

**Visionary Detect: Unleashing FRCNN for Precision Object Detection**

SmartInternz

www.smartinternz.com

# 

# Project Description:

### Develop a robust object detection and classification system capable of identifying and categorizing virtual objects in real-time from various digital environments. The system should be optimized to recognize objects across different virtual platforms, such as augmented reality (AR), virtual reality (VR), and video games, with high accuracy and low latency. This includes handling diverse visual contexts, object sizes, and orientations while maintaining performance in resource-constrained environments.

### Scenarios

**Scenario 1: Virtual Training Simulations**

In a VR-based military training simulation, the object detection system identifies and classifies virtual equipment, vehicles, and targets within the environment. Trainees interact with the virtual objects, and the system must respond in real-time, providing feedback based on the detected objects.

**Scenario 2: Smart Virtual Assistants**

A smart virtual assistant integrated within a VR workspace identifies and classifies objects in the virtual environment, such as documents, tools, or interface elements. The assistant helps users interact with the environment more efficiently by providing shortcuts and automating tasks based on the detected objects.

**Scenario 3: Virtual Social Interaction Platforms**

In a virtual social interaction platform, users create and share virtual spaces with objects like avatars, decorations, and digital art. The object detection system helps in classifying these objects for moderation, content filtering, and user preference settings.

# 

# Technical Architecture:

****

# Prerequisites:

**Python Programming**: Intermediate knowledge of Python, including experience with libraries such as NumPy, PIL, and Flask.

**Deep Learning Concepts**: Understanding of CNNs, object detection, and transfer learning.

**PyTorch Framework**: Familiarity with PyTorch, especially in building and fine-tuning models.

**Web Development**: Basic knowledge of Flask for deploying machine learning models as web applications.

### 

### ● Anaconda Navigator :

**o** Refer to the link below to download Anaconda Navigator

**o** Link: https://youtu.be/C4OPn58BLaU?si=XQtp-Ij5jtUdy3cn

### ● Python packages:

o Open anaconda prompt as administrator

o Type “pip install torch” and click enter.

o Type “pip install pycocotools” and click enter.

o Type “pip install matplotlib” and click enter.

o Type “pip install torchvision” and click enter.

o Type “pip install seaborn” and click enter.

o Type “pip install Flask” and click enter.

# Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

#### Prior Knowledge:

* **Convolutional Neural Networks (CNNs)**: Understanding of how CNNs operate, particularly for tasks like image classification and object detection.

<https://arxiv.org/pdf/1511.08458>

* **Object Detection Algorithms**: Familiarity with object detection models, such as YOLO, SSD, or earlier versions of R-CNN, will help in understanding the architecture of Faster R-CNN.

<https://arxiv.org/pdf/1506.01497>

* **Flask**: Basic experience in building and deploying web applications using Flask.

● **Flask Basics**: <https://www.youtube.com/watch?v=lj4I_CvBnt0>

# 

# Project Objectives:

The objective of this project is to develop an advanced, data-driven model capable of identifying and mitigating network anomalies with high accuracy and efficiency. This project aims to:

1. **Enhance Network Security**: By employing an autoencoder-based deep learning model, the project seeks to detect and classify anomalies such as unauthorized access, Denial of Service (DoS) attacks, and other suspicious activities in network traffic. Early detection of these anomalies can prevent potential security breaches and protect sensitive information.
2. **Improve Incident Response**: The project aims to provide real-time monitoring and alerting capabilities, enabling quicker identification and classification of anomalies. This prompt response is crucial for minimizing the impact of any detected threats, ensuring timely and appropriate action is taken to maintain network integrity and availability.
3. **Optimize Resource Allocation**: By accurately identifying high-risk anomalies, the project helps prioritize security efforts and resource allocation. This includes focusing on the most significant threats, thereby reducing unnecessary expenditures on benign or low-risk activities. The model also aims to optimize the deployment of network monitoring and security personnel, improving overall operational efficiency.

# Project Flow:

**Project Setup and Environment Preparation**

* **Install Dependencies**: Ensure all necessary Python packages like PyTorch, torchvision, Flask, and PIL are installed.
* **Dataset Preparation**: The project uses the Penn-Fudan Dataset, which contains images of pedestrians. This dataset is loaded and preprocessed in the Jupyter notebook.

**Data Preparation and Model Training**

* **Data Loading and Transformation**: Load the Penn-Fudan dataset and apply necessary transformations, such as converting images to tensors.
* **Model Definition**: Use a pre-trained Faster R-CNN model and fine-tune it to specifically detect pedestrians.
* **Training**: The model is trained in the notebook using the dataset, with monitoring of loss and performance metrics.

**Evaluation**

* **Testing the Model**: The trained model is evaluated on a validation set to measure its accuracy in detecting pedestrians.
* **Non-Maximum Suppression (NMS)**: NMS is applied to filter overlapping boxes and ensure the most confident detections are retained.

**Deployment via Flask**

* **Flask Application Setup**: The web application is structured using Flask, allowing users to upload images for pedestrian detection.
* **Model Integration**: The trained model is loaded into the Flask app, and predictions are generated for uploaded images.
* **Bounding Boxes and Labels**: The draw\_bounding\_boxes function is used to draw bounding boxes around detected pedestrians, label them with their class names , and display the confidence score.

**Testing and Deployment**

* **Testing the Web App**: The Flask app is tested locally with various images to ensure the model correctly identifies pedestrians.
* **Deployment**: The Flask application can be deployed on cloud platforms or hosted locally for real-time pedestrian detection.

# Project Structure:

Create the Project folder with the

Project name



We have to store our files and

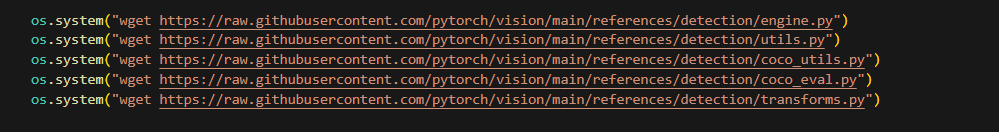
folders in this structure

# 

# Milestone 1: Project Setup and Environment Preparation

**Activity 1: Install Dependencies**

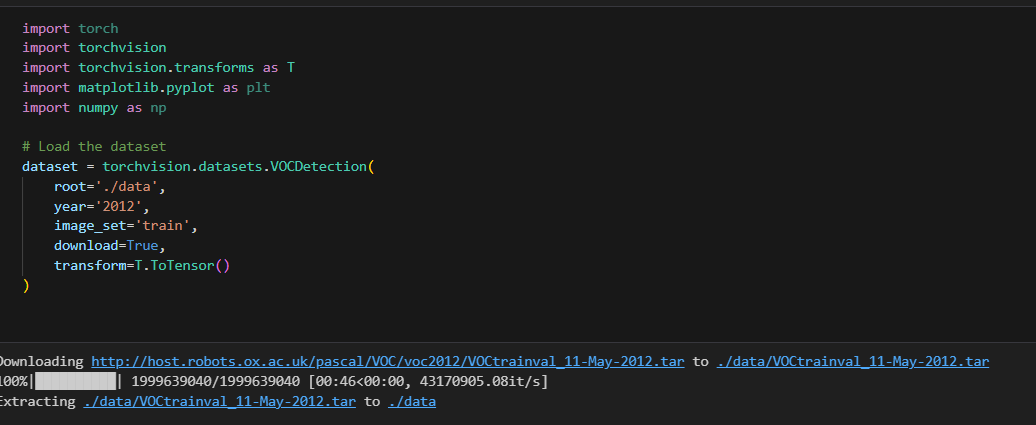
* Install necessary Python packages including PyTorch, torchvision, PIL, and Flask.
* Also download the engine.py, utils.py, coco\_utils.py, coco\_eval.py and transforms.py from the pytorch vision
* Set up the Jupyter notebook environment for model training.



# Milestone 2: Data Preparation

**Activity 1: Dataset Download**

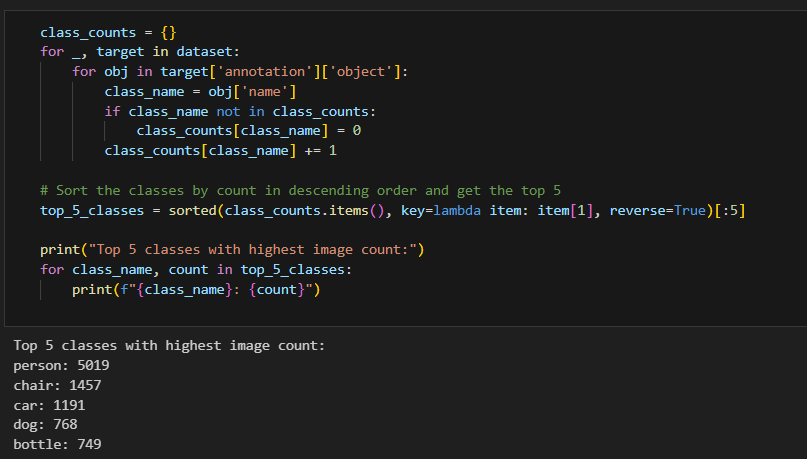
* Download the Penn-Fudan dataset, which contains annotated labelled images.
* Store the dataset in an accessible location for both training and testing.

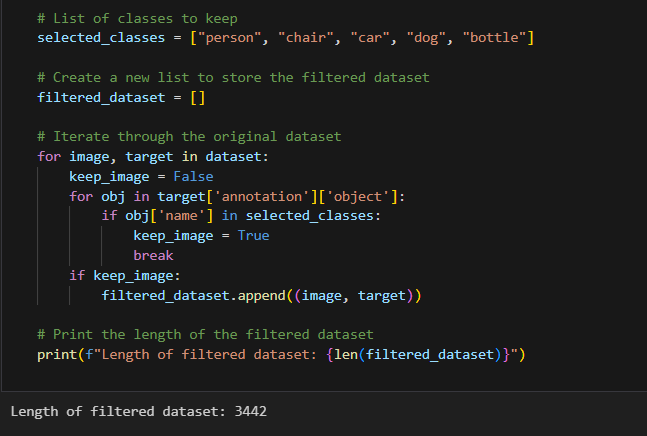


**Activity 2: Data Transformation**

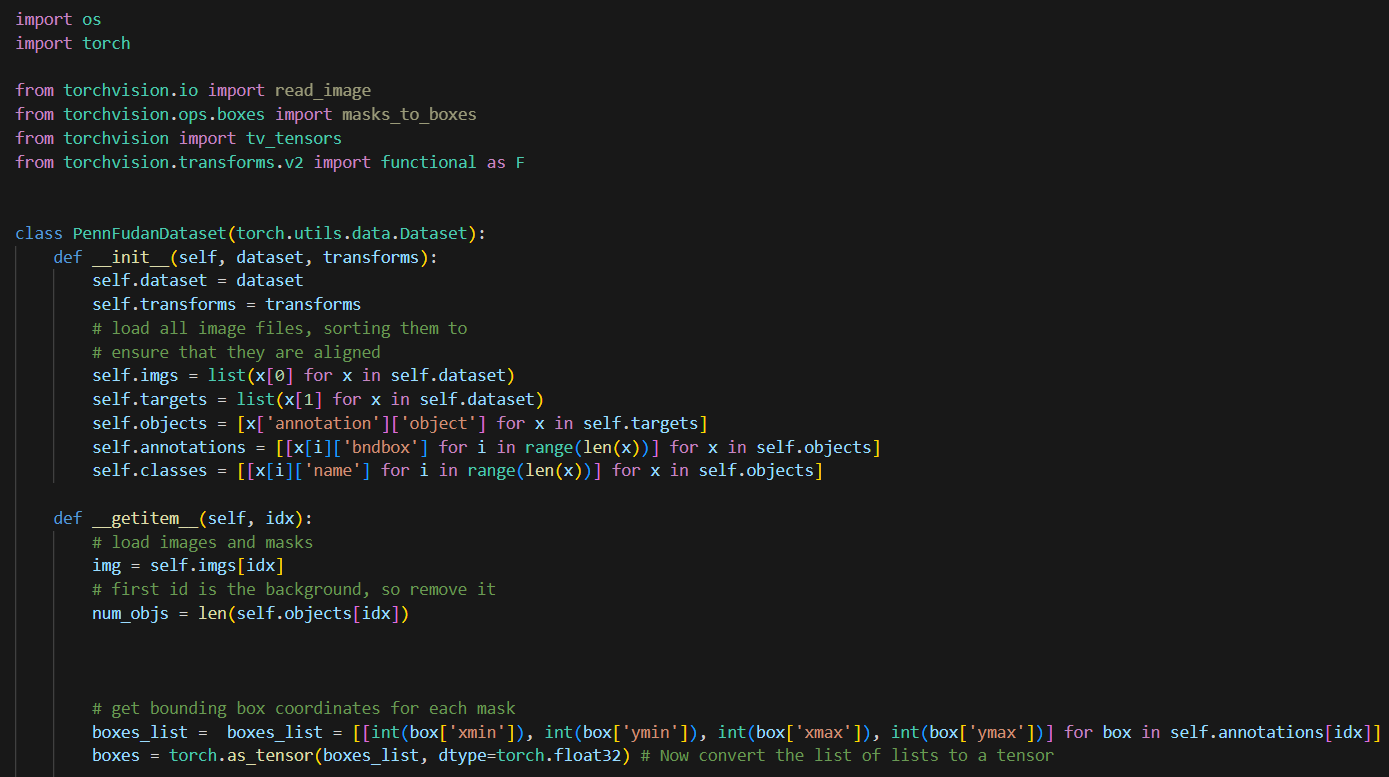
**Activity-2.1 loading only top 5 classes for the model (to get good accuracy):**

* Below code we have checked the top 5 classes which have high count of images and filtered the data.





* Load the dataset using PyTorch’s DataLoader.
* Apply necessary transformations like resizing, normalization, and tensor conversion to the images.

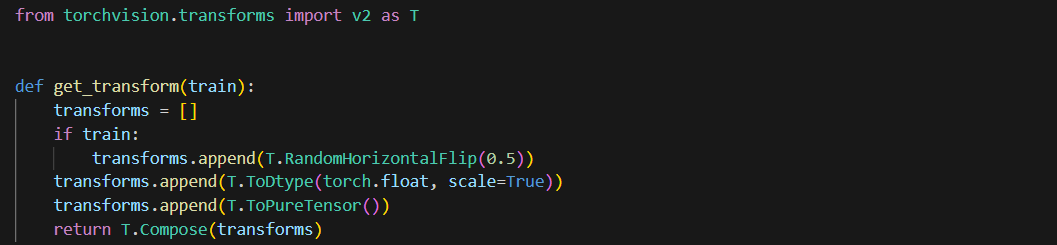




# Milestone 3: Model Training

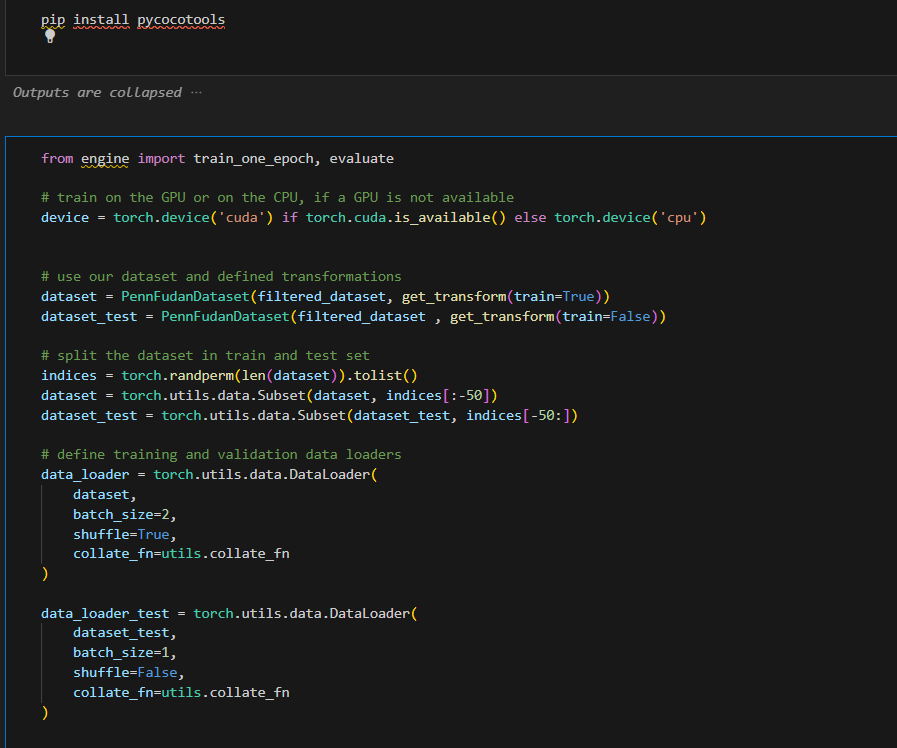
**Activity 1: Model Definition**

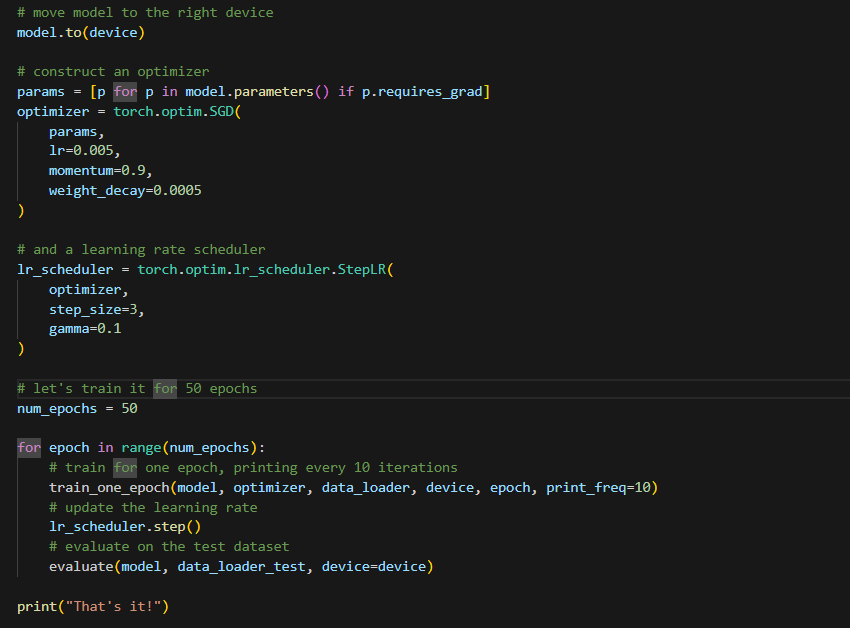
* Load a pre-trained Faster R-CNN model with a resnet50 as backbone.
* Modify the model to specifically detect objects, adjusting the number of classes accordingly.





**Activity 2: Model Training**

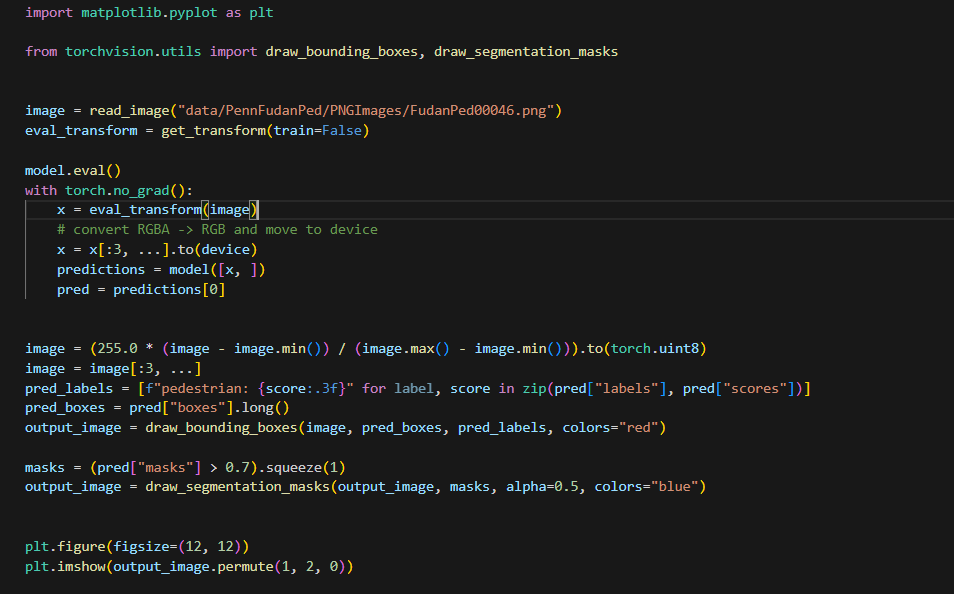
* Train the model using the transformed dataset, monitoring training loss and adjusting hyperparameters.
* Save the trained model for later use in the Flask application.



# Milestone 4: Model Building

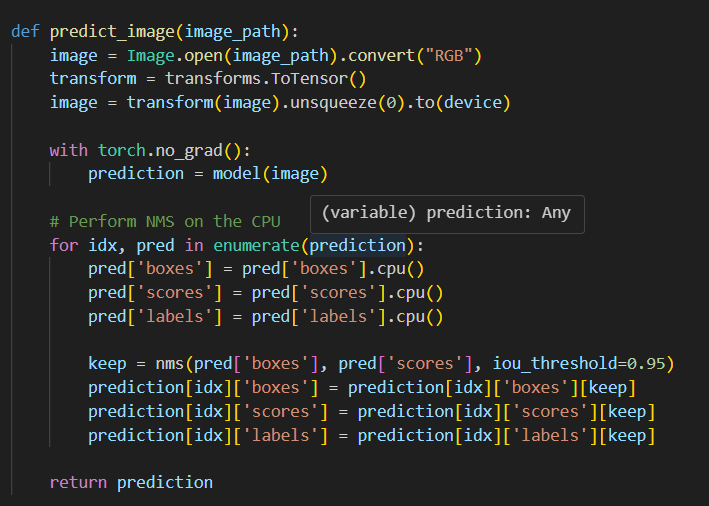
**Activity 1: Model Testing**

* Test the trained model on a separate validation set to evaluate its performance in detecting pedestrians.



**Activity 2: Non-Maximum Suppression (NMS)**

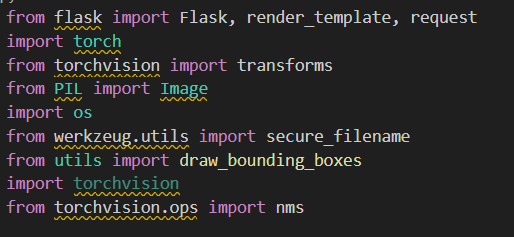
* Apply NMS to remove overlapping bounding boxes and retain the most confident detections.
* Evaluate the impact of NMS on detection accuracy and performance.

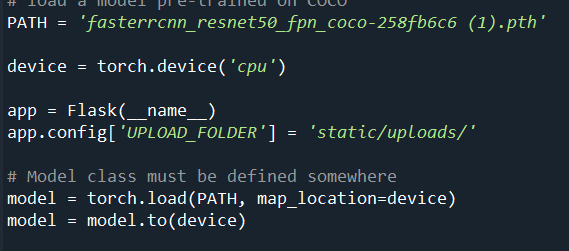


# Milestone 5: Application Building

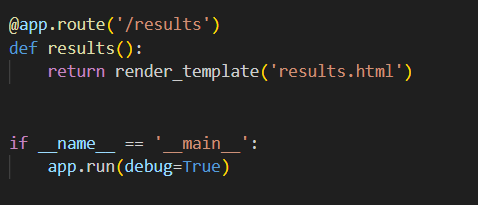
**Activity 1: Flask Application Setup**

* Set up the Flask application structure to allow image uploads and model inference.
* Create routes and views for the web interface.



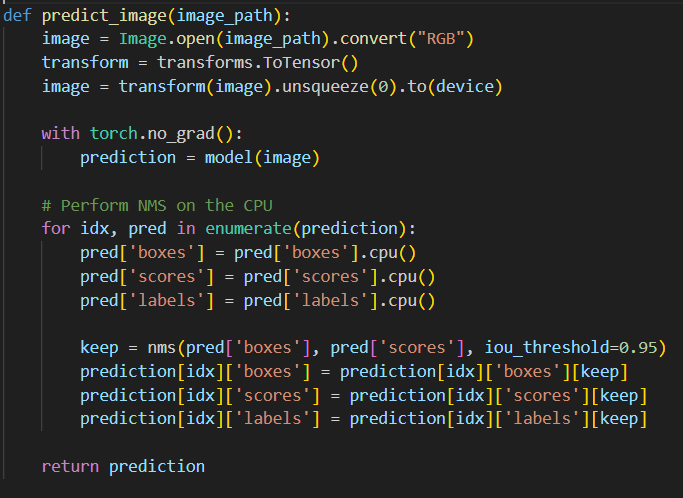


****

****

**Activity 2: Integrate Model with Flask**

* Load the trained model into the Flask app.
* Develop functionality to process uploaded images and return detection results.



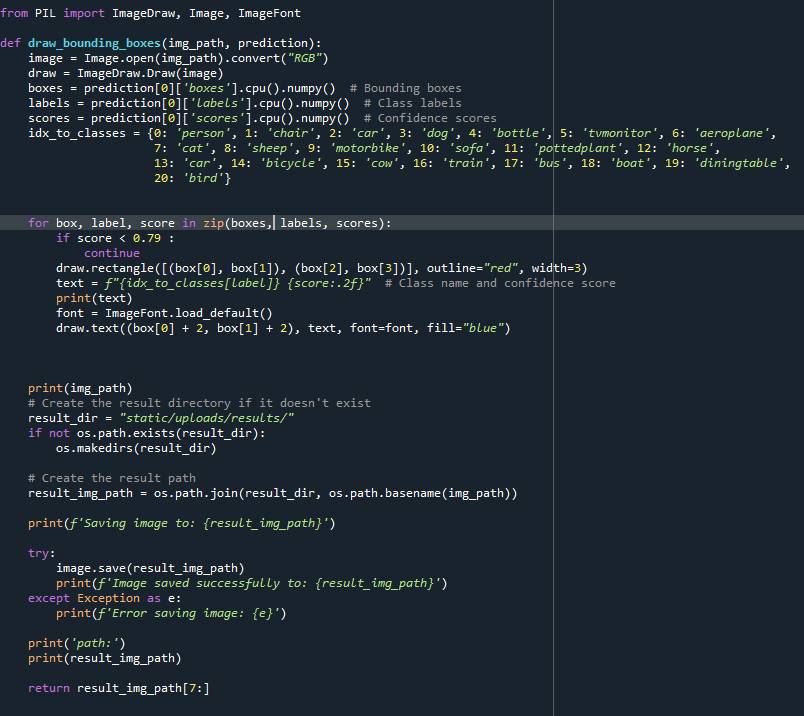
# Milestone 6: Drawing Bounding Boxes and Labels

**Activity 1: Draw Bounding Boxes**

* Use draw\_bounding\_boxes to draw rectangles around detected pedestrians.
* Annotate each box with the class names and confidence score.

**Activity 2: Save and Display Results**

* Save the processed image with bounding boxes in a designated directory.
* Display the processed image in the web application interface



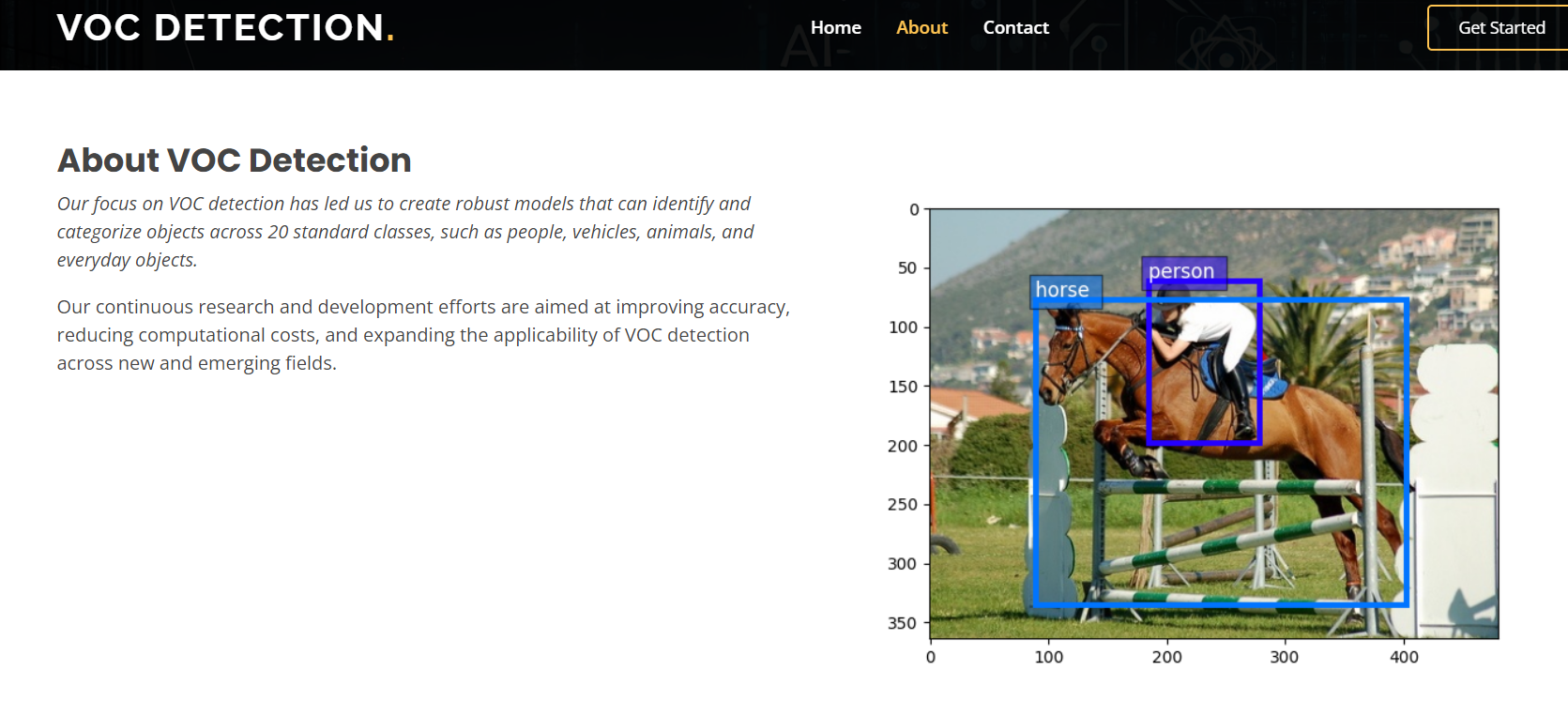
# Milestone 7: Testing and Deployment

* **Activity 1: Test the Web Application**
  + Upload various images to the Flask app to test the reliability and accuracy of the pedestrian detection system.
  + Verify that bounding boxes and labels are correctly displayed.

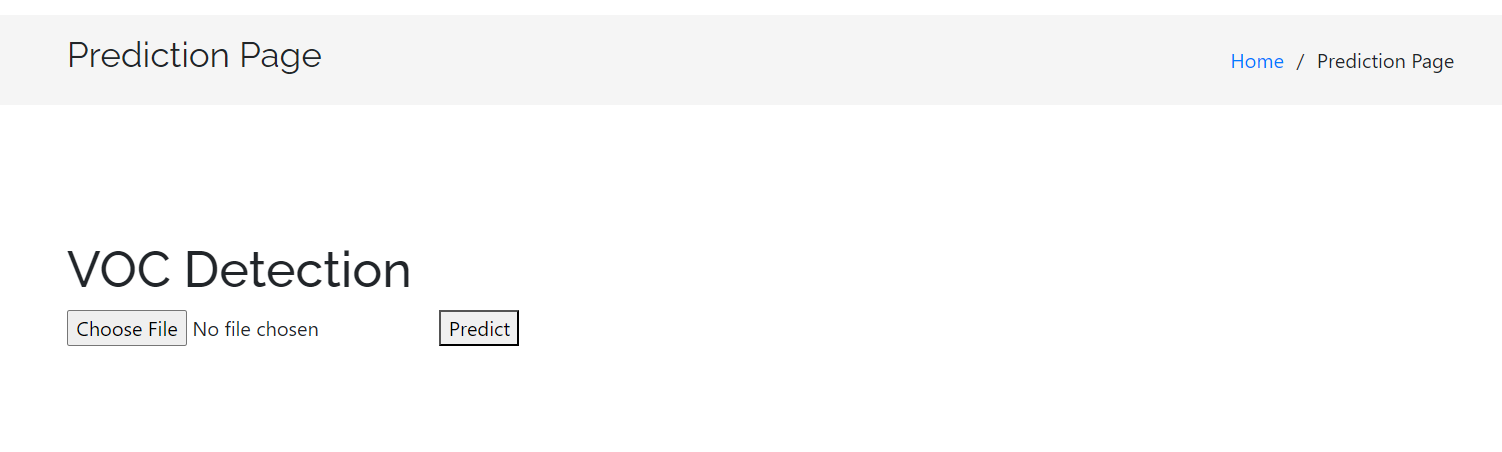
Home page :



About Section:



Prediction page:

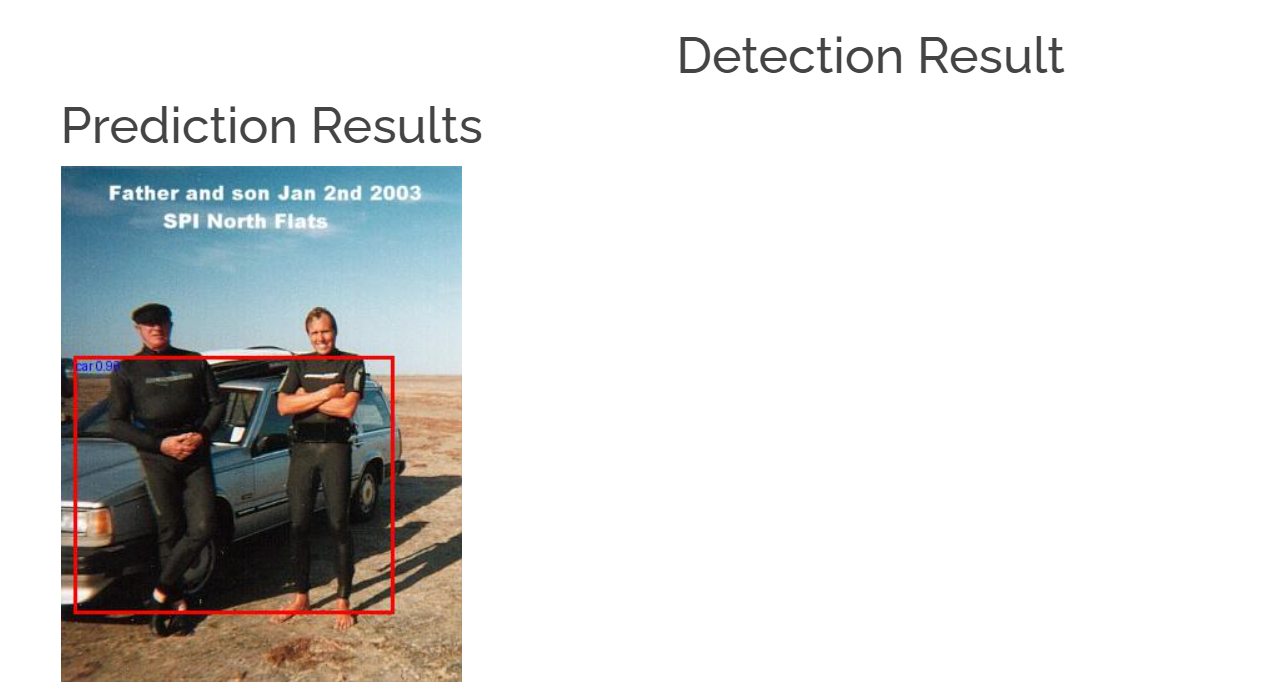


**Test case1:**

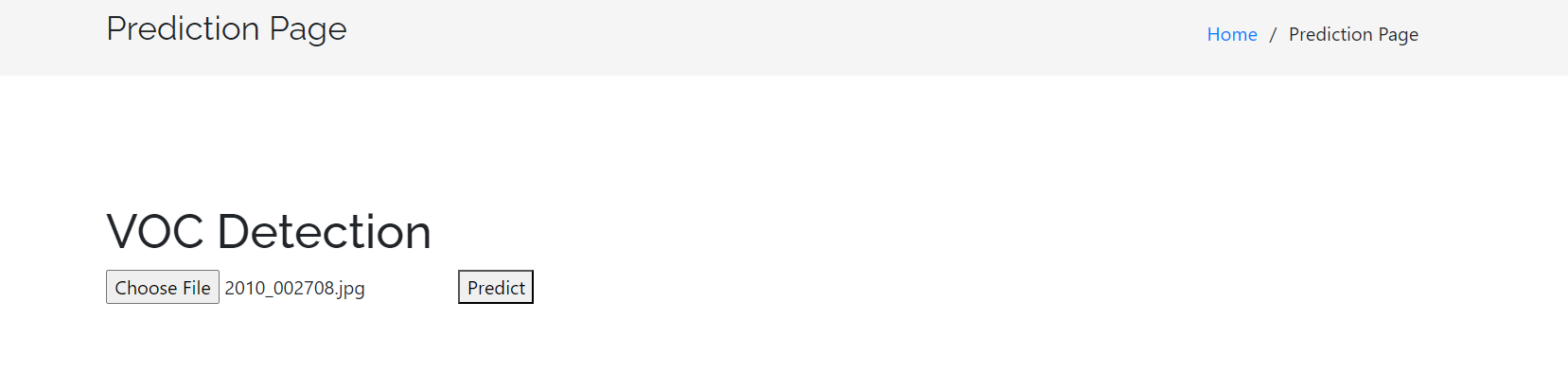
Let’s upload an image

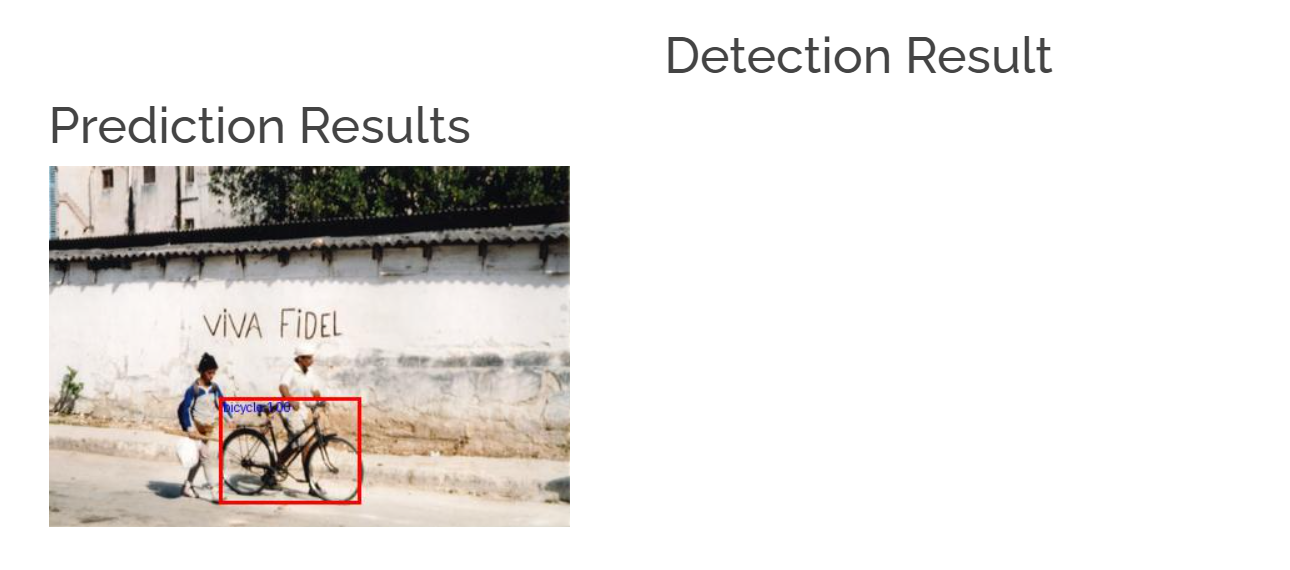


Result Page showing the object within bounding boxes

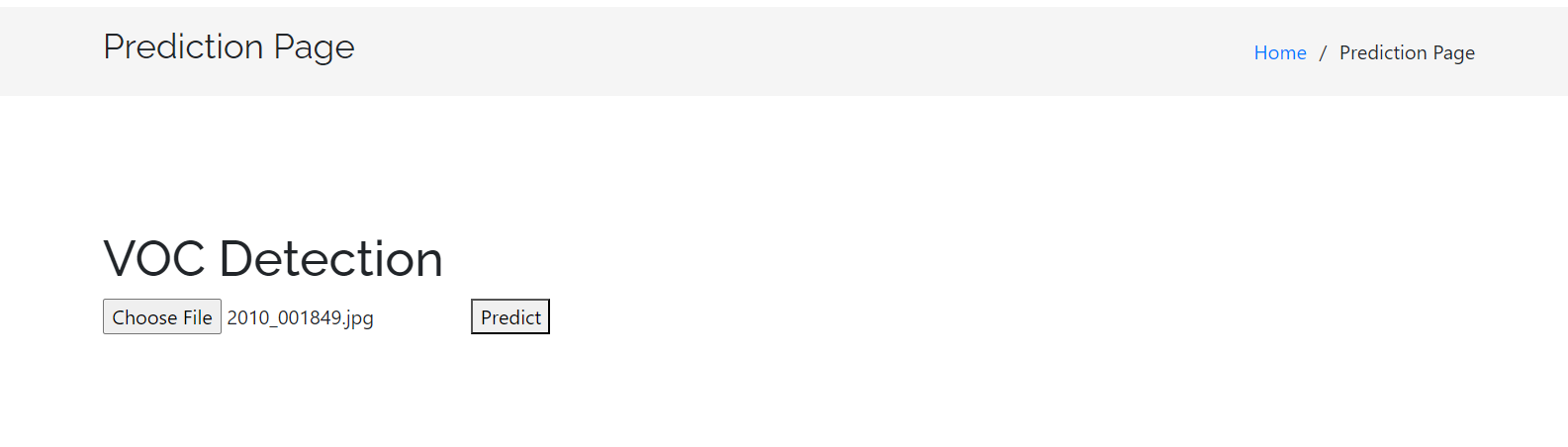


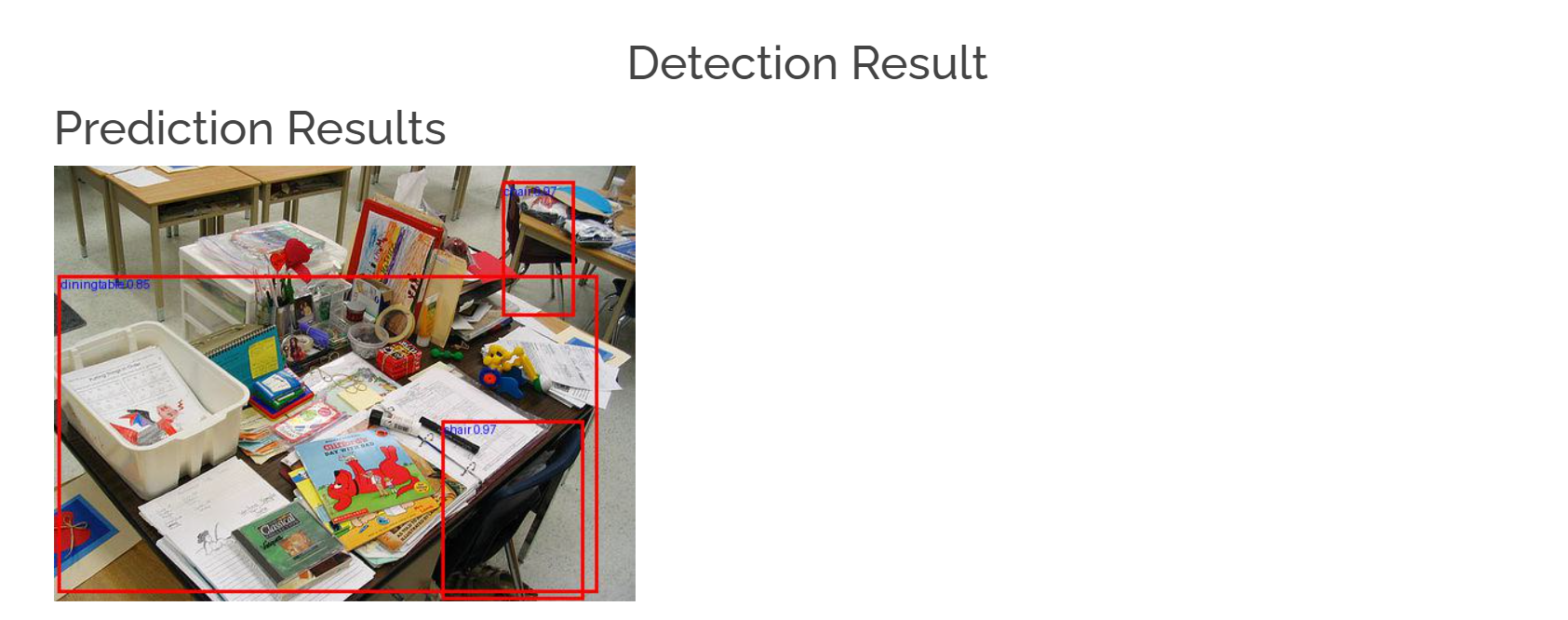
**Test case2:**

****

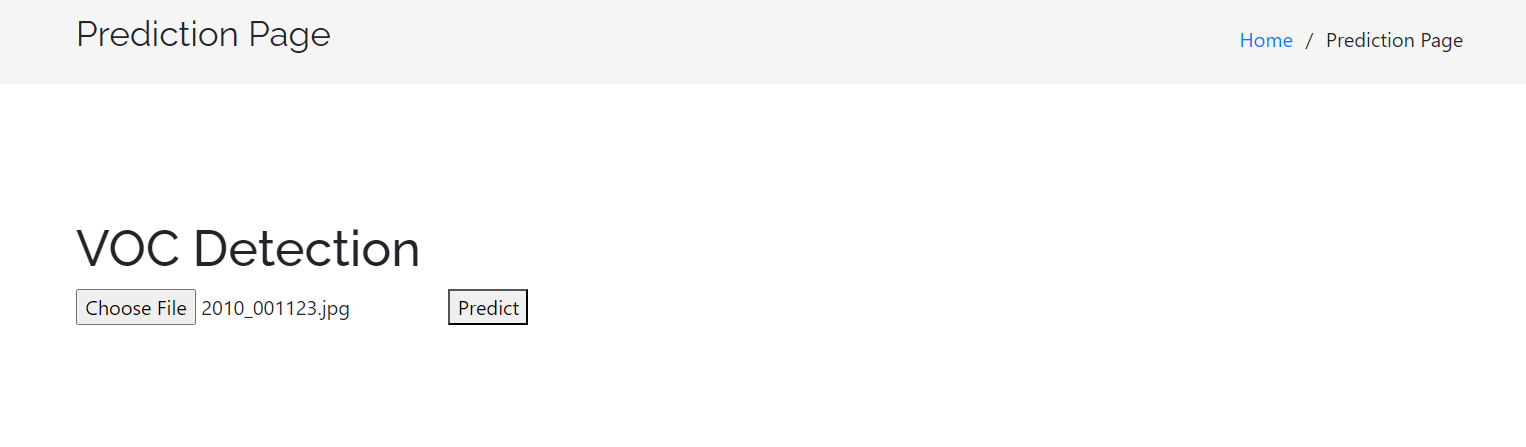
****

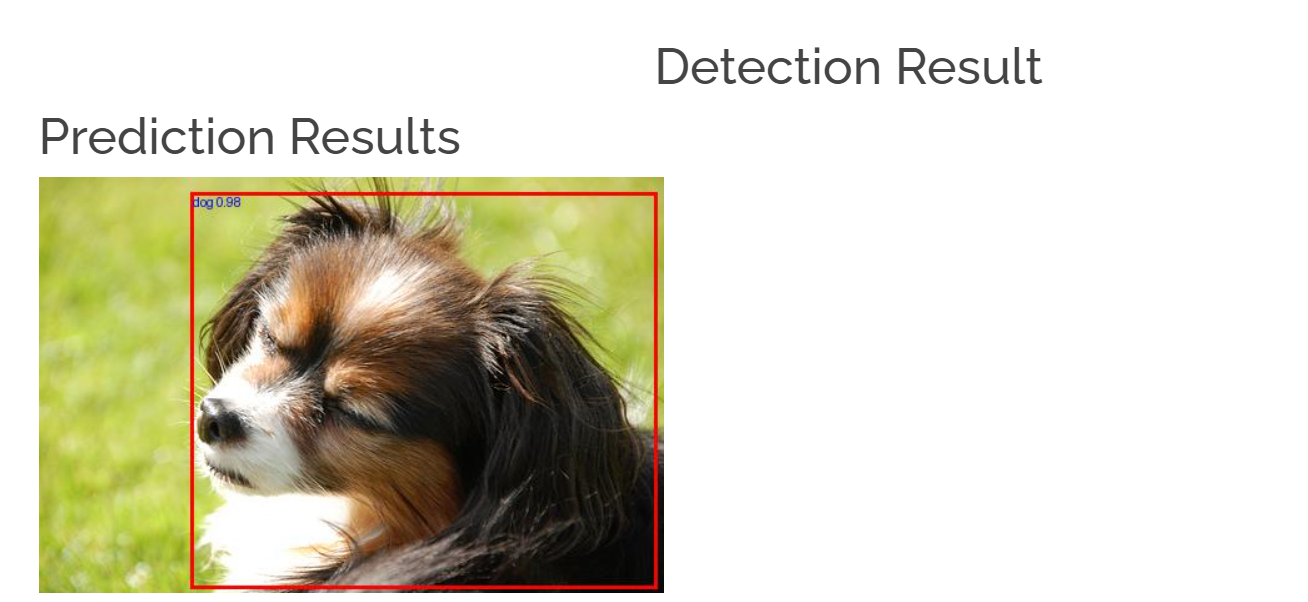
**Test case3:**

****

****

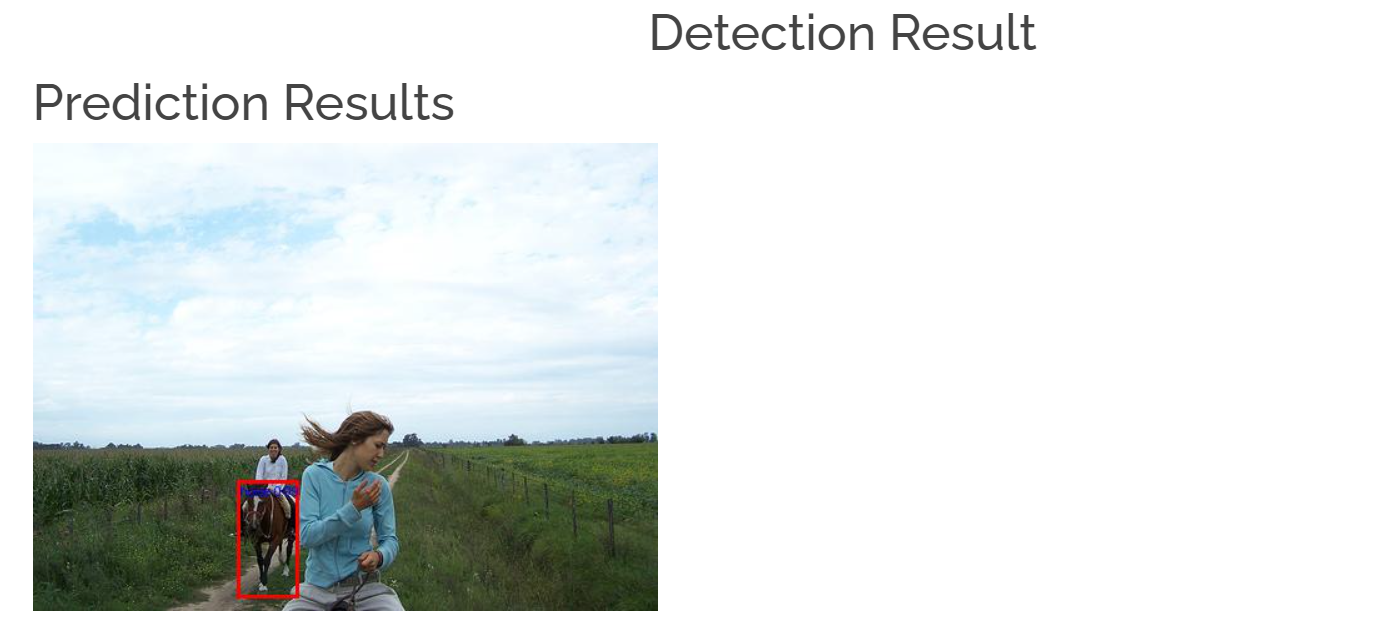
**Test case4:**

****

****

**Test case5:**

****

****