MSS-M-2: CASE STUDY INTELLIGENT SYSTEMS

PROJECT NAME: AI-Driven Adjustment of Public Images and Videos for Compliance to General Data Protection Regulation (GDPR)

FINAL REPORT

Submitted By

DHARMESH PATEL	12203573	dharmesh.patel@stud.th-deg.de
JEMTY JOSE	12202186	jemty.jose@stud.th-deg.de
MUHAMMAD SULEMAN	12203072	muhammad.suleman@stud.th-deg.de
MUHAMMAD AZHAR MEHBOOB	12202335	muhammad.mehboob@stud.th-deg.de

ARTIFICIAL INTELLIGENCE FOR SMART SENSORS/ACTUATORS

UNDER THE GUIDANCE OF PROF. JÜRGEN WITTMANN

Faculty of Applied Natural Science and Industrial Engineering

Deggendorf Institute of Technology, Cham

ACKNOWLEDGEMENT

We would like to express our sincere thanks and profound gratitude to our Project supervisor, Prof. Wittmann Jürgen for considering us worthy enough to do a project under him. His valuable help, guidance, and encouragement have been a constant source of motivation during the project work. We are also deeply indebted to the support of all the faculty members of Applied Natural Sciences and Industrial Engineering, Deggendorf Institute of Technology, during the project. We offer our regards to all who have supported us in any respect during the project.

TABLE OF CONTENTS

ABSTRACT

With the development of technology, the constant use of smartphones and cameras in public places has increased. To maintain personal safety and privacy, there are laws and regulations set by European Union authorities that must be followed when publicly sharing movies/images/videos. These laws preserve the public's right to privacy. To comply with GDPR, a person must obtain consent before using privacy-revealing images. This is a difficult task when dealing with images taken in public. Using a privacy masking tool is helpful in these situations. This shows the importance of our project. We use all modern techniques in our project to ensure that all regulations are followed accordingly. This project is primarily based on building predictive models using artificial intelligence and machine learning to achieve the results required by General Data Protection Regulations. 3 different machine-learning models were developed as part of the requirement. Collected images contain unnecessary objects. Objects that could publicly reveal the identity of any individual were hidden/obfuscated using computer vision techniques. Images were collected as data sets and machine learning models were trained on the data sets. In our project, we use the blurring technique to blur the objects which were not required. An app is developed to enhance the user interface to achieve the required results.

1 INTRODUCTION

As the name of the project refers to "Adjustments of Public Images/Videos Based on Artificial Intelligence to Comply with the General Data Protection Regulation (GDPR)", our project mainly focuses on editing publicly captured images/videos according to the GDPR. The General Data Protection Regulation is a set of regulations and legal guidelines developed by European authorities to protect people's rights to security and privacy. These restrictions apply to the processing of a person's personal data in case of full or partial use. European authorities have created legal and regulatory frameworks called General Data Protection Rules to protect people's rights to security and privacy. These limitations apply to the full or partial use of automation in the processing of an individual's personal data. "Personal data" - any data relating to an identified or identifiable natural person ("data subject"). An identifiable natural person is a person who can be identified directly or indirectly through an identifier, such as a face, passport or license plate [2]. However, our project is carried out under these rules to protect individual privacy. In our project, we mainly focus on three personal identifiers which includes Human faces, Number plates and legal documents such as passport, driver's license or identity card. According to the regulations, there are also other features that cannot be exposed when taking photos in public, such as tattoos, jewellery, etc.

Using artificial intelligence and image processing techniques. We effectively use artificial intelligence and machine learning techniques to create such a model to adapt public images/videos according to GDPR regulations [2]. This document illustrates all the requirements, design phases, and implementation of such a project. To make such an effective model, different sets of image data are needed to train the model and achieve the required results. Such results must comply with GDPR regulations. A machine learning model relies primarily on training data to make predictions or decisions without being specifically programmed. A simple neural network consists of an input layer, a hidden layer, and an output layer. The connections between nodes in a simple neural network are modelled as weights between them. Positive weights reflect excitatory connectivity, while negative weights reflect inhibitory connectivity. But in a neural network, all these weights are combined to control the amplitude of the output.

These neural networks are widely used in predictive modelling where they are trained with a given data set to produce the required results from independent data sets. Our project uses Computer Vision technology, which is used to analyse and process digital images for the classification and recognition of various objects. The "blur" function is used to blur unrelated data sets in an image data set [3].

2 LITERATURE REVIEW

The General Data Protection Regulation came into effect in 2018. Our research project started with data protection rules. All regulations have been read and taken from the official data protection website of Germany and the European Union regulations. Some blogs were also read about the rules and regulations for sharing public images/videos in Germany and other countries of the European Union. All of the identifiers were found on official websites and blogs that should be hidden and obscured according to rules and regulations [5].

The machine learning algorithm was first introduced by AI pioneer and computer scientist Arthur Samuel in the 1950s. Machine learning uses two types of techniques: supervised learning, which trains a model based on known input and output data so that it can predict future outputs, and unsupervised learning, which finds hidden patterns or internal structures in the input data. Our project, it was started with instructions and procedures mentioned in the git hub repository [3]. This repository explained all the requirements and procedures for developing an object recognition project. YOLO machine learning model was originally introduced by Joseph Redmon in Darknet. This is the latest of all the models in use to date.

Object detection is the basis for other important AI vision techniques, such as image classification, image retrieval, or object co-segmentation, which extracts meaningful information from real objects. Developers and engineers are using these technologies to build futuristic machines that deliver food and medicine to our doorsteps. In our project, we used the Gaussian blur technique to hide unwanted objects.

3 PROBLEM STATEMENT

When taking/sharing images/videos publicly, it is very important that the person's identity is not revealed in accordance with general data protection rules. Image processing techniques such as blurring of image objects etc. are used to maintain privacy and hide personal information such as faces and legal documents such as passports and license plates. According to the regulations, anyone can report their identity to the authorities. To avoid such problems and eliminate irrelevant data sets, we use predictive models with modern techniques that can hide each person's personal information that can reveal their identity. Another factor is the hundred percent accuracy of the results of such complex forecasting models. The results of these models are usually not always one hundred percent. That's why we use more than one model to get the most accurate results. In addition, a comparison is made to obtain the best results.

4 PROJECT STEPS

This project is based on the following steps:

1. Outline the General Data Protection Regulations (GDPR) required for capturing public images.
2. Collecting image data and data sets creation.
3. Annotations for collected image data sets.
4. Training Models for Object Recognition with Collected Image Data sets.
5. Blurring of the Objects.
6. App Development of machine-learning models.

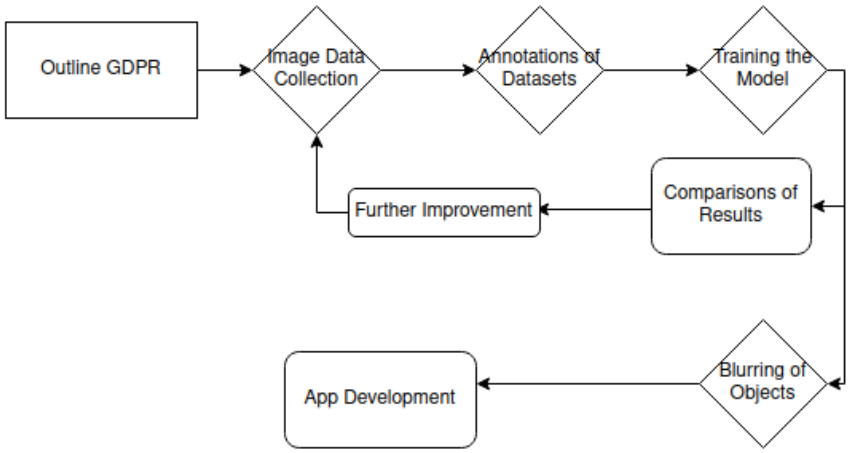


Figure 4.1- Project Flow Chart

5 PROJECT IMPLEMENTATION

5.1 Image Data Collection

The input data set is one of the most important steps in any machine learning model. In our research, data images were collected to train the model. it can be simply understood as the process of acquiring model-specific information to train AI algorithms better so that they can take proactive decisions with autonomy.3,500- 4000 different photos were collected. These images were collected from various websites including Google, Kaggle, etc. The collected images were divided into three categories: images of people along with various objects, images of cars with license plates, and images of official documents such as passports, driver's licenses, and ID cards. These collected images were made using different props, backgrounds, and colour schemes. A live webcam image was also recorded. In addition, videos containing each of the three tags were collected.

5.2 Annotations of Collected Images

The procedure and practice of image labelling to create an image database are known as image annotations. Additionally, these data sets are used as input to a machine learning model used in training. Better image annotations produce better results, increasing accuracy and precision. Due to the importance of the process, image classifications and annotations were completed after image data collection. During this process, the various things that needed to be checked were identified and classified. Only three features were recognized and classified - people, license plates, and legal documents. annotations were made using various Internet tools and software including Robo flow, Label Image, and Make Sense. Annotation marks all identifiers and creates boundaries. All described images were saved as different image data sets. Depending on the type of machine learning model used, data sets were downloaded in different formats. B. XML, YOLO format. These data sets were later used as input to various machine-learning models.

5.3 Training Datasets

Image data sets are primarily used for training algorithms and ultimately for training the model itself. Training data sets to account for 80% of all machine learning data collected and are used to teach models such as neural networks and self-learning [4].

5.4 Test Datasets

Testing the data is important to see how well a model captures the concepts. However, machine learning models are already fed a large amount of training data, which the algorithm is likely to perceive during the testing phase, so the test data set should be significantly different and out of sync with the expected results. In our project, the test data set makes up 20% of the total data [4].

5.5 Validation Datasets

Once the model is trained and tested, we need to add a validation set to ensure the final product is perfect and meets our expectations [4].

5.6 Training of Machine Learning Model

It is a set of commands used to detect a certain type of pattern. Machine learning models are trained on an input data set to identify certain patterns. An algorithm is created to repeat the process with new images to achieve the best learning result from the training data set. Three different machine-learning models were created as a requirement of the project. The programming language was Python. After collecting and annotating image data sets, we trained 3 different machine learning models on image data sets such as SSD Mobile Net, YOLO v5, and YOLO v7. Certain image data sets were also trained in Google Colab using the SSD Mobile Net machine learning model. Google Colab or Collaboratory, from Google Research, is a Jupyter notebook environment to execute

python-based code to build a machine learning or deep learning model. Certain image data sets were also trained on Google Colab for SSD Mobile Net machine learning models.

SSD Mobile Net: It is a convolutional neural network and it is the first TensorFlow portable computer vision model. However, its accuracy is low. But this is the beginning of identifiers training.

YOLO v5: YOLO is a convolutional neural network and is the most used in machine learning models. It has better speed and accuracy compared to SSD Mobile Net, which can process up to 155 frames per second. We trained the data sets with YOLOv5 and the results were achieved [30].

YOLO v7: The latest version of YOLO i-e YOLO v7, was used and the model was trained on it.

5.7 Blurring of Objects

Blurring is an image processing technique that reduces the sharpness of an image. This could be interpreted quite broadly in the context of image analysis - anything that reduces or distorts the detail of an image might apply. Applying a low pass filter, which removes detail occurring at high spatial frequencies, is perceived as a blurring effect. A Gaussian blur is a filter that makes use of a Gaussian kernel. It is often used in computer vision for image processing. Output files are obtained after training the machine learning model. In addition, this file is built with optimization technology so that identifiers can be recognized.

5.8 Comparisons of Different Machine Learning Models

After getting the neural network results, the same procedure is repeated for other neural networks such as YOLO v5 and YOLO v7. We then compared the results. Improvements were made by collecting more images and new data sets.

5.9 App Development

The final part of our project was to develop computer applications to improve the user interface. The Kivy application has been developed. All models and blur techniques were integrated into the application. A user interface and back end were developed for the application.

6 OBJECTIVES

The following are the objectives that this project aims to achieve. We are preparing 3 different machine learning models that can obfuscate/hide personal data from published images/videos according to GDPR. One of the objectives is to make sure that 100% accuracy is achieved in machine learning models. Secondly, it should follow the rules and ensure the privacy and safety of the objects/people in the images/video.

7 OBJECT DETECTION METHODOLOGIES

Object Detection is one of the most popular topics when it comes to computer vision and machine learning. We need to acquire a deep technical knowledge in Deep Learning and use a lot of hardware resources while having a collection of required images. There are multiple object detection models available for us to use in the TensorFlow model zoo which contain a deep stack of layers. And as mentioned before, it requires a lot of technical knowledge and time for them to work. It not only is limited to academia but has a huge potential in real-world business in domains like healthcare, autonomous driving, video surveillance, and vehicle detection. Nowadays, recent advances in deep learning and hardware technology have made the computer vision field a whole lot easier and more intuitive. To initiate object detection, we first need to clarify what kind of object detection we intend to use and what kind of data is required for the training of the data. We have decided to use SSD MobileNet as one of the model frameworks followed by YoloV5 and YoloV7 [7].

8 DATA COLLECTION & DATASET

In reference to machine learning, a dataset is a collection of separate pieces of data that is used to train a neural network(algorithm) to find a predictable pattern in the complete dataset. It is an essential component of a dataset that makes the ML algorithm a product to be used in a business. The data used can be of any form such as images (jpeg, png), video(mp4, avi, wmv), audio(mp3) and different formats such as CSV, excel, TXT file.

To create a reliable model having high precision, we required ample amounts of data to build the ideal dataset. Use of a high volume of images without variety would give us biased or incorrect results. A dataset with variety allows us to analyze hidden patterns and trends which makes our model make conclusions based on it. Machine Learning models follow the concept of Garbage In Garbage Out (GIGO), which states that if low-quality data is used to the train the ML Model, it will deliver a biased and less accurate result [8].

As our project is based on object detection, we take images as our data and for our model to detect images, we have defined the types of images that are needed for the object detection to be detected according to the GDP Regulations [9].These acquired images are annotated and labeled by their specified categories, namely, the faces of individuals, the registration number plates of

vehicles and documents that contain personal identification like Identification Card, Passport and Bank Card. In accordance with the GDP Regulations, we then use the user's data to be detected and blurred.

For the collection of the data we have collected images from the internet sources which include google images**,** Kaggle [10] as well as Roboflow [11]. Kaggle and Roboflow were the most reliable websites that contained not just images but free datasets that were available for the public to use. Some dataset already contained multiple angles of a face and registered traffic number plate so that the face is to be detected at every angle. But the problem arises when we need to collect the angle of the legal documents as we were not able to find the images of the passport from the internet at an angle. For that we used image manipulation techniques using Roboflow which provided us with additional images. To reduce the losses, the faces were manually marked in the legal document.

8.1 General Dataset Preparation

For the Dataset Preparation, we collected the images not just from public sources but also the personal images by accessing webcam especially for the images for capturing faces. To build a dataset, the 'LabelImg' application [12] was used which is a free to use software to build labels on the images and annotate it in formats including XML, CSV and Yolo. But later shifted to Roboflow, which has additional user-friendly options that helped us to collect and manipulate our dataset. We not only use the dataset for training purposes. A dataset is already used to split into several parts to check whether the model is working. This is why a dataset is separated into a training dataset, testing dataset and a validation dataset which is used to train the algorithm into a different type of data and unbiased prediction.

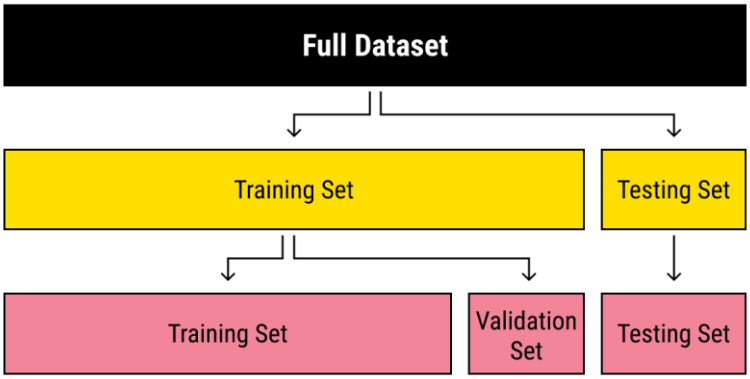


Figure 8.1 Dataset Splitting into Training, Testing and Validation dataset [13]

8.1.1 Dataset of SSD MobileNet

As explained earlier, the variety of images collected from the open sources, as well as through leveraging our webcam were later labelled and annotated in a categorized format. This data

acquisition and collection plays an important role to get a trained model with high precision and least amount of losses.

For MobileNet, we separated our dataset into a training dataset containing 80% of the data and a testing dataset containing 20% of the data. The procedure is used to approximate the performance of the algorithm which predicts on the unused data model, which in return gives an evaluation of performance while being unbiased.

We use 3 classes:

1. Person: Boundary Boxes covers the entire face
2. ‘Licence’: vehicle registration number
3. Legal-Docs: Personal documents of an individual; id card, passport, etc

These classes are marked and labelled on the pictures and later trained.

8.1.2 Dataset of YOLO

A custom data set (mentioned in section 1) was used to train the model. YOLO accepts annotations in a text format (.txt). The name of each object is taken as a class and given a class number like 1,2,3. etc. For example, if we consider the image given below.

Fig 8.2. Illustration of annotations (done with the help of the website makesense.ai[14])

Figure 8.2. shows 6 different objects, namely: Earphones, a Keyboard, a Calculator, a Mobile phone, a Diary, and a Pen. While giving the annotations, each object is given a class number. In the above case, the class numbers given were as follows:

Class 0: Calculator

Class 1: Pen

Class 2: Mobile Phone

Class 3: Ear Phone

Class 4: Keyboard

Class 5: Diary

The annotation is generated in the form of a text file mentioning the class number, x and y coordinates of the centre of the bounding box, width, and height of the bounding box in columns 1 to 4 respectively. An image of the text file is given below:

```
3 0.198458 0.380952 0.114448 0.243506
4 0.487013 0.301587 0.448052 0.477994
0 0.789773 0.347583 0.134740 0.530303
2 0.269481 0.766053 0.243506 0.277778
5 0.522727 0.747114 0.232143 0.326479
1 0.768263 0.818362 0.237825 0.057720
```

Fig 8.3. Image of the annotation file generated for YOLOv7

The coordinates of the boxes are normalized to the dimensions of the object giving them a value between 0 and 1. For example, the first object in fig 8.3. is the earphone, the class number of which was 3. The text file shows the class of the first bounding box as 3 and then the x and y coordinates of the centre of the bounding box, followed by the width and height of the bounding box [15]. The preparation of the dataset is time-consuming as each of the pictures in the dataset has to be annotated in this way.

In our project the three classes used were: Person, Vehicle Registration Number (annotated as Licence), and Legal Documents (Annotated as Legal-Docs).

9 OBJECT DETECTION MODELS

In this project, we used three different object detection models which achieved different results on the same dataset model. The models alongside their description are given below:

9.1 SSD MobileNet

A simple convolution neural network is used to build a model that contains a neural network. This classifies objects into the stated classes by figuring out the similar patterns in each and every image. A model called SSD (Single Shot Detector) MobileNet is used to compute classes and boundary boxes from an input image. As the name suggests, it can be used in mobile devices and acts as a backbone to achieve a detection model [29].

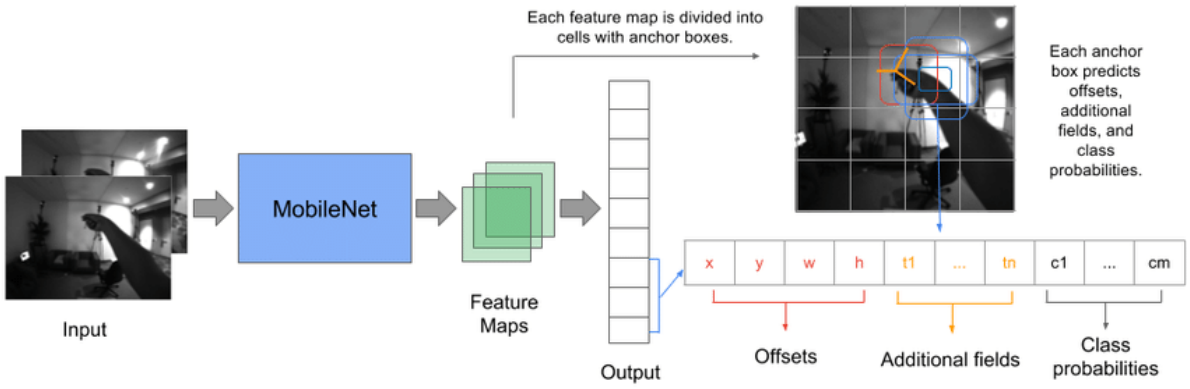


Figure 9.1. Architecture of SSD MobileNet[16]

SSD MobileNet is a CNN technique that creates multiple boundary boxes while instances of the class defined are scored if present. The more the instance of the object category is present, the confidence level inc

The SSD MobileNet Architecture has an organized models built for mobile applications that uses depth-separable convolutions to create a lightweight and small trained neural network; consist of point convolution and depth convolution. In the SSD MobileNet model, one filter is applied by deep convolution in each input channel whereas a 1x1 convolution is generated to build the deep convolutions output. The Deep convolutional has one filter to each input channel. In a depth-separable convolution splits the filtering and the combining layers making it easy to compute and reduces the size of the model [17].

9.2 YOLO

All YOLO models come under the category of single-stage object detectors. In these models, the featurization of image frames is done first. The classes and locations of the objects around which bounding boxes have to be drawn are predicted after combining and mixing these features. Then post-processing is done to achieve its final prediction. This is done using non-maximum suppression (NMS) [18]. NMS is a method for selecting one value out of many overlapping values. Depending on the desired results, the selection criteria can be determined. The criteria can be a probability number or some ratio like intersection over union. Usually, most object detectors create many bounding boxes (windows) around the objects to be detected. Most of the time many of these windows overlap. The purpose of NMS is to bring down these windows to a minimum. Here the intersection of Union (IoU) between prediction boxes is used. If we consider two bounding boxes A & B, the IoU can be written as:

$$IoU = (A \cap B) / (A \cup B)$$

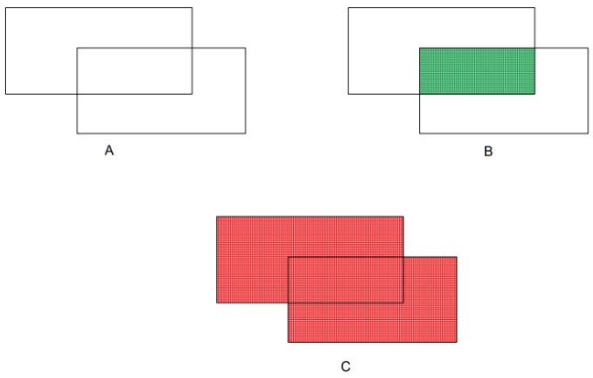


Fig 9.2. Illustrates intersection and Union of two boxes (Created using AutoCAD 2022 education license)

The shaded area (green) in Fig 9.2 (B) represents the intersection of boxes A & B and the shaded area (red) in Fig 9.2 (C) represents the union of Boxes A & B. Thus, the IoU is the green area divided by the red area [19]. In object detection models, the IoU represents the overlap of the predicted bounding box over the true bounding box. Higher IoU shows that the predicted bounding box almost resembles the ground truth.

9.2.1 The YOLO Architecture

The YOLO detection model architecture is given below.

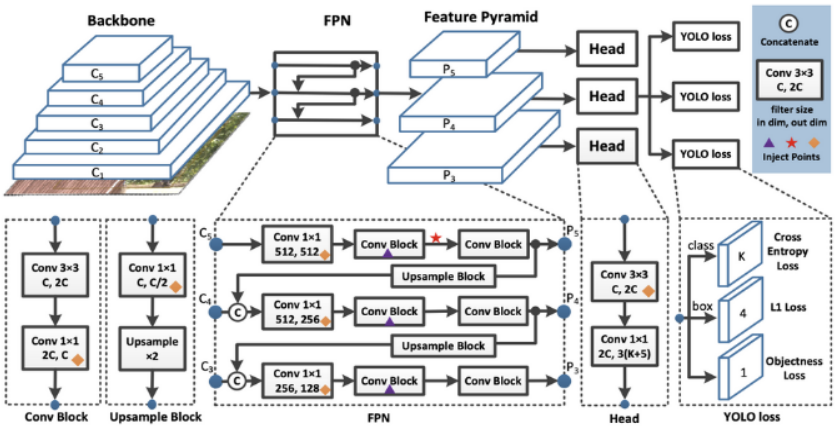


Fig 9.3. YOLO architecture (source [20])

Backbone: The backbone of the YOLO detection model is a CNN that groups pixels from images to form the features.

FPN: The FPN combines and mixes the convolution network layer representations before passing them to the head.

Head: The head makes the bounding boxes and class prediction.

The creators of the YOLOv7 model are WongKinYiu and Alexey Bochkovskiy. The advantage of YOLOv7 over its predecessors is that it predicts more accurately taking the approximately same amount of time as that of the v7 model. To bring about these results, the authors made the following changes in the network and training routines [21].

Extended Efficient Layer Aggregation: In this method, the distance taken by the gradient to back-propagate through the convolution layers in the backbone was reduced. This helped the network to learn more powerfully.

Model Scaling: To keep the model architecture optimal while scaling for different sizes, YOLOv7 scale the network depth and width in concert while compounding layers together. This method is called compound model scaling.

To run the YOLOv7 code on the local computer, an environment was set up. All required libraries were installed in this virtual environment. The YOLOv7 code available in the GitHub repository of WongKinYiu [22], was used in this project for object detection.

9.3 Training the model of SSD MobileNet

The Dataset, labelled and annotated, were divided into two sections; namely: train and test in a ratio of 80:20 respectively. The data in the train folder were used to train the model and the data in the test folder was used to test the trained model's accuracy of training. These files were then converted into a binary TFrecord format for pipelining. We have cloned a TF Object detection API that uses the protobuf files which pipelines the TFrecord for the training as well as the evaluation process [18]. During the training, a batch size of **16** was selected which means the training steps used 16 images were processed in a single gradient or step. A larger batch size, **16**, resulted in a higher RAM usage. But since 4GB VRAM was available, the 1**6** was used. The time taken for the training of the model was **4 ho**urs. Initially, we used 2000 training steps. Training for more training steps can lead to better performance, as the model has more opportunities to learn from the training data. So, for better performance, we changed our dataset to train in 20,000 training steps. Training our model in 30,000 resulted in overfitting [17]. Later, we briefly discussed the result in the report.

9.4 Training the model of YOLO

The data set along with the annotations were divided into two sections, namely: train and validate. The ratio was 80:20 respectively. The data in the train folder were used to train the model and the data in the validate folder was used to validate the accuracy of training given to the model. The model was trained for 100 epochs. In training machine learning models, an epoch refers to a single pass through the entire training dataset. At the end of one epoch, the model goes through every image in the training dataset, and the weights of the model will be updated to try and better predict the labels for the images in the training dataset. The model then validates the performance by trying to predict the class of the object in the validation dataset. Since the model will get more opportunities to learn, training for more epochs can lead to better performance. However, training for too many epochs can lead to overfitting, where the model starts to memorize the training data and performs poorly on unseen data. Therefore, it is important to carefully choose the number of epochs to use when training object detection models using YOLOv7. One way to do this is to monitor the model's performance on the validation set, and train only till a point where the model's performance on the validation set starts to degrade. This can be a sign that the model is starting to overfit the training data [23][24].

In this project, 2548 images distributed over the three classes were used for training. The batch size used was 4 to cope with the available computational facility. A larger batch size like 16 or 32 would have resulted in lesser training time but that would require more GPU VRAM. During the training of YOLOv7, one epoch processed all 2548 images in 4 batches, each batch containing 637

images. The GPU used for this training was NVIDIA RTX 3050 with 4GB VRAM. The time taken for training the model was about 9 hours. A detailed discussion of the results obtained after the training is given below. Whereas for of YOLOv5, a batch size of 8 was used since it was trained on NVIDIA RTX 3060 with 6GB VRAM. The time taken for training the model was about 9 hours. A detailed discussion of the results obtained after the training is given below.

9.4.1 The loss function

In YOLO, the loss function is used to measure the difference between the model's predicted output and the ground truth labels during training. The loss function is used to optimize the model's parameters and improve its ability to make accurate predictions on any data.

There are three main components of the loss function in YOLO: the box loss, the objectness loss the classification loss [25].

9.4.2 Box loss

When a loss function is used to train a model to forecast the bounding boxes of objects in an image, it is referred to as box loss. The box loss aims to maximize the model's capability to precisely forecast the location, dimensions, and form of the bounding boxes enclosing the image's objects. The mean squared error (MSE) between the predicted bounding boxes and the actual bounding boxes is used to construct the box loss function. For each bounding box coordinate (x, y, w, h), the MSE is determined as the sum of the squared variances between the predicted and ground truth values (x and y coordinates of the center of the bounding box, width, and height of the bounding box). The box loss is then the average MSE across all of the bounding boxes in the image.

9.4.3 Objectness loss

The loss function, which is used while training the model to estimate the probability that an object will be present in each grid cell of the feature map, is referred to as objectness loss. The objectness loss seeks to maximize the model's capacity to precisely forecast where objects will appear in the image. The cross-entropy loss between the expected objectness scores and the objectness labels from the actual value is what determines the objectness loss. The objectness labels are binary values (0 or 1) indicating if an object is present or not in each grid cell, and the objectness scores are the odds the model predicted of an object being present in each grid cell. then comes the loss of objectivity. The objectness loss is then the average cross-entropy loss across all of the grid cells in the feature map.

9.4.4 Classification loss

Classification loss refers to the loss function that is used to train a model to predict the class labels of objects in an image. The classification loss optimizes the model's ability to properly estimate the class labels of the image's objects. The classification loss is frequently based on the cross-entropy loss, which is a measurement of the difference between the predicted class labels and the actual

class labels. The cross-entropy loss is calculated as the negative log-likelihood of the actual class labels given the projected class probabilities. The classification loss and box loss are added together and weighted by the hyperparameter alpha to form the overall loss function.

The model's weights and biases are then updated using the losses, which helps to reduce the loss and enhance model performance. By modifying the model's characteristics using backpropagation and optimization approaches like stochastic gradient descent (SGD), the loss function is reduced during training. Predictions will be more accurate as losses decline.

9.4.5 Confusion Matrix

In object detection models, the effectiveness of the object detection method is assessed using a confusion matrix. For instance, it can be used to assess how well an algorithm is performing at correctly detecting the existence of an object of a particular type in an image once it has been trained to do so. For each class that the algorithm is trained to recognize, rows and columns for that class are present in the confusion matrix. The matrix's cells show how frequently each class was successfully and wrongly detected by the algorithm.

For example, if the algorithm is trained to detect pens and apples, the matrix might have a row for pens and a row for apples, and a column for true positives, true negatives, false positives, and false negatives for each class. The confusion matrix can be used to calculate various evaluation metrics, such as precision, recall, and F1 score, which can help to more accurately evaluate the performance of the object detection algorithm. Figure 9.4. Shows the concept of a confusion matrix.

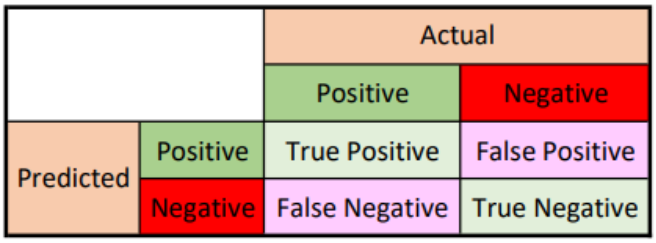


Fig 9.4. Confusion matrix (created using Microsoft excel 2021 Home and student edition)

Here,

True Positive shows the number of times a model correctly classifies a positive class as positive,

False Positive shows the number of times a model incorrectly classifies a negative class as positive,

False Negative shows the number of times a model incorrectly classifies a positive class as negative,

True Negative shows the number of times a model correctly classifies a negative class as negative.

9.4.6 Precision and Recall

Precision and recall are evaluation matrices used to determine the performance of an object detection model.

9.4.6.1 Precision

Precision is a measure of the model's ability to accurately identify the objects in an image. It is calculated as the number of true positive detections (i.e., the number of objects that the model correctly detected) divided by the total number of positive detections (i.e., the number of objects that the model detected, both correctly and incorrectly). A high precision means that the model is good at identifying the objects it detects, but it may miss some objects [26]. The precision is calculated using the following formula:

Precision = True Positive / (True Positive + False Positive)

9.4.6.2 Recall

A recall is the measure of the model's ability to detect all of the objects in an image. It is calculated as the number of true positive detections divided by the total number of objects in the image (i.e., the number of objects that the model should have detected). A high recall means that the model is good at detecting all of the objects, but it may also have a high number of false positive detections [26]. The recall is calculated using the following formula:

Precision = True Positive / (True Positive + False Negative

9.4.7 Mean Average Precision

Mean average precision (mAP) is a measure of the overall quality of the model's predictions, and it is calculated by taking the average of the average precisions at different intersection over union (IoU) thresholds. Average precision (AP) is the average precision of the model when applied to a particular dataset. It is calculated by first calculating the precision and recall at different IoU thresholds, and then plotting the precision as a function of recall. The area under this curve gives the AP.

The mAP is calculated by averaging the APs of the model across a range of different classes or categories. It is useful to compare the performance of different models on the same dataset and evaluate their overall accuracy. In general, a model with a high mAP is considered to be a high-performing model [27]. The mean average precision can be calculated using the following formula:

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i$$

Here, AP_i is the average precision of a particular class and N is the number of classes.

MAP@0.5 means, mean Average Precision at a confidence threshold of 0.5. The confidence threshold is a value that determines whether a prediction should be considered a true positive or a false positive. This value is used to filter out false positive predictions and improve the overall accuracy of the object detection model. The confidence threshold is a value between 0 and 1. A prediction is considered a true positive if the model is confident (i.e., the predicted probability of the object is

greater than the confidence threshold) that an object is present in the image. If the model is not confident (i.e., the predicted probability of the object is less than the confidence threshold) that an object is present in the image, the prediction is considered a false positive [27]. mAP@0.5:0.95 is calculated by averaging the average precision (AP) for the three object classes for all thresholds from 0.5 to 0.95 with a step size of 0.05.

10 RESULTS OF OBJECT DETECTION MODELS

10.1 Result of SSD MobileNet:

After the training steps were completed, the best parameters were computed and were used for object detection. This trained model was used to detect still images and real-time object detection using a webcam. As seen from the image below, the model detected images with a 0.81 precision after a training step of 20000.

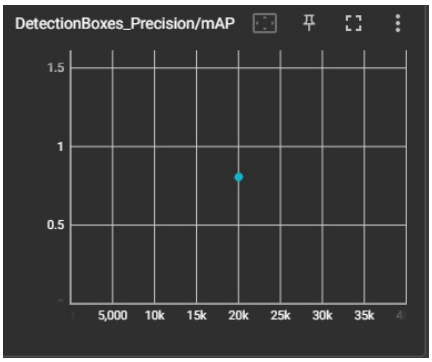


Fig 10.1. mAP Precision of SSD MobileNet (20,000 Training Steps)

In figure x, a human facing at the side was detected with a confidence level of 99%. There were few images which were detected with a lower confidence as shown in Figure y. The model was able to detect but with a lower confidence. Unfortunately, some number plates were not detected which were small and pixelated in an image.

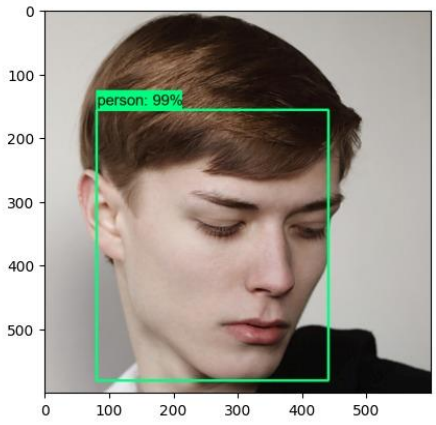


Fig 10.2. Man facing an angle (CL: 99%)

Fig 10.3. Car having Lower confidence and low accuracy.

As can be seen in the above picture, the model detects the classes with high confidence. In Fig 10.2, the face of a human was detected with a confidence of 0.92. In Fig 10.3. the number plates were detected with lower confidence. In Fig 10.3, some images of the cars were not detected, giving us a confidence of 76 percentage while some number plates which were far from the camera, having a blurred view, were not detected.

10.2 Result of Yolo Models:

The confusion matrix obtained for the YOLOv7 model used for this project using the custom data set is shown in figure 7b. All the true positive values are above 0.9, which implies that the positive classes are predicted as positive with an accuracy above 0.9. For the class ‘person’, the TP value is 0.99, for ‘license’, the value is 0.92 and for ‘Legal-Docs’, the value is 0.91. While the TP values are high, the trained set has its own flaws as can be seen from the prediction of Background as other classes. One main reason is the noises in the dataset such as skin tone which resembles the ground color and name boards which can be confused as license plates. All other boxes are blank showing that the given classes are not wrongly identified.

10.2.1 Losses

The values of losses at the end of the 99th epoch (minimum losses) is shown below. Both YOLOv5 and YOLOv7 exhibited similar behavior. In fig 10.4, all three losses are shown. It can be clearly seen that the losses were reduced with the number of epochs going ahead.

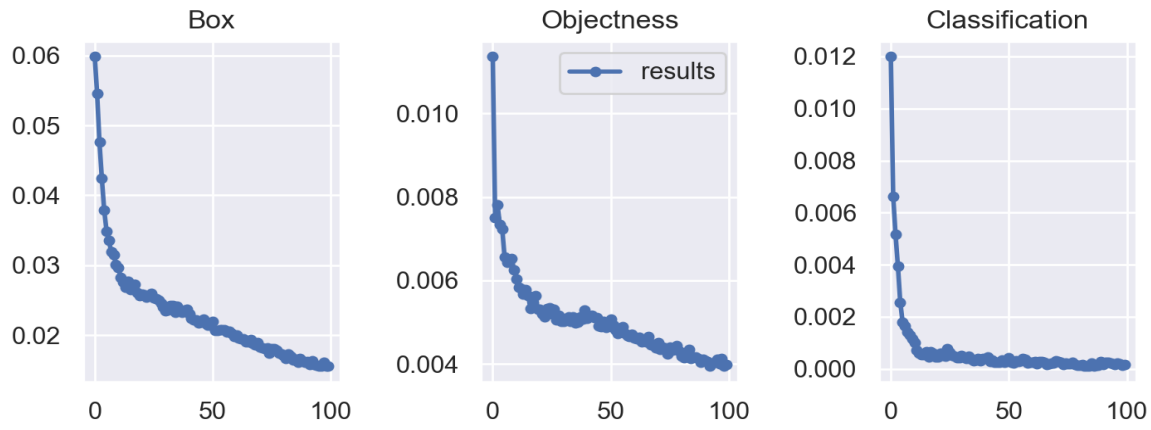


Fig 10.4. graph showing the losses for 100 epochs

10.2.2 Confusion Matrix

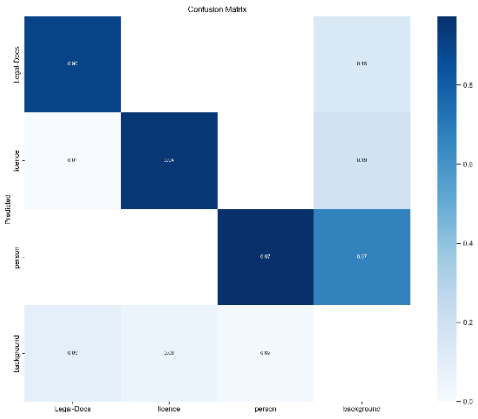


Fig 10.5 (a) YOLOv5 Confusion matrix

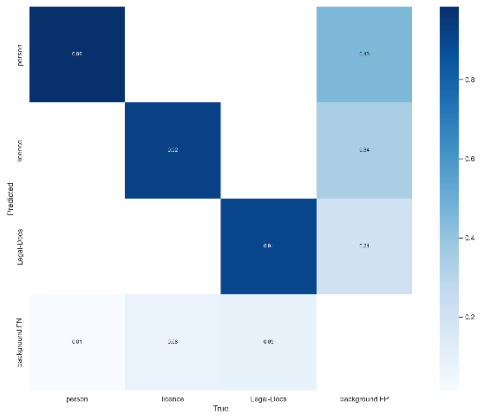
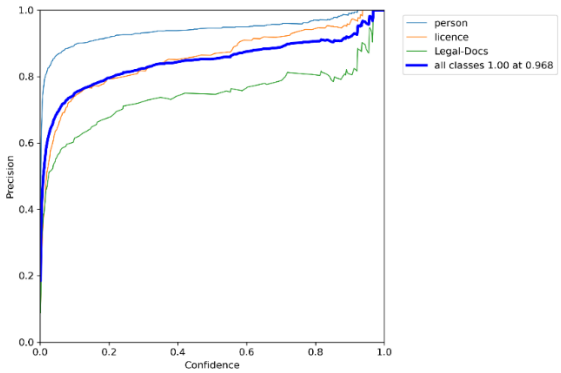
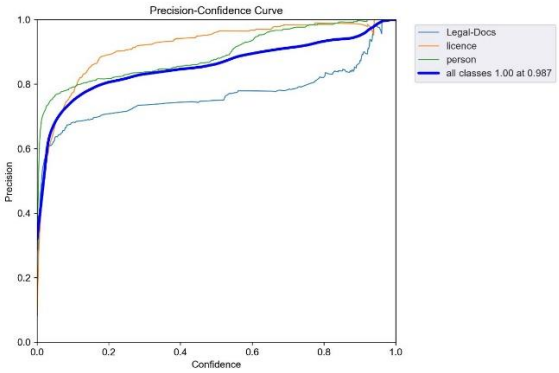


Fig 10.5 (b) YOLOv7 Confusion Matrix

The results of YOLOv5 and YOLOv7 were almost similar. While comparing the confusion matrices of YOLOv5 (figure 10.5 (a)) and YOLOv7 (figure 10.5 (b)), the background noise in YOLOv5 was higher i.e., 0.67 compared to the background noise of that of YOLOv7 i.e., 0.45. This meant that the v7 could distinguish that particular class from the background with more precision.

A graph obtained by plotting the precision against the confidence is shown in figure 10.6 given below. The value of average precision of all the classes taken together increases with an increase in confidence and reaches a maximum value of 1. In the case of YOLOv7, the maximum value is obtained at a confidence of 0.968 and for YOLOv5, the maximum value is with a confidence of 0.987.

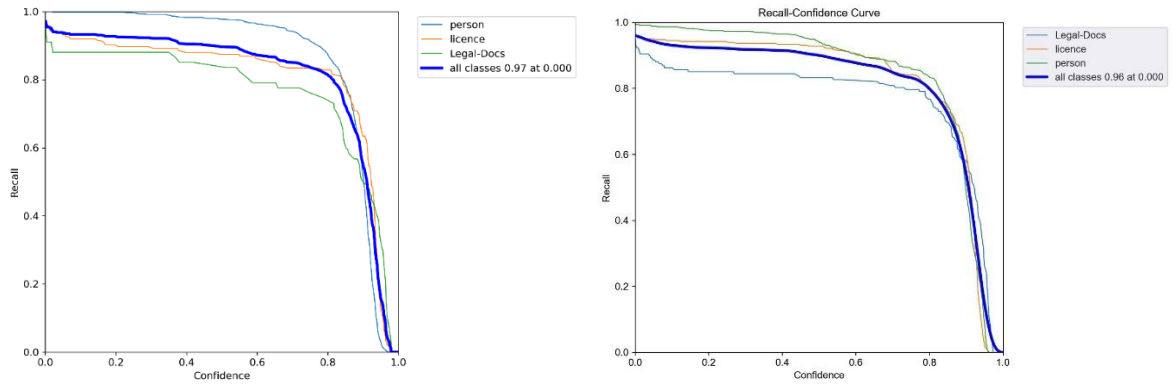




(a) YOLOv7 *(b) YOLOv5*

Fig 10.6. Confidence vs Precision

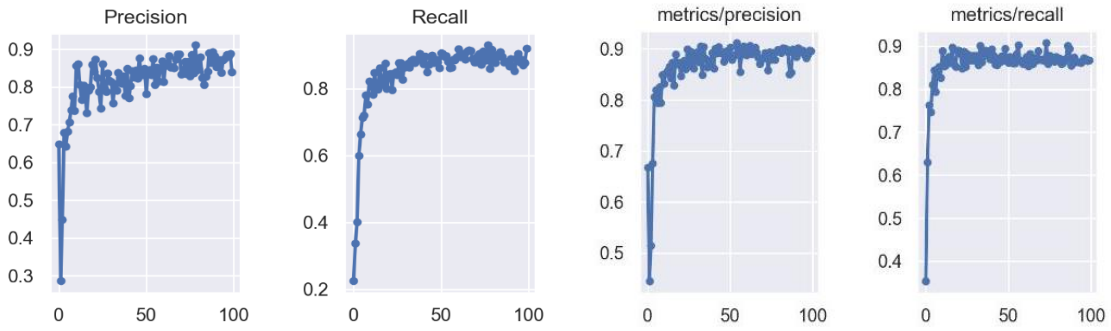
Figure 10.7 shows the recall plotted against confidence. Here, the average value of recall of all the classes taken together is above .85 up to a confidence of approximately 0.85. This means that the model is good for detecting most of the objects with a confidence of approximately 0.85. Both YOLOv5 and YOLOv7 has similar results.



(a) YOLOv7 *(b) YOLOv5*

Fig 10.7. Confidence vs Recall

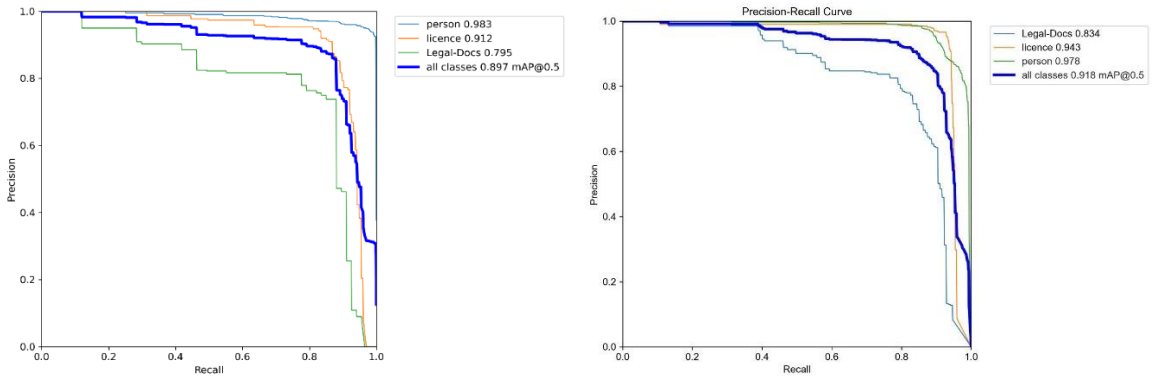
In figure 10.8., we can see that both the precision and recall curves are growing with the epochs. In the case of YOLOv7, the precision starts from 0.2866 in the beginning and reaches a value of around 0.9 by the end of the 100th epoch. This means that the model identifies the detected objects really well at the end of the training. Similar to the precision curve, the recall value also increases from 0.225 to a value of around 0.9, meaning that the model is detecting almost all of the objects. The recall curve obtained using the YOLOv5 is almost exactly the same as that of YOLOv7 as can be seen in figure 10.7.



(a) YOLOv7 *(b) YOLOv5*

Fig 10.8.. Precision, Recall, graphs.

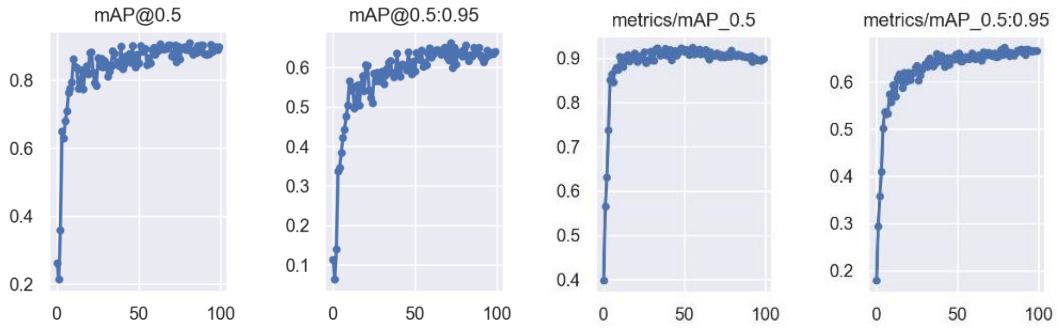
10.2.3 Precision-Recall curve



(a) YOLOv7 *(b) YOLOv5*

Fig 10.9. Precision-recall curve

The precision-recall curve shows a trade-off between the precision and recall of the model. A good model will have high precision for high values of recall. High values for both precision and recall mean the model can detect objects well and it can identify the detected objects very well [28]. Fig 10.9. (a) shows the precision-recall curve for the YOLOv7 model trained using our dataset. The curve shows that for values of precision near 0.9, the recall is also near 0.9. This result shows that the trained model can detect and identify the classes with high accuracy. The precision-recall curve obtained for the YOLOv5 model also shows comparable results. For a recall of 0.9, the precision is around 0.9.



(a) YOLOv7 *(b) YOLOv5*

Fig 10.10. mAP@ 0.5 and mAP@0.5:0.95

Figure 10.10 (a) shows the mAP plot for YOLOv7. In this figure, the mAP@0.5 goes on increasing with the epoch and reaches a value around 0.9 by the 100th epoch. This means that the precision of the model when it predicts the presence of an object in an image with a confidence of at least 0.5 is 0.9. The mAP@0.5:0.95 obtained is nearly 0.65. The curve can be seen dipping a bit towards the end of the graph suggesting that the model might have started overfitting during the training. However, since the result of the best epoch is used for object detection, this overfitting does not affect the prediction of new objects. The mAP curve for YOLOv5 is shown in figure 10.10. (b). In this case

also we can see that the values are close to that of YOLOv7. The curve for mAP @0.5 has started overfitting at about 50th epoch. But that did not affect the final result as the best epoch was chosen for comparison.

10.3 Object detection using YOLOv7 custom-trained model

After the training was done, the epoch that returned the best parameters was used for object detection. The trained model was used for detection from still images, videos, and for live detection using the webcam. The results obtained are shown below.

Fig 10.11 (a)

Fig 10.11 (b

Fig 10.11 (c) *Fig 10.11 (d)*

Fig. 10.11 (a) Detection of Legal Document (Original image source: Google images)

Fig 10.11 (b) Detection of multiple classes (Original image source: Google images)

Fig 10.11 (d) Detection from a video (Original Video Source: YouTube)

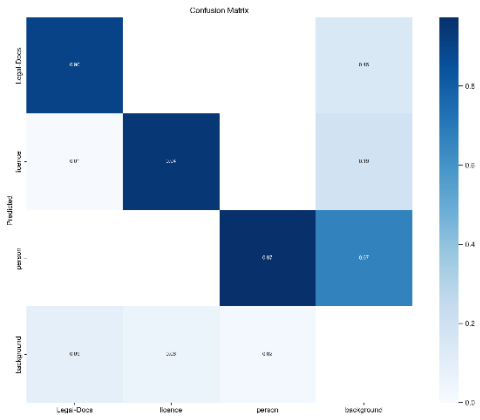
Fig 10.11 (e) Live detection using webcam

As can be seen in the above picture, the model detects the classes with high confidence. In figure 10.11 (a), the legal document was detected with a confidence of 0.92, and the face of the person was also detected separately. In figure 10.11 (b), the number plates and faces were detected with high confidence. Some faces were not detected clearly as they were mostly not distinguishable from the background (the confusion matrix showed a 0.45 False negative for this particular class). Figure 10.11 (c) illustrates the detection from a video. The face of the person was detected with a confidence of 0.86. Figure 10.11 (d) shows live detection using a webcam. The class 'person' is detected with a confidence of 0.92. These results show that the trained model is capable of multi-class detection with high confidence.

11. SELECTION OF OBJECT DETECTION MODEL FOR OBJECT BLURRING

The results obtained from the three object detection models, i.e., SSD MobileNet 320x320, YOLOv5, and YOLOv7 were compared and the model which gave the best result for our data set was chosen for object blurring. The results obtained using SSD MobileNet 320x320 were too low compared to the YOLO models. So that model was discarded initially. The results of YOLOv5 and YOLOv7 were almost similar. While comparing the confusion matrices of YOLOv5 (figure 10.12 (a)) and YOLOv7 (figure 10.12 (b)), the background noise in YOLOv5 was higher i.e., 0.67 compared to the background noise of that of YOLOv7 i.e., 0.45. This meant that the v7 could distinguish that particular class from the background with more precision. This along with the factor that YOLOv7

had a better architecture compared to the YOLOv5 led us to the conclusion that the YOLOv7 custom-trained model can be used for the object blurring purpose.



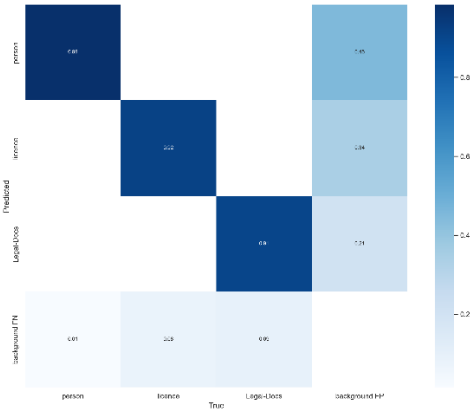


Fig 10.12 (a) YOLOv5 Confusion matrix

Fig 10.12 (b) YOLOv7 Confusion Matrix

12. OBJECT BLURRING

Blurring an object in an image using OpenCV is a process that can be achieved using the cv2.blur function or the cv2.GaussianBlur function. To blur an object, it is first need to segment the object from the rest of the image using image processing techniques such as thresholding or image detection.

Once the object is separated from the background, blur can be applied to the object by passing the object region to the cv2.blur or cv2.GaussianBlur function. cv2.blur is a function in the Python library OpenCV that allows the blurring of an image. It takes in two required arguments: the input image (which must be a grayscale or color image) and the size of the blur kernel. The blur kernel size can be adjusted to control the intensity of the blur. The function applies a blur to the image by averaging the pixels in the image within the kernel size. This can be useful for reducing noise or smoothing out the appearance of an image. cv2.blur is faster to compute when compared to cv2.GaussianBlur function. Due to this higher speed of the cv2.blur function, this function was used for object blurring in this project. For object detection and separation from the background, the YOLOv7 custom-trained object detection model was used. The blurring was done on still objects, and videos and was also incorporated for live object blurring using live videos captured through the webcam.

While using the object blurring code, a confidence threshold of 0.25 and an IoU threshold of 0.45 were used. This was to enable maximum object detection and blurring. The blur ratio (kernel) was set to 70 so that the objects are blurred with high opacity. The bounding boxes that showed the

confidence level of the object detection model were also removed before applying the blur. The results of the object blurring process are shown below.

Fig. 10.13 (a)

Fig. 10.13 (b)

Fig. 10.13 (c)

Fig. 10.13 (d)

Fig. 10.13 (e)

Figure 10.13. (a) shows face blurring applied to an image with a group of people (Original image source:[10])

Figure 10.13. (b) shows face blurring applied in a video. (Original video source: YouTube)

Figure 10.13. (c) shows image blurring applied to a legal document. (Original image source: Google images)

Figure 10.13. (d) shows image blurring applied on a live video taken using a webcam.

Figure 10.13. (e) shows image blurring applied on the vehicle registration number plates.(Original Image Source: Kaggle)

As can be seen from the above images, object detection and blurring were successfully implemented on objects abiding by the GDP Regulations. Since the function cv2.blur depends on the detected area in the image, the accuracy of blurring depended on the precision of the object detection model.

13 KIVY

13.1 Introduction

An open-source framework called Kivy is used to create programs for devices with touch screens and other natural user interfaces. Kivy is a cross-platform framework that can be used in Windows, Linux, Mac OS X, iOS, and Android, with more implementations on the way. The creation of apps based on Kivy uses the Python programming language. Kivy is ideal for graphical applications like games and information displays because it rests on top of OpenGL [31].

13.2 Architecture of Kivy

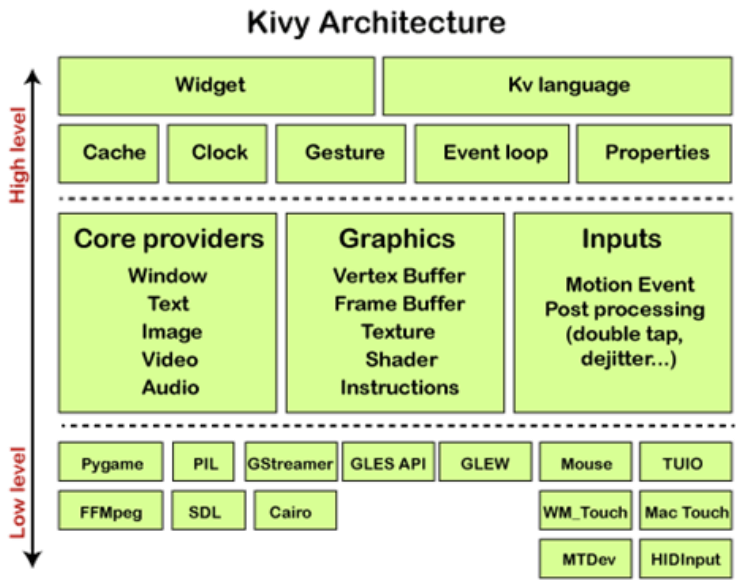


Fig. 11.1 Kivy Architecture

13.2.1 Core Providers and Input Providers

Understanding Kivy's internals requires an understanding of concepts like modularity and abstraction. Kivy seeks to abstract commonplace actions like opening a window, showing text and photos, playing audio, collecting images from a camera, correcting typos, etc. It refers to these as core tasks. Because of this, the API is simple to use and extend. (32) Moreover, it refers to a piece of code as a core provider when it uses one of these several native APIs to communicate with the operating system on the one side and with Kivy on the other (serving as an intermediary communication layer).

13.2.2. Graphics

Graphics API for Kivy is an abstraction of OpenGL. Kivy issues OpenGL-based commands for hardware-accelerated drawing at the lowest level. However, writing OpenGL code can be a little complicated. This is why Kivy offers the graphics API, which enables you to draw objects using straightforward metaphors that are missing from OpenGL (e.g., Canvas, Rectangle, etc.). This graphics API, which is built at the C level for performance reasons, is used by all of the Kivy widgets themselves [34].

13.2.2. Core

The core package provides following features:

13.2.2.1 Clock

The timer events can be planned using the clock. Periodic timers as well as one-shot timers are supported.

13.2.2.2 Cache

The cache in Kivy is where Python objects are kept. There are two techniques to manage the cache: Object limit and Timeout.

13.2.2.3 Gesture Detection

To recognize different stroke types, circles, rectangles, and squares, Kivy employs gesture detection method.

13.2.2.4 Kivy Language

User interfaces can be easily and effectively described using the kivy language.

13.2.2.5 Properties

These are not the typical characteristics that you may be familiar with from Python. They are our own property classes that connect the code for your widget with the description of the user interface.

13.2.3 UIX (Widgets & Layouts)

You can reuse the frequently used widgets and layouts in the UIX module to quickly design a user interface.

13.2.3.1 Widgets

You can add widgets to your program's user interface to add various functionalities.Examples: file browser, buttons, sliders, lists, screen and so on.

13.2.3.2 Layouts

Layouts are used to arrange widgets. Obviously, you can determine the placements of your widgets yourself, but using one of Kivy's pre-made layouts is frequently more practical.

13.2.4 Input Events

The core of the Kivy architecture is the Kivy input events. Touches, mice, TUIO, Mouse are just a few of the several input events that are included in the Kivy abstract. Every individual input event can also provide 2D onscreen positioning.

A touch () class instance serves as a general description of all input kinds. These three states are all possible for a touch () class instance.

- Down
- Move
- Up

13.3 Widgets

In GUI development, the term "widgets" is most usually used to refer to the portion of the program that interacts with the user. Widgets function as an object in Kivy that receives input events. A widget tree is used to organize every widget. One widget may have one child, many children, or none at all [33].

Kivy provides one piece of new info per touch each time it becomes accessible. The root widget in the widget tree is the first widget to receive each event.

widgets used in our application are described here:

13.3.1 Label

Text rendering is done using a label widget. Font size, color, and text color can all be added to labels to increase their attractiveness.

13.3.2 Button

A button is a Label that, when pushed (or released upon a click or touch), causes certain activities to occur.

13.3.3 Image

To display an image, use the Image widget.

13.3.4 Text Input

This widget offers a basic text input box that may be edited.

13.3.5 File manager

It gives access to a shortcut bar with connections to system and custom directories. The links bar will expand and display all the directories inside a shortcut when you touch it. Additionally, it makes it easier to specify unique paths to be included in the list of shortcuts.

13.3.6 Camera

To record and show video from a camera, use the Camera widget.

13.3.7 Screen Manager

The screen manager is a widget created specifically for handling several screens in application.

13.3.8 Boxlayout

BoxLayout can organize widgets horizontally, with one widget placed after another, or vertically, with one widget stacked on top of another.

13.3.9 Float layout

Using a relative position, a Float Layout enables us to position elements. This means that rather than specifying precise coordinates, everything will be placed using percentages or according to the width and height of the current window.

13.4 Frontend & Backend

Our frontend and backend are linked by various files, each of which performs a particular function. When calling functions in Kivy, where functions are required to carry out some task, we created the functions, imported them into a Python file, and then used this function.

Here are some file names and their descriptions that we use in our UI to connect the backend with the front end.

- **Pt file (" yolov7_object_blurring detection.pt \\yolov7"):**

  This PT file includes a trained model created by the YoloV7 algorithm.

- **Detect_and_blur.py:**

  This file is used for blurring after detection of an object by our model, and it also has an argparse module that makes it easy to write user-friendly command-line interfaces.

- **Main.py file:**

This is a Python file, and it uses to connect functions with our UI.

- **Kivy file:**

  It stores source code in the Kivy syntax, which may also contain templates, dynamic class definitions, widgets, and rule definitions. In our case, this Kivy file is written in the main python file.

13.4.1 Front End

The layer or component of an application that the user can use, view, and interact with through buttons, images, interactive elements, navigational menus, and text is known as the front-end application, also referred to as the application's interface.

13.4.1.1 Application’s Frontend

There are a total of five screens in our application, and each screen has unique characteristics that make certain tasks possible.

1st screen:

One label and one button found on the first screen. Our application's name appears on the label. using buttons that you can enter in the application. We basically use the screen manager library to move from one screen to another. To add images to the application background, we use the Kivy background image function.

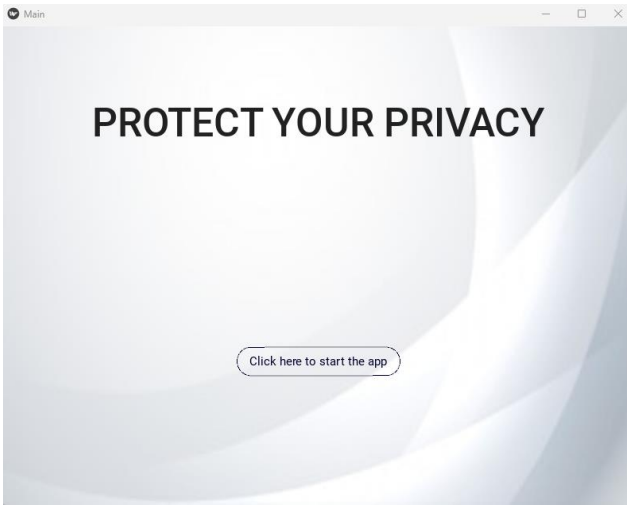


Figure 11.2. First Screen of Application

2nd screen:

Its primary display is on the second screen. Three buttons are on this display. Using the Kivy button library, we give the first button name "web camera." This button enables the web camera to recognize

objects and blur them. The same button library that we used for the first button is used for the second button, "object detection by image." You can blur an image and find things by pressing this button. The third button does similar operations to the second button, but it also allows us to recognize things and blur those objects throughout the entire video.

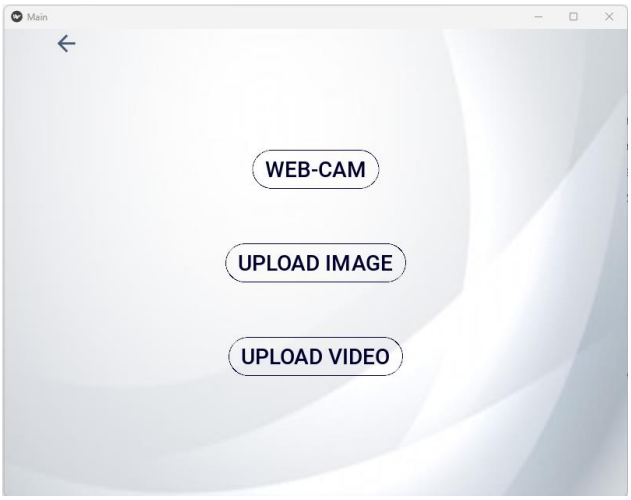


Figure 11.3. Second Screen of Application

3rd screen:

The third screen contains one camera module that displays webcam output and two buttons: one for image capture from the web camera. In Figure 11.4. the camera pops out to give us a blurred image using a webcam.

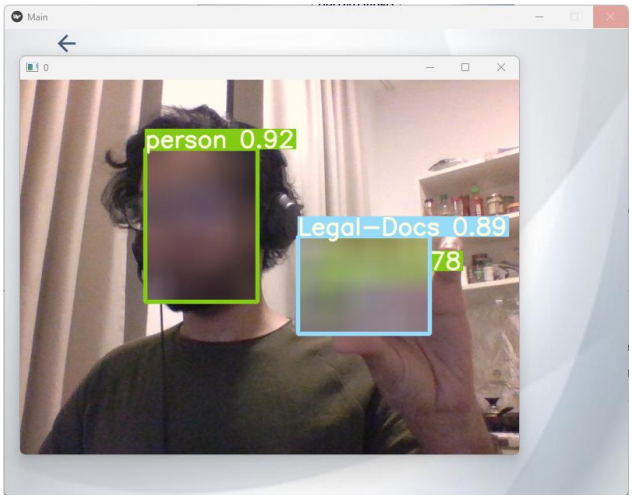


Figure 11.4. Third Screen of Application

4th screen:

The blur and object detection from the image are included in the fourth screen. To do this, we choose files from the local computer and then display them on the main screen using the Kivy file manager

library. Additionally, we added a button that, when pressed after choosing a file, will identify and blur every object on the screen and show the result.

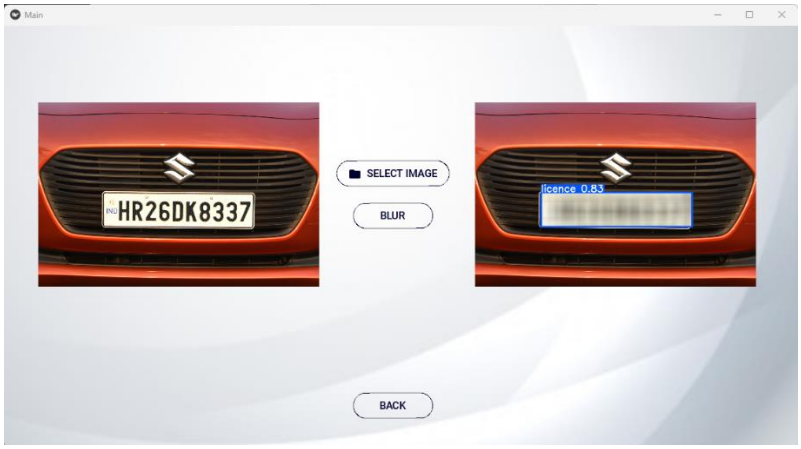


Figure 11.5. Fourth Screen of Application

5th screen:

The fifth screen is similar to the fourth screen in terms of functions, with the exception that it only finds and blurs things using video files. A Label Box shows use the video file which has been selected to be blur.

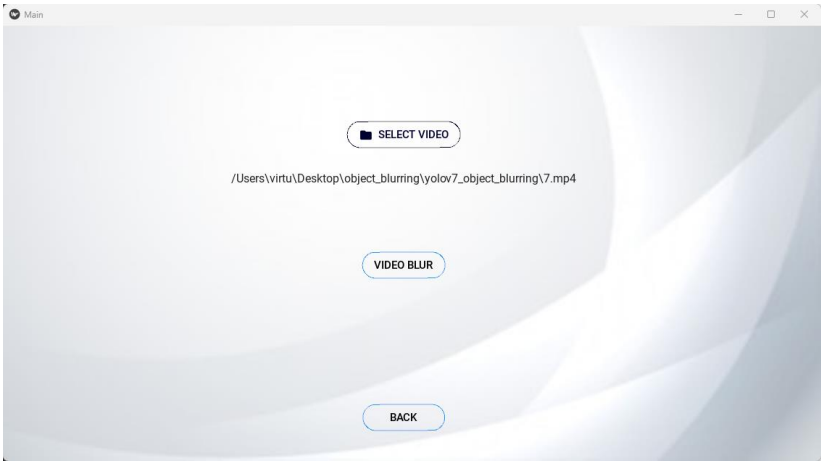


Figure 11.6. Fifth Screen of Application

13.4.2 BackEnd:

the section of your web app that the user cannot see directly. As requests come in, it prepares the information and sends it back to the user's browser. Backend code is created to run on a server, never on the user's computer.

13.4.2.1 Application's Backend:

We connected file together using importation technique from different directory.

In the third screen is used to objects and blur from the web camera output. For that, we use the Kivy camera library and a function called capture_video. It then uses os.system to run Python "python yolov7_object_blurringun_detect_and_blur.py" with --source 0. The result will be displayed on the screen.

In the back end of the fourth screen, we have create a function called "Outblur" to recognize and blur an image after using a file manager library to upload an image from a local computer. This function has an if-else condition: if the file is not selected, nothing will be output, and otherwise, we define a command variable containing our YOLOv7 model, which recognizes and blurs objects in images. When the output is ready, the Kivy image library will show result on screen.

Similarly, In the Screen 5 has the same functionality, with the exception that Screen 4 uses images to find and blur objects, whereas Screen 5 uses video to find and blur objects. We built a function called output_videoblur. This function starts by checking if the vid_path is empty. If it is, then the code will pass and nothing will happen. It then uses os.system to run python yolov7_object_blur-ringun_detect_and_blur.py with --source vid path as an argument to detect and blur a video file in that directory. The output of the detected video is saved in the detectedvid variable.

14 RISKS & LIMITATIONS

Though, the implementation of Object Detection has made significant advancements during this past few years, it still faces multiple challenges. Similarly, building an application using Kivy would has its limitations as well. In this section, we discuss these limitations:

- Objects marked for detection used in this project have varies viewpoint. For example, the head of a person looked from different side view. There were some instances that the object detection was not able to detect when there was a face, though the person was looking from the side.
- Despite many data collection efforts, it still remains substantially smaller in scale.
- Though, we used YOLO which has a significantly higher speed of detection as compared to Mobile Net, it still would be an issue in terms of GDPR which require that the image is to be detected at every instance.
- As stated in [6], tattoos are a part of a human identity due to which, we face challenge since detecting a tattoo containing artworks would require a unique detection method.
- KIVY has fewer functions and weights that make the UI less attractive and user-friendly.
- Some tasks in Kivy take longer to complete. Kivy uses a bridge scheme to compile the code, making the development of applications in it relatively slower.
- The package size of the application developed in Kivy is unnecessarily large because the python interpreter must be included with the code.

15 RECOMMENDATION & FUTURE WORK

As our project was built on a smaller scale, it has a lot of potential making it more helpful when used on a global scale. In this section, we intend to discuss the future prospects of this project:

- Addition of internet links, which include YouTube links, would help when intended to use for another project.
- Publishing a complete application in the mobile play store or building a website, would help the public get more access so that they are able to follow the law. Using cross-platform technologies such as Flutter, we can create more appealing user interactions.
- Ability to access live security cameras for private users would help in downloading a specific clip having clipped part.
- Use of face recognition helps in ignoring a certain selected individual, or legal documents.

16 REFERENCES

[1] https://gdpr.eu/working-remotely-data-security/

[2] https://digitalguardian.com/blog/what-gdpr-general-data-protection-regulation-understanding- and-complying-gdpr-data-protection

[3] https://github.com/nicknochnack/TFODCourse

[4] https://towardsai.net/p/l/how-does-ai-data-collection-work-in-relation-to-machine-learning-models#

[5] https://iclg.com/practice-areas/data-protection-laws-and-regulations/germany

[6] https://allaboutberlin.com/guides/website-compliance-germany

[7] https://labelyourdata.com/articles/what-is-dataset-in-machine-learning

[8] https://www.datatobiz.com/blog/datasets-in-machine-learning/

[9] https://gdpr-info.eu/#:~:text=General%20Data%20Protection%20 Regulation%20 GDPR

[10] https://www.kaggle.com/datasets/andrewmvd/car-plate-detection

[11] https://universe.roboflow.com/license-plate-number/license-plate-number-qrq7f

[12] https://github.com/heartexlabs/labelImg

[13] https://labelyourdata.com/articles/what-is-dataset-in-machine-learning

[14] https://www.makesense.ai

[15] Blog.paperspace.com/train-yolov7-custom-data/

[16] https://www.researchgate.net/figure/MobileNet-SSD-AF-architecture-we-use-MobileNet-as-the-feature-extractor-network-and-SSD_fig7_324584455

[17] https://www.analyticsvidhya.com/blog/2022/09/object-detection-using-yolo-and-mobilenet-ssd/#:~:text=Mobilenet%20SSD%20is%20an%20object,detection%20optimized%20for%20mobile%20devices

[18] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md

[19] https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/

[20] Blog.roboflow.com/pp-yolo-beats-yolov4-object-detection/

[21] YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, Wang, Chien-Yao and Bochkovskiy, Alexey and Liao, Hong-Yuan Mark, 2022, arXiv

[22] Github.com/WongKinYiu/yolov7

[23] Javatpoint.com/epoch-in-machine-learning

[24] Simplilearn.com/tutorials/machine-learning-tutorial/what-is-epoch-in-machine-learning

[25] Towardsdatascience.com/yolov7-a-deep-dive-into-the-current-state-of-the-art-for-object-detection-ce3ffedeeaeb#7c9e

[26] Developers.google.com/machine-learning/crash-course/classification/precision-and-recall

[27] V7labs.com/blog/mean-average-precision

[28] Analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

[29] https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470

[30] https://pyimagesearch.com/2022/06/20/training-the-yolov5-object-detector-on-a-custom-dataset/

[31] https://www.ic.unicamp.br/~celio/mc853/kivy/Kivy_Interactive_Applications_and_Games.pdf

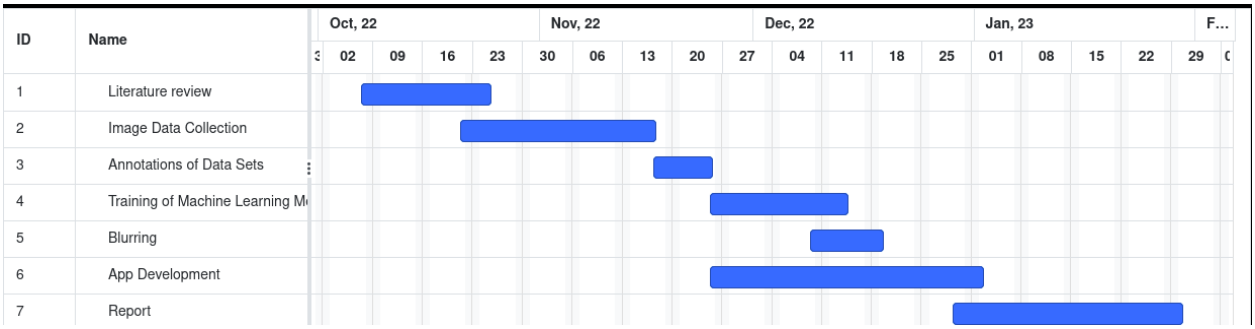
[32] https://kivy.org/doc/stable/guide/architecture.html

[33] https://buildmedia.readthedocs.org/media/pdf/kivy/latest/kivy.pdf4c-

[34] https://www.javatpoint.com/kivy

17 APPENDIX

Appendix I - Gantt Chart



Appendix II- Requirements of the Kivy (Application)

certifi==2022.12.7
charset-normalizer==3.0.1
colorama==0.4.6
contourpy==1.0.7
cycler==0.11.0
docopt==0.6.2
docutils==0.19
fonttools==4.38.0
idna==3.4
Kivy==2.1.0
kivy-deps.angle==0.3.3
kivy-deps.glew==0.3.1
kivy-deps.sdl2==0.4.5
Kivy-Garden==0.1.5
kivymd==1.1.1
kiwisolver==1.4.4
matplotlib==3.6.3
numpy==1.24.1
opencv-python==4.7.0.68
packaging==23.0
pandas==1.5.3
Pillow==9.4.0
pipreqs==0.4.11
plyer==2.1.0
Pygments==2.14.0
pyparsing==3.0.9
pypiwin32==223
python-dateutil==2.8.2
pytz==2022.7.1
pywin32==305
PyYAML==6.0
requests==2.28.2
scipy==1.10.0

```
seaborn==0.12.2
six==1.16.0
thop==0.1.1.post2209072238
torch==1.13.1
torchvision==0.14.1
tqdm==4.64.1
typing_extensions==4.4.0
urllib3==1.26.14
yarg==0.1.9
```

Appendix III - Complete Code for MobileNet (Image Collection.ipynb)

```
#####BUILD IN A JUPYTER NOTEBOOK FILE

import os
import time
import cv2
import uuid
labels = ['person','licence','Legal-Docs']
number_imgs = 5
#Directories built for MobileNet detection
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
if not os.path.exists(IMAGES_PATH):
    if os.name == 'nt':
        !mkdir {IMAGES_PATH}
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
    if not os.path.exists(path):
        !mkdir {path}
#Capture Images
#Capture Images from Live Camera
for label in labels:
    cap = cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_imgs):
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
# Image Labelling using LabelImg
!pip install --upgrade pyqt5 lxml
LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
# Option to use LabelImg for Annotation Building
if not os.path.exists(LABELIMG_PATH):
    !mkdir {LABELIMG_PATH}
```

```
    !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
if os.name =='nt':
    !cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc
!cd {LABELIMG_PATH} && python labelImg.py
# Training and Testing Partition
TRAIN_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'train')
TEST_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'test')
ARCHIVE_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'archive.tar.gz')
```

Appendix IV - Complete Code for MobileNet (Training & Dectection.ipynb)

```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUS-
TOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'ex-
port'),
    'TFJS_PATH':os.path.join('Tensorflow',  'workspace','models',CUSTOM_MODEL_NAME,  'tfjsex-
port'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME,
'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME,
'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'nt':
            !mkdir {path}
#  Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD
if os.name=='nt':
    !pip install wget
    import wget
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
# Install Tensorflow Object Detection
if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-
win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
```

```
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py && python setup.py build && python setup.py install
    !cd Tensorflow/models/research/slim && pip install -e .
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
!python {VERIFICATION_SCRIPT}
import object_detection
if os.name == 'nt':
    wget.download(PRETRAINED_MODEL_URL)
    !move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
# Create Label Map
labels = [{'name':'person', 'id':1}, {'name':'licence', 'id':2}, {'name':'Legal-Docs', 'id':3} ]
with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
# Create TF records
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
# Copy Model Config to Training Folder
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 2
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]
config_text = text_format.MessageToString(pipeline_config)
```

```
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
# TRAIN the Model
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=5000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
!{command}
# Evaluate the Model
command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
# Load Train Model From Checkpoint
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-6')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
# Detect from an Image
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'TRI (1359).jpg')
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
```

```
label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'],
          detections['detection_classes']+label_id_offset,
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=5,
          min_score_thresh=.8,
          agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
# Real Time Detections from your Webcam
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
   ret, frame = cap.read()
   image_np = np.array(frame)

   input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
   detections = detect_fn(input_tensor)
   num_detections = int(detections.pop('num_detections'))
   detections = {key: value[0, :num_detections].numpy()
                 for key, value in detections.items()}
   detections['num_detections'] = num_detections

   # detection_classes should be ints.
   detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

   label_id_offset = 1
   image_np_with_detections = image_np.copy()
   viz_utils.visualize_boxes_and_labels_on_image_array(
             image_np_with_detections,
             detections['detection_boxes'],
             detections['detection_classes']+label_id_offset,
             detections['detection_scores'],
             category_index,
             use_normalized_coordinates=True,
             max_boxes_to_draw=5,
             min_score_thresh=.8,
             agnostic_mode=False)

   cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))
   if cv2.waitKey(10) & 0xFF == ord('q'):
      cap.release()
      cv2.destroyAllWindows()
      break
```

Appendix V - REQUIREMENTS For YOLO (main.py)

```
# YOLOv5 🚀 requirements

# Usage: pip install -r requirements.txt

# Base ------------------------------------------------------------------------
gitpython
ipython  # interactive notebook
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
psutil  # system resources
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1  # FLOPs computation
torch>=1.7.0  # see https://pytorch.org/get-started/locally (recommended)
torchvision>=0.8.1
tqdm>=4.64.0
# protobuf<=3.20.1  # https://github.com/ultralytics/yolov5/issues/8012

# Logging ---------------------------------------------------------------------
tensorboard>=2.4.1
# clearml>=1.2.0
# comet

# Plotting --------------------------------------------------------------------
pandas>=1.1.4
seaborn>=0.11.0
# Export ----------------------------------------------------------------------
# coremltools>=6.0  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.4.1  # ONNX simplifier
# nvidia-pyindex  # TensorRT export
# nvidia-tensorrt  # TensorRT export
# scikit-learn<=1.1.2  # CoreML quantization
# tensorflow>=2.4.1  # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export

# Deploy ----------------------------------------------------------------------
# tritonclient[all]~=2.24.0

# Extras ----------------------------------------------------------------------
# mss  # screenshots
# albumentations>=1.0.3
# pycocotools>=2.0  # COCO mAP
# roboflow
# ultralytics  # HUB https://hub.ultralytics.com
```

Appendix VI - Complete Code for Training YOLOv5 Model(main.py)

Pre-built Codes are used to make YOLOv5 Model

Code Source: yolov5/models at master · ultralytics/yolov5 (github.com)

Appendix VII - Complete Code for Detection in YOLOv5 Model (main.py)

Pre-built Codes are used to make YOLOv7 Model

Code Source: WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (github.com)

Appendix VIII - Complete Code for Kivy (main.py)

```
from kivy.core.window import Window
from kivy.lang import Builder
from kivy.uix.screenmanager import Screen, ScreenManager
from kivymd.app import MDApp
from kivymd.uix.filemanager import MDFileManager
from kivy.uix.boxlayout import BoxLayout
import subprocess
from kivymd.toast import toast
import sys
import os
from plyer import filechooser
from kivy.uix.button import Button
from kivy.properties import ListProperty
from kivy.app import App
from textwrap import dedent
from os import startfile

import cv2
from PIL import Image
import time
import numpy as np
from fordetectingandkivy import *
#To maximize window size
Window.maximize()
#Kivy file
KV = '''
#Windowmanger operate different screen
WindowManager:
    First:
    Second:
    Third:
    Fourth:
    Fifth:
#First screen
<First>:
    name: 'First'
    ScreenWallpaper:
        size:320,500
        AsyncImage:
```

```
            allow_stretch: True
            keep_ratio:False
            source:'https://img.freepik.com/free-vector/shiny-white-gray-background-with-wavy-
lines_1017-25101.jpg?w=360'
        MDLabel:
            text:'PROTECT YOUR PRIVACY
            font_style: 'Button'
            font_size: 50
            halign: "center"
            size_hint_y: None
            pos_hint: {"center_x":.5,"center_y":.8}
            padding_y: 10
            text_color: rgba(0,0,45,255)

        MDRoundFlatButton:
            text:'Click here to start the app'
            font_size: 15
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
            line_color: rgba(0,0,45,255)
            pos_hint: {"center_x":.5,"center_y":.3}
            on_press:
                root.manager.current = 'second'
                root.manager.transition.direction = 'left'
#Second screen
<Second>:
    name: 'second'
    ScreenWallpaper:
        size :320,500
        AsyncImage:
            allow_stretch: True
            keep_ratio:False
            source:'https://img.freepik.com/free-vector/shiny-white-gray-background-with-wavy-
lines_1017-25101.jpg?w=360'
        MDIconButton:
            icon: "arrow-left"
            pos_hint:{"center_x":0.1,"center_y":0.97}
            icon_size: "35dp"
            theme_text_color: "Custom"
            text_color: rgba(71,92,119,255)
            on_press:
                root.manager.current = 'First'
                root.manager.transition.direction = 'left'
        MDRoundFlatButton:
            text:'Web-Cam'
            font_style: 'Button'
            font_size: 28
            halign: "left"
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
```

```
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.5,"center_y":.7}
        on_press:
            root.capture_video()
    MDRoundFlatButton:
        text:'Upload Image'
        font_style: 'Button'
        font_size: 28
        halign: "left"
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.5,"center_y":.5}
        on_press:
            root.manager.current = 'fourth'
            root.manager.transition.direction = 'left'
    MDRoundFlatButton:
        text:'Upload Video'
        font_style: 'Button'
        font_size: 28
        halign: "left"
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.5,"center_y":.3}
        on_press:
            root.manager.current = 'fifth'
            root.manager.transition.direction = 'left'
#Third screen
<Third>:
    name:'third'

    ScreenWallpaper:
        size:320,500
        AsyncImage:
            allow_stretch: True
            keep_ratio:False
            source:'https://img.freepik.com/free-vector/shiny-white-gray-background-with-wavy-
lines_1017-25101.jpg?w=360'
    MDCard: #card to hold components
        orientation: "vertical"
        padding: "10dp"
        spacing: "5dp"
        pos_hint: {"center_x":0.5,"center_y":0.68}
        md_bg_color: rgba(200,200,210,255)
        radius:(20,20,20,20)
        size_hint: 0.6,0.6
        elevation: 6
    MDRoundFlatButton:
        text:'Play'
```

```
        font_style: 'Button'
        font_size: "15dp"
        size_hint:0.10,0.05
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.3,"center_y":.2}

    MDRoundFlatButton:
        text:'Capture'
        font_style: 'Button'
        font_size: "15dp"
        size_hint:0.10,0.05
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.5,"center_y":.2}
    MDRoundFlatButton:
        text:'Blur'
        font_style: 'Button'
        font_size: "15dp"
        size_hint:0.10,0.05
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.7,"center_y":.2}
        on_release: root.blur_img()
    MDRoundFlatButton:
        text:'Back'
        font_style: 'Button'
        font_size: "15dp"
        size_hint:0.10,0.05
        background_color: 1, 1, 1, 1
        text_color: rgba(0,0,45,255)
        theme_icon_color: "Custom"
        icon_color: rgba(0,0,45,255)
        line_color: rgba(0,0,45,255)
        pos_hint: {"center_x":.5,"center_y":.1}
        on_press:
            root.manager.current = 'second'
            root.manager.transition.direction = 'left'
#Fourth screen
<Fourth>:
    name:"fourth"
    ScreenWallpaper:
        size:320,500
        AsyncImage:
            allow_stretch: True
```

```
            keep_ratio:False
            source:'https://img.freepik.com/free-vector/shiny-white-gray-background-with-wavy-
lines_1017-25101.jpg?w=360'
    Screen:
        size:root.height,root.width
        MDRoundFlatIconButton:
            text: "Select Image"
            icon: "folder"
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
            line_color: rgba(0,0,45,255)
            pos_hint: {'center_x': .49, 'center_y': .65}
            on_release: root.file_manager_open()
#To add image in this screen
        Image:
            id:image
            source: 'photo.png'
            pos_hint: {'center_x': .22, 'center_y': .6}
            size_hint: None,None
            size: 400,400
        MDRoundFlatButton:
            text:'Blur'
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
            line_color: rgba(0,0,45,255)
            pos_hint: {"center_x":.49,"center_y":.55}
            on_release: root.output_blur()
        Image:
            id:newimage
            source: ''
            pos_hint: {'center_x': .77, 'center_y': .6}
            size_hint: None,None
            size: 400,400
        MDRoundFlatButton:
            text:'Back'
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
            line_color: rgba(0,0,45,255)
            pos_hint: {"center_x":.49,"center_y":.1}
```

```
            on_press:
                root.manager.current = 'second'
                root.manager.transition.direction = 'left'
    #Fifth screen
<Fifth>:
    name:"fifth"
    ScreenWallpaper:
        size:320,500
        AsyncImage:
            allow_stretch: True
            keep_ratio:False
            source:'https://img.freepik.com/free-vector/shiny-white-gray-background-with-wavy-
lines_1017-25101.jpg?w=360'
    Screen:
        size:root.height,root.width
        MDRoundFlatIconButton:
            text: "Select Video"
            icon: "folder"
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            background_color: 1, 1, 1, 1
            text_color: rgba(0,0,45,255)
            theme_icon_color: "Custom"
            icon_color: rgba(0,0,45,255)
            line_color: rgba(0,0,45,255)
            pos_hint: {'center_x': .49, 'center_y': .75}
            on_release: root.file_manager_open()
        MDLabel:
            id:txt
            text:""
            halign: "center"
            pos_hint: {"center_x":.49, "center_y":.65}
        MDRoundFlatButton:
            text:'Video Blur'
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            theme_text_color: "Custom"
            text_color: 0, 0, 0, 1
            pos_hint: {"center_x":.49,"center_y":.45}
            on_release: root.output_videoblur()
        MDRoundFlatButton:
            text:'Back'
            font_style: 'Button'
            font_size: "15dp"
            size_hint:0.10,0.05
            theme_text_color: "Custom"
            text_color: 0, 0, 0, 1
            pos_hint: {"center_x":.49,"center_y":.1}
            on_press:
                root.manager.current = 'second'
                root.manager.transition.direction = 'left'
'''
```

```
class FileChoose(Button):
    pass
pic_path=''
class First (Screen):
    pass
class Second (Screen):
    def capture_video(self):
        os.system('python yolov7_object_blurring\\un_detect_and_blur.py --source 1')
class Third(Screen):
    pass
class Fourth(Screen):
    global pic_path
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.manager_open = False
        self.file_manager = MDFileManager(
            exit_manager=self.exit_manager,
            select_path=self.select_path,
            #preview=True
        )
        self.rot_count=0
    def file_manager_open(self):
        PATH ="/"
        self.file_manager.show(PATH)  # output manager to the screen
        self.manager_open = True
    def select_path(self, path):
        global pic_path
        '''It will be called when you click on the file name
        or the catalog selection button.
        :type path: str;
        :param path: path to the selected directory or file;
        '''
        print(path)
        pic_path=path
        self.ids.image.source = path
        self.exit_manager()
    def exit_manager(self, *args):
        '''Called when the user reaches the root of the directory tree.'''
        self.manager_open = False
        self.file_manager.close()
    def output_blur(self):
        global pic_path
        if pic_path == '':
            pass
        else:
            print ('---------------------------------------------------------- ', type(pic_path))
            command = 'python yolov7_object_blurring\\un_detect_and_blur.py' + ' --source ' + pic_path
            os.system(command=command)
            detectedimg=output_of_detected_image(pic_path)
            self.ids.newimage.source = str(detectedimg)
#Fifth screen
class Fifth (Screen):
    global vid_path
    def __init__(self, **kwargs):
```

```
        super().__init__(**kwargs)
        self.manager_open = False
        self.file_manager = MDFileManager(
            exit_manager=self.exit_manager,
            select_path=self.select_path,
            #preview=True
        )
        self.rot_count=0
    def file_manager_open(self):
        PATH ="/"
        self.file_manager.show(PATH)  # output manager to the screen
        self.manager_open = True
    def select_path(self, path):
        global vid_path
        '''It will be called when you click on the file name
        or the catalog selection button.
        : type path: str;
        : param path: path to the selected directory or file;
        '''
        print(path)
        vid_path=path
        self.ids.txt.text=path
        self.exit_manager()
    def exit_manager(self, *args):
        '''Called when the user reaches the root of the directory tree.'''
        self.manager_open = False
        self.file_manager.close()
    def output_videoblur(self):
        global vid_path
        if vid_path == '':
            pass
        else:
            print('------------------------------------------------------------ ', type(pic_path))
            command = 'python yolov7_object_blurring\\un_detect_and_blur.py' + ' --source ' + vid_path
            os.system(command=command)
            detectedvid= output_of_detected_video(vid_path)
            startfile(detectedvid)
class ScreenWallpaper(BoxLayout):
    pass

#Class for Screen Manager
class WindowManager(ScreenManager):
    pass
class CameraClick(BoxLayout):
    pass
class MainApp(MDApp):
    def build(self):
        global sm
        #Adding screens with screen manager
        sm = ScreenManager()
        sm.add_widget(First(name='First'))
        sm.add_widget(Second(name='Second'))
        sm.add_widget(Third(name='third'))
        sm.add_widget(Fourth(name='fourth'))
```

```
        sm.add_widget(Fifth(name='fifth'))
        return Builder.load_string(KV)
if __name__ == "__main__":
    MainApp().run()
```

Appendix IX - Complete Code for Blurring (fordetectingandkivy) :

```
from yolov7_object_blurring.un_detect_and_blur import *
import cv2
import math
import numpy as np
import os
import time

def source_of_image(path_of_image):
    detect(source= path_of_image )
    return
def output_of_detected_video(path_of_image):
    project= 'runs/detect'
    name='exp'
    file_name = os.path.basename(path_of_image)
    file_name_without_extension, file_extension = os.path.splitext(file_name)
    print(file_name)
    print(file_extension)
    print(file_name_without_extension)
    print(file_extension)
    print(file_name_without_extension)
    directoryofdetectedimage = Path(project) / name/ file_name
    print(file_extension)
    print(file_name_without_extension)
    return directoryofdetectedimage
def output_of_detected_image(path_of_image):
    project='runs/detect'
    file_name = os.path.basename(path_of_image)
    name='exp'
    directoryofdetectedimage = Path(project) / name/ file_name
    return directoryofdetectedimage
```