Machine Learning-Based Approach Implementation

Milestone 4

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 19th, 2024

Introduction:

Last Milestone, we applied a Machine Learning-Based Approach implementation over the traditional Classical-Based Approach.
A Machine Learning-Based Approach for detection was applied to detect the Arabic number plate position in an image, After successfully detecting the car's plates in the previous milestone, we need to use the Machine Learning - Based Approach to recognize the Arabic characters found on the Arabic number plate that we already detected in the image, and then output the recognized characters as text.

1 - Dataset Acquisition:

- For the Dataset Preparation, we found a dataset that has more than 1400 images that will be beneficial to test our Machine Learning-Based Approach and will be of great help in training the model.
- Roboflow will be the tool used for the annotation of the plates and all the photos will be annotated, this tool was used because it generates the annotated dataset in a format compatible with the online frameworks that provide the object detection models, the same tool that was used in the last milestone 3.

2 - Model Training:

- After getting the images from the dataset that was given to us and annotating them we will be able to use them to train the Neural Network that will be used to detect the number plate and then obtain the characters from the plate and output them as text.
- For the Neural Network that will be used, we will use the YOLO (You Only Look Once), which was also used in the last milestone and we mentioned why it is a great option for this project, Ultralytics was used to train the YOLO model on the pre-created custom dataset.

After training the model, we will try to obtain the characters from the number plate and output them as text above the characters on the number plate to see if our model does the job that is required or not.

Code Explanation:

- Package Installation:

```
!pip install ultralytics
import ultralytics
ultralytics.checks()
```

- `!pip install ultralytics`: This command installs the ultralytics package, which includes tools for training and deploying YOLO (You Only Look Once) models, specifically the latest versions like YOLOv8.
- `import ultralytics`: This imports the ultralytics package into the Python environment.
- `ultralytics.checks()`: This function performs checks to ensure that the environment is correctly set up to use Ultralytics tools. It typically checks for necessary dependencies, verifies installations, and provides information about the system's configuration.

- Roboflow Setup:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="ucVCFgSzMC30dhBFdcBb")
project = rf.workspace("smart-ad-ezzae").project("plate-detect-oydq2")
version = project.version(1)
dataset = version.download("yolov8")
```

This part installs the Roboflow package and sets up access to a dataset for training the YOLOv8 model and then starts to get output in the following code after localizing the letters.

- `!pip install roboflow`: Installs the roboflow package, which is a tool for managing datasets, particularly for computer vision tasks. Roboflow helps with dataset preprocessing, versioning, and integration with machine learning frameworks.
- `from roboflow import Roboflow`: Imports the Roboflow class from the roboflow package.

- `rf = Roboflow(api_key="ucVCFgSzMC30dhBFdcBb")`: Initializes a Roboflow object using an API key. The API key grants access to the user's Roboflow account and associated projects.

- `project = rf.workspace("smart-ad-ezzae").project("plate-detect-oydq2")` : Accesses a specific project within a workspace. In this case, the project is "plate-detect-oydq2" within the "smart-ad-ezzae" workspace.

- `version = project.version(1)`: Selects a specific version of the project. Versions allow tracking of changes and improvements to the dataset.

- `dataset = version.download("yolov8")`: Downloads the dataset in a format compatible with YOLOv8. This dataset will be used for training the YOLO model.

- Model Training:

```
from ultralytics import YOLO

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

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

This section initializes the YOLOv8 model and trains it using the provided dataset for 50 epochs.

- `from ultralytics import YOLO`: Imports the YOLO class from the ultralytics package. This class provides functionalities for loading, training, and using YOLO models.

- `model = YOLO('yolov8n.pt')`: Loads a pretrained YOLOv8 model. The 'yolov8n.pt' file is a pretrained weight file for YOLOv8 (likely the "nano" version, which is the smallest and fastest variant suitable for training and inference).

- `results = model.train(data='/content/plate-detect-1/data.yaml', epochs=50, imgsz=512)`: Trains the YOLOv8 model using the provided dataset. Key parameters include:

- `data='/content/plate-detect-1/data.yaml'`: Path to the dataset configuration file, which contains information about the dataset, including paths to training and validation data, class names, etc.

- `epochs=50`: Number of training epochs. An epoch is one complete pass through the training dataset.

- `imgsz=512`: Image size used for training. The images are resized to 512x512 pixels.

- Object Detection:

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

result = model.predict('/content/1.jpeg', save=True, conf=0.1,
show_conf=False, agnostic_nms=True)
result = model.predict('/content/2.jpeg', save=True, conf=0.1,
show_conf=False, agnostic_nms=True)
result = model.predict('/content/3.jpeg', save=True, conf=0.1,
show_conf=False, agnostic_nms=True)
result = model.predict('/content/4.jpeg', save=True, conf=0.1,
show_conf=False, agnostic_nms=True)
result = model.predict('/content/5.jpeg', save=True, conf=0.1,
show_conf=False, agnostic_nms=True)
```

In this section, the trained YOLOv8 model is loaded with the best weights obtained during training. Then, the model's `predict` method is called to make predictions on multiple images. Each call to `predict` takes the path to an image (`'/content/1.jpeg'`, `'/content/2.jpeg'`, etc.) and several optional parameters:

- `save=True`: Saves the output image with bounding boxes drawn around detected objects.

- `conf=0.1`: Confidence threshold for object detection. Objects with a confidence score lower than this threshold are ignored.

- `show_conf=False`: Determines whether to display the confidence scores of detected objects.

- `agnostic_nms=True`: Indicates whether to use agnostic Non-Maximum Suppression (NMS) during post-processing. Agnostic NMS treats all classes equally during NMS, which can be useful for certain scenarios.

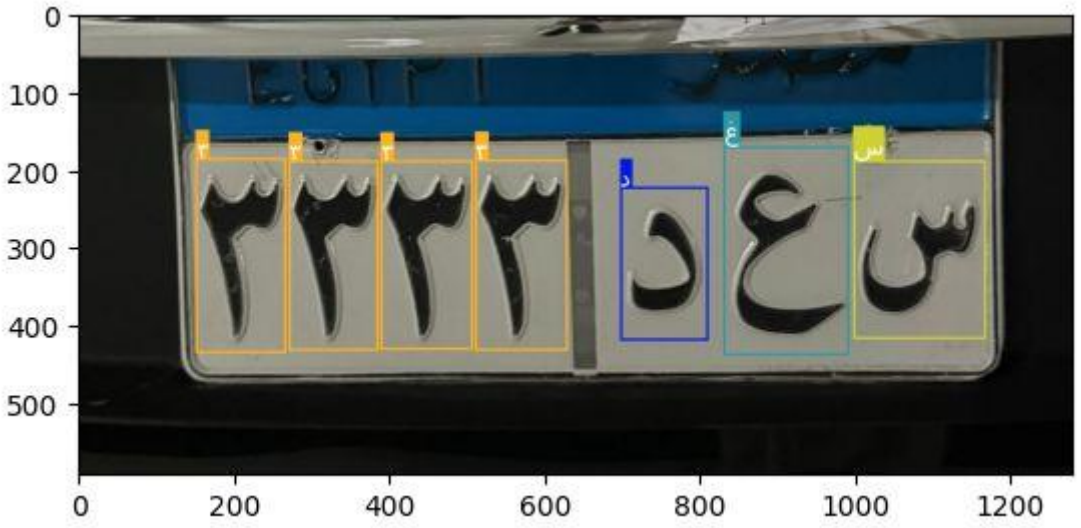
- **Post-processing and Visualization:**

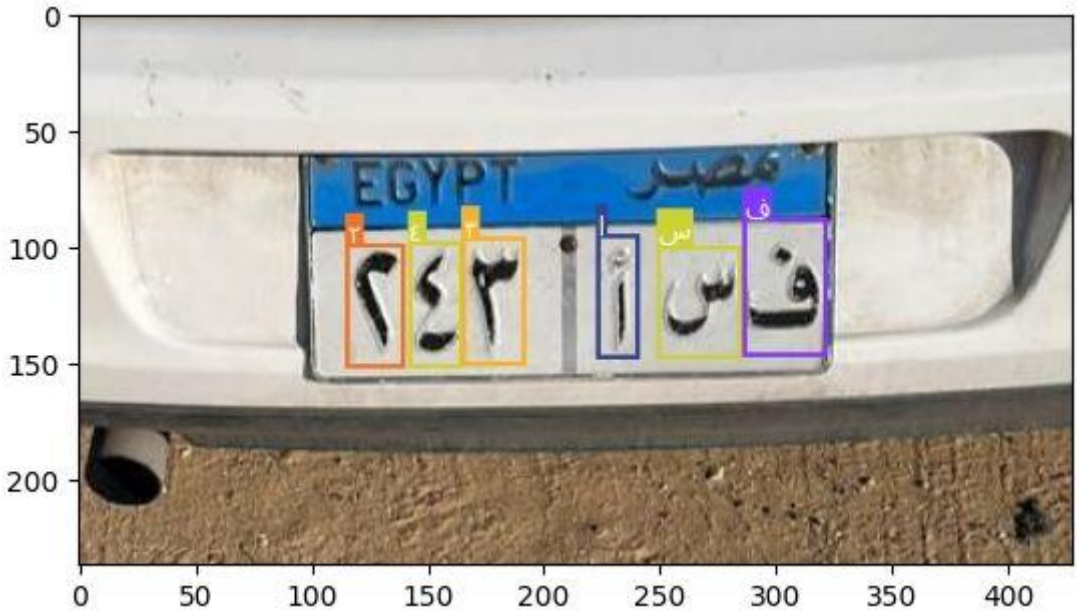
```
import numpy as np

results= model('/content/runs/detect/predict/1.jpeg')
results= model('/content/runs/detect/predict/2.jpeg')
results= model('/content/runs/detect/predict/3.jpeg')
results= model('/content/runs/detect/predict/4.jpeg')
results= model('/content/runs/detect/predict/5.jpeg')

for r in results:
  boxes = r.boxes.data.cpu().numpy()
  sorted_indices = np.argsort(boxes[:, 0])
  sorted_texts = [boxes[i] for i in sorted_indices]

  print("Ordered Text: ", sorted_texts)
import matplotlib.pyplot as plt
from PIL import Image
img = Image.open('/content/runs/detect/predict/1.jpeg')
plt.imshow(img)
img = Image.open('/content/runs/detect/predict/2.jpeg')
plt.imshow(img)
img = Image.open('/content/runs/detect/predict/3.jpeg')
plt.imshow(img)
img = Image.open('/content/runs/detect/predict/4.jpeg')
plt.imshow(img)
img = Image.open('/content/runs/detect/predict/5.jpeg')
plt.imshow(img)
```

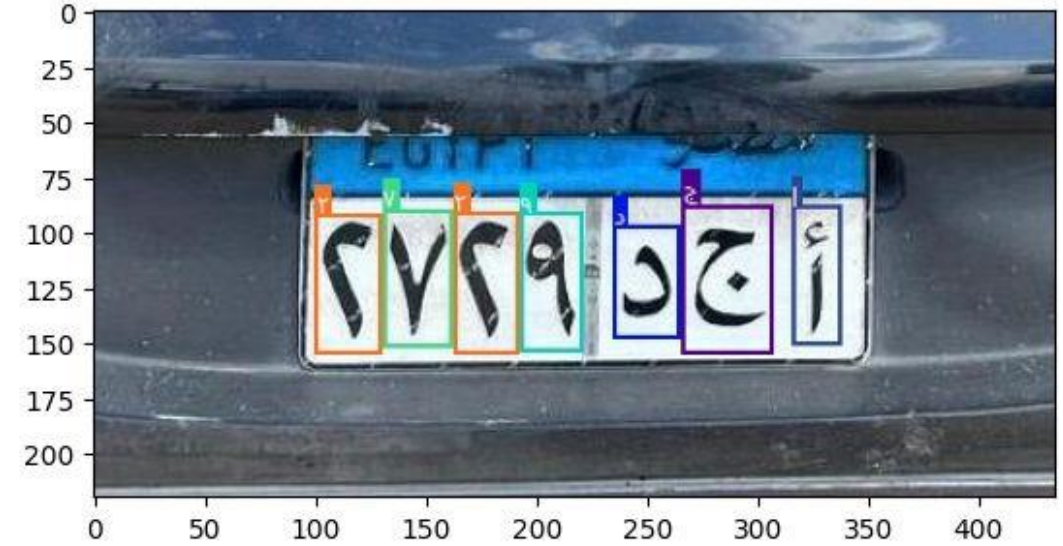
After making predictions, the code processes the results for each image. It iterates over the results and extracts bounding box coordinates (`boxes`). Then, it sorts the bounding boxes based on their x-coordinates (`boxes[:, 0]`). The sorted bounding box coordinates are stored in `sorted_texts`. Finally, it prints the ordered bounding box coordinates for each image.

For visualization, the code uses matplotlib and PIL (Python Imaging Library). It opens each predicted image using PIL's `Image.open` method, then displays the image using matplotlib's `imshow` function. This allows you to see the original image with bounding boxes drawn around detected objects.

The following images are 5 images that were the output of the model which shows that it succeeded in recognizing the letters and numbers on the plates, highlighting the characters, and writing the number or letter above each one of them after it finished the localizing and prediction process.


EGYPT
٣٣٣٣
د
ع
س


EGYPT
مصر
٢٤٣
أ
س
ف


EGYPT
٢٧٢٩
د
ج
أ

Conclusion:

In this project, we successfully implemented a machine learning-based approach to recognize Arabic characters on number plates. By using a big dataset of over 1400 annotated images and using the YOLOv8 model for object detection, we achieved huge results in both the detection and recognition phases.

The usage of the YOLOv8 model proved to be great due to its efficiency and accuracy in real-time object detection tasks. Through training and testing, the model demonstrated a high level of performance in recognizing Arabic characters from number plates, as shown by the successful predictions on multiple test images. The results were consistently reliable, with the model accurately identifying and annotating the characters, thus showing that our approach is successful and better than the Classical-Based Approach when using a Neural Network.

Potential improvements could include expanding the dataset for even greater accuracy and exploring other neural network models to further optimize performance.

All things considered, this significant achievement shows how machine learning methods can be effectively applied to solve real-world issues, opening up possibilities for automatic recognition system applications. The results of this project support the viability and efficiency of applying YOLO-based models for precise and effective number plate character recognition, making significant contributions to the fields of computer vision and machine learning.