Number Plate Detection & OCR

Presented By:

Mostafa Hamada,

Yasin Elmezayen,

Omar Alsabahi

Content

- Overview
- Motives
- Project Architecture
- Challenges
- Future aspects
- Conclusion
- Life Demo

Introduction

This project focuses on developing a robust system to accurately detect and recognize car number plates. The system utilizes advanced technologies in computer vision, deep learning, and optical character recognition.

Key Motives

- Ability to identify and manage traffic violations efficiently.
- Automated Toll Collection and enhancing the efficiency and accuracy of toll collection.

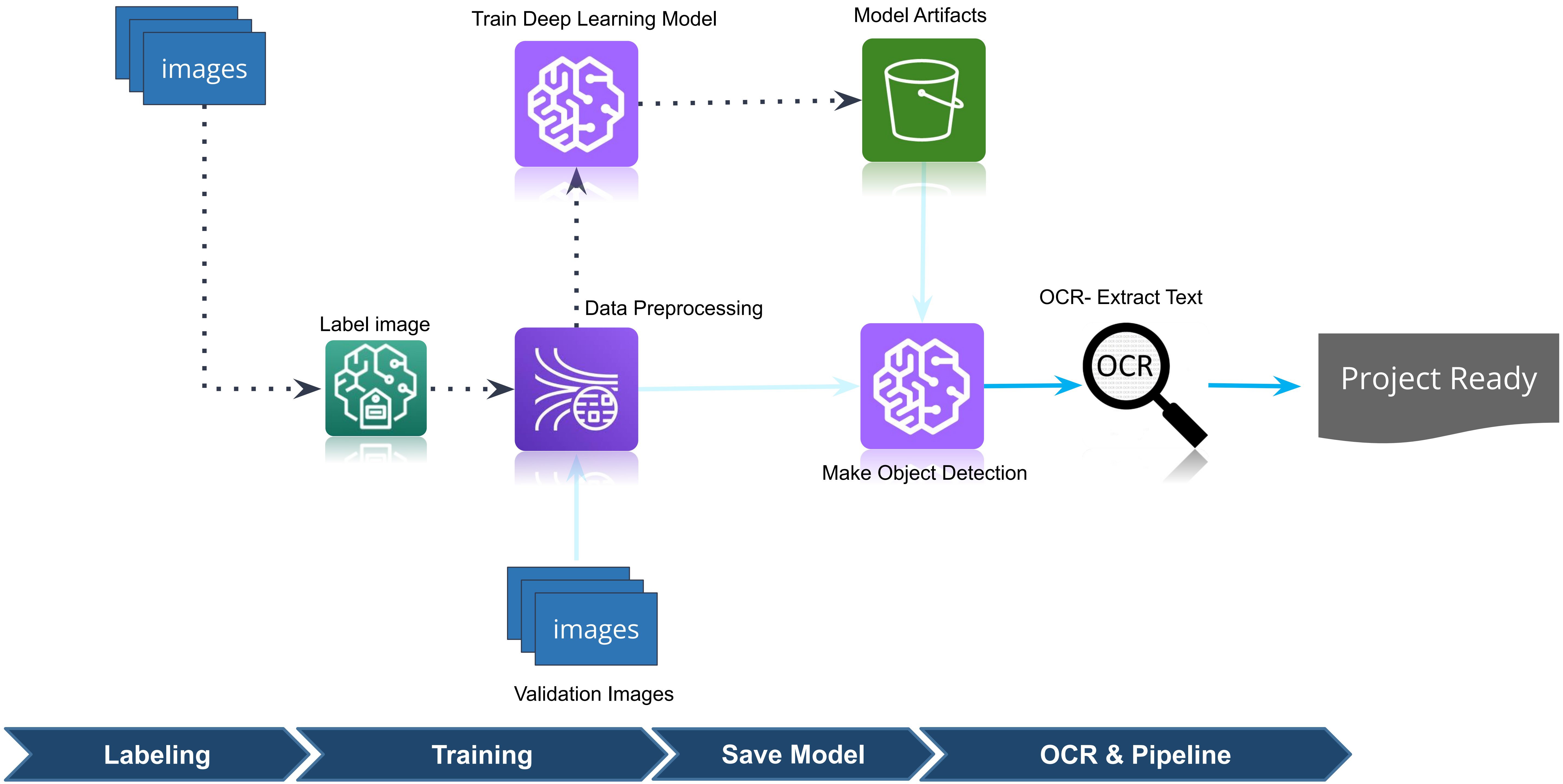
Law Enforcement and Security:

- Facilitating the identification of stolen or wanted vehicles.
- Enhancing surveillance and security in high-risk areas.

Parking Management:

- Improving the management of parking spaces in urban areas.

Project Architecture



Dataset Overview

Dataset Description:

- **Total images: 228**
- **Images contain cars with visible number plates.**
- **Variety of angles and lighting conditions to ensure robustness.**

Label Studio

- Label Studio is an open-source data labeling tool that supports multiple data types.
- User-friendly interface for annotating images with bounding boxes.

Data Annotation

- **Step-by-Step Process:**
  1. **Import Images:**
     - **Import the dataset of 228 car images into Label Studio.**
     - **Supported formats include JPEG, PNG, etc.**
  2. **Annotation:**
     - **Use the bounding box tool to draw rectangles around the car number plates.**
     - **Assign a label called "Number Plate" to each annotation.**
  3. **Review and Edit:**
     - **Review annotations for accuracy and consistency.**
     - **Edit annotations if necessary.**

Exporting Annotations


```
▼<annotation verified="yes">
  <folder>images</folder>
  <filename>N1.jpeg</filename>
  <path>C:\Users\Mostafa\Desktop\Project_Files\1_Labeling\images\N1.jpeg</path>
 ▼<source>
    <database>Unknown</database>
  </source>
 ▼<size>
    <width>1920</width>
    <height>1080</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
 ▼<object>
    <name>number_plate</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
   ▼<bndbox>
      <xmin>1099</xmin>
      <ymin>647</ymin>
      <xmax>1402</xmax>
      <ymax>729</ymax>
    </bndbox>
  </object>
</annotation>
```


Exporting Annotations

- **After labeling all images, export the annotations.**
- **Choose the XML file format for exporting.**
- **XML format includes details about bounding boxes and**

  image file paths.

Converting XML to CSV

Converting XML to CSV

- Why Convert to CSV:
  - CSV format is easier to manipulate and use in model training.
  - Simplifies data loading and preprocessing in machine learning pipelines.a
- Conversion Process:
  - Use a Python script to convert XML files to a CSV format.
  - The CSV file includes columns for image file paths and bounding box coordinates (xmin, ymin, xmax, ymax).

Splitting the Dataset

Splitting the Dataset

- **Importance of Splitting the Dataset:**
  - Ensures the model is trained on one subset of data and tested on another to evaluate performance.
  - Prevents overfitting and provides a measure of how the model generalizes to unseen data.
- **Splitting Process:**
  - Split the dataset into 80% training and 20% testing sets.
  - Use the `train_test_split` function from `sklearn.model_selection`.

Model Architecture

```
inception_resnet = InceptionV3(weights="imagenet",include_top=False,input_tensor=Input(shape=(224,224,3)))
inception_resnet.trainable=False
# --------------------
headmodel = inception_resnet.output
headmodel = Flatten()(headmodel)
# headmodel = Dense(500,activation="relu")(headmodel)
headmodel = Dense(64,activation="relu")(headmodel)
headmodel = Dense(4,activation='sigmoid')(headmodel)
# --------- model
model = Model(inputs=inception_resnet.input,outputs=headmodel)
# complie model
model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.Adam(learning_rate=1e-6))
model.summary()
```

Model Architecture

- Pre-trained InceptionV3 Model:
  - InceptionV3 used for feature extraction.
  - Pre-trained on ImageNet dataset.
  - Include top layer removed to adapt for new task.
- Custom Head for Bounding Box Regression:
  - Flattening the output of InceptionV3.
  - Adding dense layers for predicting bounding box coordinates.

Model Training Setup

```
history = model.fit(x=x_train,y=y_train,batch_size=10,epochs=100,
                    validation_data=(x_test,y_test),callbacks=[tfb])
```

Model Training Setup

- Training Configuration:
  - Optimizer: Adam with a learning rate of 1e-6
  - Loss function: Mean Squared Error (MSE)
  - Batch size: 10
  - Number of epochs: 100

Model Performance

Model Performance

- **Evaluation Metric:**
  - **Mean Squared Error (MSE)**
- **Results:**
  - **Training vs. validation performance**

Model Training Performance Analysis

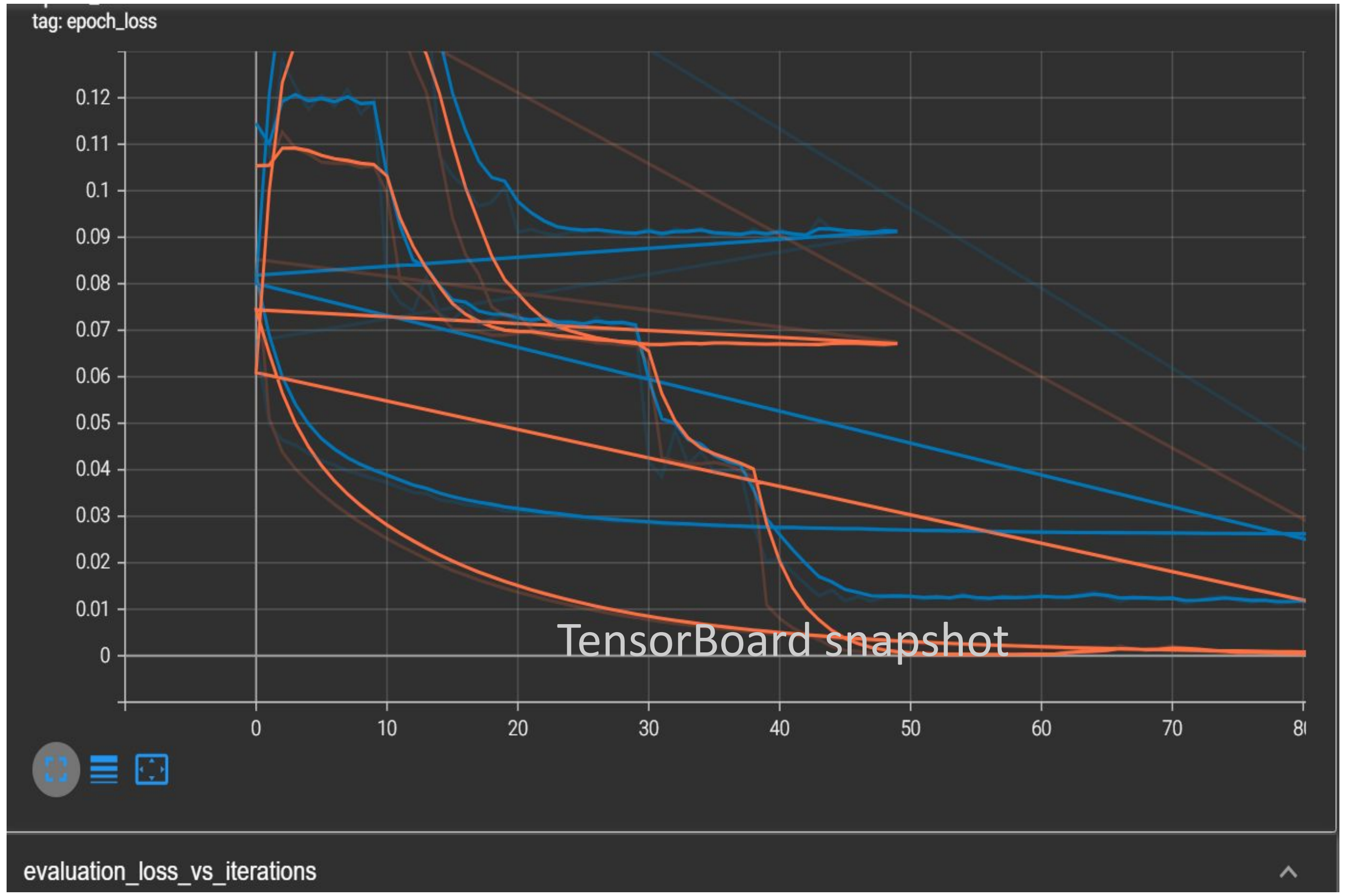
Model Training Performance Analysis

First Epoch :

Name	Smoothed	Value	Step
train	0.1054	0.1054	0
validation	0.1145	0.1145	0

Final Epoch:

Name	Smoothed	Value	Step
train	3.8996e-4	3.6717e-4	99
validation	0.02601	0.02599	99



Model Training

- At the beginning (initial epochs), both training and validation loss start relatively high and begin to decrease quickly.
- The model rapidly improves during the initial epochs, which is expected as it learns the fundamental patterns in the data.

Convergence:

- As training progresses, both the training and validation loss continue to decrease.
- The training loss tends to decrease more smoothly, whereas the validation loss has some fluctuations.
- Towards the end of the training (final epochs), the losses start to stabilize, indicating the model is converging.

Predictions

- Our model demonstrated excellent performance in detecting the number plate, even when the plate was not in the regular position.

- This example highlights the robustness and accuracy of our model in diverse conditions.

Other Examples of The Model Predictions

Integrating OCR with Tesseract

- Tesseract is an open-source OCR engine that can recognize text in images.
- It is highly effective for recognizing characters on number plates.

Why Tesseract:

- High accuracy in text recognition.
- Supports multiple languages and scripts.
- Easy to integrate with Python using the `pytesseract` library.

Example of Extracting the Numbers of a plate


```python
# extract text from image
text = pt.image_to_string(roi)
print(text)
```

```
TS 08 FM 8888
```

Limitations of Tesseract OCR

Common Limitations:

1. **Image Quality:**
2. **Varied Fonts and Styles:**
3. **Complex Backgrounds:**
4. **Lighting Conditions:**
    - **Inconsistent results under different lighting conditions**

Technical Limitations:

- **Processing Speed:**
    - **Slower processing times compared to some commercial OCR solutions, especially for large batches of images.**

Future work and challenges

- Transition to real-time video feeds for continuous, immediate recognition.
- Improve OCR to handle low-quality images, noise, and distortions more effectively.

Conclusion

- **Successful Development**:
  - Robust system for number plate detection and recognition.
- **Key Achievements**:
  - High accuracy in diverse conditions.
  - Effective integration of Tesseract OCR.
- **Applications**:
  - Enhanced traffic management, law enforcement, and parking.
- **Challenges**:
  - Managed variations in lighting, angles, and image quality.
- **Future Work**:
  - Real-time video recognition.
  - Improved OCR for low-quality images.

Thanks for listening

Any Questions?