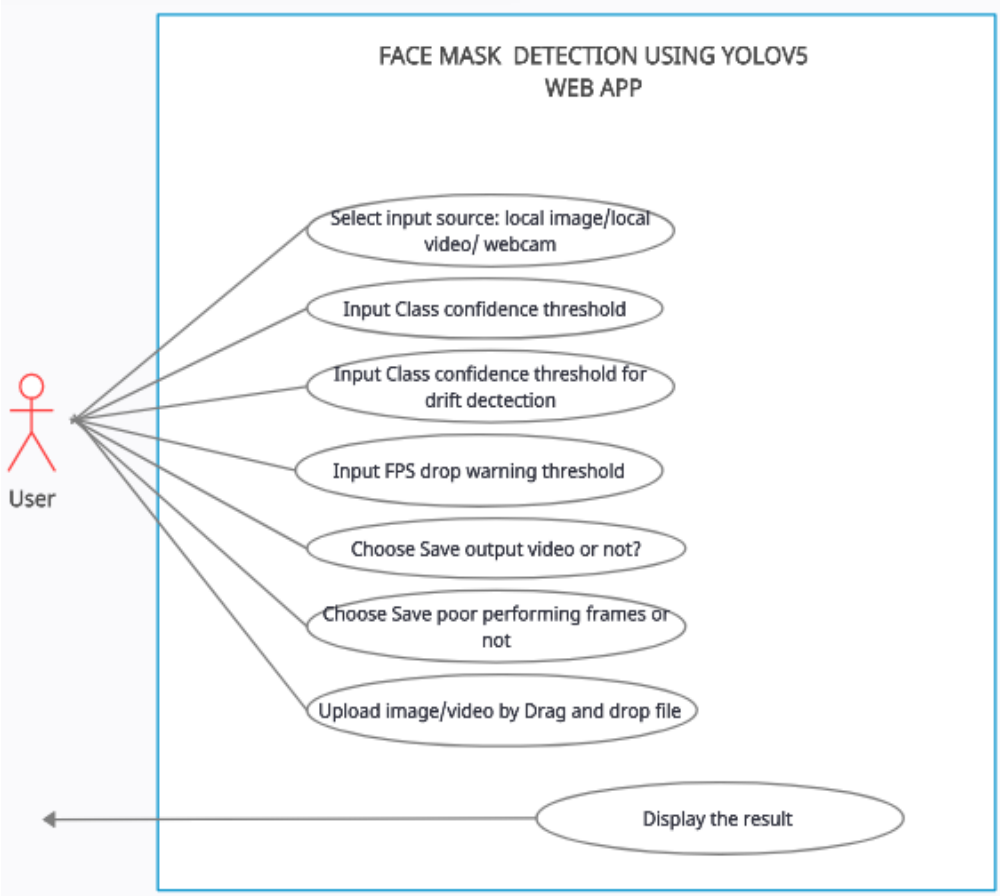


Figure 40: Use Case Diagram of Facemask Detection web application uisng yolov5

2.2.2: Detect on image:

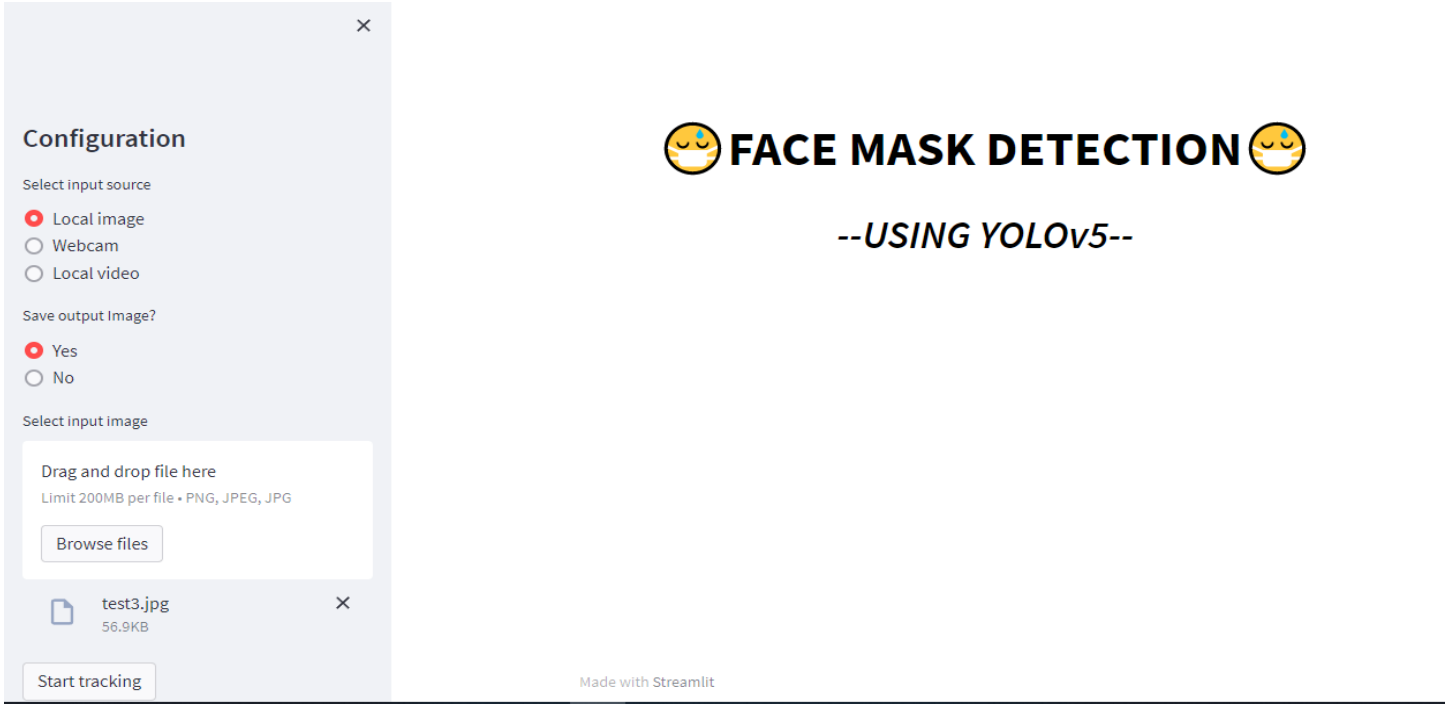


Figure 41: The user interface at detection on image mode

Step 1: In Select input source part, choose Local image.

Step 2: Save the output image part choose Yes if you want to save the image after detecting it.

Step 3: Browse files / Drag or drop files

Step 4: Click "Start Tracking"

Inference Stats

Total Detected objects


```
{
  "with_mask" : 4
  "without_mask" : 9
}
```


Figure 42: The result of detecting on image

We have used the same image with the testing image of Resnet50 and MobileNetv2 (in 1.2 of Chapter 6) to test detection on images applying YOLOv5. We can see that YOLOv5 has performed a better result than Resnet50 or Mobilenetv2. Although some people are blurred, YOLOv5 still detected those people. With Resnet50 and MobileNetv2, we just detected only 7 people (including with mask and without mask), whereas there were 13 people detected (4 people with mask, 9 people without mask) when applying YOLOv5.

2.2.3: Detect on video:

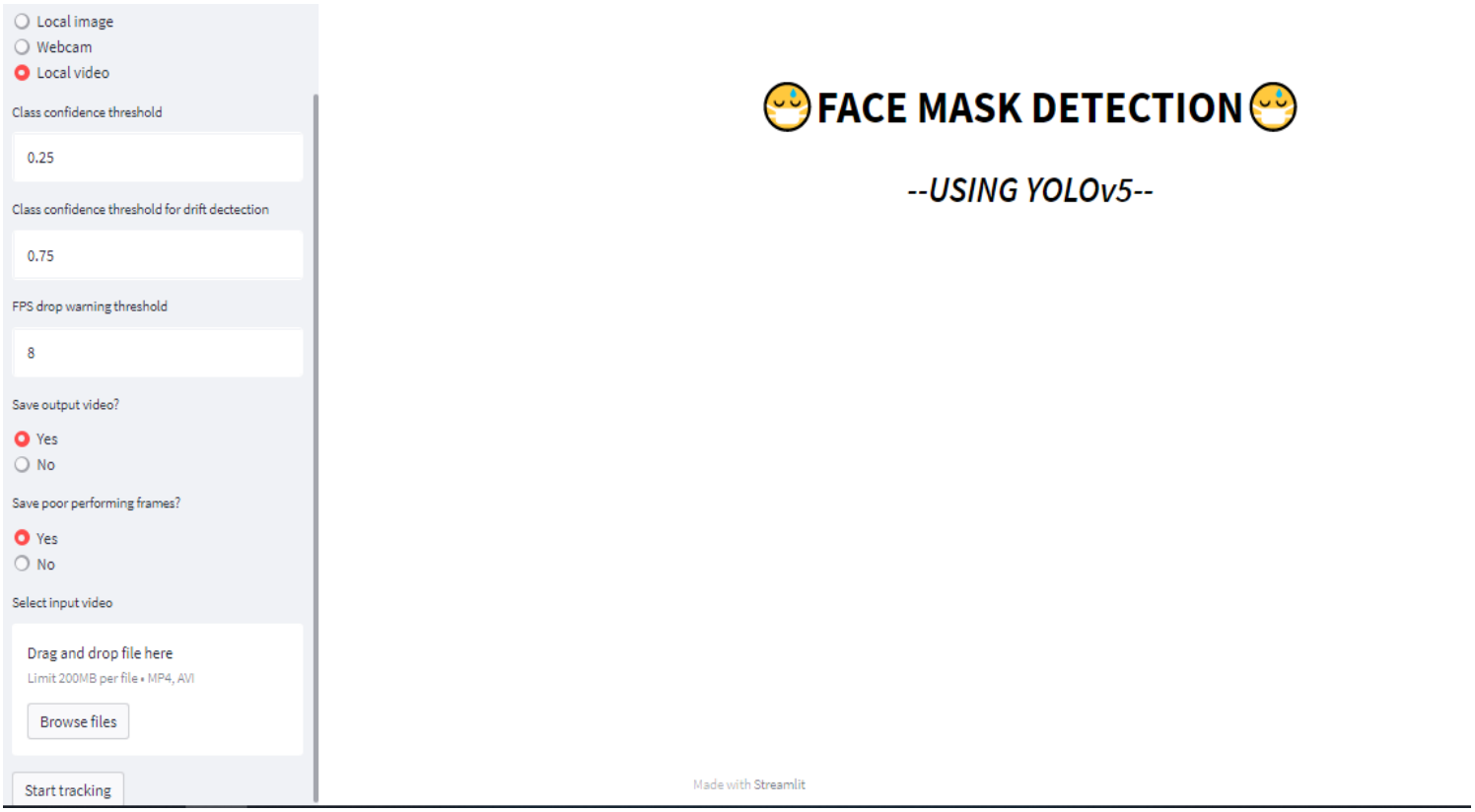


Figure 43: The user interface at detection in video mode

Step 1: In the Select input source part, choose Local video.

Step 2: Input class confidence threshold

Step 3: Input drop warning threshold

Step 4: Save output image part choose Yes if you want to save a frame of video after detecting

Step 5: Save poor performance frames part choose Yes if you want to save.

Step 6: Browse files / Drag or drop files

Step 7: Click "Start Tracking"

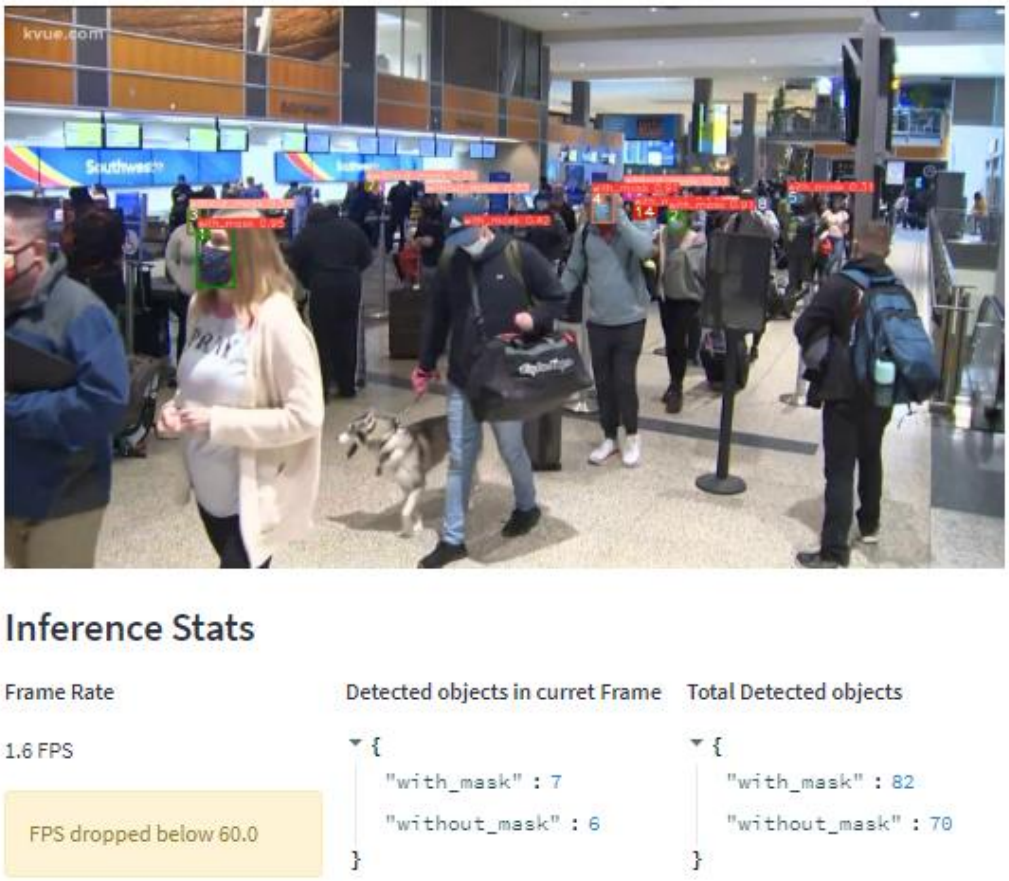


Figure 44: The result of detecting on video

2.2.4: Detect on webcam:

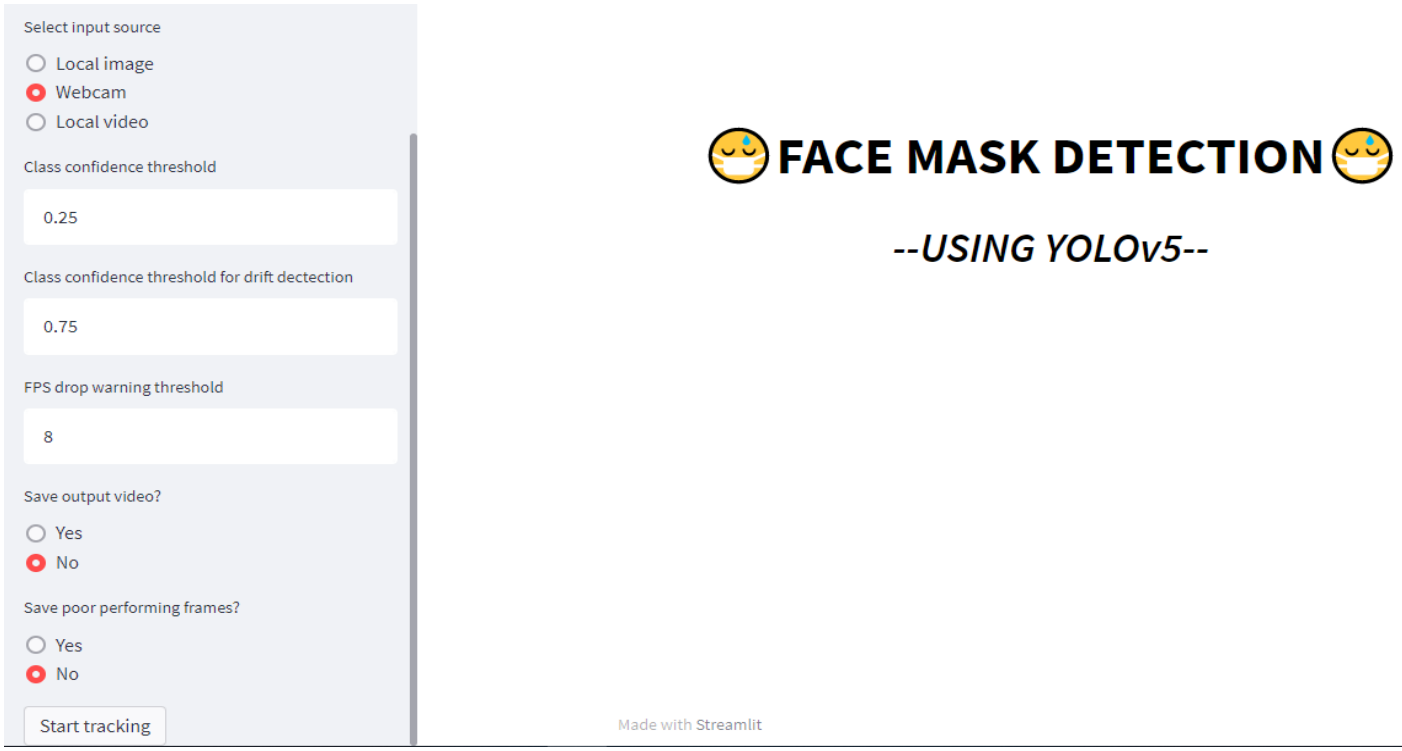


Figure 45: User interface at detection on webcam mode

Step 1: In the Select input source part, choose Local video.

Step 2: Input class confidence threshold

Step 3: Input drop warning threshold

Step 4: Save output image part choose Yes, if you want to save a frame of video after detecting

Step 5: Save poor performance frames part choose Yes if you want to save.

Step 6: Browse files / Drag or drop file

Step 7: Click "Start Tracking"

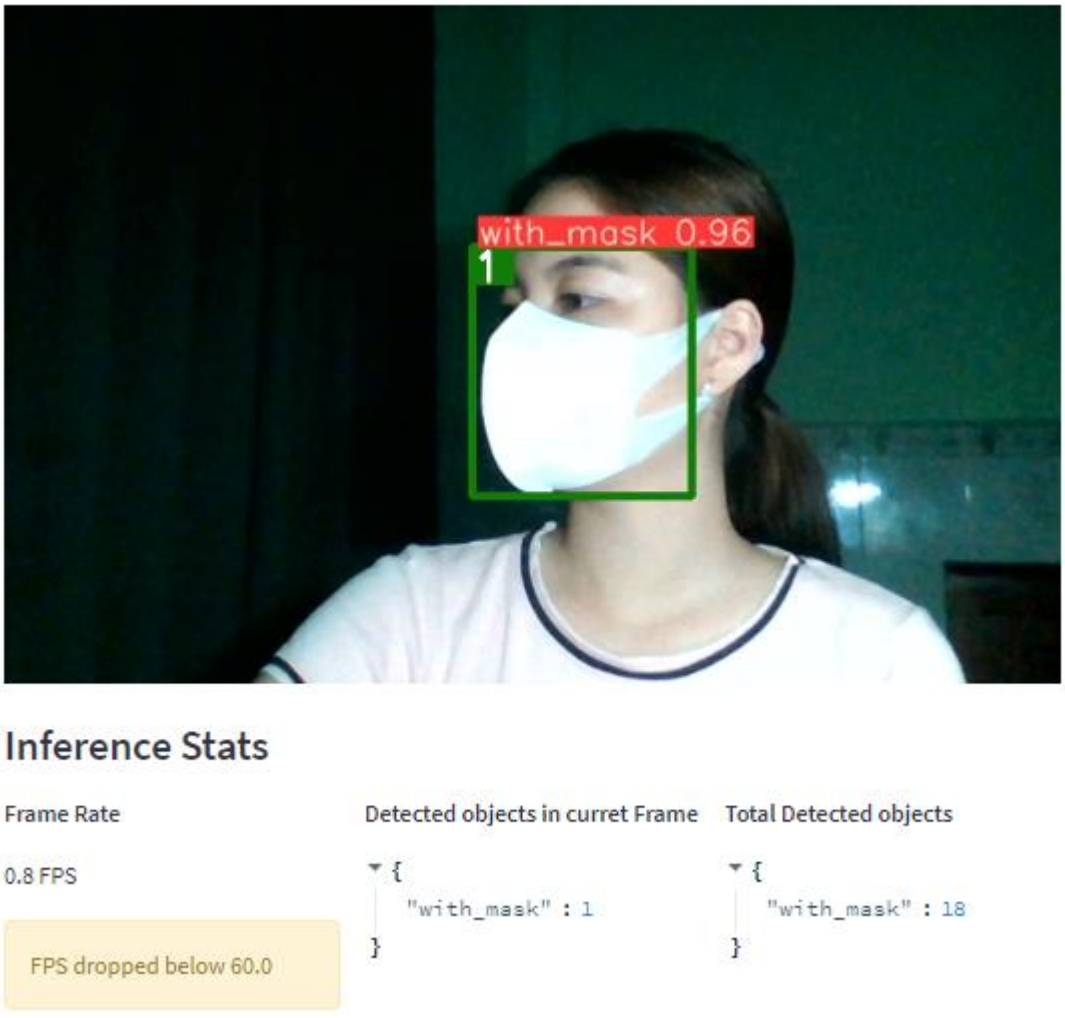


Figure 46: The result of detecting on webcam

CHAPTER 5: DISCUSSION AND FUTURE WORK

The model of the facemask detector is presented, which may help public health. In my first work, I utilized OpenCV, TensorFlow, Keras,and mobilenetV2/Resnet50 to assess whether or not persons were wearing facemask. The model is evaluated using images and video in real-time. A person with a mask can be identified with 99 percent accuracy. However, it can be seen that Object detection (particularly yolov5) provides better accurate results than Classification for the challenge of detecting persons wearing masks. The latest version of the YOLO series, YOLOv5, is more flexible than the previous single-stage detection algorithms. Because YOLOv5 is initially developed in PyTorch, it is supported and deployed via the existing PyTorch ecosystem. It is evident from the findings shown above that MobileNetv2 and Resnet50 produced pretty decent results, however Yolov5 produced superior outcomes. Even if the face of some people in image is blurry, object detection could find them. It has found more people wearing masks or not wearing masks than Classification, despite the fact that we used the same image.

Although it provides a beneficial outcome for the facemask detection problem. Nonetheless, this web application has some limitations. For instance, it sometimes recognizes accurately whether one person wears facemask only when he or she is directly facing the camera. In addition, the speed of detection on video or camera is still slow since this Web application must detect on frame by frame of video. To sum up, this web app is quite beneficial at public place such as super markets and airports. Improving this web app's ability to detection faces of persons who are not directly facing the camera is still a work in progress.

In the future, we want to add some features to this thesis in order to increase real-time mask detection speed and efficiency. This technique should be improved to recognize the faces of persons who are not immediately facing the camera. In addition, an alarm system that emits a signal when someone without a mask pass the area and a notification system that alerts the observer, such as authorities, may be implemented. This approach might minimize the spread of COVID-19. It can be installed in supermarkets and public areas, for instance. This helps our measures to counter the spreading of the COVID-19 virus,b ecause wearing masks limits the transmission of the COVID-19 virus in the population. The system involves a comprehensive check of all individuals that pass through the main gate. The application includes a thorough review of all persons who enter the entrance gate. The door will open , if the individual is wearing a facemask,; else, an error message such as "Please wear a mask before entering " may be shown. This method may be utilized

to identify persons who are not wearing masks not only during the COVID epidemic, but also in hospitals, chemical industries, and cement companies where workers come into frequent contact with hazardous chemicals and fine dust...

REFERENCES

[1] Ha Nguyen, “Vietnam Imposes Hefty Fines for Going Maskless”, Viet Times Newspaper [April , 2020].[Online].Available: https://www.voanews.com/a/science-health_coronavirus-outbreak_vietnam-imposes-hefty-fines-going-maskless/6186797.html

[2] AHuynh TD. The more I fear about COVID-19, the more I wear medical masks: A survey on risk perception and medical masks uses. [2020]

[3] Anh Le, “ Thủ tướng Nguyễn Xuân Phúc yêu cầu mọi người dân đeo khẩu trang ở nơi đông người”.Viet Times Newspaper .[Online].Available: https://viettimes.vn/thu-tuong-nguyen-xuan-phuc-yeu-cau-moi-nguoi-dan-deo-khau-trang-o-noi-dong-nguoi-post126311.html

[4] Swami Sivasubramanian, “How AI and machine learning are helping to tackle COVID-19World Economic Forum”. The World Economic Forum. [May 2020]. [Online]. Available: https://www.weforum.org/agenda/2020/05/how-ai-and-machine-learning-are-helping-to-fight-covid-19/

[5] Kabir, A. I., Mitra, S., Jakowan, & Das, S. S. “Development of a face-mask detection software using artificial intelligence (AI) in python for Covid-19 protection.”. Journal of Management Information and Decision Sciences. [2021]. [Online]. Available:https://www.abacademies.org/articles/Development-of-a-face-mask-detection-software-using-artificial-intelligence-AI-in-python-for-Covid-19-protection-1532-5806-24-6-292.pdf

[6] Chavez, S., Long, B., Koyfman, A., & Liang, S. Y. “Coronavirus Disease (COVID-19): A primer for emergency physicians.” [2020]

[7] Feng, S., Shen, C., Xia, N., Song, W., Fan, M., & Cowling, B. J. “Rational use of face masks in the COVID-19 pandemic”. [2020]

[8] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A.& LiangChieh. “MobileNetV2: Inverted Residues and Linear Bottlenecks”. [2018]

[9] Boyko, N., Basystiuk, O., & Shakhovska, N. “Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library”. [2018]

[10] Kalas, M. S. “Real-Time Face Detection and Tracking using OpenCV. International Journal of Soft Computing and Artificial Intelligence”. [2019]

[11] Militante, S. V.,& Dionisio, N. V. “Real-Time Face Mask Recognition with Alarm System using Deep Learning”. [2020]

[12] Das, A., Ansari, M. W., & Basak, R. “Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV.” [2020]

[13] Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. (2021). “A Hybrid Deep Transfer Learning Model with Machine Learning Methods for Face Mask Detection in the Era of the COVID-19 Pandemic”. [2021]

[14] Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. “Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection”. [2021]

[15].E. Susanto, R. Rudiawan, D. Analia, S. Pamungkas and H. Soebakti. "The deep learning development for real-time ball and goal detection of barelang-FC". 2017 International Electronics Symposium on Engineering Technology and Applications (IES-ETA). [2017].

[16].C. Liu, Y. Tao, J. Liang, K. Li and Y. Chen. "Object detection based on YOLO network". 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC). [2018].

[17].W. Yang and Z. Jiachun. "Real-time face detection based on YOLO". 2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII) .[2018].

[18].D. Garg, P. Goel, S. Pandya, A. Ganatra and K. Kotecha. "A deep learning approach for face detection using YOLO". 2018 IEEE Punecon, [2018].

[19].C. Zhao and B. Chen. "Real-time pedestrian detection based on improved YOLO model”. 11th International Conference on Intelligent Human-Machine Systems and Cybernetics [2019].

[20].J. Redmon and A. Farhadi. “YOLO9000: Better Faster Stronger” [2016].

[21].C. Kim et al."Implementation of Yolo-v2 image recognition and other testbenches for a CNN accelerator". 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin). [2019].

[22].H. V and K. R. "Real time pedestrian detection using modified YOLO V2". 2020 5th International Conference on Communication and Electronics Systems” (ICCES). [2020]

[23].J. Redmon and A. Farhadi. “YOLOv3: An Incremental Improvement”.[2018]

[24].J. Hu, X. Gao, H. Wu and S. Gao. "Detection of workers without the helments in videos based on YOLO V3". 2019 12th International Congress on Image and Signal Processing BioMedical Engineering and Informatics (CISP-BMEI). [2019].

[25] A. Bochkovskiy, C-Y. Wang and H-Y. M. Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. [2020]

[26] Dostdar Hussain,Muhammad Ismail, Israr Hussain, Roobaea Alroobaea, Saddam Hussain, and Syed Sajid Ullah. “Face Mask Detection Using Deep Convolutional Neural Network and MobileNetV2-Based Transfer Learning”. [2022]

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. [2015]

[28] M. Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. [2018]

APPENDIX

A. Classification with Resnet50 vs MobileNetv2 (SOURCE CODE)

1. Requirement

requirements.txt

```
tensorflow>=2.5.0*
keras==2.4.3
imutils==0.5.4
numpy==1.19.5
opencv-python>=4.2.0.32
matplotlib==3.4.1
argparse==1.4.0
scipy==1.6.2
scikit-learn==0.24.1
pillow>=8.3.2
streamlit==1.10.0
onnx==1.10.1
tf2onnx==1.9.3
```

2. Training model:

2.1 Training model with Resnet50

Resnet50_trainning.py 9+

ResNet50_v2 > Resnet50_trainning.py > ...

```
# -*- coding: utf-8 -*-

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import tensorflow_hub as hub
import tensorflow as tf
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras import layers
from tensorflow.keras.layers import Input
import argparse
import os
from tensorflow.keras.applications import ResNet50V2

ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to output face mask detector model")
args = vars(ap.parse_args())

```

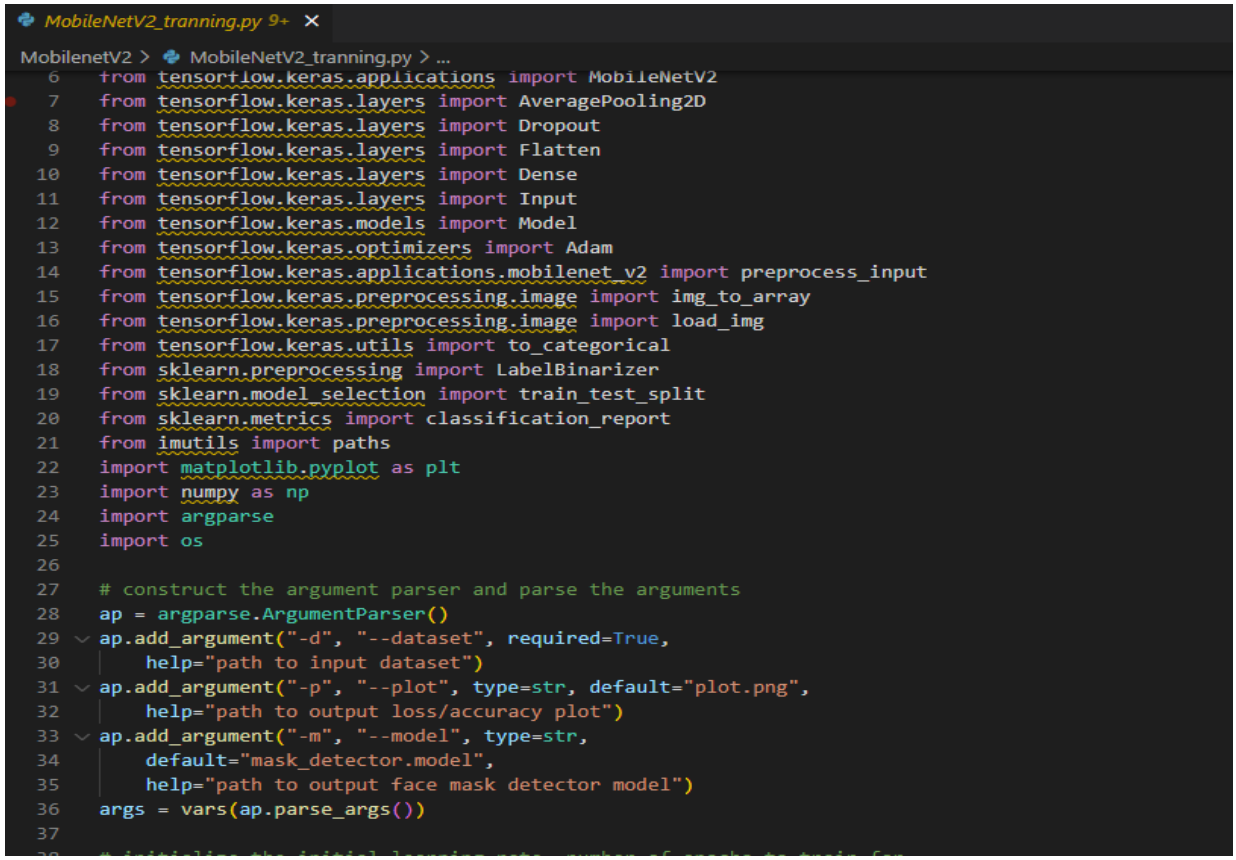
```
31
32    print("[INFO] loading images...")
33    imagePaths = list(paths.list_images(args["dataset"]))
34    data = []
35    labels = []
36
37    IMG_SIZE = 224
38    CHANNELS = 3
39    N_LABELS=2
40
41
42    # loop over the image paths
43    for imagePath in imagePaths:
44        # extract the class label from the filename
45      label = imagePath.split(os.path.sep)[-2]
46        # load the input image (224x224) and preprocess it
47      image = load_img(imagePath, target_size=(IMG_SIZE, IMG_SIZE))
48      image = img_to_array(image)
49      image = image/255
50    #image = preprocess_input(image)
51
52        # update the data and labels lists, respectively
53      data.append(image)
54      labels.append(label)
55
56    data = np.array(data, dtype="float32")
57    labels = np.array(labels)
58
59    lb = LabelBinarizer()
60    labels = lb.fit_transform(labels)
61    labels = to_categorical(labels)
```

```
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.20, stratify=labels, random_state=42)

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")


feature_extractor_layer = ResNet50V2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(IMG_SIZE,IMG_SIZE,CHANNELS)))

feature_extractor_layer.trainable = False

model = tf.keras.Sequential([
    feature_extractor_layer,
    layers.Flatten(name="flatten"),
    layers.Dense(1024, activation='relu', name='hidden_layer'),
    layers.Dropout(0.5),
    layers.Dense(N_LABELS, activation='sigmoid', name='output')
])

model.summary()

LR = 1e-5 # Keep it small when transfer learning
EPOCHS = 20
BS = 256
```

```
model.compile(
  optimizer=tf.keras.optimizers.Adam(learning_rate=LR),
  loss="binary_crossentropy",
  metrics=["accuracy"])

import time
start = time.time()
history = model.fit(aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    epochs=EPOCHS)
print('\nTraining took {}'.format((time.time()-start)))

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

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

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

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

N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), history.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), history.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), history.history["accuracy"], label="accuracy")
plt.plot(np.arange(0, N), history.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
```

```
  plt.title("Training Loss and Accuracy")
  plt.xlabel("Epoch #")
  plt.ylabel("Loss/Accuracy")
  plt.legend(loc="lower left")
  plt.savefig(args["plot"])
```

2.2 Training model with MobileNetv2



```python
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]

    # load the input image (224x224) and preprocess it
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)

    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)

# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)

# construct the 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")

# load the MobileNetV2 network, ensuring the head FC layer sets are
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
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)

# place the head FC model on top of the base model (this will become
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
for layer in baseModel.layers:
    layer.trainable = False

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

# train 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)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")

# plot the training loss and accuracy
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(args["plot"])
```

3. Detection:

3.1 Detection on image:

```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import cv2
import os
def mask_image():
    # construct the argument parser and parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--image", required=True,
        help="path to input image")
    ap.add_argument("-f", "--face", type=str,
        default="face_detector",
        help="path to face detector model directory")
    ap.add_argument("-m", "--model", type=str,
        default="mask_detector.model",
        help="path to trained face mask detector model")
    ap.add_argument("-c", "--confidence", type=float, default=0.5,
        help="minimum probability to filter weak detections")
    args = vars(ap.parse_args())

    # load our serialized face detector model from disk
    print("[INFO] loading face detector model...")
    prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
    weightsPath = os.path.sep.join([args["face"],
        "res10_300x300_ssd_iter_140000.caffemodel"])
    net = cv2.dnn.readNet(prototxtPath, weightsPath)

    # load the face mask detector model from disk
    print("[INFO] loading face mask detector model...")
    model = load_model(args["model"])

```

```
    image = cv2.imread(args["image"])
    orig = image.copy()
    (h, w) = image.shape[:2]

    # construct a blob from the image
    blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    print("[INFO] computing face detections...")
    net.setInput(blob)
    detections = net.forward()

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the confidence is greater than the minimum confidence
        if confidence > args["confidence"]:
            # compute the (x, y)-coordinates of the bounding box for the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            # ensure the bounding boxes fall within the dimensions of the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel ordering, resize it to 224x224, and preprocess it
            face = image[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
```

```
            face = img_to_array(face)
            face = preprocess_input(face)
            face = np.expand_dims(face, axis=0)

            # pass the face through the model to determine if the face
            # has a mask or not
            (mask, withoutMask) = model.predict(face)[0]

            # determine the class label and color we'll use to draw
            # the bounding box and text
            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

            # include the probability in the label
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100

            # display the label and bounding box rectangle on the output
            # frame
            cv2.putText(image, label, (startX, startY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)

    # show the output image
    cv2.imshow("Output", image)
    cv2.waitKey(0)

if __name__ == "__main__":
    mask_image()

```

3.2 Detection on webcam:

```
from pickle import FRAME
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2
import os
import streamlit as st


def detect_and_predict_mask(frame, faceNet, maskNet):
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
                                 (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    faces = []
    locs = []
    preds = []
```

```
    faceNet.setInput(blob)
    detections = faceNet.forward()
    faces = []
    locs = []
    preds = []

    # loop over the detections
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.5:
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
            face = frame[startY:endY, startX:endX]
            if face.any():
                face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                face = cv2.resize(face, (224, 224))
                face = img_to_array(face)
                face = preprocess_input(face)

                # add the face and bounding boxes to their respective
                # lists
                faces.append(face)
                locs.append((startX, startY, endX, endY))
```

```python
def Load_WebCam(faceNet, maskNet):

    vs = VideoStream(src=0).start()
    time.sleep(2.0)

    # loop over the frames from the video stream
    while True:

        frame = vs.read()
        frame = imutils.resize(frame, width=400)
        # face mask or not
        (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
        # FRAME_WINDOW = st.image([])
        for (box, pred) in zip(locs, preds):
            # unpack the bounding box and predictions
            (startX, startY, endX, endY) = box
            (mask, withoutMask) = pred
            # the bounding box and text
            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

            # include the probability in the label
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
            # display the label and bounding box rectangle on the output frame

            cv2.putText(frame, label, (startX, startY - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
            # frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

        # show the output frame
        cv2.imshow("Frame", frame)
```

```python
        # show the output frame
        cv2.imshow("Frame", frame)

        # FRAME_WINDOW.image(frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

    cv2.destroyAllWindows()
    vs.stop()
```

3.3. Web app

```python
import streamlit as st
from PIL import Image, ImageEnhance
import numpy as np
import cv2
import os
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import detect_mask_image
from detect_mask_video import Load_WebCam
st.set_page_config(page_title='Face Mask Detection using Resnet50/MobileNet_v2', page_icon='😷', layout='centered', initia


def local_css(file_name):
    """ Method for reading styles.css and applying necessary changes to HTML"""
    with open(file_name) as f:
        st.markdown(f'<style>{f.read()}</style>', unsafe_allow_html=True)


def mask_image():
    global RGB_img
    print("[INFO] loading face detector model...")
    prototxtPath = os.path.sep.join(["face_detector", "deploy.prototxt"])
    weightsPath = os.path.sep.join(["face_detector",
                                    "res10_300x300_ssd_iter_140000.caffemodel"])
    net = cv2.dnn.readNet(prototxtPath, weightsPath)

    # load the face mask detector model
    print("[INFO] loading face mask detector model...")
    model = load_model("mask_detector.model")

    image = cv2.imread("./images/out.jpg")
    (h, w) = image.shape[:2]
```

```python
    blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
                                 (104.0, 177.0, 123.0))

    print("[INFO] computing face detections...")
    net.setInput(blob)
    detections = net.forward()

    # loop over the detections
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]

        if confidence > 0.5:
            # compute the (x, y)-coordinates of the bounding box

            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # ensure the bounding boxes fall within the dimensions of the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            face = image[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)
            face = np.expand_dims(face, axis=0)

            (mask, withoutMask) = model.predict(face)[0]

            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
```

```
                label = "Mask" if mask > withoutMask else "No Mask"
                color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

                label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

                cv2.putText(image, label, (startX, startY - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
                RGB_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    mask_image()

    # ----------------------WebCam-----------------------------
    # load our serialized face detector model from disk
    print("[INFO] loading face detector model...")
    prototxtPath = os.path.sep.join(["face_detector", "deploy.prototxt"])
    weightsPath = os.path.sep.join(["face_detector",
                                    "res10_300x300_ssd_iter_140000.caffemodel"])
    faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

    # load the face mask detector model from disk
    print("[INFO] loading face mask detector model...")
    maskNet = load_model("mask_detector.model")

    # initialize the video stream and allow the camera sensor to warm up
    print("[INFO] starting video stream...")
```

```
def mask_detection():
    local_css("css/styles.css")
    st.markdown('<h1 align="center">😷 Face Mask Detection</h1>', unsafe_allow_html=True)
    activities = ["Image", "Webcam"]
    st.set_option('deprecation.showfileUploaderEncoding', False)
    st.sidebar.markdown("# Mask Detection on?")
    choice = st.sidebar.selectbox("Choose among the given options:", activities)

    if choice == 'Image':
        st.markdown('<h2 align="center">Detection on Image</h2>', unsafe_allow_html=True)
        st.markdown("### Upload your image here ⬇")
        image_file = st.file_uploader("", type=['png', 'jpeg', 'jpg'],  accept_multiple_files=False)  # upload image
        if image_file is not None:
            our_image = Image.open(image_file)  # making compatible to PIL
            im = our_image.save('./images/out.jpg')
            saved_image = st.image(image_file, caption='', use_column_width=True)
            st.markdown('<h3 align="center">Image uploaded successfully!</h3>', unsafe_allow_html=True)
            if st.button('Process'):
                st.image(RGB_img, use_column_width=True)

    if choice == 'Webcam':
        st.markdown('<h2 align="center">Detection on Webcam</h2>',
                    unsafe_allow_html=True)
        if st.sidebar.button('Start tracking'):
            Load_WebCam(faceNet, maskNet)
        st.sidebar.markdown(
            '<h3 style="text-align: center; color: red; font-style: italic; ">please press the Q(quit) key to turn off the
mask_detection()
```

B. Object detection with YOLOv5 (SOURCE CODE)

1. Training model using CoLab:

yolov5-training.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:36 AM

Comment Share

+ Code + Text Connect Editing

```
[ ] ## importing required libraries
    import os
    import shutil
    import random
```

```
[ ] ## connecting to the google drive

    from google.colab import drive
    drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
[ ] # ## unzipping the data
    # !unzip /content/drive/MyDrive/face_mask_detection/face_mask_detection_yolov5.zip
```

```
[ ] train_path = "/content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/train/images"
    val_path = "/content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/val/images"
    test_path = "/content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test"
```

▾ Clone the yolov5 repo from https://github.com/ultralytics/yolov5.git

```
[ ] %cd /content/drive/MyDrive/Face_mask_detection_yolov5
```

```
    /content/drive/MyDrive/Face_mask_detection_yolov5
```

```
[ ] !git clone https://github.com/ultralytics/yolov5.git
```

+ Code + Text Connect Editing

```
### change the dir to yolov5
%cd /content/drive/MyDrive/Face_mask_detection_yolov5/yolov5
```

```
/content/drive/MyDrive/Face_mask_detection_yolov5/yolov5
```

```
[ ] ### install all requirements

    !pip install -r requirements.txt
```

```
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
     |████████████████████████████████| 596 kB 8.0 MB/s
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 10)) (2.23.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 11)) (1.4.1)
Requirement already satisfied: torch>=1.7.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 12)) (1.11.0+cu113)
Requirement already satisfied: torchvision>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 13)) (0.12.0+cu113)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 14)) (4.64.0)
Requirement already satisfied: protobuf<=3.20.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 15)) (3.17.3)
Requirement already satisfied: tensorboard>=2.4.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 18)) (2.8.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 22)) (1.3.5)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 23)) (0.11.2)
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 35)) (5.5.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 36)) (5.4.8)
Collecting thop
  Downloading thop-0.1.0.post2206102148-py3-none-any.whl (15 kB)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (1.4.3)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (3.0.9)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (0.11.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r requirements.txt (line 10)) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r requirements.txt (line 10)) (2022.6.15)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r requirements.txt (line 10)) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r requirements.txt (line 10)) (2.10)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch>=1.7.0->-r requirements.txt (line 12)) (4.1.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-packages (from protobuf<=3.20.1->-r requirements.txt (line 15)) (1.15.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r requirements.txt (line 18)) (0.37.1)
```

+ Code + Text

Connect ▾ | Editing

Download pre-trained weights

- yolov5l.pt: https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5l.pt
- yolov5m.pt: https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5m.pt
- yolov5s.pt: https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.pt
- for other models: https://github.com/ultralytics/yolov5/releases

```
[ ] !wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.pt
```

```
--2022-06-19 13:57:16--  https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.pt
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/264818686/eab38592-7168-4731-bdff-ad5ede2002be?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2
--2022-06-19 13:57:16--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/264818686/eab38592-7168-4731-bdff-ad5ede2002be?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWN
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.111.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 14698491 (14M) [application/octet-stream]
Saving to: 'yolov5s.pt.3'

yolov5s.pt.3        100%[===================>]  14.02M  43.1MB/s    in 0.3s

2022-06-19 13:57:17 (43.1 MB/s) - 'yolov5s.pt.3' saved [14698491/14698491]
```

```
[ ] !wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5l.pt
```

```
--2022-06-12 17:33:10--  https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5l.pt
Resolving github.com (github.com)... 52.69.186.44
Connecting to github.com (github.com)|52.69.186.44|:443... connected.
```

+ Code + Text

Connect ▾ | Editing

Training the model

```
!python train.py --img 416 --batch 8 --epochs 50 --data /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/data.yaml --weights /content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/yolov5x.pt
```

```
  2                -1  4    309120  models.common.C3                        [160, 160, 4]
  3                -1  1    461440  models.common.Conv                      [160, 320, 3, 2]
  4                -1  8   2259200  models.common.C3                        [320, 320, 8]
  5                -1  1   1844480  models.common.Conv                      [320, 640, 3, 2]
  6                -1 12  13125120  models.common.C3                        [640, 640, 12]
  7                -1  1   7375360  models.common.Conv                      [640, 1280, 3, 2]
  8                -1  4  19676160  models.common.C3                        [1280, 1280, 4]
  9                -1  1   4099840  models.common.SPPF                      [1280, 1280, 5]
 10                -1  1    820480  models.common.Conv                      [1280, 640, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12          [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  4   5332480  models.common.C3                        [1280, 640, 4, False]
 14                -1  1    205440  models.common.Conv                      [640, 320, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16          [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  4   1335040  models.common.C3                        [640, 320, 4, False]
 18                -1  1    922240  models.common.Conv                      [320, 320, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  4   4922880  models.common.C3                        [640, 640, 4, False]
 21                -1  1   3687680  models.common.Conv                      [640, 640, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  4  19676160  models.common.C3                        [1280, 1280, 4, False]
 24      [17, 20, 23]  1     47103  models.yolo.Detect                      [2, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [320, 640, 1280]]
Model summary: 567 layers, 86224543 parameters, 86224543 gradients

Transferred 739/745 items from /content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/yolov5x.pt
AMP: checks passed ✅
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 123 weight (no decay), 126 weight, 126 bias
albumentations: version 1.0.3 required by YOLOv5, but version 0.1.12 is currently installed
train: Scanning '/content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/train/labels.cache' images and labels... 678 found, 0 missing, 19 empty, 0 corrupt: 100% 678/678 [00:00<?, ?it/s]
```

+ Code + Text

Connect ▾ Editing

Analysing the results using tensorboard

```
[ ] # Start tensorboard
    # Launch after you have started training
    # logs save in the folder "runs"
    %load_ext tensorboard
    %tensorboard --logdir runs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 2269), started 0:00:37 ago. (Use '!kill 2269' to kill it.)
```

+ Code + Text

Connect ▾ Editing

inference or detection on new images

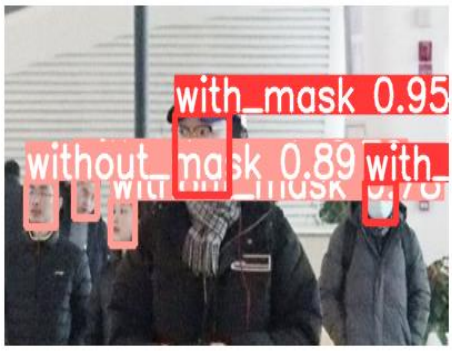
```
[ ] ## inference or detection on new images
    !python detect.py --source /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test --weights  /content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/r
```

```
detect: weights=['/content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/runs/train/exp/weights/last.pt'], source=/content/drive/MyDrive/Face_mask_detection_yol
YOLOv5 🚀 v6.1-250-g6adc53b Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla P100-PCIE-16GB, 16281MiB)

Fusing layers...
Model summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
image 1/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss70.png: 256x416 2 with_masks, 3 without_masks, Done. (0.016s)
image 2/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss71.png: 256x416 5 with_masks, 1 without_mask, Done. (0.011s)
image 3/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss72.png: 416x320 1 without_mask, Done. (0.016s)
image 4/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss73.png: 320x416 3 with_masks, Done. (0.016s)
image 5/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss74.png: 320x416 26 with_masks, Done. (0.012s)
image 6/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss75.png: 224x416 13 with_masks, 1 without_mask, Done. (0.016s)
image 7/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss76.png: 288x416 3 with_masks, Done. (0.016s)
image 8/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss77.png: 256x416 3 with_masks, Done. (0.013s)
image 9/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss78.png: 416x320 1 with_mask, Done. (0.014s)
image 10/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss79.png: 416x416 5 with_masks, Done. (0.014s)
image 11/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss80.png: 256x416 1 with_mask, Done. (0.012s)
image 12/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss81.png: 256x416 3 with_masks, Done. (0.011s)
image 13/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss82.png: 256x416 6 with_masks, Done. (0.013s)
image 14/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss83.png: 320x416 2 with_masks, 1 without_mask, Done. (0.013s)
image 15/28 /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/test/maksssksksss84.png: 288x416 3 with_masks, Done. (0.012s)
```

+ Code + Text Connect Editing

display result images

```
[ ] #display result images

    import glob
    from IPython.display import Image, display

    for imageName in glob.glob('/content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/runs/detect/exp/*.png'): #assuming JPG
        display(Image(filename=imageName))
        print("\n")
```



download weights

```
[ ] #export your model's weights for future use
    from google.colab import files
    files.download('./runs/train/exp/weights/last.pt')
```

Retrain the model from last saved weight or from a perticular weight

```
[ ] !python train.py --img 416 --batch 8 --epochs 150 --data /content/drive/MyDrive/Face_mask_detection_yolov5/yolo_data/data.yaml --weights /content/drive/MyDrive/F
```

```
train: weights=/content/drive/MyDrive/Face_mask_detection_yolov5/yolov5/runs/train/exp2-yolov5l/weights/last.pt, cfg=, data=/content/drive/MyDrive/Face_mask_det
github: ⚠ YOLOv5 is out of date by 8 commits. Use `git pull` or `git clone https://github.com/ultralytics/yolov5` to update.
YOLOv5 🚀 v6.1-250-g6adc53b Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla P100-PCIE-16GB, 16281MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/

                from  n    params  module                                  arguments
  0               -1  1      7040  models.common.Conv                      [3, 64, 6, 2, 2]
  1               -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  2               -1  3    156928  models.common.C3                        [128, 128, 3]
```

2. Detection:

2.1 Requirement for web app:

requirements.txt

```
# pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# Logging -------------------------------------
tensorboard>=2.4.1

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

thop

# Extras ----------
streamlit
psutil
```

2.2 Detect on image:

```python
from graphs import bbox_rel, draw_boxes
from deep_sort_pytorch.deep_sort import DeepSort
from deep_sort_pytorch.utils.parser import get_config
from yolov5.utils.torch_utils import select_device, time_sync
from yolov5.utils.plots import Annotator, colors, save_one_box, plot_one_box
from yolov5.utils.general import set_logging
from yolov5.utils.general import (LOGGER, check_file, check_img_size, check_imshow, check_requirements, colorstr,
                                  increment_path, non_max_suppression, print_args, scale_coords, strip_optimizer, xyxy2xywh)
from yolov5.utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from yolov5.models.common import DetectMultiBackend
import argparse
import time
from pathlib import Path
from collections import Counter

import cv2
import torch
import torch.backends.cudnn as cudnn
import os
import sys
import psutil
import subprocess
# Path for internal module without changing base
sys.path.insert(0, './yolov5')


FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

```

```python
def get_gpu_memory():
    result = subprocess.check_output(
        [
            'nvidia-smi', '--query-gpu=memory.used',
            '--format=csv,nounits,noheader'
        ], encoding='utf-8')
    gpu_memory = [int(x) for x in result.strip().split('\n')]
    return gpu_memory[0]


@torch.no_grad()
def detectI(weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
            source=ROOT / 'yolov5/data/images',  # file/dir/URL/glob, 0 for webcam
            data=ROOT / 'yolov5/data/coco128.yaml',  # dataset.yaml path
            stframe=None,
            kpi2_text="", kpi3_text="",
            js1_text="", js2_text="", js3_text="",
            imgsz=(416, 416),  # inference size (height, width)
            conf_thres=0.25,  # confidence threshold
            iou_thres=0.45,  # NMS IOU threshold
            max_det=1000,  # maximum detections per image
            device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
            view_img=False,  # show results
            save_txt=False,  # save results to *.txt
            save_conf=False,  # save confidences in --save-txt labels
            save_crop=False,  # save cropped prediction boxes
            nosave=False,  # do not save images/videos
            classes=None,  # filter by class: --class 0, or --class 0 2 3
            agnostic_nms=False,  # class-agnostic NMS
            augment=False,  # augmented inference
            visualize=False,  # visualize features
```

```python
            update=False,  # update all models
            project=ROOT / 'runs/detect',  # save results to project/name
            name='exp',  # save results to project/name
            exist_ok=False,  # existing project/name ok, do not increment
            line_thickness=1,  # bounding box thickness (pixels)
            hide_labels=False,  # hide labels
            hide_conf=False,  # hide confidences
            half=False,  # use FP16 half-precision inference
            dnn=False,
            config_deepsort="deep_sort_pytorch/configs/deep_sort.yaml"  # Deep Sort configuration
            ):
    save_img = not nosave and not source.endswith(
        '.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # initialize deepsort
    cfg = get_config()
    cfg.merge_from_file(config_deepsort)
    deepsort = DeepSort(cfg.DEEPSORT.REID_CKPT,
                        max_dist=cfg.DEEPSORT.MAX_DIST, min_confidence=cfg.DEEPSORT.MIN_CONFIDENCE,
                        nms_max_overlap=cfg.DEEPSORT.NMS_MAX_OVERLAP, max_iou_distance=cfg.DEEPSORT.MAX_IOU_DISTANCE,
                        max_age=cfg.DEEPSORT.MAX_AGE, n_init=cfg.DEEPSORT.N_INIT, nn_budget=cfg.DEEPSORT.NN_BUDGET,
                        use_cuda=True)

    # Directories
    save_dir = increment_path(Path(project) / name,
                              exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
                                                          exist_ok=True)  # make dir

```

```
    set_logging()
    device = select_device(device)
    half &= device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data)
    stride, names, pt, jit, onnx, engine = model.stride, model.names, model.pt, model.jit, model.onnx, model.engine
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Half
    # FP16 supported on limited backends with CUDA
    half &= (pt or jit or onnx or engine) and device.type != 'cpu'
    if pt or jit:
        model.model.half() if half else model.model.float()

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load(
            'weights/resnet101.pt', map_location=device)['model']).to(device).eval()

    # Dataloader
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
```

```
Search (Ctrl+Shift+F)= time.time()

    dt, seen = [0.0, 0.0, 0.0], 0
    mapped_ = dict()
    global_graph_dict = dict()
    for path, im, im0s, vid_cap, s in dataset:
        t1 = time_sync()
        im = torch.from_numpy(im).to(device)
        im = im.half() if half else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
        visualize = increment_path(
            save_dir / Path(path).stem, mkdir=True) if visualize else False
        pred = model(im, augment=augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2

        # NMS
        pred = non_max_suppression(
            pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
        dt[2] += time_sync() - t3
```

```
169         for i, det in enumerate(pred):  # per image
170             seen += 1
171             if webcam:  # batch_size >= 1
172                 p, im0, frame = path[i], im0s[i].copy(), dataset.count
173                 s += f'{i}: '
174             else:
175                 p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)
176 
177             p = Path(p)  # to Path
178             save_path = str(save_dir / p.name)  # im.jpg
179             txt_path = str(save_dir / 'labels' / p.stem) + \
180                 ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
181             s += '%gx%g ' % im.shape[2:]  # print string
182             # normalization gain whwh
183             gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]
184             imc = im0.copy() if save_crop else im0  # for save_crop
185             annotator = Annotator(
186                 im0, line_width=line_thickness, example=str(names))
187 
188             if len(det):
189                 # Rescale boxes from img_size to im0 size
190                 det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()
191 
192                 # Print results
193                 names_ = []
194                 cnt = []
195                 for c in det[:, -1].unique():
196                     n = (det[:, -1] == c).sum()  # detections per class
197                     # add to string
198                     s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "
```

```
199                     names_.append(names[int(c)])
200                     cnt.append(int(n.detach().cpu().numpy()))
201                 mapped_.update(dict(zip(names_, cnt)))
202 
203                 global_graph_dict = Counter(
204                     global_graph_dict) + Counter(mapped_)
205 
206                 bbox_xywh = []
207                 confs = []
208                 # Adapt detections to deep sort input format
209                 for *xyxy, conf, cls in det:
210                     x_c, y_c, bbox_w, bbox_h = bbox_rel(*xyxy)
211                     obj = [x_c, y_c, bbox_w, bbox_h]
212                     bbox_xywh.append(obj)
213                     confs.append([conf.item()])
214 
215                 xywhs = torch.Tensor(bbox_xywh)
216                 confss = torch.Tensor(confs)
217 
218                 # Pass detections to deepsort
219                 outputs = deepsort.update(xywhs, confss, im0)
220 
221                 # draw boxes for visualization
222                 if len(outputs) > 0:
223                     bbox_xyxy = outputs[:, :4]
224                     identities = outputs[:, -1]
225                     draw_boxes(im0, bbox_xyxy, identities)
226 
227                 # Write MOT compliant results to file
228                 if save_txt and len(outputs) != 0:
229                     for j, output in enumerate(outputs):
```

```
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                        with open(f'{txt_path}.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                        annotator.box_label(xyxy, label, color=colors(c, True))
                    if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

            else:
                deepsort.increment_ages()

            # Print time (inference + NMS)
            print(f'{s}Done. ({t2 - t1:.3f}s)')

            # Stream results
            if view_img:
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
```

```
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                            save_path += '.mp4'
                        vid_writer = cv2.VideoWriter(
                            save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer.write(im0)

    js1_text.write(str(psutil.virtual_memory()[2])+"%")
    js2_text.write(str(psutil.cpu_percent())+'%')
    js3_text.write(str(get_gpu_memory())+' MB')
    stframe.image(im0, channels="BGR", use_column_width=True)

    kpi3_text.write(global_graph_dict)

    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        print(f"Results saved to {save_dir}{s}")

    if update:
        strip_optimizer(weights)  # update model (to fix SourceChangeWarning)

```

2.3 Detect on video / webcam:

```python
import argparse
import time
from pathlib import Path
import streamlit as st
import cv2
import torch
import torch.backends.cudnn as cudnn
import os
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
sys.path.insert(0, './yolov5') # Path for internal module without changing base
import numpy as np
from yolov5.models.common import DetectMultiBackend
from yolov5.utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from yolov5.utils.general import (LOGGER, check_file, check_img_size, check_imshow, check_requirements, colorstr,
                                  increment_path, non_max_suppression, print_args, scale_coords, strip_optimizer, xyxy2xywh)
from yolov5.utils.general import set_logging
from yolov5.utils.plots import Annotator, colors, save_one_box, plot_one_box
from yolov5.utils.torch_utils import select_device, time_sync


from deep_sort_pytorch.utils.parser import get_config
from deep_sort_pytorch.deep_sort import DeepSort

from graphs import bbox_rel,draw_boxes
from collections import Counter

import psutil
import subprocess

```

```python
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

def get_gpu_memory():
    result = subprocess.check_output(
        [
            'nvidia-smi', '--query-gpu=memory.used',
            '--format=csv,nounits,noheader'
        ], encoding='utf-8')
    gpu_memory = [int(x) for x in result.strip().split('\n')]
    return gpu_memory[0]

@torch.no_grad()
def detect(weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
        source=ROOT / 'yolov5/data/images',  # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'yolov5/data/coco128.yaml',  # dataset.yaml path
        stframe=None,
        #stgraph=None,
        kpi1_text="",
        kpi2_text="", kpi3_text="",
        js1_text="",js2_text="",js3_text="",
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
```

```
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=2,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,
        display_labels=False,
        config_deepsort="deep_sort_pytorch/configs/deep_sort.yaml", #Deep Sort configuration
        conf_thres_drift = 0.75,
        save_poor_frame__ = False,
        inf_ov_1_text="", inf_ov_2_text="",inf_ov_3_text="", inf_ov_4_text="",
        fps_warn="",fps_drop_warn_thresh=8
        ):
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    ## initialize deepsort
    cfg = get_config()
    cfg.merge_from_file(config_deepsort)
    deepsort = DeepSort(cfg.DEEPSORT.REID_CKPT,
                        max_dist=cfg.DEEPSORT.MAX_DIST, min_confidence=cfg.DEEPSORT.MIN_CONFIDENCE,
                        nms_max_overlap=cfg.DEEPSORT.NMS_MAX_OVERLAP, max_iou_distance=cfg.DEEPSORT.MAX_IOU_DISTANCE,
                        max_age=cfg.DEEPSORT.MAX_AGE, n_init=cfg.DEEPSORT.N_INIT, nn_budget=cfg.DEEPSORT.NN_BUDGET,
                        use_cuda=True)


    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir
    if save_poor_frame__:
        try:
            os.mkdir("drift_frames")
        except:
            print("Folder exists, overwriting...")

    # Initialize
    set_logging()
    device = select_device(device)
    half &= device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data)
    stride, names, pt, jit, onnx, engine = model.stride, model.names, model.pt, model.jit, model.onnx, model.engine
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Half
    half &= (pt or jit or onnx or engine) and device.type != 'cpu'  # FP16 supported on limited backends with CUDA
    if pt or jit:
        model.model.half() if half else model.model.float()

    # Second-stage classifier
    classify = False
```

```
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']).to(device).eval()

    # Dataloader
    if webcam:
        #view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1  # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    t0 = time.time()

    dt, seen = [0.0, 0.0, 0.0], 0
    prev_time = time.time()
    selected_names = names.copy()
    global_graph_dict = dict()
    global_drift_dict = dict()
    test_drift = []
    frame_num = -1
    poor_perf_frame_counter=0
    mapped_ = dict()
    min_FPS = 10000
    max_FPS = -1
    for path, im, im0s, vid_cap, s in dataset:
        frame_num = frame_num+1
        t1 = time_sync()
```

```
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
        pred = model(im, augment=augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2

        # NMS
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
        dt[2] += time_sync() - t3

        # Process predictions
        class_count = 0

        drift_dict = dict()

        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
```

```
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness, example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                names_ = []
                cnt = []
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
                    names_.append(names[int(c)])
                    cnt.append(int(n.detach().cpu().numpy()))
                mapped_.update(dict(zip(names_, cnt)))

                global_graph_dict = Counter(global_graph_dict) + Counter(mapped_)

                bbox_xywh = []
                confs = []
                # Adapt detections to deep sort input format
                for *xyxy, conf, cls in det:
                    x_c, y_c, bbox_w, bbox_h = bbox_rel(*xyxy)
                    obj = [x_c, y_c, bbox_w, bbox_h]
                    bbox_xywh.append(obj)
                    confs.append([conf.item()])
                    # print("conf : {0}, conf_t : {1}".format(conf, conf_thres))
                    if conf<conf_thres_drift:
                        if names[int(cls)] not in test_drift:
```

```
                            test_drift.append(names[int(cls)])
                        if save_poor_frame__:
                            cv2.imwrite("drift_frames/frame_{0}.png".format(frame_num), im0)
                            poor_perf_frame_counter+=1
                    # print(type(conf_thres))

                xywhs = torch.Tensor(bbox_xywh)
                confss = torch.Tensor(confs)

                # Pass detections to deepsort
                outputs = deepsort.update(xywhs, confss, im0)

                # draw boxes for visualization
                if len(outputs) > 0:
                    # print("Outputs :", outputs)
                    bbox_xyxy = outputs[:, :4]
                    identities = outputs[:, -1]
                    draw_boxes(im0, bbox_xyxy, identities)

                # Write MOT compliant results to file
                if save_txt and len(outputs) != 0:
                    for j, output in enumerate(outputs):
                        bbox_left = output[0]
                        bbox_top = output[1]
                        bbox_w = output[2]
                        bbox_h = output[3]
                        identity = output[-1]
                        with open(txt_path, 'a') as f:
                            f.write(('%g ' * 10 + '\n') % (frame_idx, identity, bbox_left,
                                                           bbox_top, bbox_w, bbox_h, -1, -1, -1, -1))  # label format

```

```
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                    with open(txt_path + '.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img or display_labels:  # Add bbox to image
                    c = int(cls)  # integer class
                    label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                    plot_one_box(xyxy, im0, label=label, color=colors(c, True), line_thickness=line_thickness)
                    if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

        else:
            deepsort.increment_ages()

        # Stream results
        if view_img:
            cv2.imshow(str(p), im0)
            cv2.waitKey(1)  # 1 millisecond

        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
            else:  # 'video' or 'stream'
                if vid_path != save_path:  # new video
                    vid_path = save_path
                    if isinstance(vid_writer, cv2.VideoWriter):
                        vid_writer.release()  # release previous video writer
                    if vid_cap:  # video
```

```
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                    else:  # stream
                        fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path += '.mp4'
                    vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer.write(im0)

            curr_time = time.time()
            fps_ = curr_time - prev_time
            fps_ = round(1/round(fps_, 3),1)
            prev_time = curr_time

        js1_text.write(str(psutil.virtual_memory()[2])+"%")
        js2_text.write(str(psutil.cpu_percent())+'%')
        js3_text.write(str(get_gpu_memory())+' MB')

        kpi1_text.write(str(fps_)+' FPS')
        if fps_ < fps_drop_warn_thresh:
            fps_warn.warning(f"FPS dropped below {fps_drop_warn_thresh}")
        kpi2_text.write(mapped_)
        kpi3_text.write(global_graph_dict)

        inf_ov_1_text.write(test_drift)
        inf_ov_2_text.write(poor_perf_frame_counter)

        if fps_<min_FPS:
            inf_ov_3_text.write(fps_)
            min_FPS = fps_
        if fps_>max_FPS:
```

```
320             inf_ov_4_text.write(fps_)
321             max_FPS = fps_
322
323         stframe.image(im0, channels="BGR", use_column_width=True)
324
325     if save_txt or save_img:
326         s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
327         print(f"Results saved to {save_dir}{s}")
328
329     if update:
330         strip_optimizer(weights)
331
332     if vid_cap:
333         vid_cap.release()
334
```

2.4. Web app:

app.py > ...

```
1   import cv2
2   import streamlit as st
3   from deep_list import *
4   from detect_sort import detectI
5   import torch
6   from PIL import Image
7
8   # Setting custom Page Title and Icon with changed layout and sidebar state
9   st.set_page_config(page_title='Face Mask Detection with Yolov5', page_icon='😷', layout='centered', initial_sidebar_state='expanded')
10
11  def main():
12      st.markdown("<h1 style='text-align: center; color: black; '>😷FACE MASK DETECTION😷 </h1>", unsafe_allow_html=True)
13      st.markdown("<h2 style='text-align: center; color: black;font-style: italic;'> --USING YOLOv5--</h2>", unsafe_allow_html=True)
14      inference_msg = st.empty()
15      st.sidebar.title("Configuration")
16
17      input_source = st.sidebar.radio(
18          "Select input source",
19          ('Local image', 'Webcam', 'Local video'))
20
21      # -----------------------------------------------------
22      if input_source == "Webcam" or input_source == "Local video":
23          conf_thres = st.sidebar.text_input(
24              "Class confidence threshold", "0.25")
25
26          conf_thres_drift = st.sidebar.text_input(
27              "Class confidence threshold for drift dectection", "0.75")
28
29          fps_drop_warn_thresh = st.sidebar.text_input(
30              "FPS drop warning threshold", "60")
31          # Video
32          save_output_video = st.sidebar.radio(
33              "Save output video?", ('Yes', 'No'))
```

```
        if save_output_video == 'Yes':
            nosave = False
            display_labels = False
        else:
            nosave = True
            display_labels = True

        # Frame
        save_poor_frame = st.sidebar.radio(
            "Save poor performing frames?", ('Yes', 'No'))
        if save_poor_frame == "Yes":
            save_poor_frame__ = True
        else:
            save_poor_frame__ = False

    # ------------------------- LOCAL IMAGE -----------------------------
    if input_source == "Local image":
        # Image
        save_output_Image = st.sidebar.radio(
            "Save output Image?", ('Yes', 'No'))
        if save_output_Image == 'Yes':
            nosave = False
            display_labels = False
        else:
            nosave = True
            display_labels = True

        image = st.sidebar.file_uploader(
            "Select input image", type=['png', 'jpeg', 'jpg'], accept_multiple_files=False)

        if st.sidebar.button("Start tracking"):
```

```
            stframe = st.empty()

            st.subheader("Inference Stats")
            kpi3, kpi1, kpi2 = st.columns(3)

            st.subheader("System Stats")
            js1, js2, js3 = st.columns(3)

            # Updating Inference results

            with kpi3:
                st.markdown("**Total Detected objects**")
                kpi3_text = st.markdown("0")
            # Updating System stats

            with js1:
                st.markdown("**Memory usage**")
                js1_text = st.markdown("0")

            with js2:
                st.markdown("**CPU Usage**")
                js2_text = st.markdown("0")

            with js3:
                st.markdown("**GPU Memory Usage**")
                js3_text = st.markdown("0")

            detectI(weights="last.pt", source=image.name, stframe=stframe, kpi3_text=kpi3_text, js1_text=js1_text, js2_text=js2_text, js3_text=js
                    device='0', nosave=nosave)
```

```python
    # ------------------------- LOCAL VIDEO -----------------------------
    if input_source == "Local video":

        video = st.sidebar.file_uploader("Select input video", type=[
                                         "mp4", "avi"], accept_multiple_files=False)

        if st.sidebar.button("Start tracking"):

            stframe = st.empty()

            st.subheader("Inference Stats")
            kpi1, kpi2, kpi3 = st.columns(3)

            st.subheader("System Stats")
            js1, js2, js3 = st.columns(3)

            # Updating Inference results

            with kpi1:
                st.markdown("**Frame Rate**")
                kpi1_text = st.markdown("0")
                fps_warn = st.empty()

            with kpi2:
                st.markdown("**Detected objects in curret Frame**")
                kpi2_text = st.markdown("0")

            with kpi3:
                st.markdown("**Total Detected objects**")
                kpi3_text = st.markdown("0")
```

```python
            with js1:
                st.markdown("**Memory usage**")
                kpi1_text = st.markdown("0")

            with js2:
                st.markdown("**CPU Usage**")
                js2_text = st.markdown("0")

            with js3:
                st.markdown("**GPU Memory Usage**")
                js3_text = st.markdown("0")

            st.subheader("Inference Overview")
            inf_ov_1, inf_ov_2, inf_ov_3, inf_ov_4 = st.columns(4)

            with inf_ov_1:
                st.markdown(
                    "**Poor performing classes (Conf < {0})**".format(conf_thres_drift))
                inf_ov_1_text = st.markdown("0")

            with inf_ov_2:
                st.markdown("**No. of poor peforming frames**")
                inf_ov_2_text = st.markdown("0")

            with inf_ov_3:
                st.markdown("**Minimum FPS**")
                inf_ov_3_text = st.markdown("0")

            with inf_ov_4:
                st.markdown("**Maximum FPS**")
                inf_ov_4_text = st.markdown("0")
```

```
            detect(weights="last.pt", source=video.name, stframe=stframe, kpi1_text=kpi1_text, kpi2_text=kpi2_text, kpi3_text=kpi3_text, js1_text
                  conf_thres_drift=float(conf_thres_drift), save_poor_frame__=save_poor_frame__, inf_ov_1_text=inf_ov_1_text, inf_ov_2_text=inf_

            inference_msg.success("Inference Complete!")

    # ------------------------- WEBCAM --------------------------------
    if input_source == "Webcam":

        if st.sidebar.button("Start tracking"):

            stframe = st.empty()

            st.subheader("Inference Stats")
            kpi1, kpi2, kpi3 = st.columns(3)

            st.subheader("System Stats")
            js1, js2, js3 = st.columns(3)

            # Updating Inference results

            with kpi1:
                st.markdown("**Frame Rate**")
                kpi1_text = st.markdown("0")
                fps_warn = st.empty()

            with kpi2:
                st.markdown("**Detected objects in curret Frame**")
                kpi2_text = st.markdown("0")

            with kpi3:
                st.markdown("**Total Detected objects**")
```

app.py > ...

```
            with js1:
                st.markdown("**Memory usage**")
                js1_text = st.markdown("0")

            with js2:
                st.markdown("**CPU Usage**")
                js2_text = st.markdown("0")

            with js3:
                st.markdown("**GPU Memory Usage**")
                js3_text = st.markdown("0")

            st.subheader("Inference Overview")
            inf_ov_1, inf_ov_2, inf_ov_3, inf_ov_4 = st.columns(4)

            with inf_ov_1:
                st.markdown(
                    "**Poor performing classes (Conf < {0})**".format(conf_thres_drift))
                inf_ov_1_text = st.markdown("0")

            with inf_ov_2:
                st.markdown("**No. of poor peforming frames**")
                inf_ov_2_text = st.markdown("0")

            with inf_ov_3:
                st.markdown("**Minimum FPS**")
                inf_ov_3_text = st.markdown("0")

            with inf_ov_4:
                st.markdown("**Maximum FPS**")
                inf_ov_4_text = st.markdown("0")

            detect(weights="last.pt", source='0', stframe=stframe, kpi1_text=kpi1_text, kpi2_text=kpi2_text, kpi3_text=kpi3_text, js1_text=js1_te
```

```
    torch.cuda.empty_cache()


if __name__ == "__main__":
    try:
        main()
    except SystemExit:
        pass
```