# Machine Learning-Based Approach Implementation

## Final Report


**German International University**
**Department of Mechanical Engineering**
**Faculty of Engineering**

**Project Members/IDs:**

Yahia Adam (7005813)
Youssef Desouki (7001816)
Nour El Fewy (7002342)
Kiroll Farid (7005201)
Amr Khaled Mesbah (7005368)


**Deadline: May 24th, 2024**

## Introduction:

This final milestone, requires us to merge both of the Machine Learning-Based Approaches (Milestone 3 & Milestone 4) into one code and make it detect the location of the number plate and highlight it when an image is inserted in the model like it was made in Milestone 3 then after that the model localizes the letters and highlight them in the detected image and then output the recognized characters as text.

A scenario that was required from us is used to apply the new merged model to test our new code, Here is the scenario to be addressed: an input image that will be used in the new model should contain multiple cars.
First, we used the model from the third milestone to detect each car's license plate.
Then, feed the cropped images containing the plates to the model from the fourth milestone to recognize the text inside each plate.

## Code Explanation:

### Install Dependencies

```
!pip install -r requirements.txt
!pip install roboflow
!pip install ultralytics
!pip install pandas ultralytics
```

markdown

These lines install the necessary Python packages. The `-r requirements.txt` installs all dependencies listed in a `requirements.txt` file. `roboflow`, `ultralytics`, and `pandas` are explicitly installed because they are necessary for the subsequent code to function.

### Import Libraries and Setup Roboflow

```
!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")
```

Python

These lines import the Roboflow library, initialize it with an API key, access a specific project and version within the Roboflow workspace, use Project Version 6, and download the dataset in the YOLOv8 format.

### Load and Train a YOLO Model

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)

# Train the model
results = model.train(data='/content/Milestone3-6/data.yaml', epochs=50, imgsz=512)
```

Here, the YOLO model from the Ultralytics library is loaded. The model is trained using a dataset specified in `data.yaml`, for 50 epochs with an image size of 512.

## Predict an Image

```
import sys
from ultralytics import YOLO

# Load the model
model = YOLO('/content/runs/detect/train/weights/best.pt')

#Predict on an image
result = model.predict('/content/1.jpg', save=True, conf=0.1, hide_conf=True, agnostic_nms=True)
```

This segment loads a YOLO model and performs prediction on an image (`/content/1.jpg`). The predictions are saved, and non-maximum suppression is applied in an agnostic way.

## Additional Setup and Model Training

```
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="J931xHdL95fBvr5e446M")
project = rf.workspace("smart-ad-ezzae").project("plate-detect-oydq2")
version = project.version(1)
dataset = version.download("yolov8")
```

Another Dataset is imported for additional setup and model training for working the model of milestone 4.

```
import sys
from ultralytics import YOLO

# Load a pretrained model
model = YOLO('yolov8n.pt')

# Train the model
results = model.train(data='/content/plate-detect-1/data.yaml', epochs=50, imgsz=512)
model = YOLO('/content/runs/detect/train/weights/best.pt')
```

This code imports the sys module, which provides access to some variables used or maintained by the interpreter and functions that interact strongly with the interpreter, it also re-import and re-train a YOLO model using a different dataset. After training, it loads the best model weights from the training run.

### Prediction of Another Image

```
result = model.predict('/content/2.jpeg', save=True, conf=0.1, agnostic_nms=True)
```

This code makes predictions on a new image (`/content/2.jpeg`) using the previously trained model.

### Extract Bounding Boxes from Results

```
# Extract bounding boxes from the results
bounding_box_coordinates = []
for resul in result:
    for box in resul.boxes:
        x_min, y_min, x_max, y_max = box.xyxy[0].tolist()  # Get bounding box coordinates
        bounding_box_coordinates.append([x_min, y_min, x_max, y_max])

# Print the bounding box coordinates
for coord in bounding_box_coordinates:
    print(f"Bounding box coordinates: {coord}")
```

This part extracts bounding box coordinates from the prediction results and prints them.

### Extract Images Based on Bounding Boxes

```
import cv2
import os

def extract_images(image_path, bounding_boxes_array):
    # Load the original image
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Image at {image_path} could not be loaded.")
        return

    # Create a directory to save cropped images
    output_dir = os.path.splitext(image_path)[0] + "_cropped"
    os.makedirs(output_dir, exist_ok=True)

    # Iterate over each bounding box
    for i, box in enumerate(bounding_boxes_array):
        x_min = int(box[0])
        y_min = int(box[1])
        x_max = int(box[2])
        y_max = int(box[3])
```

```
        # Crop the region of interest from the original image
        cropped_image = image[y_min:y_max, x_min:x_max]

        # Save the cropped image
        output_path = os.path.join(output_dir, f"box_{i + 1}.jpg")
        cv2.imwrite(output_path, cropped_image)
        print(f"Box {i + 1} saved to {output_path}")

# Example usage
image_path = '/content/runs/detect/predict4/2.png'
extract_images(image_path, bounding_box_coordinates)
```

This function extracts images from specified bounding boxes and saves them. It reads the original image, creates a directory for cropped images, iterates over each bounding box, crops the region of interest, and saves the cropped images.

## Predict on Cropped Images

```
model = YOLO('/content/runs/detect/train/weights/best.pt')

result = model.predict('/content/runs/detect/predict4/2_cropped/box_1.jpg', save=True, conf=0.1, agnostic_nms=True, hide_conf= False, save_txt= True)
```

```
model = YOLO('/content/runs/detect/train/weights/best.pt')

result = model.predict('/content/runs/detect/predict4/2_cropped/box_2.jpg', save=True, conf=0.1, agnostic_nms=True, hide_conf= False)
```

These lines load the trained YOLO model and make predictions on the cropped images saved previously.

## Combine Images

```python
import cv2

# Read the two images
image1 = cv2.imread('/content/2.png')

image2_list = [cv2.imread('/content/runs/detect/predict8/box_1.jpg'), cv2.imread('/content/runs/detect/predict9/box_2.jpg')]

# Resize each image in image2_list to the same size as the region in image1 where it will be placed
for i, image2 in enumerate(image2_list):
    region_height, region_width, _ = image1[int(bounding_box_coordinates[i][1]):int(bounding_box_coordinates[i][3]),
                                            int(bounding_box_coordinates[i][0]):int(bounding_box_coordinates[i][2])].shape
    image2_list[i] = cv2.resize(image2, (region_width, region_height))

# Iterate over each bounding box
for i, bbox in enumerate(bounding_box_coordinates):
    # Extract the coordinates of the bounding box
    x_min, y_min, x_max, y_max = bbox

    # Place image2 onto image1 at the specified coordinates
    image1[int(y_min):int(y_max), int(x_min):int(x_max)] = image2_list[i]

image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
```

```python
import matplotlib.pyplot as plt
plt.imshow(image1)
plt.show()
```

Finally, this code combines images by placing cropped images back into the original image based on bounding box coordinates. It reads the images, resizes the cropped images to fit the bounding boxes, places the resized images into the original image, and displays the result.

## Summary

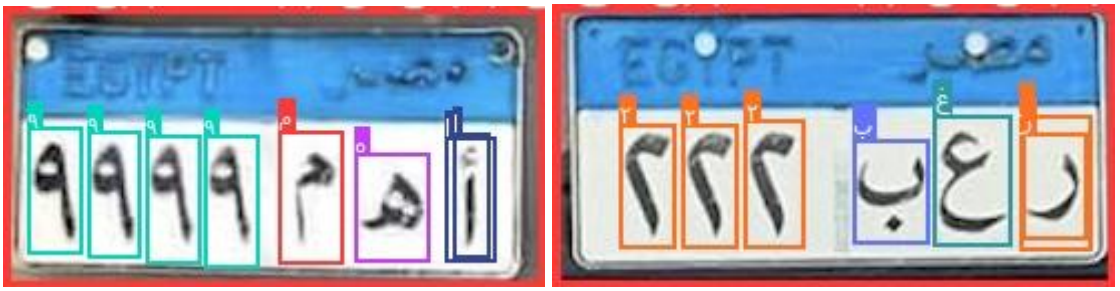
To summarize the whole code, the code installs necessary packages, sets up and trains a YOLOv8 model using datasets from Roboflow like what was done in the previous milestones, makes predictions on images, extracts bounding boxes, crops images based on these boxes makes predictions on the cropped images, and finally combines the results by placing cropped images back into their original locations in a larger image. This process allows for detailed object detection and manipulation using the YOLO model, a Machine Learning-Based Model that shows way more accuracy and better results than the Classical-Based Approach.

**The following images are the results of the new model and show that the code succeeded in doing the requirements:**

The first image is the beginning of the new code and shows that it was able to detect both number plates on both cars and highlight them in the red box, it seems like it is two different images but it is one image, we just got two photos and merged them and inserted them into the model as one, so it verified that milestone 3 works as intended.


The second image recognizes the letters in the number plate and highlights them into boxes, then writes the Arabic letter of each character on each letter.

The following two images show what happens when the two milestones are merged, it first detects the number plate and highlights the plate with a red box like milestone 3, then after that, each letter is highlighted and then the Arabic text is written above each one as the milestone 4 works.



The last image shows that the code works when an image with multiple vehicles is involved, which covers what was needed in the scenario required and its successful run.