Machine Learning-Based Approach Implementation

Milestone 3

German International University

Faculty of Engineering

Project Members/IDs:

Yahia Adam – 7005813
Youssef Desouki – 7001816
Nour El Fewy – 7002342
Kiroll Farid – 7005201

Deadline: May 8th, 2024

Introduction:

In this milestone, the goal is to detect the location of Egyptian car plates in an image. This task involves several steps:

1 – Dataset Acquisition:

- 62 photos of Egyptian Car plates were obtained to do the milestone.
- Roboflow was the tool used for the annotation of the plates and all the photos were annotated, and this tool was used because it generates the annotated dataset in a format compatible with the online frameworks that provide the object detection models.

2 – Model Training:

- After obtaining the photos of the car plates and annotating them we need to use them to train a Neural Network which is an example of a machine learning model that will be used for object detection.
- In our case on only need to detect the cars' number plates, so one of the Neural Networks that can be used for object detection is called YOLO (You Only Look Once).

This is why YOLO Neural Network was used in this project and here is an explanation of how it works and its strengths:

YOLO (You Only Look Once) is a state-of-the-art object detection algorithm known for its speed and accuracy. Unlike traditional object detection algorithms that perform detection on multiple sub-regions of an image, YOLO divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously in a single pass through the neural network. Here's a brief description highlighting its strengths:

1. **Speed**: YOLO is exceptionally fast compared to other object detection algorithms. By processing the entire image in a single pass, YOLO avoids the need for multiple sliding window operations or region proposal networks, leading to faster inference times. This makes it suitable for real-time applications where speed is crucial, such as autonomous driving, surveillance systems, and robotics.

2. **Single-stage Detection**: YOLO is a single-stage detector, which means it directly predicts bounding boxes and class probabilities without requiring multiple stages of processing. This simplifies the architecture and reduces the computational complexity, resulting in faster inference times and lower memory requirements.

3. **High Accuracy**: Despite its speed, YOLO maintains high accuracy in object detection tasks. Its architecture is designed to effectively capture both global context and fine-grained details within the input image, enabling accurate localization and classification of objects. This makes it suitable for a wide range of applications where precise object detection is essential.

4. **Unified Framework**: YOLO provides a unified framework for object detection, localization, and classification. By jointly optimising these tasks within a single neural network, YOLO can learn rich representations of objects and their spatial relationships, leading to more accurate and coherent predictions. This simplifies the training process and improves overall performance.

5. **Real-time Applications**: YOLO's combination of speed and accuracy makes it well-suited for real-time applications where timely detection of objects is critical. It has been successfully applied in various domains, including surveillance, traffic monitoring, object tracking, and augmented reality, enabling intelligent systems to make rapid decisions based on real-time input.

Then, Ultralytics was used to train the YOLO Model on the pre-created custom dataset.

A training process with 400 iterations was made to test the YOLO Neural Network model and to see if it works effectively or not, tests were accomplished.

It is worth noting some technical difficulties during the implementation of the project. Ultralytics if installed using the pip command, doesn’t work. To solve this problem, ultralytics was downloaded as a zip file and then uploaded to colab drive.

Code Explanation:

This code essentially downloads a dataset from Roboflow, installs required packages, trains a YOLOv8 model, and performs inference on two images using the trained model.

Installing Required Packages

```
!pip install -r requirements.txt
```

This command installs the Python dependencies listed in the requirements.txt file of the cloned repository. These dependencies are necessary for running the subsequent scripts. Requirements.txt contains all the versions required for the libraries that will make YOLO library work.

Installing Roboflow Library:

```
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="j1hGwcjQkh3taS7YL3C8")
project = rf.workspace("signalproject").project("milestone3-4glyw")
version = project.version(6)
dataset = version.download("yolov8")
```

This code installs the Roboflow Python library, which provides functionalities for interacting with Roboflow, a platform for managing and preprocessing image datasets, then the code also initializes an instance of the Roboflow class with the specified API key, enabling access to Roboflow functionalities, in addition, specify the Roboflow workspace, project, and version to be used. It selects the project "milestone3-4glyw" from the workspace "signalproject" and version 6 of that project. Then lastly, downloads the dataset associated with the specified version of the project. It downloads the dataset in YOLOv8 format. This code was extracted from Roboflow itself to have the unique API key. Once run, the dataset was visualised in the colab environment.

Importing Ultralytics from Zipfile:

```
import zipfile
with zipfile.ZipFile('/content/ultralytics.zip', 'r') as zip_ref:
    # Extract all the files in the zip file
    zip_ref.extractall('/content/Ultralytics_lib')
```

This code extracts and imports the ultralytics library in order to do the YOLO neural network processing on. There were technical difficulties when dealing with ultralytics, therefore, it was downloaded as a zip file, and then uploaded to colab environment. After that using the above code,

the ultralytics file was unzipped.

Training The YOLOv8 Model:

```
!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/train.py model=yolov8n.pt data=/content/Licence-
Plate-Detection-using-YOLO-V8/Milestone3-6/data.yaml epochs=400
```

This command trains the YOLOv8 model using the specified configuration line (data.yaml) and parameters. It trains the model for 400 images.

After roboflow code run, we opened the data.yaml file and then edited the destination file paths so that YOLO can read the file paths of the images. Using YOLO train algorithm in the ultralytics library, the model was trained. The training iterations were 400. As you can see 400 epochs mean 400 training iterations.

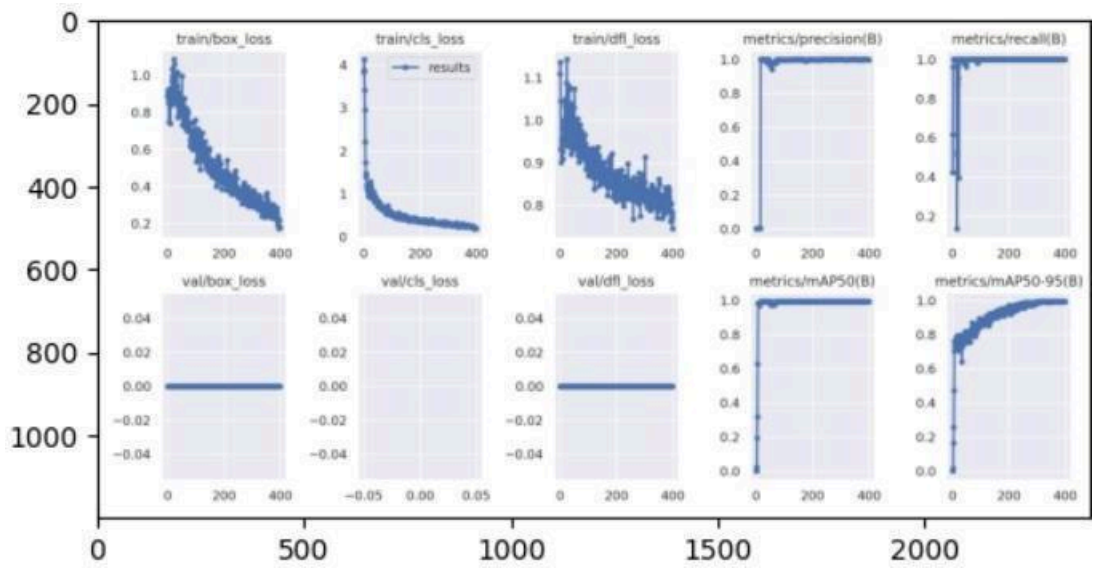
Importing The matplotlib Library:

```
# Import the matplotlib library
import matplotlib.pyplot as plt

# Load the image
image = Image.open("/content/runs/detect/train/results.png")

# Display the image
plt.imshow(image)
plt.show()
```

This code imports the matplotlib for usage.



And this is a visualization for the training process, and the code plots the different graphs to show how the training process went.

Here is also a visualization for the trained images.

Code:

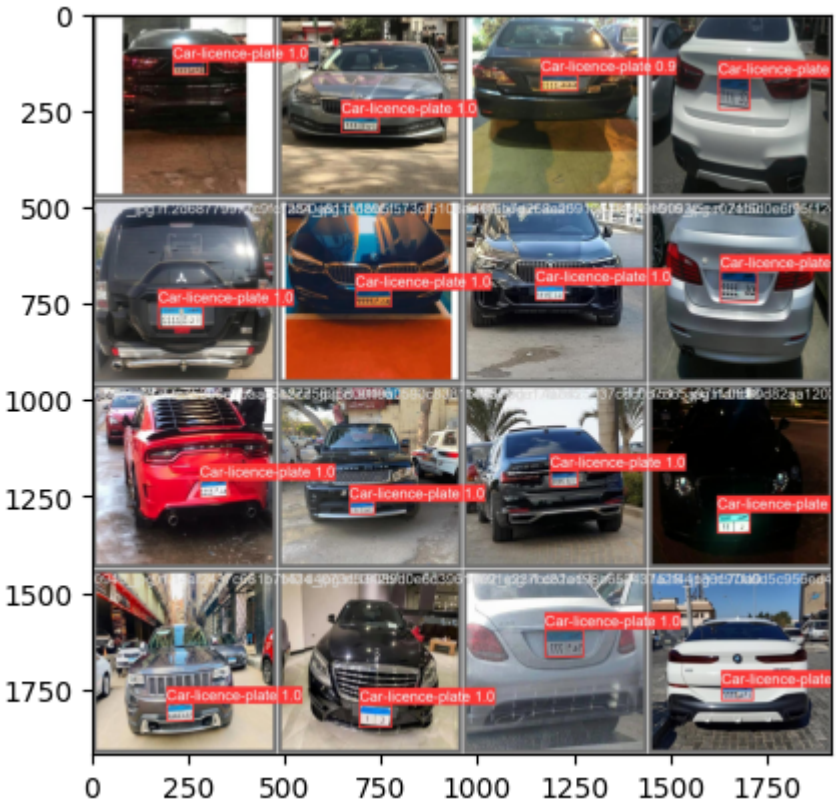
```
# Load the image

image = Image.open("/content/runs/detect/train/val_batch1_pred.jpg")


# Display the image

plt.imshow(image)

plt.show()
```


0
250
500
750
1000
1250
1500
1750
0
250
500
750
1000
1250
1500
1750
Car-licence-plate 1.0
Car-licence-plate 0.9
Car-licence-plate

Performing Testing on Images:

```
result=!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection-
using-YOLO-V8/runs/detect/train3/weights/best.pt' source='/content/0067.jpg'

result=!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection-
using-YOLO-V8/runs/detect/train3/weights/best.pt' source='/content/0210.jpg'

result=!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection-
using-YOLO-V8/runs/detect/train3/weights/best.pt' source='/content/0228.jpg'

result=!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection-
using-YOLO-V8/runs/detect/train3/weights/best.pt' source='/content/0261.jpg'

result=!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection-
using-YOLO-V8/runs/detect/train3/weights/best.pt' source='/content/0582.jpg'
```

These commands perform inference on different images (0067.jpg, 00210.jpg, 0.228.jpg, 0.261.jpg and 0.582.jpg) using the trained YOLOv8 model. The model weights are loaded from the specified path, and the predictions are generated for each image. In short, in this code, we passed the YOLO built-in predict function which uses the weights to test the images. Then, we passed to the neural network, the best weights obtained from the results of the training process. After that, the image file paths were passed as well. The external images were uploaded manually.

7. Outputting The test result Images:

```
# Import the PIL library
from PIL import Image

# Load the image
image = Image.open("/content/Licence-Plate-Detection-using-YOLO-
V8/runs/detect/train10/0067.jpg")

# Display the image
plt.imshow(image)

# Load the image
image = Image.open("/content/Licence-Plate-Detection-using-YOLO-
V8/runs/detect/train9/0210.jpg")

# Display the image
plt.imshow(image)
```

```
# Load the image
image = Image.open("/content/Licence-Plate-Detection-using-YOLO-
V8/runs/detect/train8/0228.jpg")

# Display the image
plt.imshow(image)

# Load the image
image = Image.open("/content/Licence-Plate-Detection-using-YOLO-
V8/runs/detect/train6/0261.jpg")

# Display the image
plt.imshow(image)

# Load the image
image = Image.open("/content/Licence-Plate-Detection-using-YOLO-
V8/runs/detect/train4/0582.jpg")

# Display the image
plt.imshow(image)
```

The first line imports the Image module from the Python Imaging Library (PIL). PIL is a library in Python used for opening, manipulating, and saving many different image file formats.

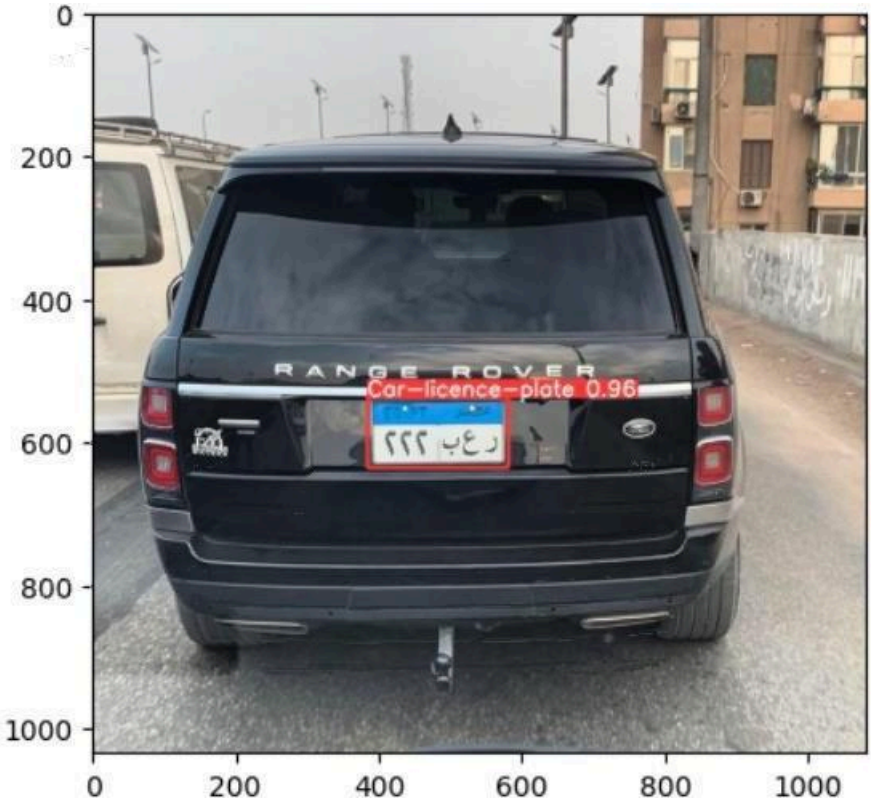
Then the rest of the code outputs the 5 sample images with boxes around the number plate and their confidence percentage. The output images were redirected to a specific file that was visible to us, we copied its path to pass to the PIL library to visualise the image.

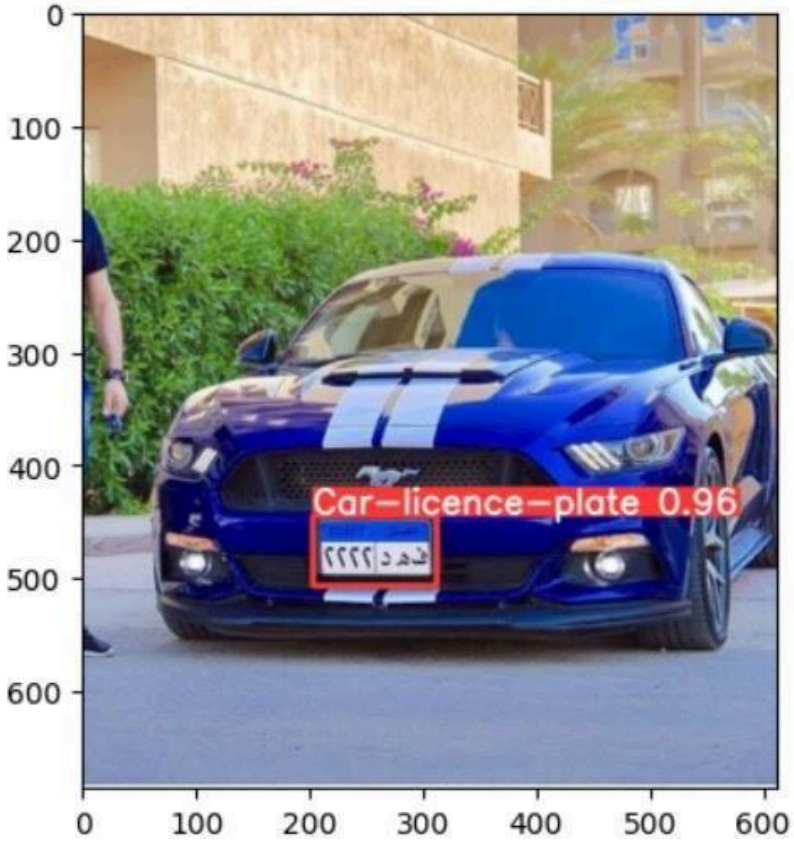
Results:

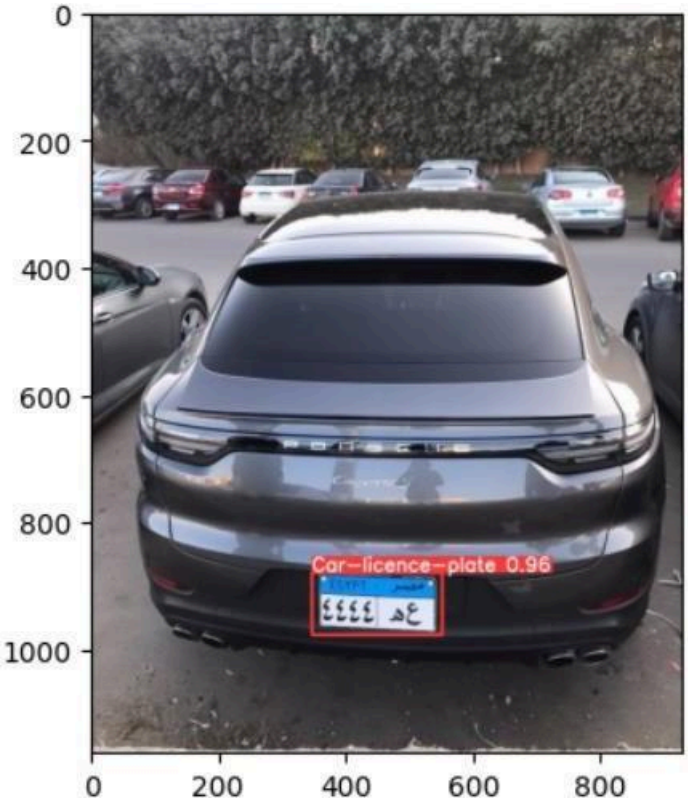
The photos shown downwards are the output of the code, 5 external sample images with the boxes surrounding the number plates and showing the confidence number with accuracy ranging from (94-95%). This was to test the validity of our code. The output is between 0 and 1 showing the percent of confidence, the higher the value the more the model is confident regarding its test result.

This confirms that the YOLO Neural Network model that was used works greatly and way better than the classical approach that was used in the last milestone.

Here are the 5 photos:


RANGE ROVER
Car-licence-plate 0.96


Car-licence-plate 0.96


Car-licence-plate 0.96
0
200
400
600
800
1000
0
200
400
600
800