Contents

1. Introduction

1.1 System Overview

The system aims to calculate the cost of parking without any human intervention, so the system should deal with cars entering to the garage and exiting the garage to calculate time-in and time-out, We have to deal with something unique in all cars which is license plates, there are a lot of steps to get the number of the plate, the system should take a video or picture containing a car and detect the license plate from it and segment the letters and numbers from the plate then separate letters and numbers from the plate then send them to third phase to classify the entire plate then send results and time in our time out to the database to calculate the customer cost

1.2 Problem Statement

It is very difficult for one model to do all these jobs so we choose to make more than one process to make it easy for models, the following are some preparations to adjust our models:

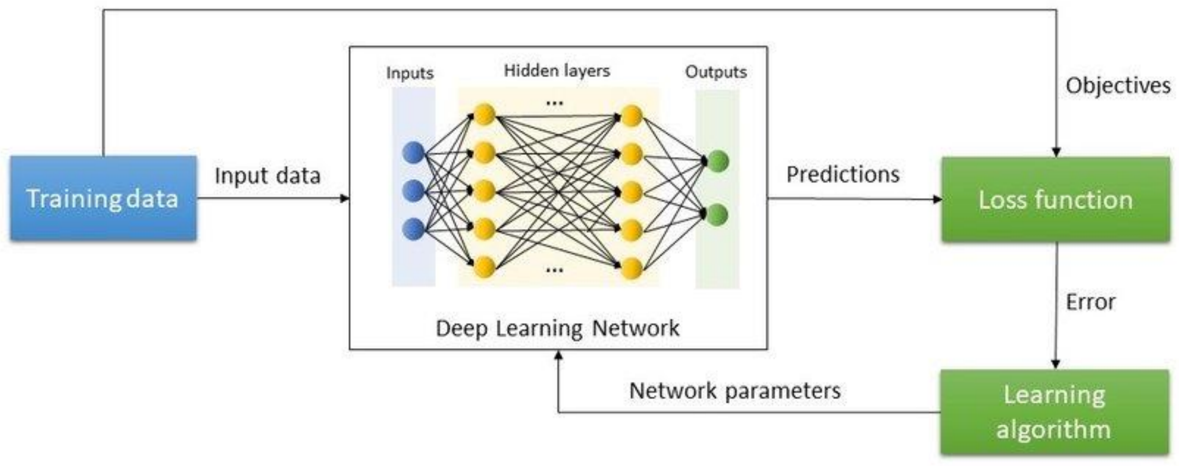
1. Search and download suitable data (Egyptian license plate)
2. Train state-of-art object detectors and evaluate them
3. Choose the best detector
4. Detect the license plate from the car and crop it
5. Try to use state of art ocr algorithms
6. Try to use segmentation to separate symbols from plate
7. Download and prepare Arabic letters and numbers
8. Explore the dataset and make sure that is suitable for our task
9. Use different pre-trained CNN models to classify letters and numbers
10. Evaluate each model and choose the best one
11. Evaluate the entire system and choose the best design which fits our task

2 Background and Related Work

2.1 Background

2.1.1 The Learning Life-Cycle

In the field of AI, there are common components for training any model, regardless of the task you are working on. Therefore, it is important to know them before starting to build the models.



The figure above shows the important steps for training any algorithm, not only neural networks. In the first step, we define the format or the method by which the model will make predictions. This format can take the form of linear equations, such as in linear regression, or it can be more complex, such as in the decision tree algorithm, or even more complex like deep learning models, such as neural networks.

The second step involves passing all predictions to the loss function. The job of the loss function is to calculate whether the model predicts true or not for each training example, where a higher loss indicates a bad model performance. The loss function plays an important role in the training process, so choosing a good one, such as Cross-Entropy is essential.

The third step is the core of the training process. The learning algorithm takes the loss value and makes some derivations based on the loss function equation and parameters to update the model parameters which are responsible for the predictions. Whenever the loss number increases, the optimizer becomes tougher for updating the parameters, and vice versa. The most common optimizers are Gradient descent, RMSprop, and Adam.

These three steps are iterative when the model loops on an entire dataset called a single epoch.

2.1.2 Deep Neural Network

Deep learning is a subfield of artificial intelligence that utilizes artificial neural networks to perform complex tasks such as image and speech recognition, natural language processing, and decision-making. Deep learning models are designed to learn from large volumes of data and improve their performance over time through the process of iterative training. This has led to remarkable breakthroughs in various fields such as computer vision and robotics.

the core component of deep learning is the neural network, which is composed of multiple layers of interconnected nodes or neurons. Each neuron receives input from other neurons and applies a mathematical function to that input to produce an output. The output of one layer of neurons becomes the input of the next layer, and this process continues until the final output is produced. The network is trained using large datasets through a process known as backpropagation, where the error between the predicted output and the actual output is minimized.

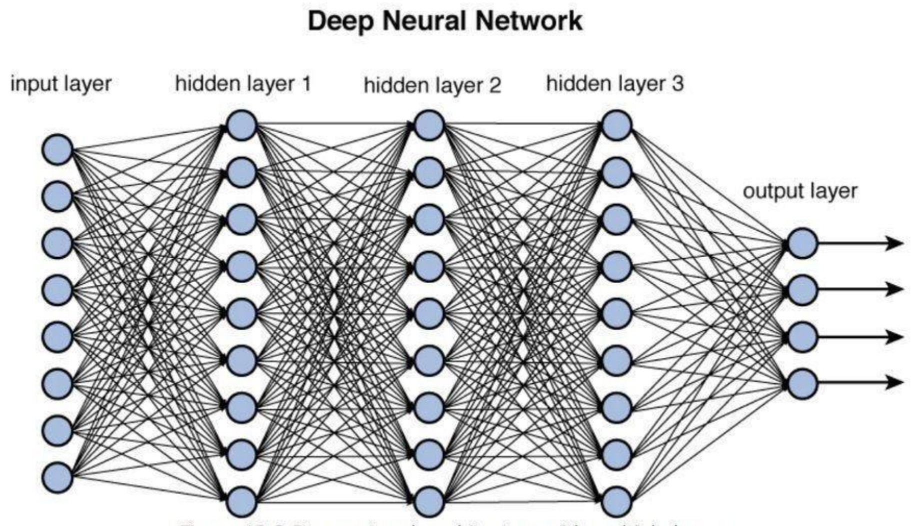
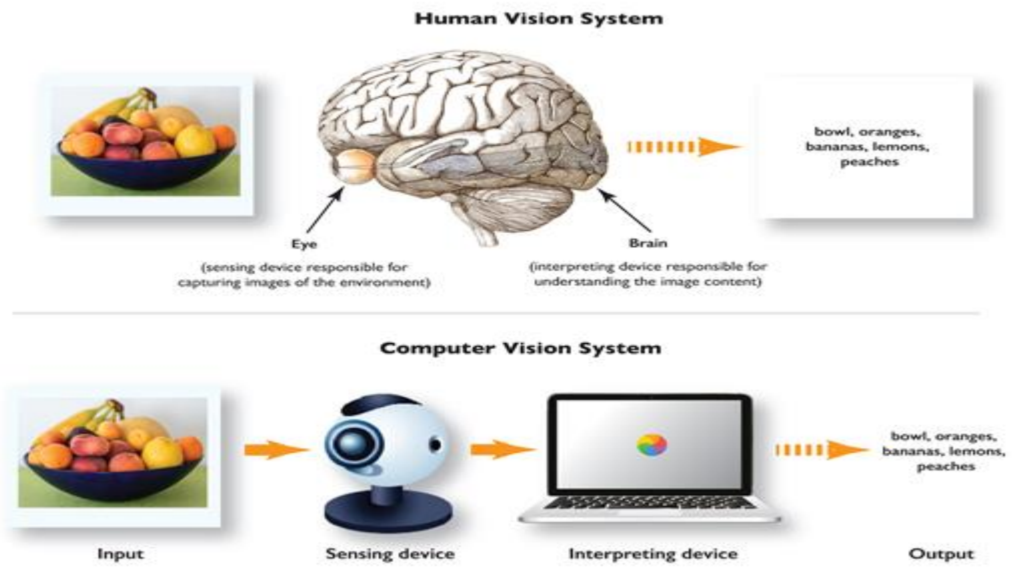


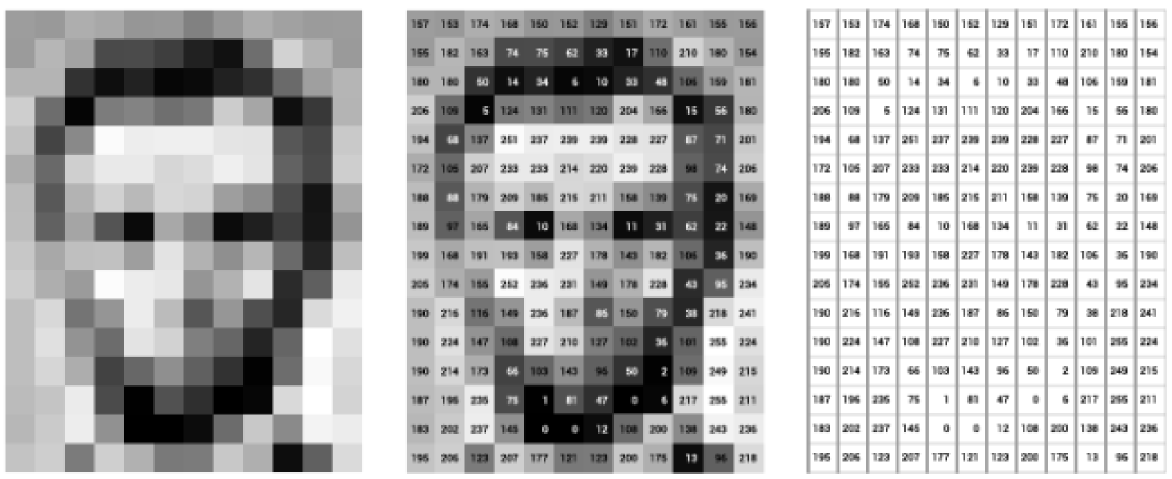
Figure 12.2 Deep network architecture with multiple layers.

2.1.3 Computer Vision

Computer Vision is an area of research in the field of Artificial Intelligence and Deep Learning, which aims to enable machines to perceive and interpret their surroundings in a manner similar to humans.


Human Vision System
bowl, oranges, bananas, lemons, peaches
Eye
(sensing device responsible for capturing images of the environment)
Brain
(interpreting device responsible for understanding the image content)
Computer Vision System
bowl, oranges, bananas, lemons, peaches
Input
Sensing device
Interpreting device
Output

Among the various subfields of Computer Vision, Image Processing, which entails the manipulation and enhancement of images via algorithms, has been a key precursor to the development of modern-day computer vision techniques. The digital image is composed of individual picture elements, or pixels, with each pixel representing a discrete unit of the image. In order to process an image, computers represent it as an array of pixels, with each pixel having values corresponding to the intensity of the three primary colors, namely red, green, and blue. Consequently, a digital image is essentially a matrix, and computer vision, in turn, becomes the study of matrices. Although simple computer vision algorithms employ linear algebra to manipulate these matrices, more advanced applications typically rely on convolutional neural networks with learnable kernels and pooling techniques for downsampling.

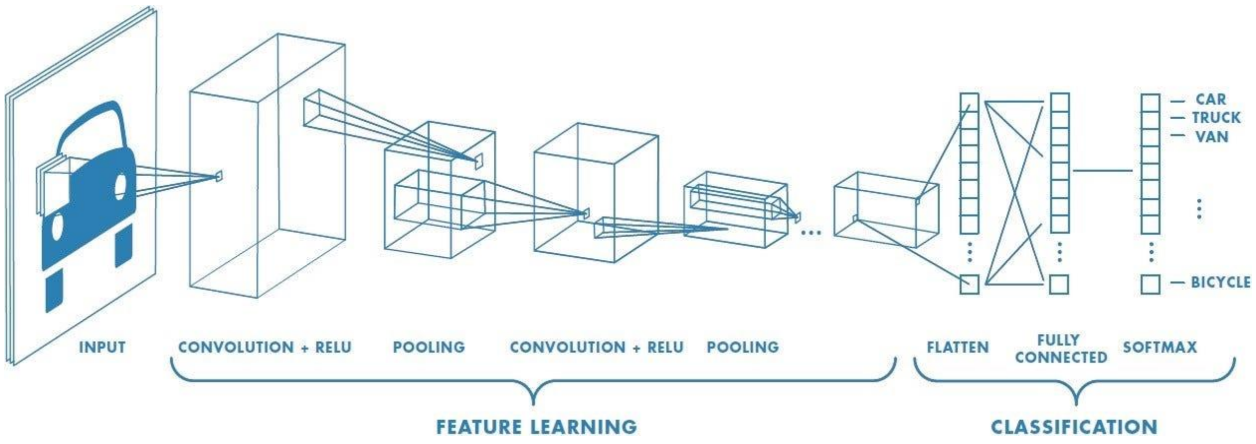


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

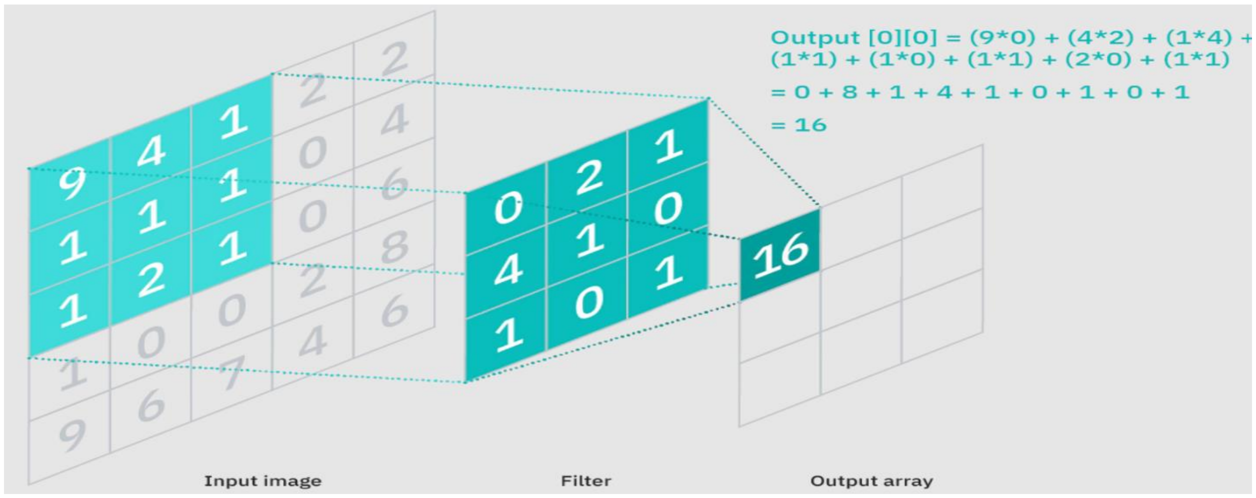
Computer vision has a number of various tasks that it is capable of doing, the most famous tasks of computer vision are image classification, object detection, and image segmentation.

2.1.4 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNNs) are a type of neural network that are particularly suited to handling image, speech, or audio signal inputs. CNNs consist of three types of layers: convolutional, pooling, and fully-connected (FC) layers.



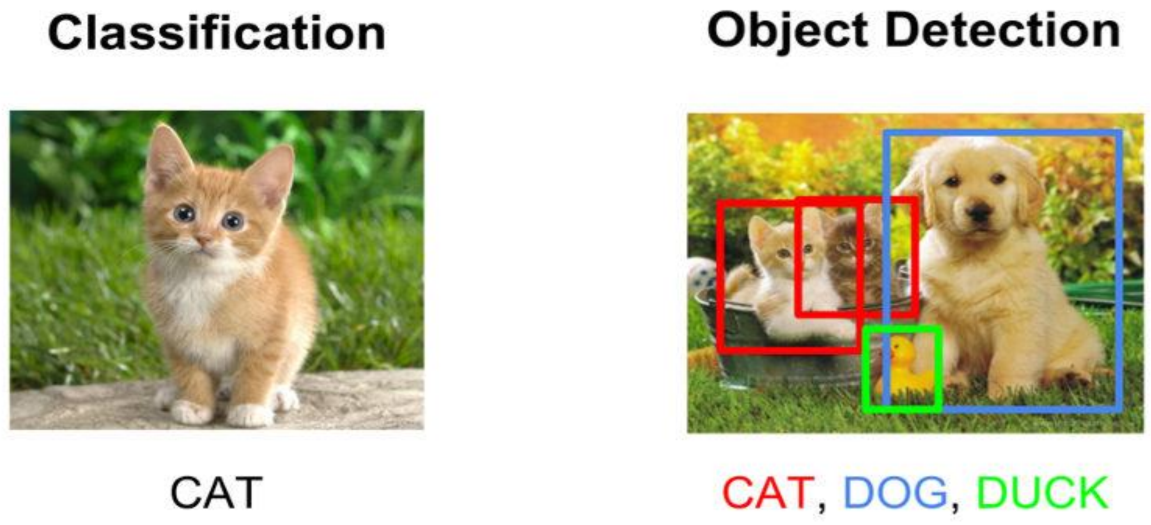
The convolutional layer is the core building block of a CNN, where computation occurs. It requires input data, a filter, and a feature map. The filter is applied to the image using convolution, and a dot product is calculated between the input pixels and the filter.The final output from this process is known as a feature map, activation map, or a convolved feature. In this process, high-level features of the image are extracted.


Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array
16

The pooling layer reduces the number of parameters in the input, improving efficiency, reducing complexity, and limiting the risk of overfitting. The fully-connected layer performs classification based on the features extracted through previous layers and their different filters, and each node in the output layer connects directly to a node in the previous layer. ReLu functions are typically used for convolutional and pooling layers, while FC layers use a softmax activation function to classify inputs appropriately. The hierarchical structure of the CNN can recognize increasingly complex features of the input as it progresses through the layers.

2.1.5 Overview of Object Detection Problem

The image classification problem is interested in determining what is in the image, while in object detection, the interest lies in determining what is in the image and where it is located within the image.



Object detection is also based on CNN and uses common architecture used in image classification as the backbone. There are two types of object detection models: one-stage detector and two-stage detector. A one-stage detector means the model can predict the class of the object and its location in the image in just one network. However, in a two-stage detector, there is a network connection with the base network to perform the bounding box of the object. Thus, one-stage detectors have high inference speeds and two-stage detectors have high localization and recognition accuracy.

State-of-the-art models can easily train data and perform object detection. The mean average precision (mAP) number and how many frames the model can process are two key factors that distinguish these models. The following are the most popular models used for object detection:

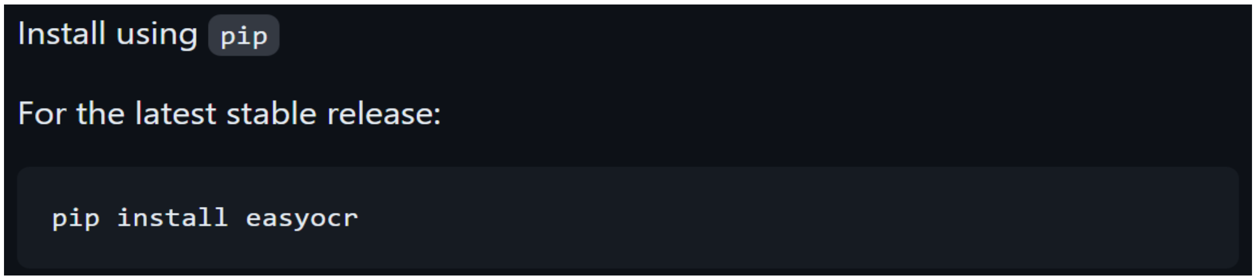
1- You only look once (YOLO) (one-stage detectors)
2- Single-shot multi-box Detector (SSD)(one-stage detectors)
3- Ratinanet(one-stage detectors)
4- EfficientDet(one-stage detectors)
5- Faster-Rcnn(two-stage detectors)

2.2 Related Work

2.2.1 pretrained Optical Character Recognition Algorithm

Optical Character Recognition (OCR) is a technology that allows for the extraction of text from an image. For example, extracting information from an ID or any other text-based document or image.

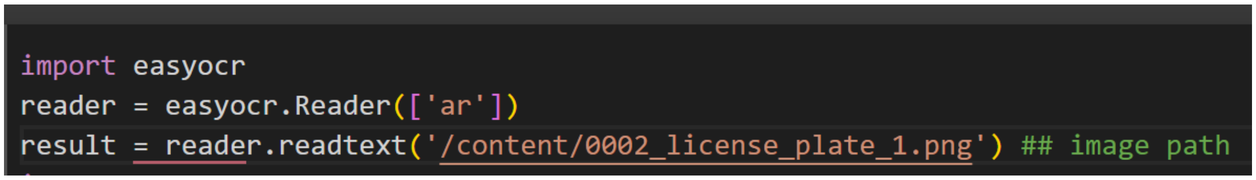
Our problem requires OCR to extract the numbers and letters from license plates. So we have to search for popular OCR algorithms. From the popular OCRs is easyocr from jaidedAI in this open source repository. So let's try it, we will install the library by


Install using `pip`

For the latest stable release:

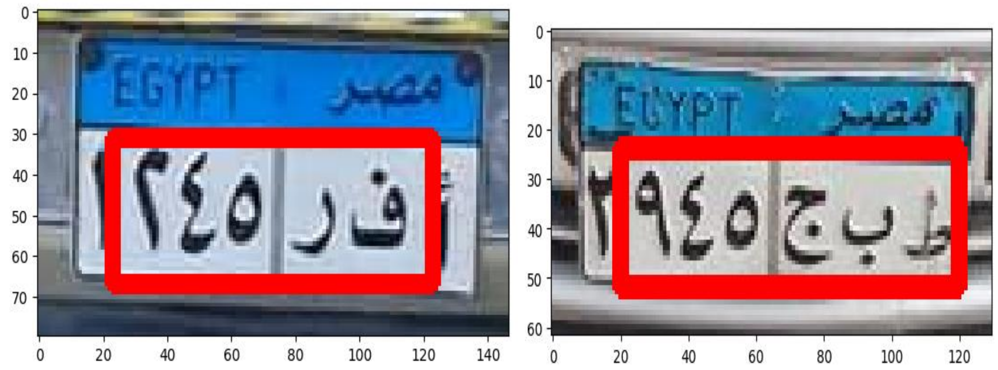
```
pip install easyocr
```

To begin, we will import the necessary library and select the language of the license plate, which in this case is Arabic (ar). We will then test the image to ensure that the OCR algorithm is working correctly

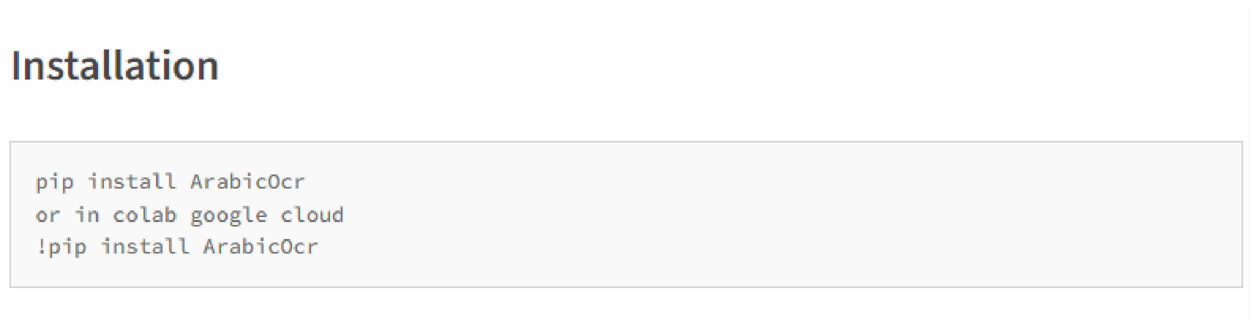


```
import easyocr
reader = easyocr.Reader(['ar'])
result = reader.readtext('/content/0002_license_plate_1.png') ## image path
```

We used two random images and will now examine their results to analyze the efficacy of the OCR algorithm.

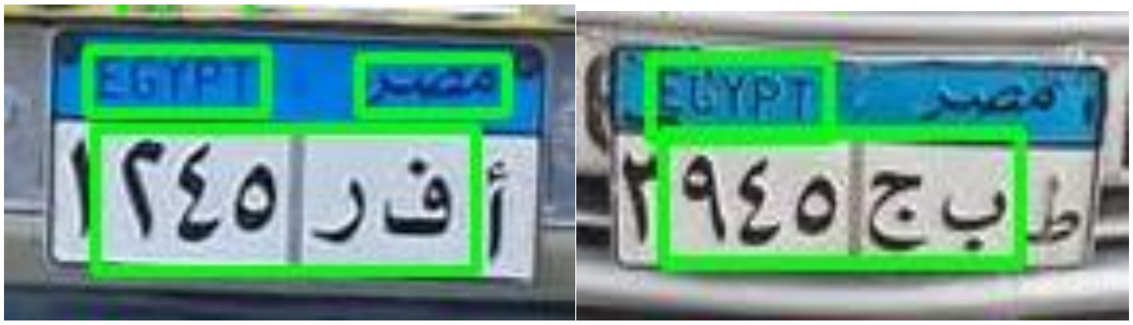


As shown in the image, the OCR model could not successfully segment all of the letters and numbers in the easy images, compared to the difficult images. So let's try another one. Our new choice is Arabic Ocr let's see the installation and results.



Installation

```
pip install ArabicOcr
or in colab google cloud
!pip install ArabicOcr
```



If the model detects the Egypt word that is not the problem, it can be handled, but the problem is when the model cannot detect all numbers and letters, so this model also failed.

OCR libraries are designed to be general-purpose and may not always be the most efficient solution for our specific needs. While they may be suitable for processing documents or images with a large amount of text, we may need to explore more specialized OCR models that are tailored to our specific problem of license plate recognition.

3 Methodology

3.1 Environment

TensorFlow, Keras, and PyTorch are three of the most popular deep-learning frameworks used today. TensorFlow is an open-source software library developed by Google, which offers a wide range of tools and resources for building and training machine learning models. Keras is a high-level neural networks API that is built on top of TensorFlow, making it easier to write and run deep learning models. PyTorch is another open-source machine learning framework developed by Facebook's AI research team, which is known for its flexibility and ease of use. All three frameworks offer support for GPUs, distributed training, and automatic differentiation, which are essential for developing and training deep learning models.

Our system is a combination of PyTorch and TensorFlow (Keras). In phases one and two, we used PyTorch due to its flexibility in handling complex tasks such as detection. However, in the third phase, we switched to TensorFlow because of its ease of use in handling simpler tasks. By using PyTorch and TensorFlow (Keras) together, we were able to leverage the strengths of each framework to develop a more robust and efficient system.

Training deep learning models can be a time-consuming process, and often requires more computational power than a local host device can provide. To address this issue, researchers and developers often turn to cloud computing platforms to train their models. Two popular cloud platforms that offer free GPU acceleration are Kaggle and Colab. Both platforms provide access to powerful GPUs, specifically the P100 (16GB), which can significantly speed up the training process.

3.2 Metrics

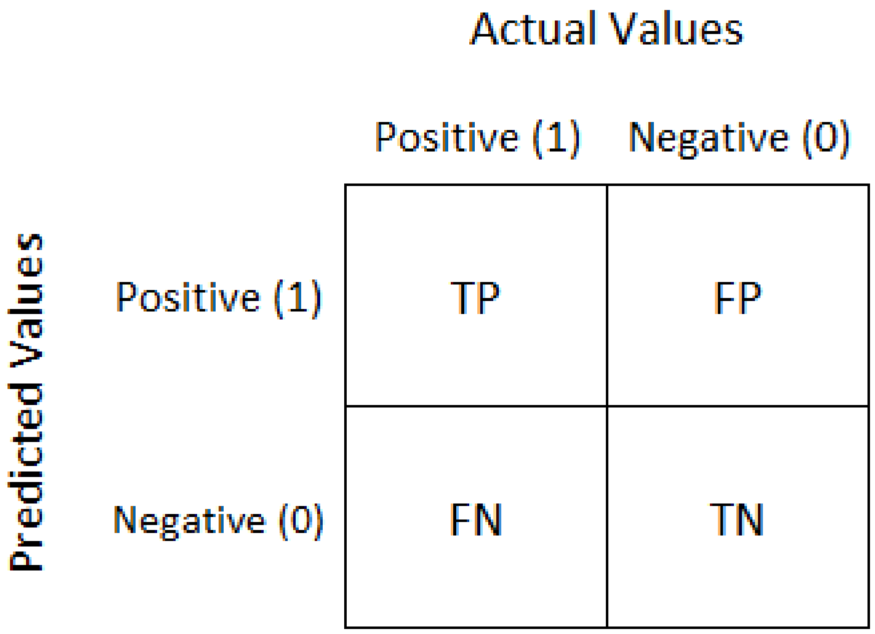
To evaluate the performance of our system, we must evaluate each of our models individually. Since each model is designed to perform a specific task, we need metrics that can accurately measure the performance of each model. These metrics should be chosen carefully to ensure that they are appropriate for the specific task at hand. In addition to evaluating individual models, we also need metrics that can be used to compare the performance of different models against each other.

The accuracy metric is a commonly used evaluation metric in many fields, including AI. The basic idea behind the accuracy metric is to measure the number of correctly classified examples out of all examples in the dataset. It is calculated by dividing the number of correctly classified examples by the total number of examples.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

While the accuracy metric is a commonly used measure of model performance, it has limitations and should not be the only way to evaluate a model. One major limitation of metric accuracy is that it can be misleading when dealing with imbalanced datasets. other evaluation metrics such as precision, recall and F1 score can be used.

To better understand precision, recall, and F1 score, it is important to first understand the confusion matrix.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Let's take an example to understand the confusion matrix. Suppose we have a cancer detection problem, where the goal is to build a model that can predict whether a patient has cancer or not. If the model predicts that the patient has cancer and the patient does have cancer, this is called a true positive. On the other hand, if the model predicts that the patient has cancer but the patient does not have cancer, this is called a false positive. Similarly, if the model predicts that the patient does not have cancer but the patient actually does have cancer, this is called a false negative.

The precision is calculated using the following formula:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

The precision number is the ratio between the number of True positives and the number of all positives made by a model which indicates how many examples the model predicted as true and already was true. A high precision score indicates that the model is making fewer false positive predictions and vice versa.

The recall is calculated using the following formula:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

The recall number indicates the ratio between the number of True positives and the number of all actual positive examples in the dataset.

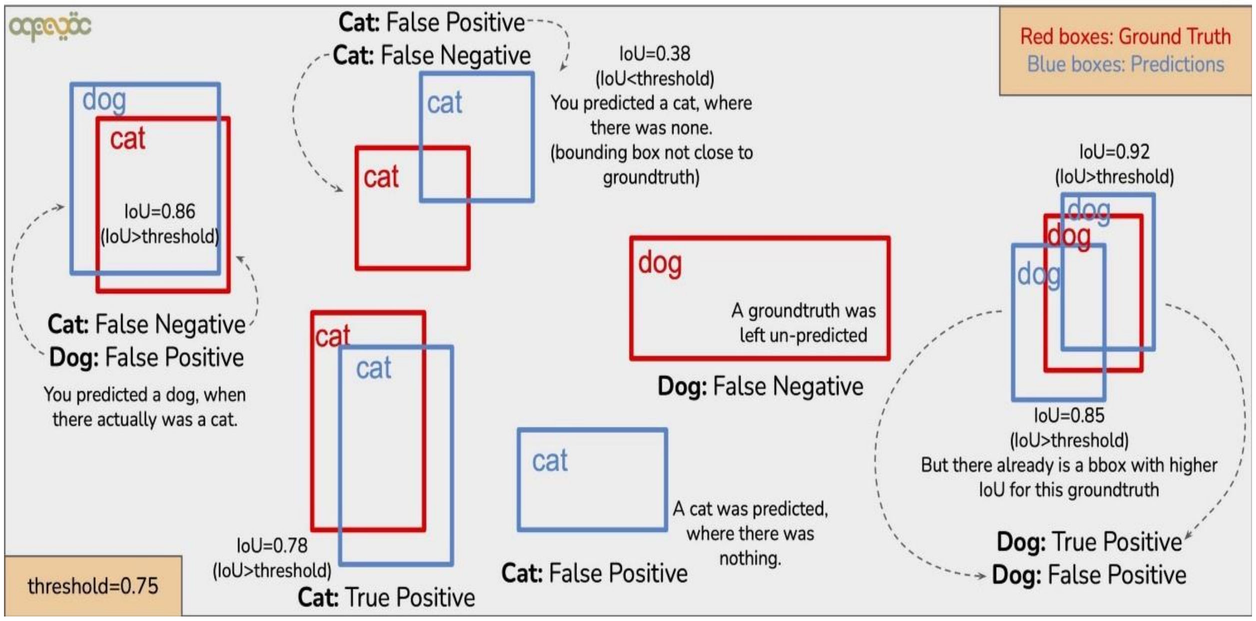
The recall is calculated using the following formula:

$$F1\ Score = 2 \times \frac{recall \times precision}{recall + precision}$$

The F1 score metric is considered a combination of recall and precision.

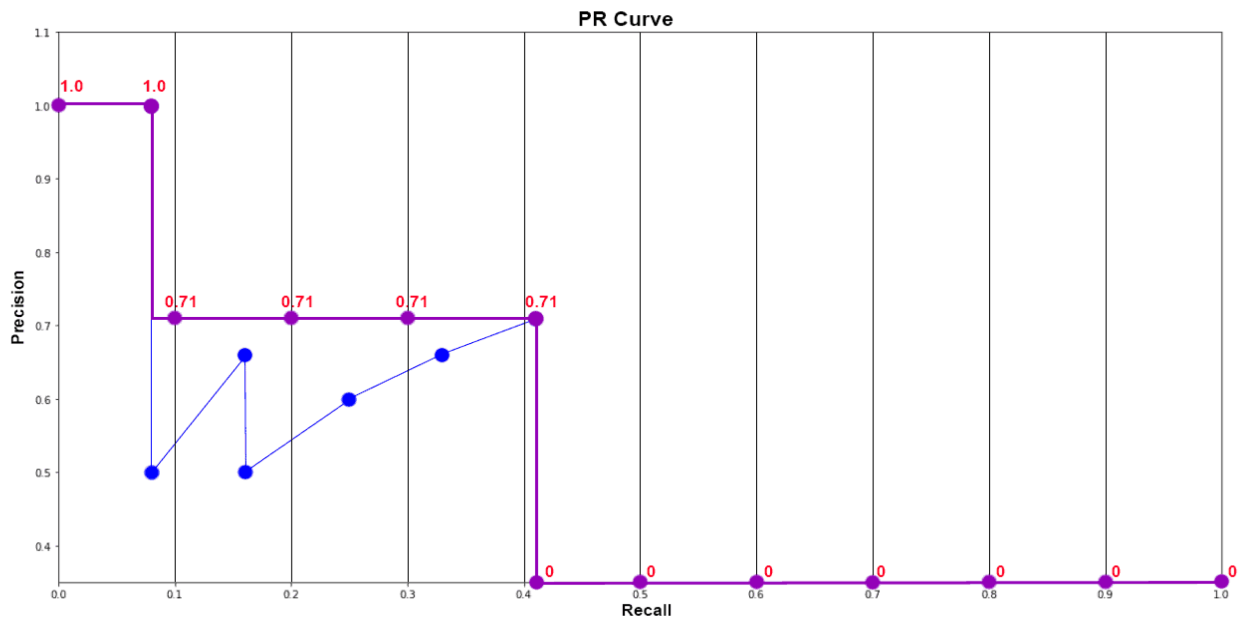
There is a distinction between image classification and object detection tasks when it comes to evaluation metrics. In image classification, the crucial factor is identifying what object appears in the image. However, in object detection, I must assess the model's performance not only in detecting the object but also in determining its location within the image and whether it is correctly identified.

Mean average precision is a very popular metric in measuring performance of the object detection models but before diving into it, we will know about intersection over union (IOU) first.



In object detection models, determining whether a predicted bounding box is a true positive or false positive is essential to calculate recall and precision. The IoU (Intersection over Union) metric helps to solve this issue by measuring the overlap between the predicted box and the ground truth box. To calculate the IoU, we find the intersection between the predicted bounding box and the ground truth bounding box, which is the area where the two boxes overlap. If the IoU value between the predicted box and the ground truth box is greater than or equal to a certain threshold, usually 0.5 or 0.7, we consider the predicted box a true positive, else we consider the predicted box a false positive.

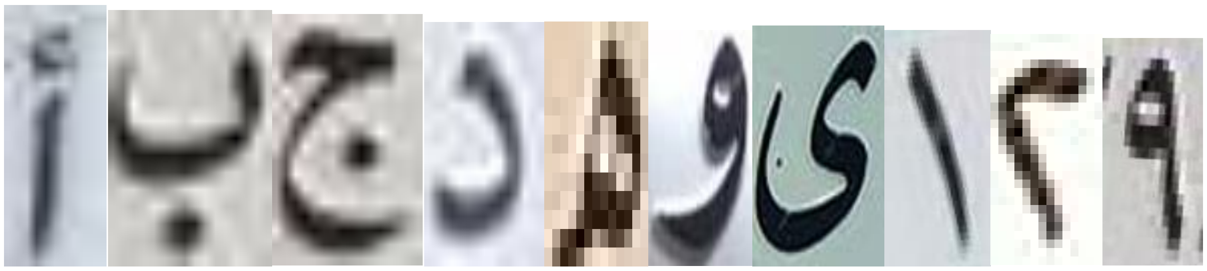
To calculate mAP, we first need to calculate AP by sorting all examples by their confidence scores. Then, we calculate the precision and recall for each example and all the examples that come before it. For instance, to calculate the AP for the third example, we need to compute the precision and recall for the first three examples. Next, we plot a precision-recall curve, The average precision is the area under the curve (AUC). The mAP is the mean of the AP values for all classes.



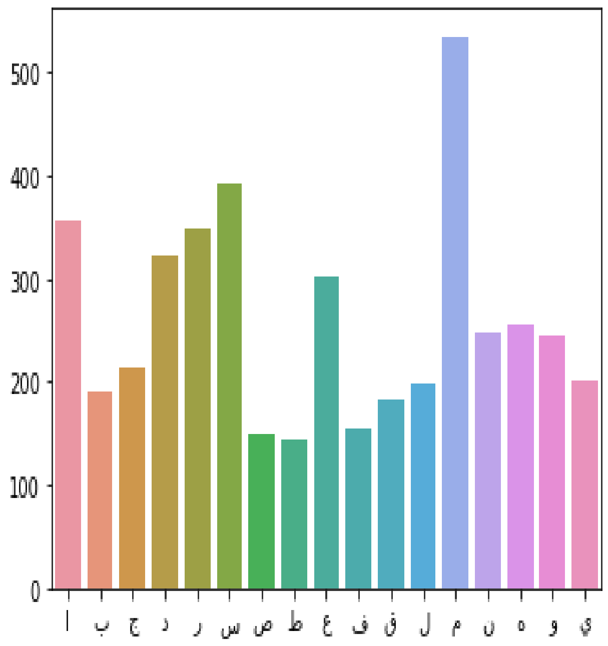
3.3 Data Exploration

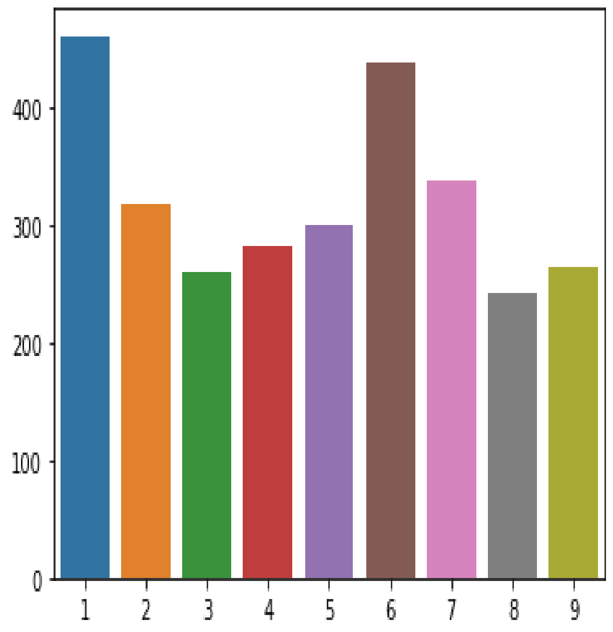
Datasets of Egyptian license plates are very rare so getting data clear is a very difficult thing, after searching we chose this dataset from ahmedramadan96, It contains about 2100 images of Egyptian cars in different shapes, different positions, and different lighting, each image has text file containing the annotation of plates in Yolo format (class_id, x_center, y_center, width, height),we will split the data into 80% (1700 images) for training and 20% (400 images) for testing

Also in this dataset, there are separated letters and numbers cropped from plates that will help us to train the third phase with data from the same distribution in the same plate font, it contains about 4450 letter images in 17 classes (in Egypt, the government uses only 17 of 28 Arabic letters) and 2900 number images in 9 classes.



I will split the data into 60% to train, 20% for validation, and 20% for testing in both models.





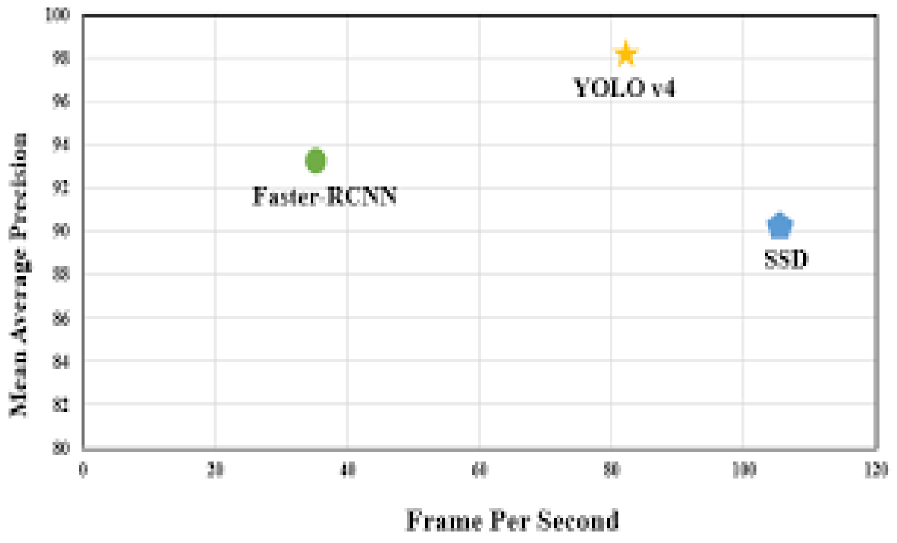
In the letters dataset on average, each letter has about 200 images, as we can see in the letters bar plot there is a tough class imbalance, in the numbers dataset, each number has about 250 images, and also suffering from class imbalance

3.4 First Phase

3.4.1 Idea Overview

our task in the first phase is to detect the license plate so we have a detection problem, In the last few years tremendous progress has been made in computer vision, especially in object detection and segmentation, The models become more robust without any preprocessing or any feature engineering on datasets.

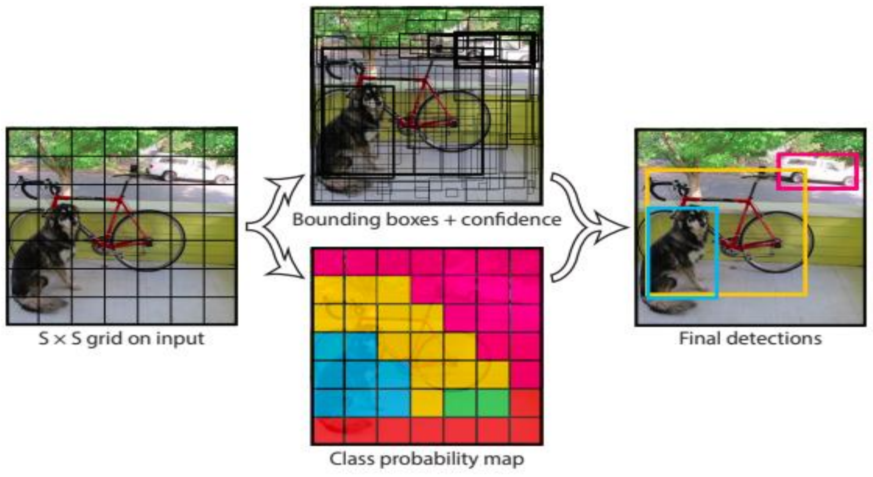
Currently, the most active area of research in object detection is focused on improving the YOLO (You Only Look Once) algorithm. YOLO has consistently achieved state-of-the-art results in object detection and has become a popular choice for many computer vision applications. As a result, we have decided to use YOLO in the first phase of our project.



3.4.2 The idea of Yolo?

Yolo paper was published in 2015, it was the faster algorithm, which can process images in real-time at 45 frames per second meanwhile, the state of art object detector was faster-Rcnn and can process 8 frames per second

YOLO (You Only Look Once) is a one-stage detector that treats object detection as a single. This approach allows the model to detect and classify objects in one phase, which contributes to its speed and efficiency. The basic idea behind YOLO is to divide the input image into a grid of SxS cells. Each cell predicts the probability of containing an object, the bounding box (x_center, y_center, width, height) of the object if present, and the probability of the object's class.



The meaning of containing an object or not is that a grid contains the midpoint of the object. Therefore if an image contains three objects, there are three grids responsible for detecting them and their classes. The range of objectness score is between 0 and 1, 0 if there is no object and 1 if the grid is sure 100% there is an object.

That was the idea of YOLO version one, this version aimed to produce a one-stage detector to use in real-time applications like self-driving cars so the mAP of yolov1 was a low little bit compared to Faster-R-CNN.

After the first versions, different versions have been proposed with many modifications based on the Yolo idea.

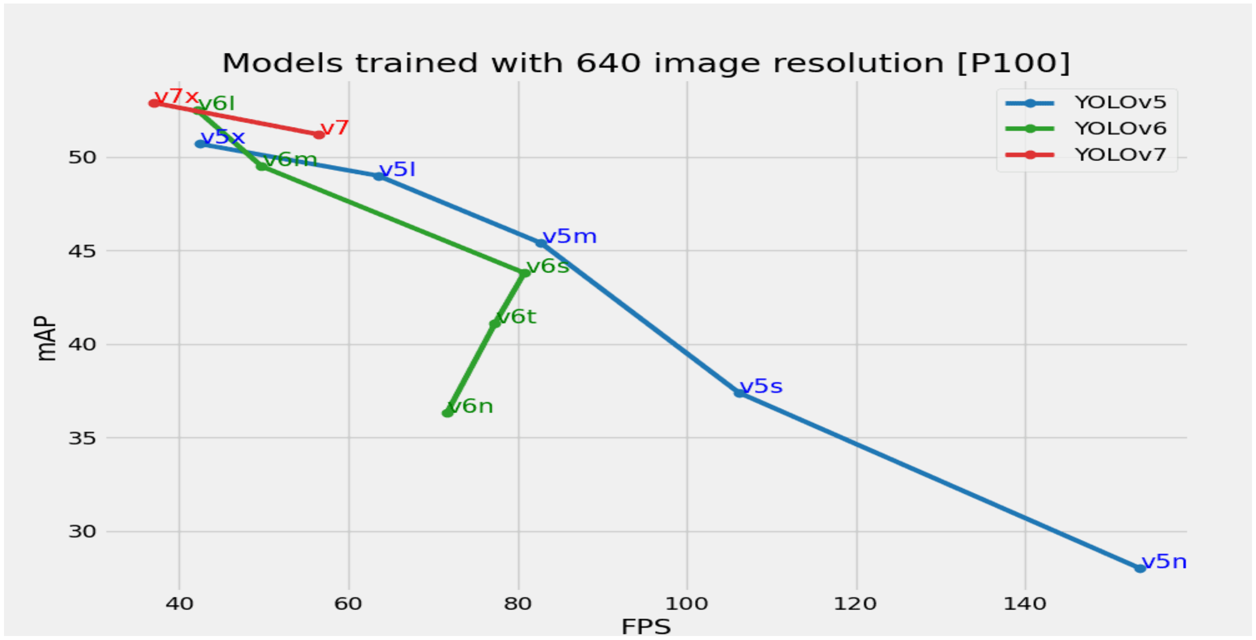
The following are some examples of these modifications:

1- architecture of network
2- the idea of anchor boxes
3- adding new blocks like(SSP,PAN)
4- new backbones
5- new augmentations techniques
6- new modifications in the loss function

Each version of the architecture comes in different types based on its size, resulting in varying mean average precision (mAP) and number of frames per second (FPS). As the size of the architecture increases, the mAP should also increase but the FPS may decrease.

3.4.3 Yolo Version 5 small

we believe plate detection is a straightforward task, and therefore, we don't require a complex model. Using a complex model would increase the number of parameters and FLOPs, therefore decreeing the FPS, which is unnecessary.

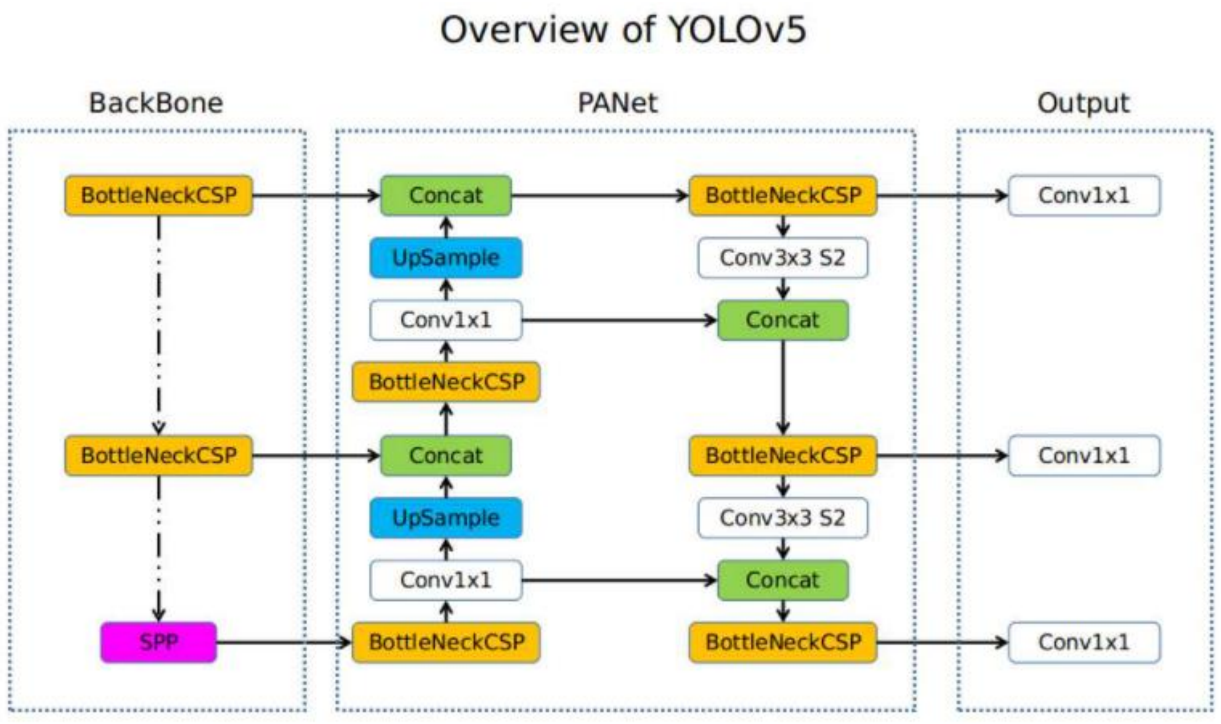


As shown in the above Figure YOLOv5 Nano has a lower mAP but higher frames per second. However, for our task, we do not require a high number of FPS at the cost of a low mAP. Therefore, YOLOv5 Nano is not suitable for our needs. As previously mentioned, our task involves a single class and is relatively straightforward. Therefore, we require a model that offers a high FPS while also providing a good mAP. YOLOv5 Small meets our requirements.

Yolov5 came a few months after v4 has published, with improvements in yolov4 on:

1- the implementation was in PyTorch not in Darknet as usual in previous versions
2- mosaic data augmentation
3- auto-learning bounding box anchors
4- loss function and training process be easier
5- Yolo v5 is extremely fast than Yolo v4

Yolov5 architecture:



The YOLOv5s architecture consists of a CSPDarknet53 backbone, an SPP (Spatial Pyramid Pooling) neck, and a YOLOv5 head. Here's a brief overview of each component:

Backbone: The backbone is responsible for extracting high-level features from the input image. YOLOv5s uses a CSPDarknet53 backbone. With using

Neck: The neck is responsible for combining features from different scales and resolutions to improve the accuracy of object detection. YOLOv5s uses an SPP (Spatial Pyramid Pooling) neck, which applies multiple pooling operations with different kernel sizes to capture features at different scales. And Path Aggregation Network (PANet) modified with BottleNeckCSP.

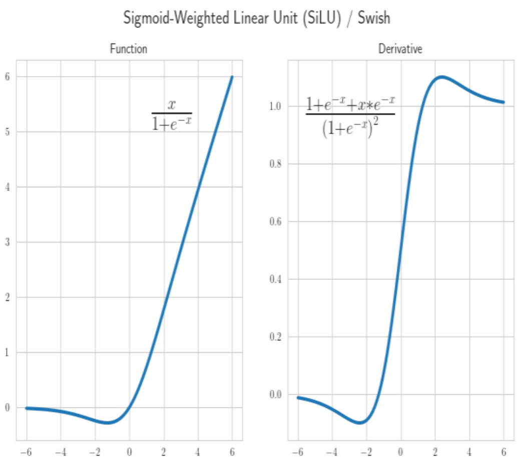
Head: The head is responsible for predicting the bounding boxes and class probabilities for the objects in the input image. YOLOv5s uses a YOLOv5 head, which consists of several convolutional layers followed by a detection layer. The detection layer uses anchor boxes to predict the bounding boxes and class probabilities for the objects in the input image. Yolo v5 makes improvements in detection layers to solve a lot of problems such as grid sensitivity.

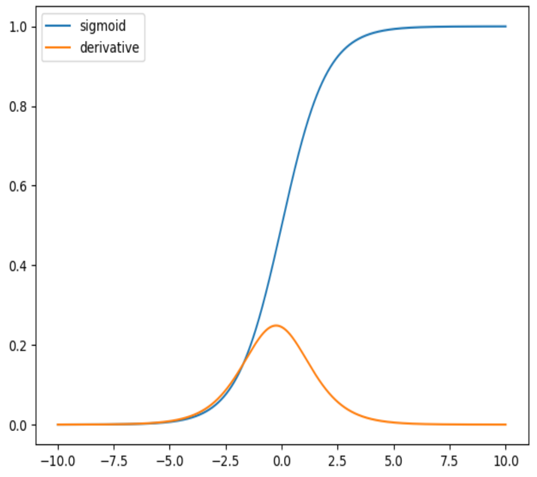
$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w \cdot e^{t_w} \\ b_h &= p_h \cdot e^{t_h} \end{aligned} \qquad \begin{aligned} b_x &= (2 \cdot \sigma(t_x) - 0.5) + c_x \\ b_y &= (2 \cdot \sigma(t_y) - 0.5) + c_y \\ b_w &= p_w \cdot (2 \cdot \sigma(t_w))^2 \\ b_h &= p_h \cdot (2 \cdot \sigma(t_h))^2 \end{aligned}$$

(a) (b)

(a) for previous versions (b) for version 5

In the entire network without detection layers, the authors used Sigmoid-weighted Linear Unit for the activation function but in 3 detection layers, they used Sigmoid.





Yolov5 loss:

Yolov5 loss consists of three 3 final losses one for bounding box coordinate, one for objectness, and the last one for classes **(if there is more than one class)**

Bounding box loss is responsible for penalizing the model if predicts boxes are not accurate or not like the ground truth. Complete IoU Loss (Ciou) based on Iou and Distance IoU Loss is the function that calculates the box loss.

The objectness loss is responsible for penalizing the model if it predicts that a grid contains an object when it does not, or if it predicts that a grid does not contain an object when it does. This ensures that the model learns to accurately detect objects and predict their locations in an image. In version 5, the Objectness loss is calculated by the Binary cross-entropy Function.

The class loss is responsible for penalizing the model if it predicts the wrong class for the object.

And also calculated by Binary cross-entropy Function.

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

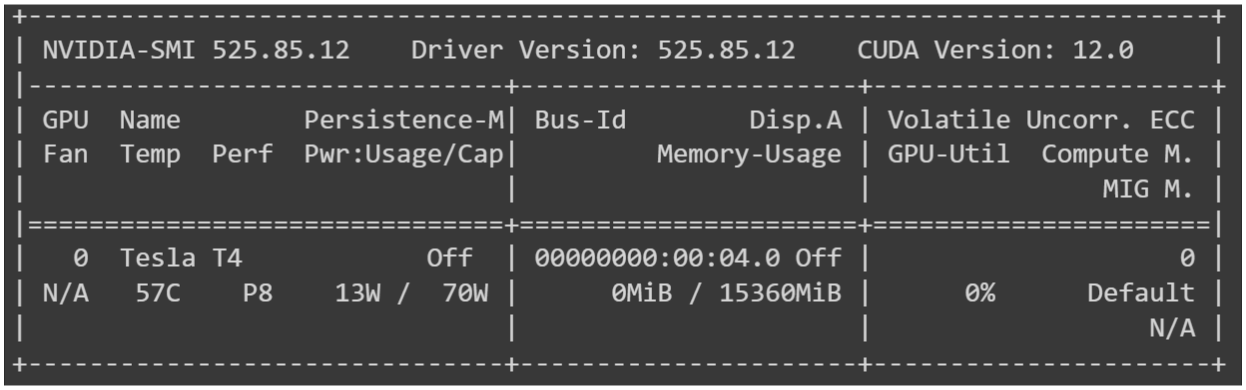
In the paper's repository, there are many modifications to the loss function, such as assigning anchors to the dataset based on the ratio between the object's width and height and the anchors, and assigning three grid cells (not only one) that are responsible for predicting the presence of an object.

Yolov5 small:

The FLOPs of Yolov5s is 16.5 at 640 image size with 7.2 M parameters, it's agreeable resource allocation in our environment. We will use the paper's implementation from Ultralytics.

3.4.4 Training Process

Now that we have obtained what we need to begin training, it is time to prepare our environments and start the training process. Our cloud will be google colab with GPU Tesla T4 15360MiB.



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   57C    P8    13W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

To begin, we will clone the repository that contains the dataset. Once we have done so, we will split the dataset into two folders: 'train' and 'validation'. The 'train' folder will contain 1688 images and their annotation files, while the 'validation' folder will contain 400 randomly selected images and their annotation files. then, we will save all folders to our Drive.

Next, we will create a YAML file containing the file paths for the 'train' and 'validation' datasets and the classes for our object detection task.

Next, we will set up the ultralytics/yolov5 package and modify the hyperparameters for our specific task. The following are the key hyperparameters that we will adjust:

1- Image size: 416x416
2- Batch size: 16
3- Number of epochs: 50
4- Learning rate: 0.01
5- Optimizer: 'Adam'
6- Augmentation: True
7- Pretrained with yolov5s.pt

Once we have set up the package and adjusted the hyperparameters, we can begin training the model. During the training process, we will save the results and weights to our Google Drive for each epoch.

3.5 Second Phase

3.5.1 Idea Overview

As we have seen with pretrained OCRs, they cannot be used effectively due to difficulties in segmenting letters and numbers. Therefore, a new solution is needed to accurately segment these characters.

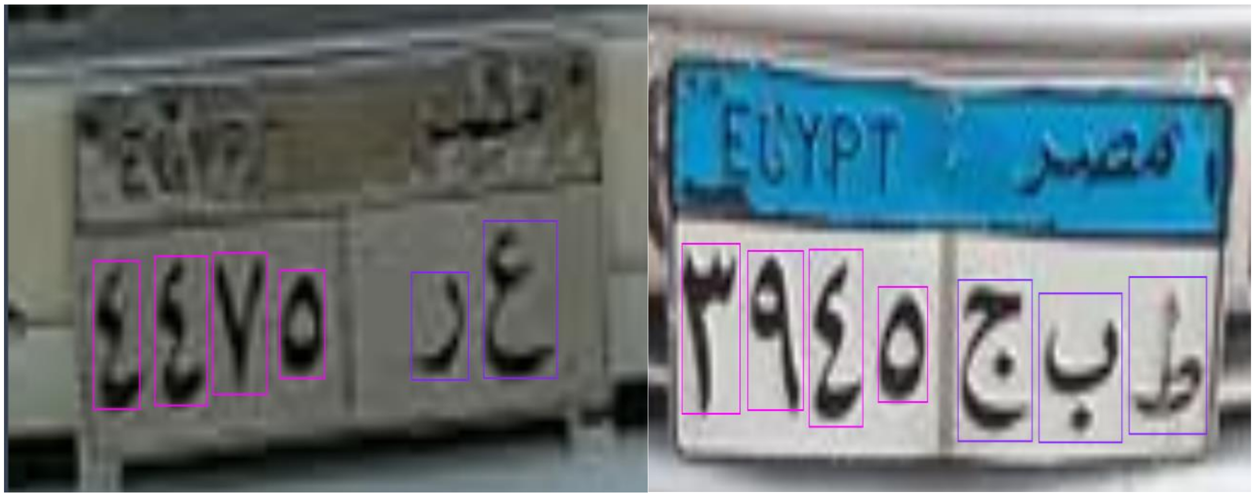
When using the 'find contours' method to segment characters in an image, it is necessary to apply filters such as dilation and Gaussian blur and adjust the parameters for each filter. However, this can result in overfitting the code to specific images. A more general approach is to use deep learning for character segmentation, as this allows the CNN to learn the best filter values to apply to all images.

To recognize characters in an image, we can use an object detection model to detect and crop them from the image. There are two approaches we can take: the first is to create a model with two classes, one for letters and one for numbers. The second approach is to create a model with a single class and separate letters and numbers based on the area of all letters and the area of all numbers on the plate. We can evaluate both methods and choose the best approach based on the results.

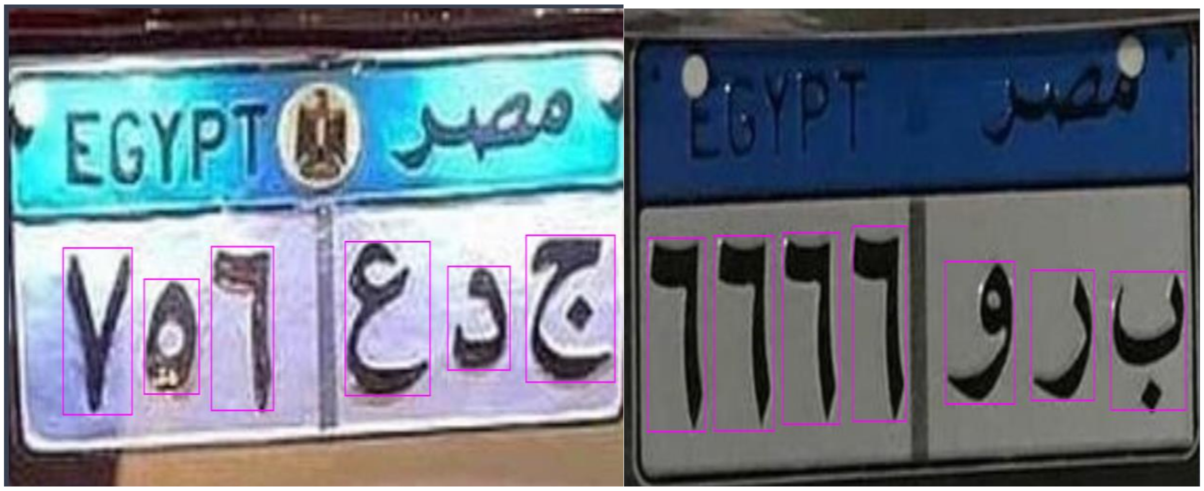
3.5.2 Data annotation

To implement our idea, we need to prepare the datasets for training on YOLO models. We will use Roboflow to annotate our dataset, as it is an easy-to-use tool for annotating data for tasks such as object detection, segmentation, and pose estimation. Additionally, Roboflow allows us to add image augmentation to improve the diversity and quality of our dataset.

To test our idea without wasting time, we will start with a small sample of data. We will annotate 240 images in YOLO format and apply augmentation techniques such as blur and rotation. Because most of the license plates are centred in the image and the cameras are often positioned to the left or right of the vehicle, we will apply a small rotation range of (-12,12) degrees to simulate this rotation. We made a version for two classes and a version for one class.



Examples of annotated data with two classes



Examples of annotated data with one class

During the annotation phase, our main concern is to ensure accurate and precise annotations. The bounding boxes should be placed as close to the object as possible to allow the model to learn the object and avoid confusion with other entities in the image. To increase the diversity of our dataset, we will generate two augmented versions of each image after the augmentation process is applied to the original data. So the total training dataset is 422 images and 50 images for validation.

3.5.3 Train Multiple Detection Models

To choose models to evaluate, we have to figure out the trade-offs. In this phase need to get the best result whatever the time it takes. We don't need high FPS in this phase. We believe that this task is somewhat challenging, we plan to use a more complex model with more epochs to achieve the best possible fit. We will also experiment with different optimizers to improve the performance of the model. we hope to achieve better accuracy and make the most of the limited dataset we have.

We plan to use default settings as a starting point to test the differences between models. By using default settings, we can establish a baseline and compare the performance of different models under the same conditions, the default setting is the following:

1- Image size: 360x360
2- Batch size: 32
3- Number of epochs: 150
4- Learning rate: 0.01
5- Optimizer: ‘SGD’
6- Augmentation: True
7- Pretrained

Now, we can start training and save the results of each model. We will try:

1-Yolo v5 medium

2-Yolo v5 Xlarge

3.6 Third Phase

3.6.1 Idea Overview

After we have cropped the letters and numbers from the license plate, the next step is the final phase of classifying these characters. This task is relatively easy for most deep-learning models available today. However, we need to ensure that the model we choose achieves high accuracy while requiring low FLOPS. This means we want a model that can accurately classify the characters without requiring a significant amount of computational power.

We have two options for developing a model to classify license plates: the first is to create a single model that can classify both letters and numbers, while the second is to create two separate models, one for letters and one for numbers. It is important to note that in the Arabic language, some letters can resemble Arabic numerals, which may confuse the model, especially when dealing with not clear or anointed license plates in Egypt.

After considering our options, we have decided to develop two separate image classification models using Convolutional Neural Networks (CNNs). The first model will be trained to classify letters, with 17 classes representing the subset of Arabic letters used on license plates. The second model will be trained to classify numbers, with 9 classes representing the numerical digits found on license plates.

3.6.2 Data Preparation

It's important to be aware of the types of letters used on license plates by the traffic police, as the government doesn't typically use all Arabic letters. After conducting research, we discovered that the government only utilizes 17 Arabic letters on license plates.

For our model to perform well, we require cropped images of letters and numbers that match the font of the license plate. Using data with the same classes but in a different font could negatively impact the model's accuracy. Therefore, we will be using the dataset from ahmedramadan96, which has a smaller number of data sets but is sufficient for this simple task. We downloaded the dataset and split it into two folders: one for letters with 17 classes and another for numbers with 9 classes.

3.6.3 Augmentation

Since the dataset is small, we will need to use augmentation techniques to expose the model to more variations of the dataset. This will prevent the model from overfitting and help it generalize across different classes.

In our case, we will apply augmentation techniques that are relevant to the potential variations that could occur in license plate images. These techniques will help the model learn to recognize license plates in different orientations, scales, positions, and levels of clarity. By doing so, we can create a more diverse dataset that better represents the real-world scenarios the model may encounter.

To apply augmentation techniques to our dataset, we will use the ImageDataGenerator function from TensorFlow Keras. This function works by randomly applying a set of techniques based on the settings we specify, we can efficiently generate a larger and more diverse dataset without having to manually apply each technique to each image.

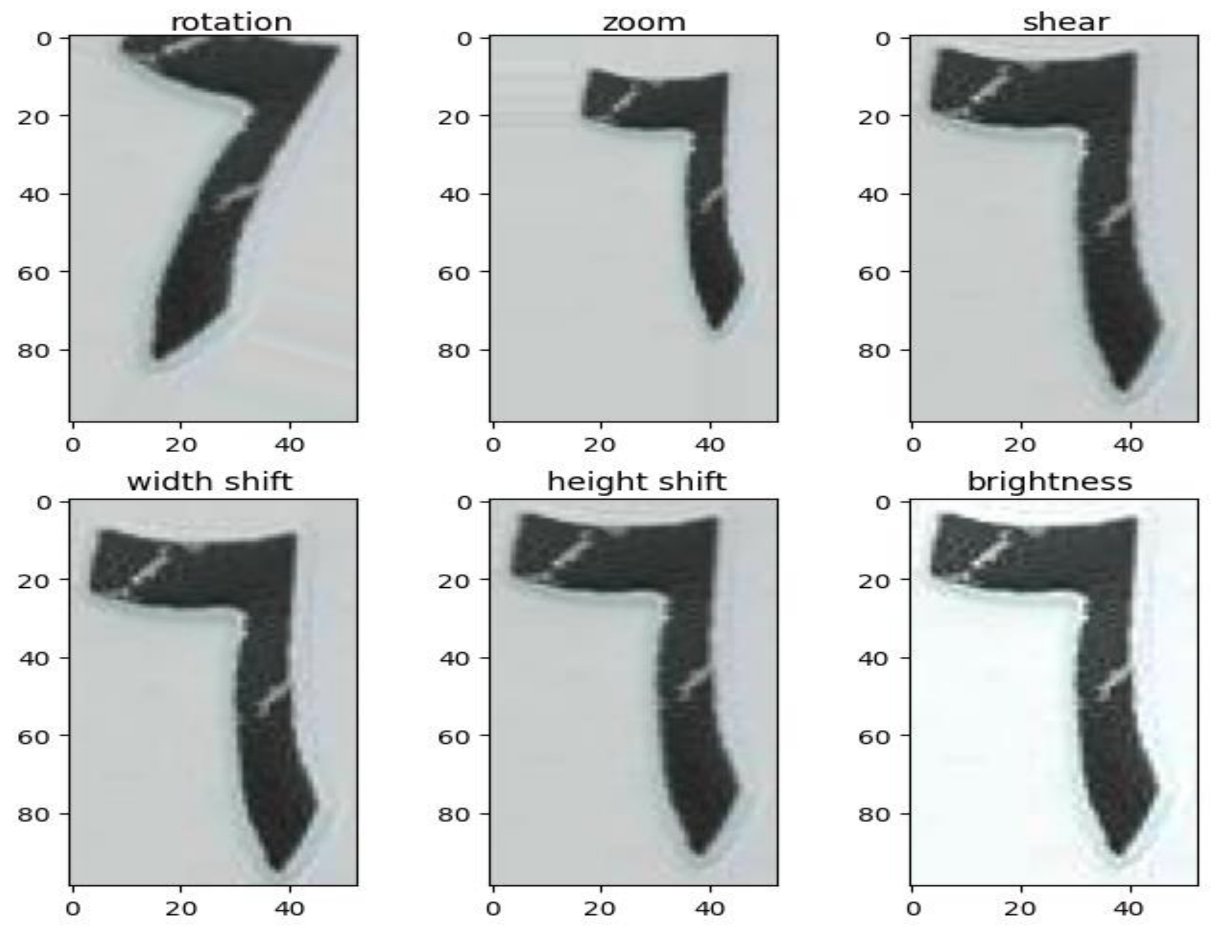
The first and most important preprocessing step is to normalize the RGB channels by dividing them by 255. This ensures that the pixel values are in the range of 0 to 1, which is necessary for the model to learn effectively.

Next, we will apply a rotation of 30 degrees to some of the images in the dataset. This is because there are images in the dataset that are taken from the front, and rotating them can help the model learn to recognize license plates that are not perfectly aligned.

We will also apply additional augmentation techniques, including zooming, shearing, and shifting the width and height of the images. These techniques can help the model learn to recognize license plates that have been cropped or shifted in various ways during the segmentation process. For example, zooming can help the model recognize plates that have been cropped too tightly while shearing can help the model recognize plates that have been tilted or distorted. Shifting the width and height of the images can help the model learn to recognize plates that have been shifted during the segmentation process. By applying these techniques, we can create a more robust and accurate model that can handle a wide range of license plate variations.

The last technique is changes in brightness. This involves adjusting the brightness of an image by increasing or decreasing the intensity of its pixels. It can be useful in improving image quality when there are significant differences in brightness, such as between day and night.

There are examples of augmentation on 6’s image



3.6.4 Training process

Now we will start with model mobilenetv2, with this easy task if it fails to generalize the object we will try other model bigger than mobilenetv2 such as DenseNet

We will use categorical cross-entropy as the loss function, which is based on the maximum likelihood estimation and calculated by:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

The optimizer will be Adaptive Moment Estimation (Adam), which is considered a combination of RMSProp and Gradient Descent, with a default learning rate

We will set up callbacks such as reducing the learning rate on a plateau, early stopping, and saving the best weights.

For the classification part of the whole network, we will use two layers with 1024 nodes each, with the ReLU activation function and a dropout rate of 0.3. The final layer will be the output layer with the number of classes and the softmax activation function.

the number of epochs needed to train a model is dependent on the complexity of the model and the desired level of loss reduction.

Now we can define models and start evaluating them.

MobileNet V2

MobileNet is a convolutional neural network architecture designed for efficient deployment on mobile and embedded devices with limited computational resources. It was developed by Google researchers (Howard et al.) in 2017 .

The primary goal of MobileNet is to provide a lightweight and efficient model for tasks such as image classification and object detection. It achieves this by utilizing depthwise separable convolutions, which split the standard convolution operation into separate depthwise and pointwise convolutions. This approach significantly reduces the computational cost while still capturing important spatial and channel-wise information.

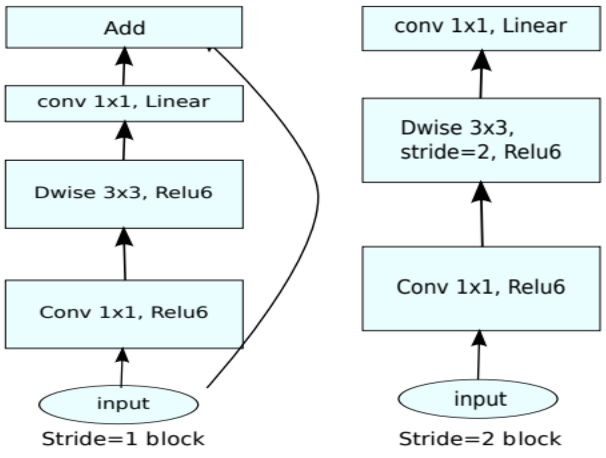
MobileNet offers different versions, such as MobileNetV1, MobileNetV2, and MobileNetV3, each introducing improvements and optimizations over the previous version. These versions vary in terms of network depth, number of parameters, and computational requirements, allowing users to choose the most suitable variant based on their specific needs and constraints.

With its emphasis on efficiency and lightweight design, MobileNet has become popular in various real-time and resource-constrained applications, including mobile and embedded vision applications, where processing power and memory limitations are critical considerations.

MobileNetV2 was introduced by the authors in 2018 in their paper MobileNetV2: Inverted Residuals and Linear Bottlenecks.

The MobileNet V2 model consists of 53 convolution layers and 1 average pooling layer, with a computational capacity of approximately 350 GFLOPs (giga floating-point operations per second). It incorporates two main components: the Inverted Residual Block and the Bottleneck Residual Block.

The architecture of MobileNet V2 employs two types of convolution layers: 1x1 convolutions and 3x3 depthwise convolutions. Each block within the model comprises three distinct layers: a 1x1 convolution layer with the ReLU6 activation function, a depthwise convolution layer, and another 1x1 convolution layer without any linearity.



(d) Mobilenet V2

There are two types of blocks in MobileNet V2: Stride 1 Blocks and Stride 2 Blocks. The internal components of these blocks are as follows:

Stride 1 Block:

- Input
- 1x1 Convolution with the ReLU6 activation function
- Depthwise Convolution with the ReLU6 activation function
- 1x1 Convolution without any linearity
- Element-wise addition (with the input)

Stride 2 Block:

- Input
- 1x1 Convolution with the ReLU6 activation function
- Depthwise Convolution with a stride of 2 and the ReLU6 activation function
- 1x1 Convolution without any linearity

These components and structures allow MobileNet V2 to efficiently extract features from input images while reducing the computational complexity. The use of depthwise convolutions and the specific block design contribute to the model’s effectiveness in resource-constrained scenarios, such as mobile and embedded devices, where computational efficiency is crucial.

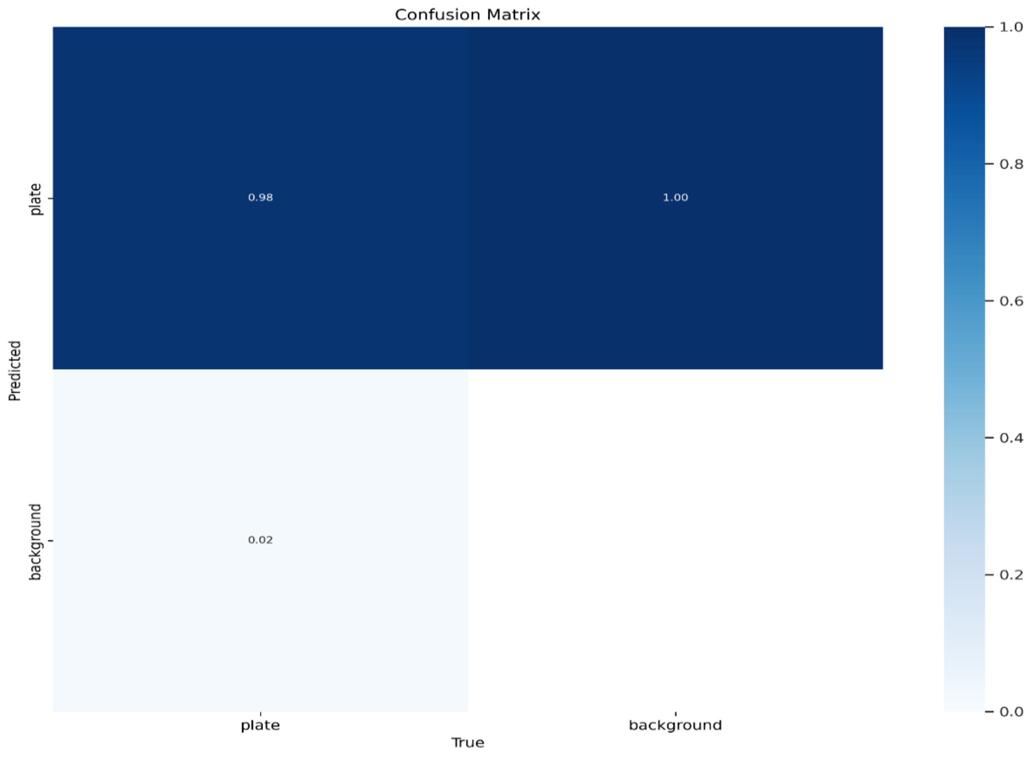
4 Results Analysis

4.1 First Phase

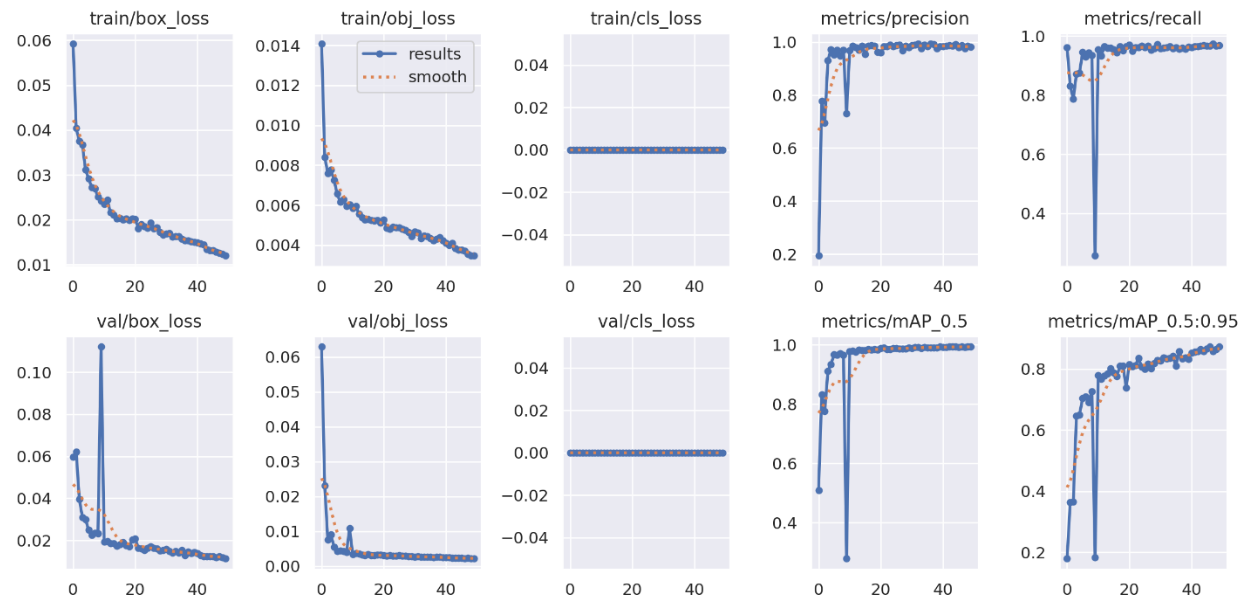
4.1.1 Yolov5 Small

As we anticipated, plate detection in the first phase was a straightforward task for a model like YOLOv5s. After training for 100 epochs with the Adaptive Moment Estimation(Adam) optimizer, the model easily achieved a good fit, with a performance of 0.993 @mAP50 and 0.872 @mAP50-90 on images that were considered more challenging than our task. The datasets included images with multiple plates and plates that were far from the camera.

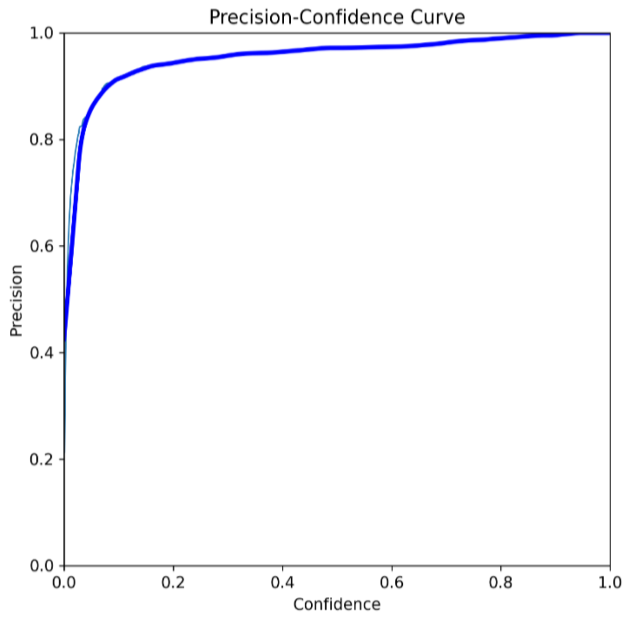
We will now analyze our metrics to ensure that the model's performance satisfies our requirements.

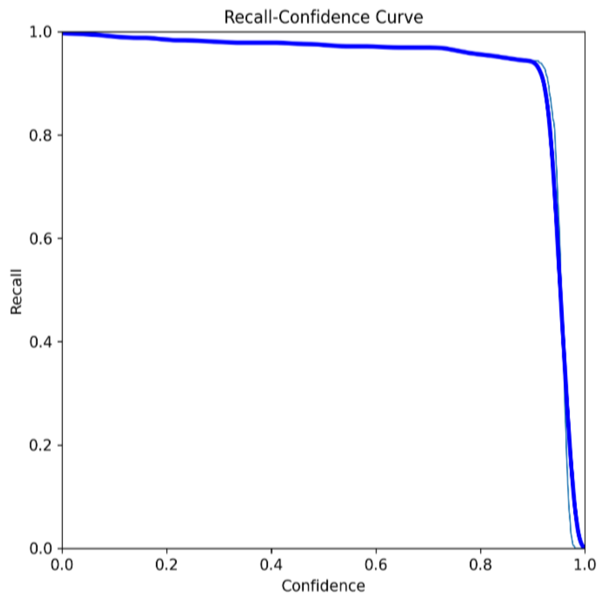


Based on the confusion matrix, the model has demonstrated a strong ability to differentiate between license plates and other objects in the image (background). The model's only error was in a false negative, achieving a rate of just 0.02.



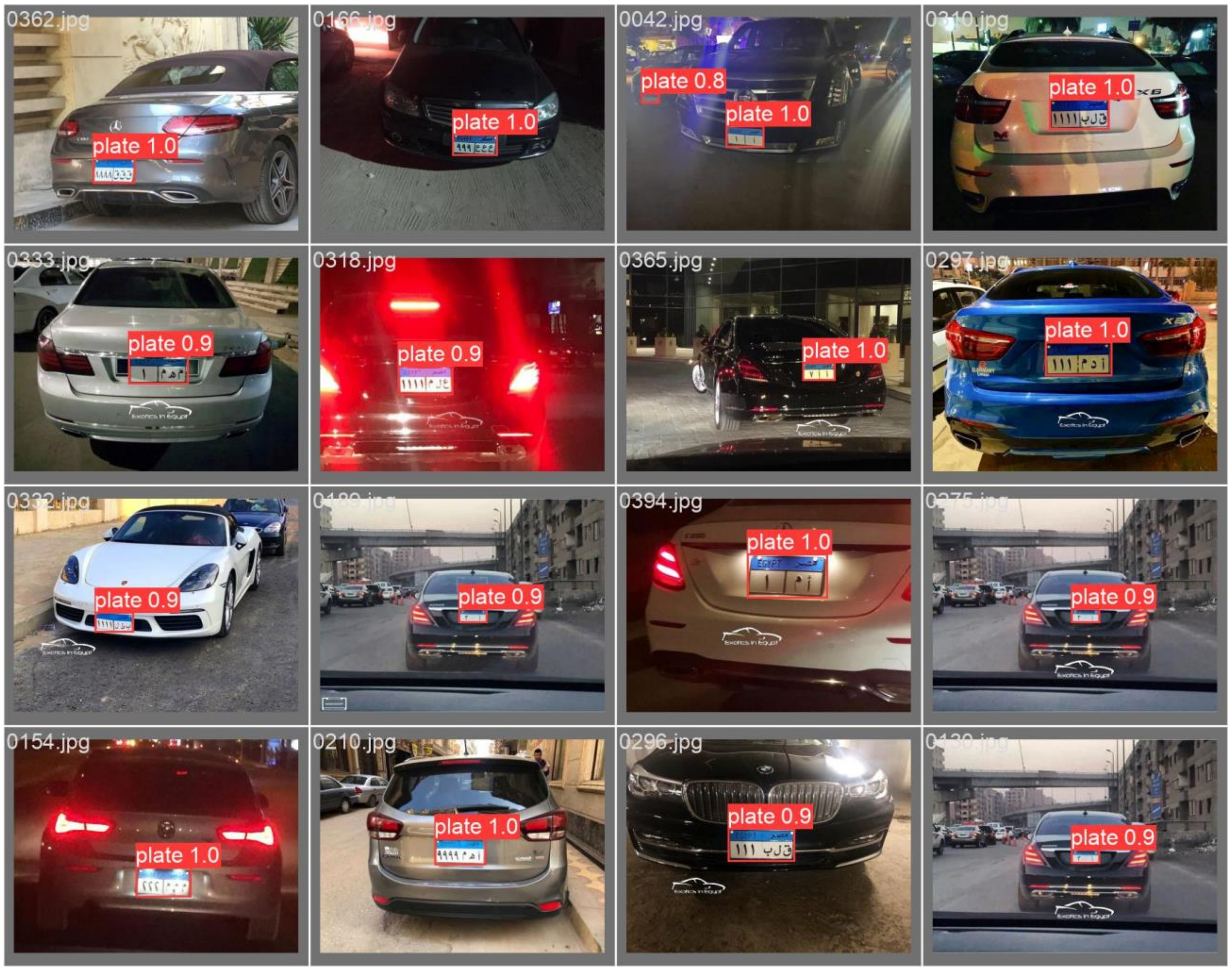
Based on the figure above, the model achieved a precision of approximately 0.982, a recall of 0.969, 0.993 @mAP50-90, and 0.872 @mAP50-95 when evaluated on 400 images with a total of 422 instances as a validation set.





Recall and precision are considered to have a reverse relationship, so it is necessary to choose a confidence value that balances them. Based on the two figures above, we have decided to choose a confidence value of 0.7.

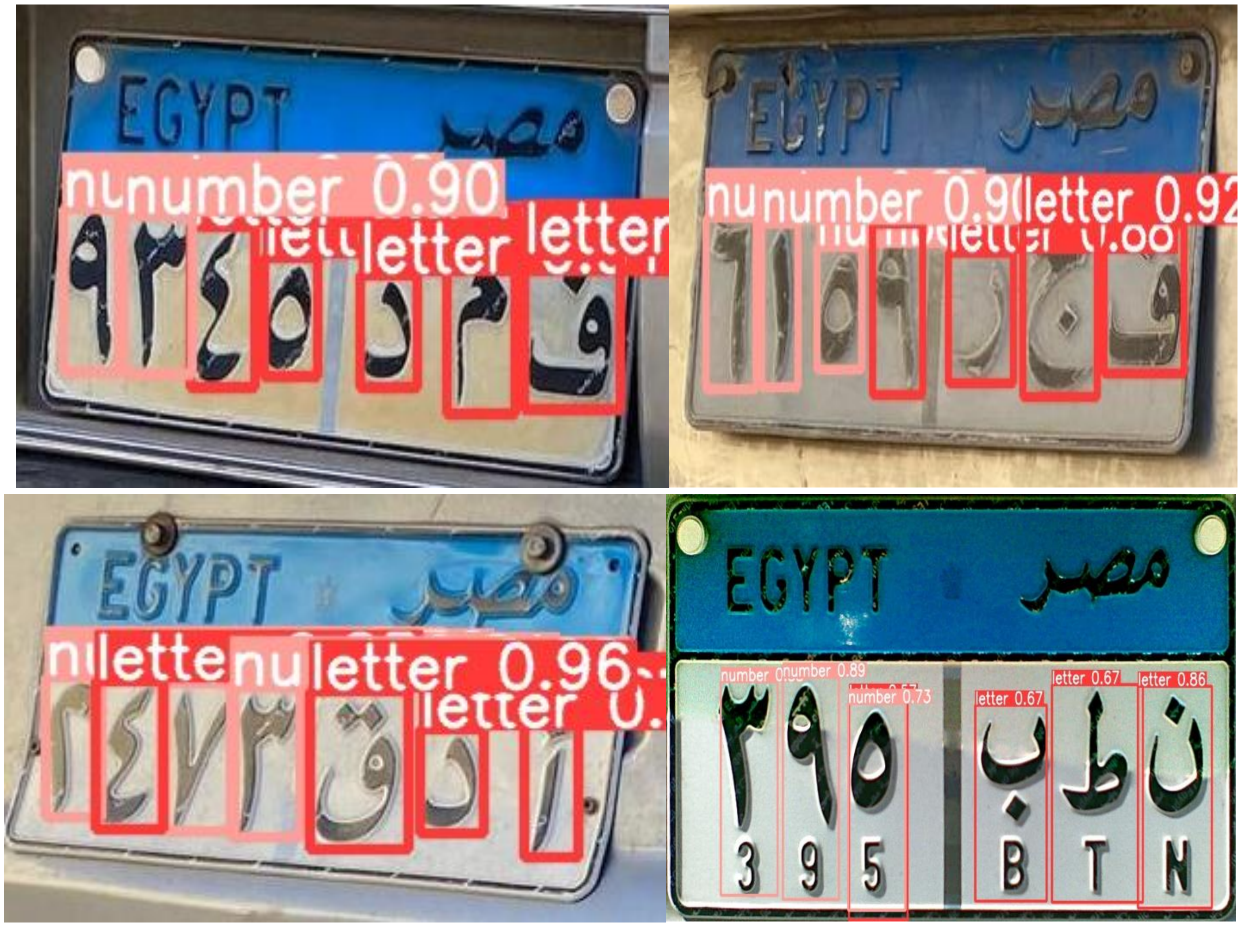
The image below shows some of the predictions made on the validation set.



As evident from the results, the model performs well in detecting license plates. Therefore, we have decided to choose YOLOv5s small for the first phase, as it meets our needs without requiring us to try other models.

4.2 Second Phase

We attempted to perform segmentation with two classes: letters and numbers. However, after training the model with this approach, we observed some failures. Due to the small size of our dataset and the similarity between some letters and numbers, there is confusion between these classes. As a result, the model sometimes segments letters as numbers and vice versa. Additionally, the model sometimes detects a character as both a number and a letter.

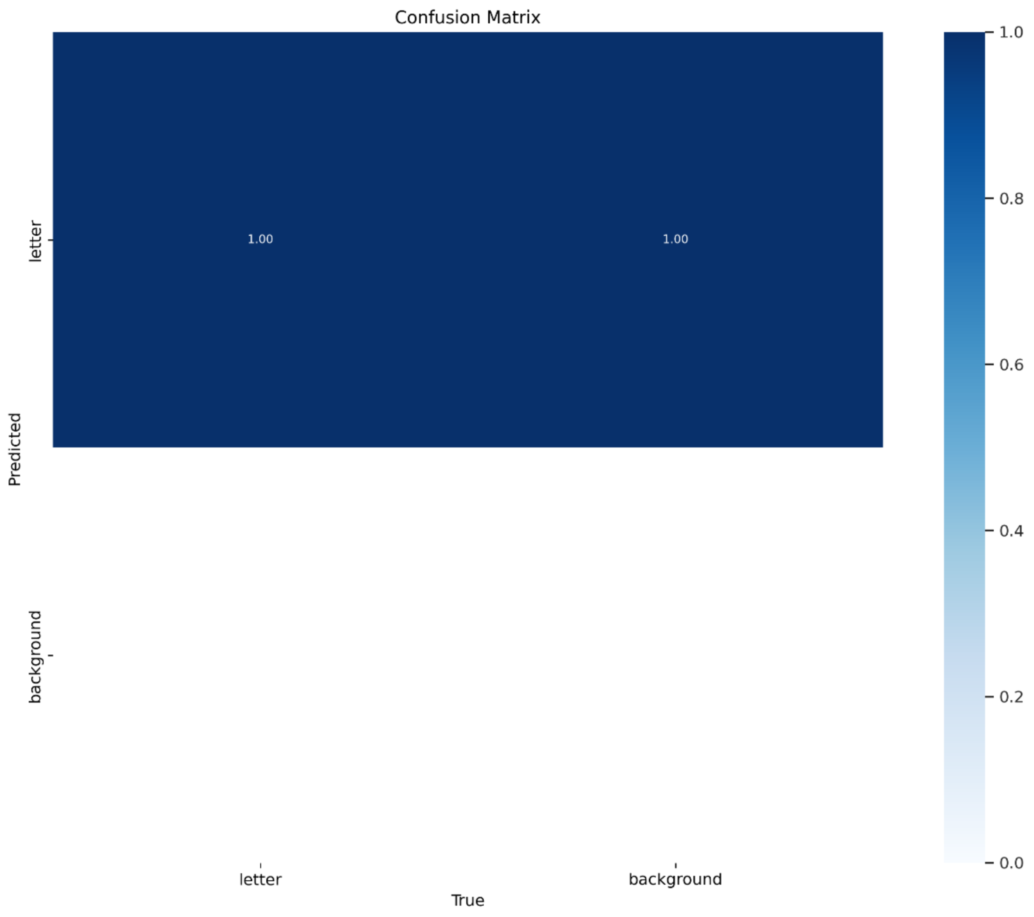


The red frame indicates the letters, while the cashmere frame indicates the numbers. In the plates shown above, the confusion between letters and numbers is apparent.

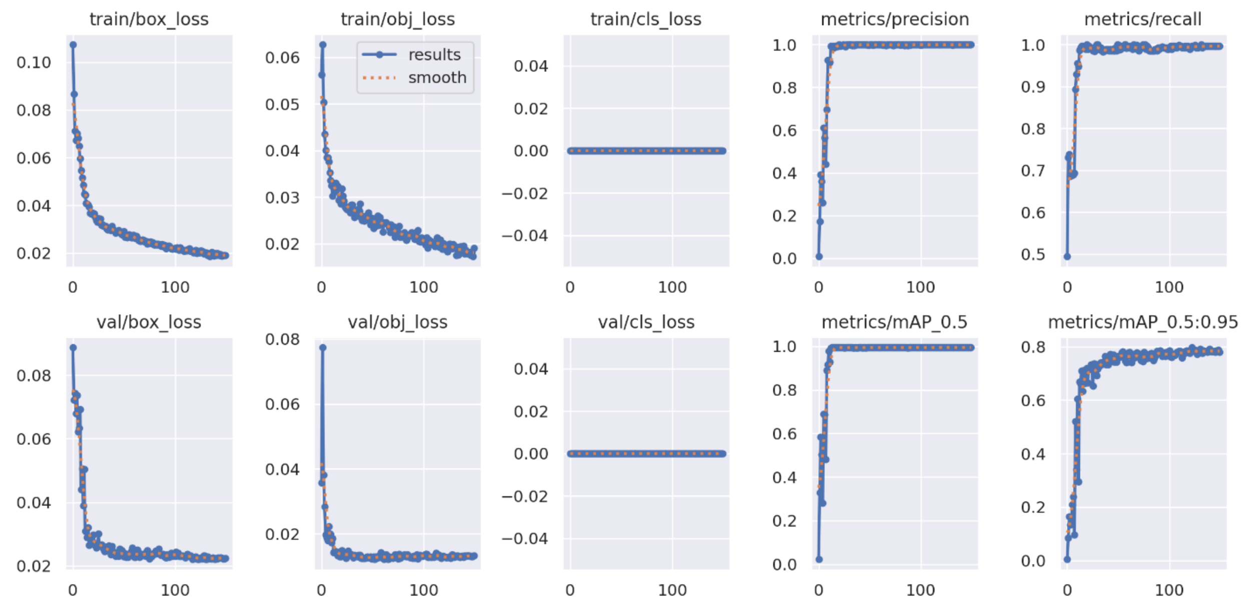
As a solution to the confusion between letters and numbers, we have decided to train the models in only one class. We will then separate between these classes in the third phase of our project.

4.2.1 Yolov5 medium

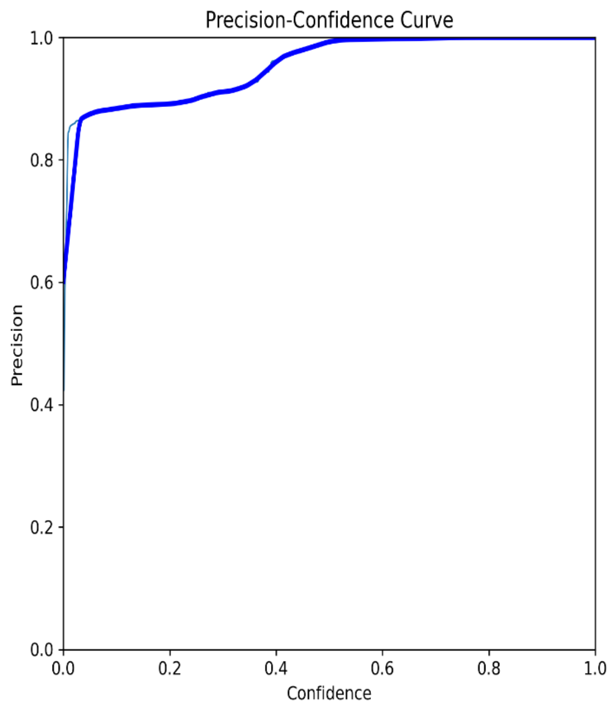
Our first model is YOLOv5 medium, which we think performs well on this problem. The model achieved 0.995 @mAP50 and 0.797 @mAP50-95. We will examine all of the metrics, as well as some of the predictions, to determine whether or not we will choose this model.

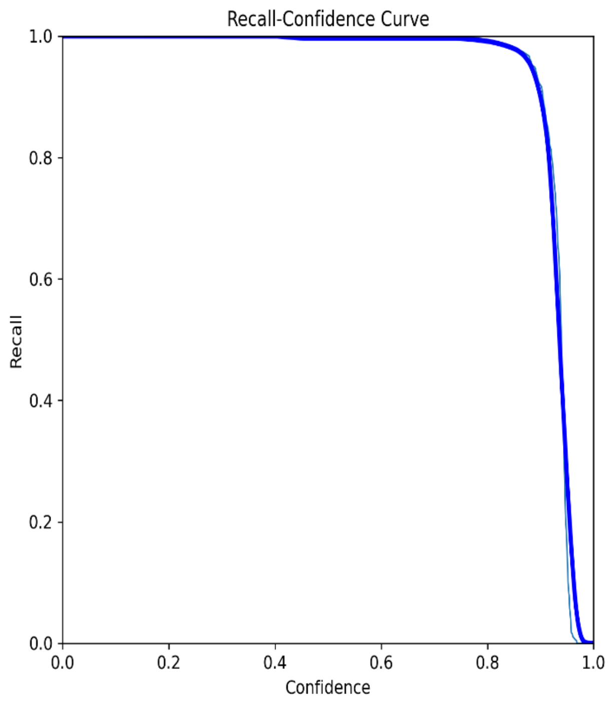


The confusion matrix shows impressive results, as there are no false positives or false negatives.



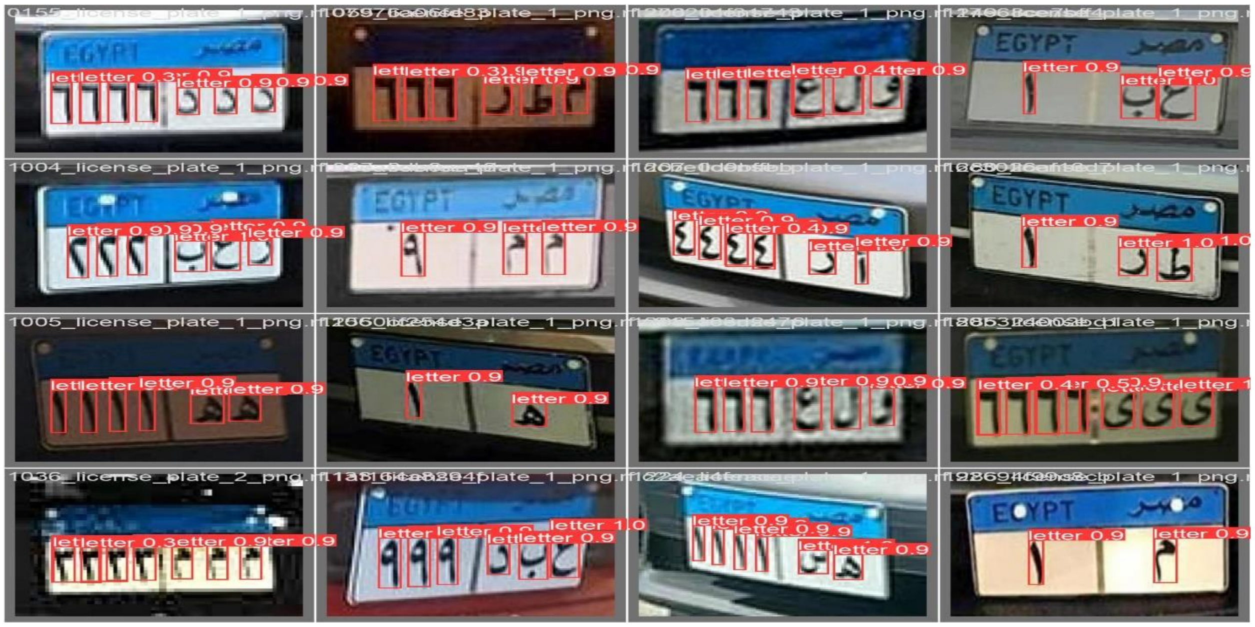
The model achieved a precision of 0.999, a recall of 0.996, 0.995 @mAP50, and 0.797 @mAP50-95 on the validation dataset, which consisted of 50 images with 271 instances.





Based on the two figures above, we have decided to choose a confidence value of 0.6.

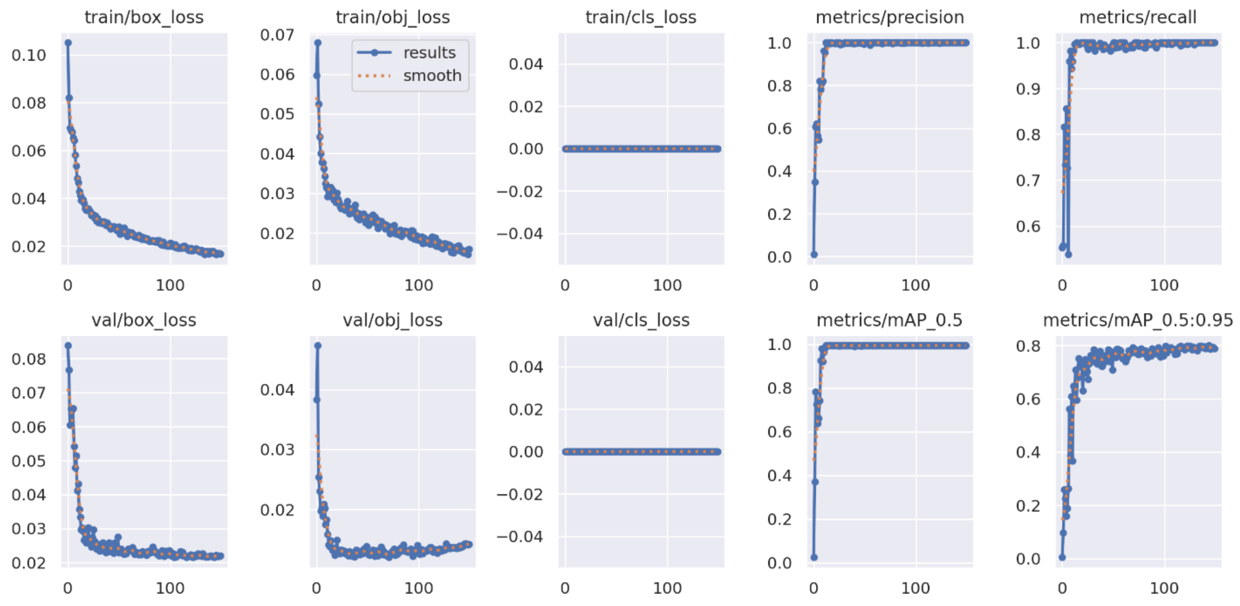
The image below shows some of the predictions made on the validation set.



We will try another model in an attempt to increase the mAP50-90 and accurately capture the characters without additional spaces.

4.2.3 Yolov5 X Large

We have attempted to use the biggest version of YOLOv5, but the model did not achieve a noticeable difference from YOLOv5 medium. which is not enough for the high FLOPS in the YOLOv5 XLarge.



The model achieved a precision of 0.999, a recall of 1.0, 0.995 @mAP50, and 0.801 @mAP50-99.

We have found that there is no significant difference between YOLOv5 small and YOLOv5 XLarge, except for @mAP50-99.

We have attempted several versions of YOLO, but we were unable to improve the mAP. However, the results we have achieved are acceptable and meet our requirements. After comparing the models, we have decided to choose YOLOv5 medium with a confidence of 0.6 because it has lower FLOPS compared to the other models.

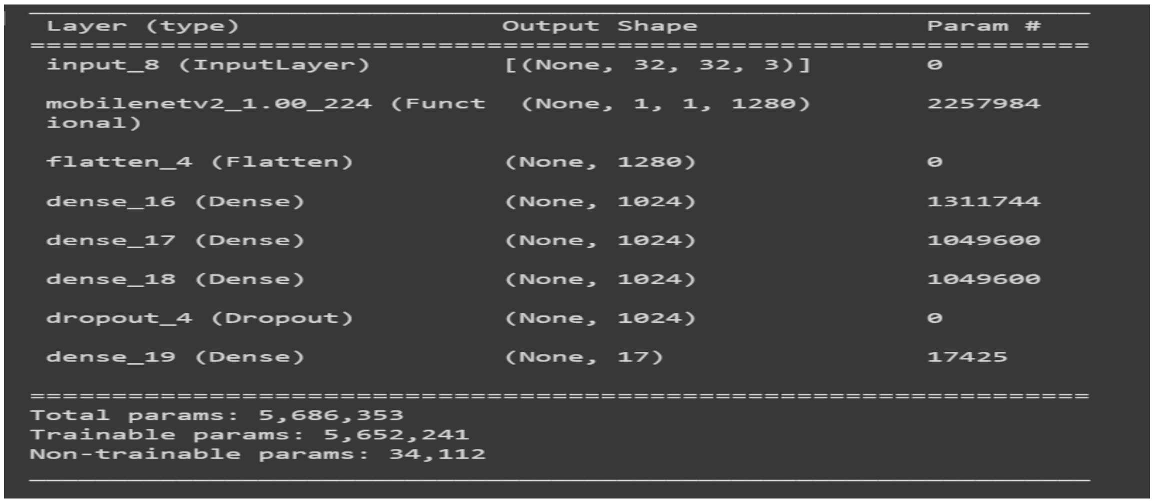
4.3 Third Phase

In this section, we attempted to use three models. First, we used MobileNetV2 to achieve the best FLOPS and utilization. We then tried ResNet, which is a larger network. If ResNet failed, we believed that DenseNet could handle this problem with high accuracy.

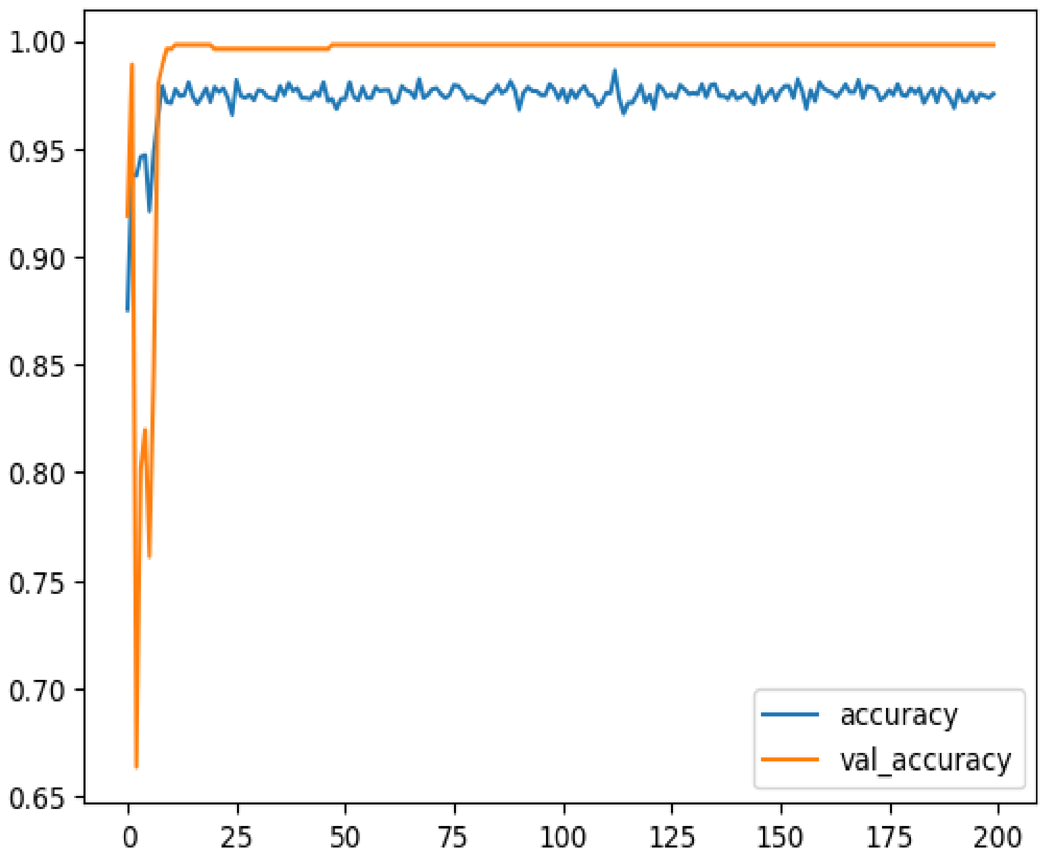
We use trained all the following models with TensorFlow and Keras

4.3.1 Mobilenet v2

After training the MobilenetV2 model for 150 epochs, the model achieved a 0.996 accuracy and 0.250 loss on the training data and a 0.989 accuracy and 0.0443 loss on the validation dataset. The numbers model, on the other hand, achieved a 0.991 accuracy and 0.04 loss on the training dataset after 200 epochs, and a 0.998 accuracy and 0.007 loss on the validation dataset. We didn't freeze any layer from the mobilenet architecture.

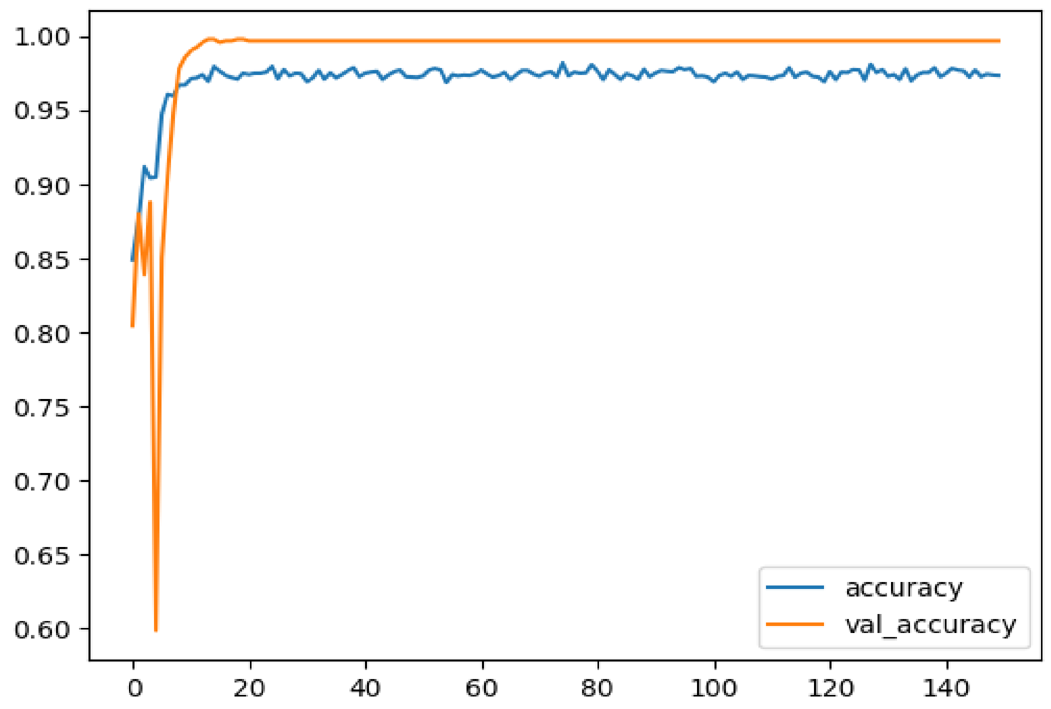


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 32, 32, 3)]       0

 mobilenetv2_1.00_224 (Funct  (None, 1, 1, 1280)       2257984
 ional)

 flatten_4 (Flatten)         (None, 1280)              0

 dense_16 (Dense)            (None, 1024)              1311744

 dense_17 (Dense)            (None, 1024)              1049600

 dense_18 (Dense)            (None, 1024)              1049600

 dropout_4 (Dropout)         (None, 1024)              0

 dense_19 (Dense)            (None, 17)                17425

=================================================================
Total params: 5,686,353
Trainable params: 5,652,241
Non-trainable params: 34,112
_________________________________________________________________
```



Numbers mobilenet v2

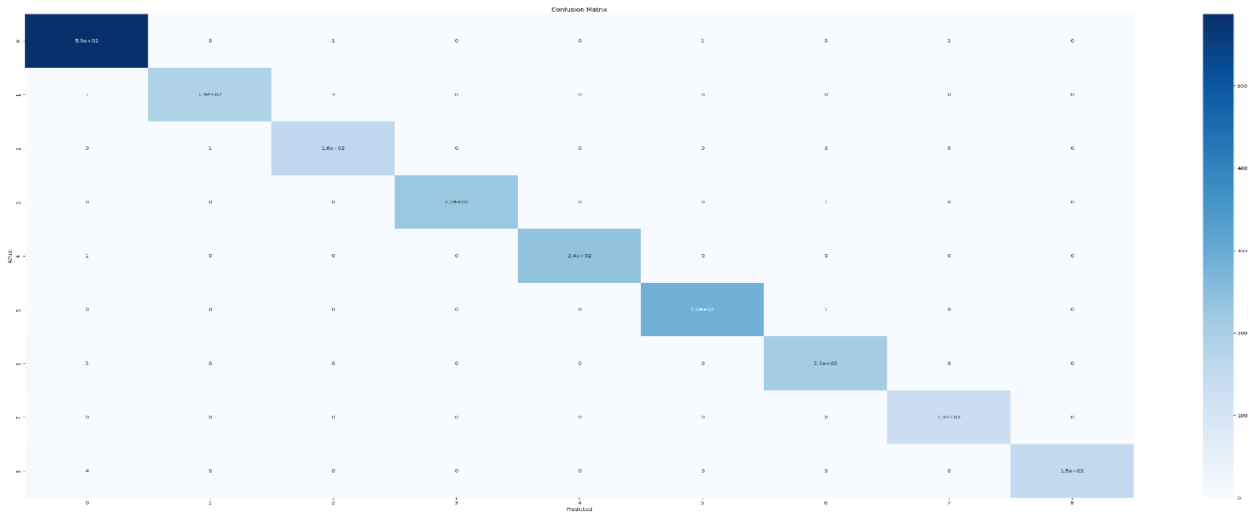
After about 25 epochs, the number model was able to achieve the best possible fit, with an accuracy of about 1.0 on the validation dataset. However, during training, the accuracy fluctuated up and down due to the augmentation, which caused the model to see different images each epoch.



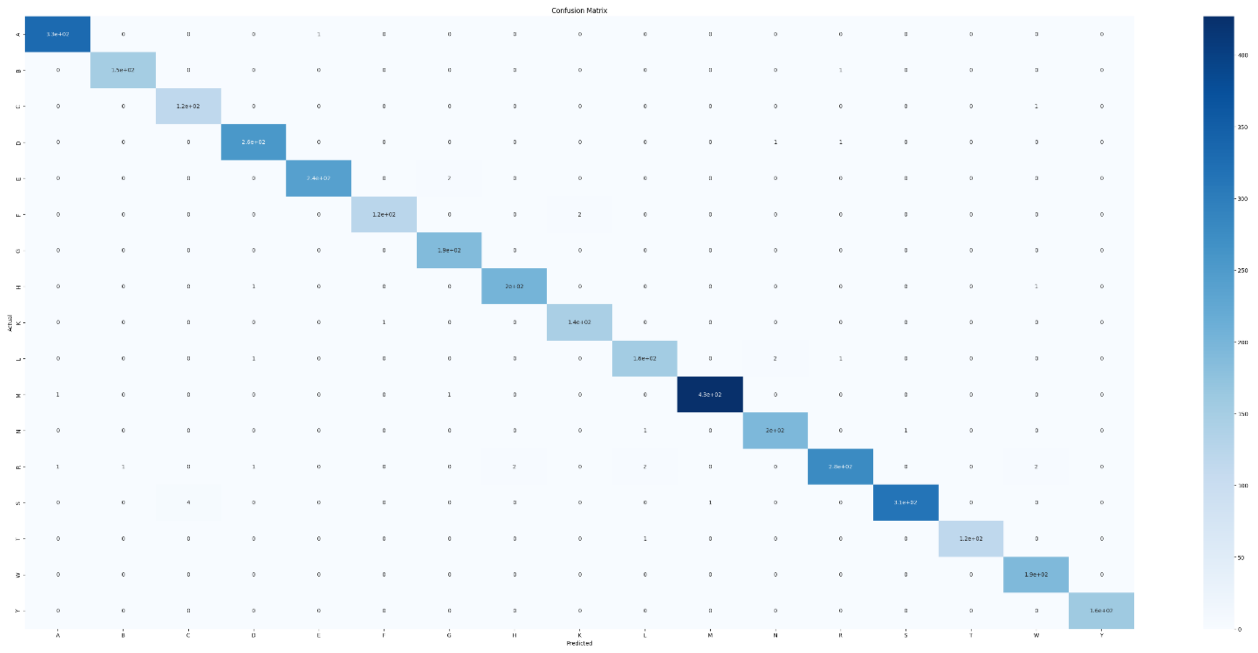
letter mobilenet v2

After about 20 epochs, the letter model was able to achieve the best possible fit, with an accuracy of about 1.0 on the validation dataset. However, during training, the accuracy fluctuated up and down due to the augmentation.

Since both models achieved a 1.0 accuracy, we can analyze the confusion between the classes by creating a confusion matrix on the training data with augmentation. This will help us understand how the models are performing in individual classes and identify any potential misclassifications or biases.

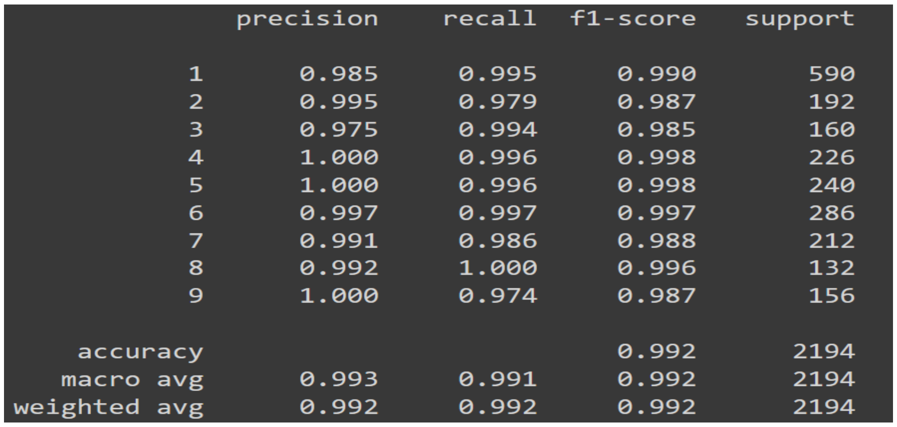


Confusion matrix of number model



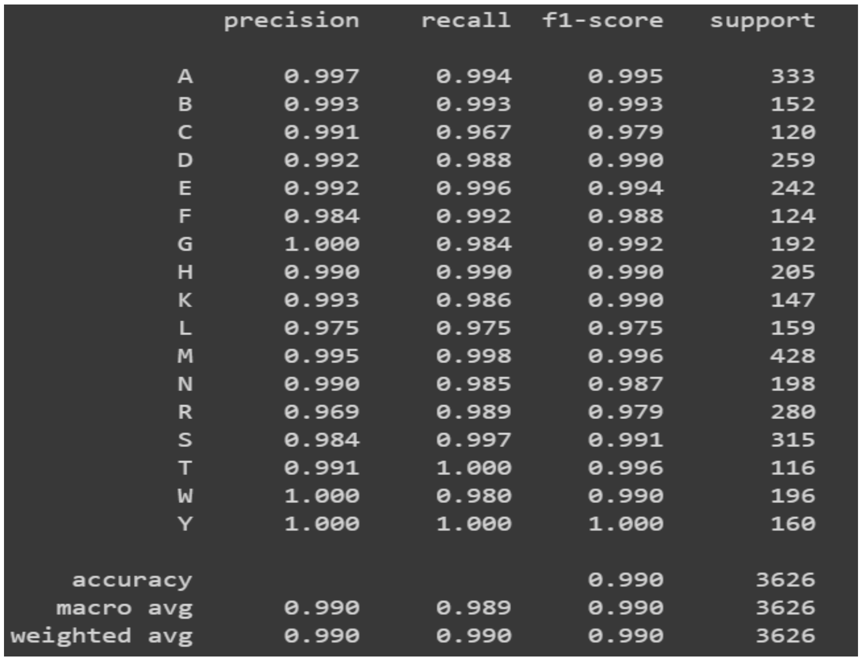
Confusion matrix of letter model

There doesn't seem to be anything abnormal in the confusion matrix, so let's calculate the precision, recall, and F1 score for each class. This will provide us with a more detailed understanding of the models' performance and help identify any areas for improvement.



```
              precision    recall  f1-score   support

           1      0.985     0.995     0.990       590
           2      0.995     0.979     0.987       192
           3      0.975     0.994     0.985       160
           4      1.000     0.996     0.998       226
           5      1.000     0.996     0.998       240
           6      0.997     0.997     0.997       286
           7      0.991     0.986     0.988       212
           8      0.992     1.000     0.996       132
           9      1.000     0.974     0.987       156

    accuracy                          0.992      2194
   macro avg      0.993     0.991     0.992      2194
weighted avg      0.992     0.992     0.992      2194
```

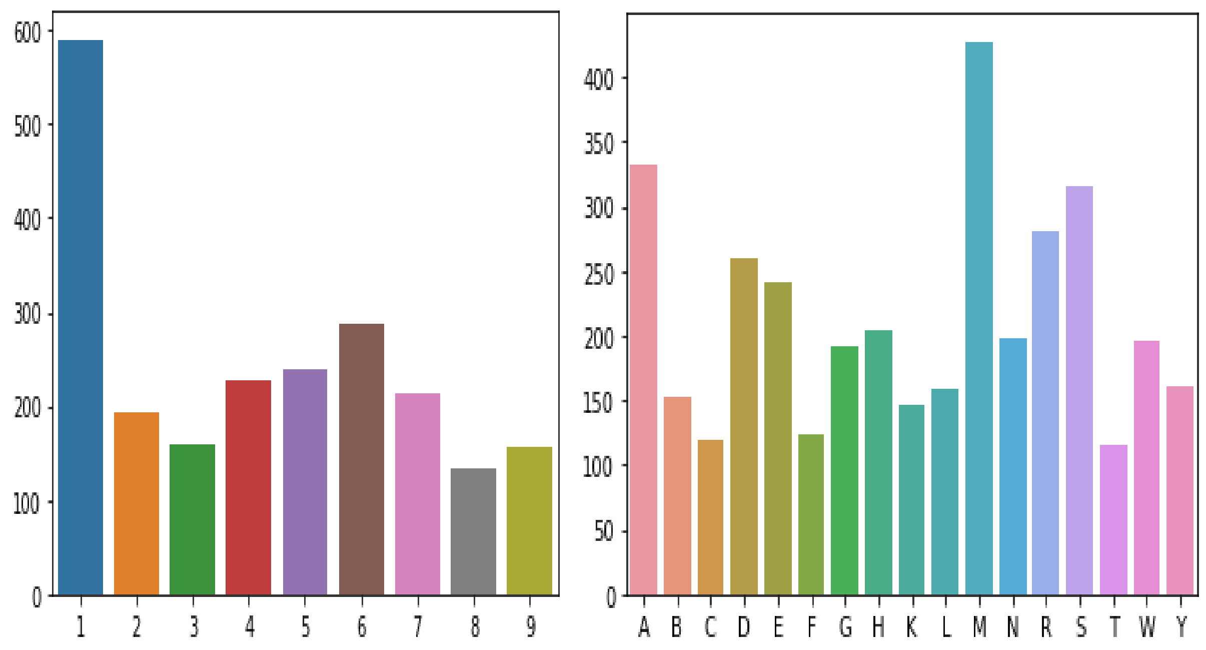
The report of the number model



```
              precision    recall  f1-score   support

           A      0.997     0.994     0.995       333
           B      0.993     0.993     0.993       152
           C      0.991     0.967     0.979       120
           D      0.992     0.988     0.990       259
           E      0.992     0.996     0.994       242
           F      0.984     0.992     0.988       124
           G      1.000     0.984     0.992       192
           H      0.990     0.990     0.990       205
           K      0.993     0.986     0.990       147
           L      0.975     0.975     0.975       159
           M      0.995     0.998     0.996       428
           N      0.990     0.985     0.987       198
           R      0.969     0.989     0.979       280
           S      0.984     0.997     0.991       315
           T      0.991     1.000     0.996       116
           W      1.000     0.980     0.990       196
           Y      1.000     1.000     1.000       160

    accuracy                          0.990      3626
   macro avg      0.990     0.989     0.990      3626
weighted avg      0.990     0.990     0.990      3626
```

The report of the letter model

It appears that the model is performing well, despite its small size. This is likely because the task at hand is relatively straightforward, and the model has learned to recognize the relevant patterns in the data. In some cases, a smaller model may be sufficient for a particular task, especially if the data is relatively simple or if computational resources are limited. However, for more complex tasks or larger datasets, a larger and more complex model may be necessary to achieve optimal performance.



The figures above show the number of instances for each class in the training dataset for both models.

4.4 Entire System

We selected 350 images from the dataset to test the entire system, from detection to classification. We then annotated all the images with the True value of the license plate recognition process and recorded the results in an Excel spreadsheet.

We chose a varied dataset for testing the model, including both intact and damaged license plates. This was done to ensure that the model can accurately recognize license plates in real-world scenarios, where plates may be damaged or obscured in various ways

After all the processing, the entire system achieved an accuracy of 0.91 in our task the images should be a good comparison to this image so we expected higher accuracy in production.

4.5 Tips For Improve Results In production

There are a lot of tips that we can improve the accuracy in the garage like the camera angle and the lighting

1- One potential modification that could be made to improve the system's performance is to reposition the camera to detect the license plate from the back of the vehicle instead of the front. This is because in low-light conditions, such as at night, many cars will have their headlights turned on, which can create glare or reflections on the license plate and make it more difficult for the model to accurately recognize the plate.

2- A second modification that could be made to improve the system's performance is to ensure that the input and output cameras are positioned at the same angle to capture the license plate from the same perspective, For example, if the input camera captures the license plate from the driver's angle, the output camera should also be positioned to capture the license plate from the driver's angle. This way, if the input camera make a mistake in classification then we need that the output camera make the same mistake so we can calculate the time without problems

3- We recommend that initially, the system should not be put into operation and instead, data should be collected from visitors for the purpose of retraining the model to achieve higher accuracy.

5 Conclusion and Future Work

5.1 Conclusion

The main aim of our system is to reduce the cost of labour and make the parking process smarter and easier for users, with simplified payment and improved ability to locate their cars. Additionally, our system can inform users of the availability of parking sections in the mall, making it easier for them to find a suitable parking spot.

5.2 Future Work

Our goal is to increase the dataset as soon as the mall starts operating. By doing so,we can improve the accuracy of our model by retraining it with the larger dataset