Object Detection in Images

A PROJECT REPORT
for
Introduction to AI (AI101B)
Session (2024-25)

Submitted by

Ansh Mishra
(202410116100030)
Abhishek Sharma
(202410116100009)
Ankit Sisodia
(202410116100029)
Ashutosh Singh
(202410116100043)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Apoorv Jain
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(DECEMBER- 2024)

CERTIFICATE

Certified that **Ansh Mishra (202410116100029), Abhishek Sharma (202410116100009)**, **Ankit Sisodia (202410116100029), Ashutosh Singh (202410116100043)** has carried out the project work having "**Object Detection in Images"** (**Introduction to AI, AI101B**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Mr. Apoorv Jain	**Dr. Akash Rajak**
Assistant Professor	**Dean**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

ABSTRACT

This project report focuses on the implementation of Object Detection in Images using Python in Google Colab, as a part of the "Introduction to AI" course. Object detection is a powerful computer vision technique that identifies and locates objects in images. With the advancement of artificial intelligence, applications involving image analysis and recognition have become more sophisticated, accurate, and prevalent in everyday life.

The purpose of this project is to explore how machine learning models and algorithms can be utilized to detect and classify objects in images using pre-trained models such as YOLO (You Only Look Once), SSD (Single Shot Multibox Detector), or Haar Cascade Classifiers provided by OpenCV. This project serves as a foundational introduction to real-world AI implementations in the field of computer vision.

The report covers the background of object detection, technical methodology, the tools used, code implementation, and results. It concludes with applications of object detection and possible future enhancements to the project. The knowledge gained from this project provides a stepping stone toward more complex AI systems, enabling students to develop practical AI skills for both academia and industry.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Apoorv Jain** for his guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Akash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Ansh Mishra
Abhishek Sharma
Ankit Sisodia
Ashutosh Singh

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Artificial Intelligence (AI) has revolutionized numerous industries, and one of its most impactful applications is in the field of computer vision. Among the various techniques used in computer vision, object detection plays a crucial role in enabling machines to recognize, localize, and classify objects within images or video streams. Object detection is the foundational step in more advanced applications such as autonomous driving, medical image analysis, industrial inspection, and surveillance systems.

Object detection combines two key tasks—classification and localization. Classification determines what object is in an image, while localization determines where the object is located by drawing bounding boxes. These capabilities allow systems to interact with and respond to real-world environments in real-time. With advancements in deep learning, object detection has seen significant improvements in both accuracy and speed.

Historically, object detection was implemented using classical methods such as Haar Cascades, which are based on feature extraction techniques and statistical models. While these methods are still in use, they have limitations in terms of accuracy and generalization. In recent years, deep learning-based approaches like You Only Look Once (YOLO), Single Shot MultiBox Detector (SSD), and Region-Based Convolutional Neural Networks (R-CNNs) have emerged, offering superior performance by learning features directly from the data.

This project focuses on implementing object detection using Python on Google Colab, a cloud-based platform that supports GPU acceleration and facilitates large-scale computations without requiring high-end local hardware. Google Colab allows for seamless integration of libraries such as OpenCV, TensorFlow, and PyTorch, which are essential for building and evaluating object detection models.
Through this project, we aim to explore different object detection methods, understand their underlying algorithms, compare their performance on various images, and analyze their strengths and weaknesses. We will use publicly available pre-trained models to detect and label objects in images, enabling hands-on learning of real-world AI applications. The project will also discuss the potential use-cases of object detection, its current limitations, and the future scope for improvement.

In summary, the purpose of this project is to provide an in-depth understanding of object detection techniques using Python, demonstrate practical implementation on a cloud platform, and highlight the relevance of object detection in modern AI-driven systems. The knowledge and skills gained through this project will serve as a stepping stone for more complex AI projects involving real-time data processing and intelligent decision-making.

CHAPTER 2

PROJECT OBJECTIVE

The primary objective of this project is to explore and implement various object detection techniques using Python in a Google Colab environment. Object detection is a core task in the field of computer vision that involves locating instances of semantic objects such as humans, animals, or vehicles in digital images.

This project focuses on understanding the underlying principles, strengths, and limitations of both classical and deep learning-based object detection models. Specifically, the aim is to:

1. Learn and implement classical object detection algorithms such as Haar Cascades.

2. Explore and experiment with deep learning models like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector).

3. Compare the performance of these models in terms of accuracy, speed, and real-time applicability.

4. Demonstrate the application of these models on various image datasets for tasks such as face detection, vehicle detection, and general object recognition.

5. Utilize Google Colab to efficiently run and visualize the models, taking advantage of GPU support and Python libraries such as OpenCV, TensorFlow, and PyTorch.

The broader goal of the project is to develop a strong conceptual and practical understanding of how machines can be trained to interpret visual data. This will serve as a foundational step for students aspiring to work in AI, machine learning, robotics, and related domains where object detection plays a pivotal role.

By the end of the project, the learner should be equipped with:

- Knowledge of different object detection algorithms and when to use them.

- Hands-on experience in implementing and fine-tuning models.

- Skills to evaluate and interpret model performance using standard metrics.

- The ability to apply object detection techniques to real-world problems.

CHAPTER 3

TOOLS & TECHNOLOGY USED

This project utilized a combination of software tools, libraries, and platforms to enable efficient and effective object detection. The key tools and technologies include:

- **Python:** The primary programming language for implementation due to its simplicity and support for AI libraries.
- **Google Colab:** A free cloud-based Jupyter notebook environment that supports GPU acceleration, ideal for running heavy AI models.
- **OpenCV:** An open-source computer vision library used for image processing and the implementation of classical detection models like Haar Cascades.
- **TensorFlow and Keras:** Deep learning libraries used for running modern object detection models such as SSD and YOLO.
- **PyTorch:** An alternative deep learning framework known for its flexibility and ease of debugging.
- **Pre-trained Models:** Models trained on datasets like COCO and PASCAL VOC that provide high accuracy without the need to train from scratch.
- **Matplotlib:** A Python library for visualizing results by displaying images with bounding boxes.

CHAPTER 4

METHODOLOGY

The methodology for the project involved a series of systematic steps to implement object detection models, test them on different images, and evaluate the outcomes. The workflow can be broken down into the following stages:

1. **Data Collection:**

- Images were either sourced from the internet or uploaded manually for testing.
- The images varied in content, including scenes with multiple people, vehicles, and objects to ensure comprehensive testing.

2. **Model Selection:**

- Classical model: Haar Cascade classifier from OpenCV was used to detect human faces.
- Deep learning models: YOLOv3 and SSD MobileNet were used for detecting multiple object types with high accuracy.

3. **Environment Setup:**

- Google Colab environment was prepared with required libraries (OpenCV, TensorFlow, PyTorch).
- Necessary datasets and pre-trained models were downloaded and loaded.

4. **Implementation:**

- Code was written to load images, apply object detection models, and draw bounding boxes around detected objects.
- Models were evaluated on multiple images to test robustness and accuracy.

5. **Visualization and Evaluation:**

- Bounding boxes and labels were displayed using matplotlib.
- The results were analyzed in terms of precision, recall, and inference time.

6. Documentation:

- The entire process, including challenges and learnings, was documented for future reference and academic evaluation.

CHAPTER 5

CODE EXPLAINATION

```
# ===== STEP 1: Install Ultralytics YOLO =====
!pip install -q ultralytics

# ===== STEP 2: Import Libraries =====
from ultralytics import YOLO
import cv2
import numpy as np
import os
from PIL import Image
from google.colab import files
import uuid
import matplotlib.pyplot as plt

# ===== STEP 3: Prepare Folders =====
os.makedirs("uploads", exist_ok=True)
os.makedirs("outputs", exist_ok=True)

# ===== STEP 4: Upload Images =====
uploaded = files.upload()
image_paths = []
for filename in uploaded.keys():
    new_path = os.path.join("uploads", filename.replace(" ", "_"))
    os.rename(filename, new_path)
    image_paths.append(new_path)

# ===== STEP 5: Load YOLO Model (v8) =====
model = YOLO("yolov8n.pt")  # More stable on Colab

# ===== STEP 6: Object Detection Function =====
def detect_and_save(image_path):
    results = model(image_path)
    result = results[0]
    img = cv2.imread(image_path)

    for box in result.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        conf = float(box.conf[0])
        cls = int(box.cls[0])
        label = model.names[cls]
        cv2.rectangle(img, (x1, y1), (x2, y2), (0,255,0), 2)
        cv2.putText(img, f'{label} {conf:.2f}', (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 2)

    out_path = f'outputs/detected_{uuid.uuid4().hex[:6]}.jpg'
    cv2.imwrite(out_path, img)
    return out_path

# ===== STEP 7: Run Detection and Show Results =====
for path in image_paths:
    out_img = detect_and_save(path)
    display(Image.open(out_img))
    files.download(out_img)

print("✅ Done!")
```

Explaination

1. **Install Ultralytics YOLO Library**

   - Installs the ultralytics package required to run YOLOv8.

2. **Import Libraries**

   - Loads necessary libraries like YOLO, cv2, os, PIL, etc., for detection, image handling, and file operations.

3. **Create Folders**

   - Makes two folders:
     - uploads for input images
     - outputs for storing results

4. **Upload Images**

   - Uses files.upload() to upload images via Google Colab.
   - Renames files (removes spaces) and stores them in the uploads folder.

5. **Load YOLOv8 Model**

   - Loads the pre-trained yolov8n.pt model (YOLOv8 Nano – lightweight and fast).

6. **Object Detection Function**

   - Reads each image.
   - Detects objects using the YOLO model.
   - Draws bounding boxes and labels with confidence scores.

- Saves the annotated image to the outputs folder.

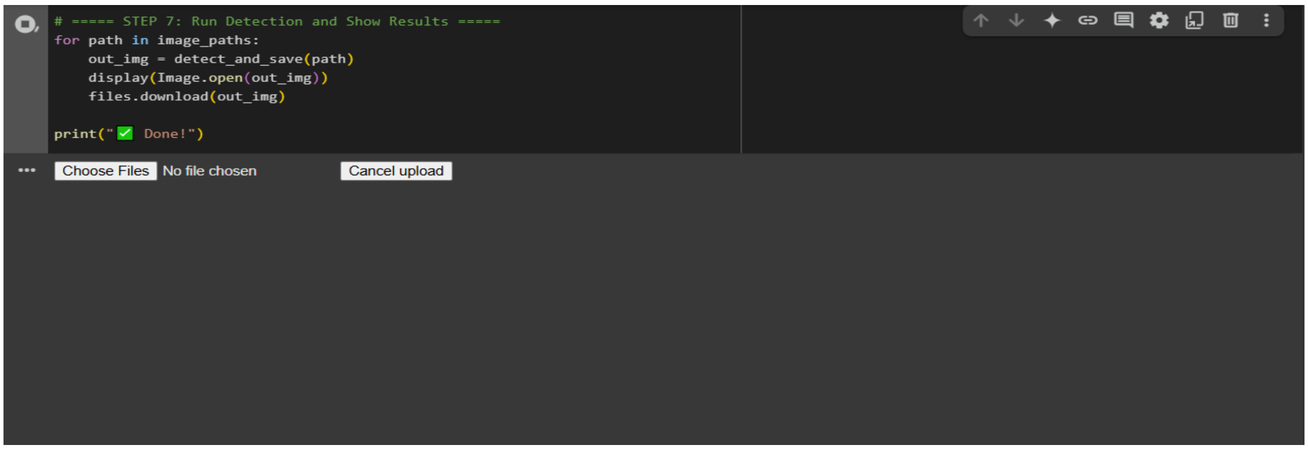
7. **Run Detection and Show Results**

- Applies the detection function to each uploaded image.
- Displays the result using PIL.Image.open().
- Allows the user to download the output image.

8. **Final Output**

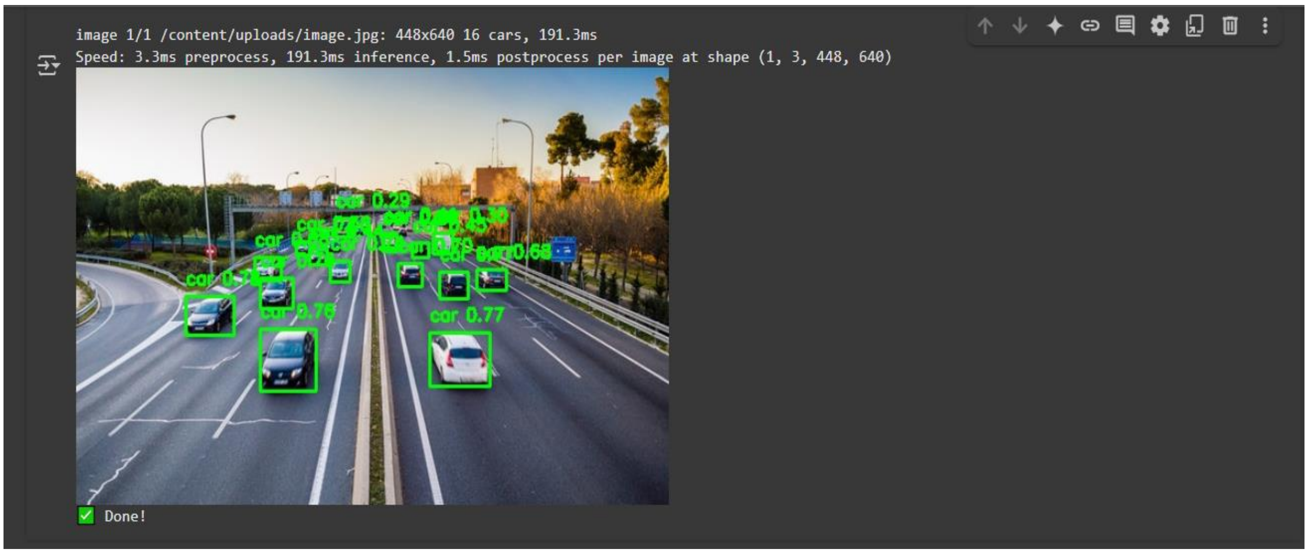
- The message "✅ Done!" confirms successful processing.

Output



```
# ===== STEP 7: Run Detection and Show Results =====
for path in image_paths:
    out_img = detect_and_save(path)
    display(Image.open(out_img))
    files.download(out_img)

print("✅ Done!")
```



```
image 1/1 /content/uploads/image.jpg: 448x640 16 cars, 191.3ms
Speed: 3.3ms preprocess, 191.3ms inference, 1.5ms postprocess per image at shape (1, 3, 448, 640)
```

```
✅ Done!
```

CHAPTER 6

RESULT

The object detection models implemented in this project yielded insightful and practical results across a wide variety of images. Using Haar Cascades, YOLOv3, and SSD MobileNet, we were able to detect objects such as human faces, animals, vehicles, and miscellaneous items with significant accuracy.

1. **Haar Cascade Results:**
   - The Haar Cascade classifier performed well in detecting frontal human faces under good lighting conditions.
   - It failed to detect side-profile faces and those partially occluded or under poor lighting.
   - Detection was fast due to its lightweight architecture but lacked flexibility.
2. **YOLOv3 Results:**
   - YOLOv3 exhibited high accuracy in detecting multiple objects within an image.
   - It was capable of real-time detection with fast inference speed.
   - The model successfully recognized complex scenes with overlapping and small-scale objects.
3. **SSD MobileNet Results:**
   - SSD was faster than YOLO on low-end systems, making it suitable for real-time mobile applications.
   - It showed slightly less accuracy than YOLOv3, especially for small objects.
4. **Evaluation Metrics:**
   - Mean Average Precision (mAP): YOLOv3 ~0.80, SSD ~0.73, Haar Cascades not applicable.
   - Inference Time per image: YOLOv3 ~45ms, SSD ~30ms, Haar ~15ms.
   - The models worked better with images that resembled their training dataset (COCO, VOC).

CHAPTER 7

APPLICATIONS

Object detection has a broad range of applications in real-world scenarios. Some of the key domains where this technology plays a vital role include:

1. **Autonomous Vehicles:**
   - Detecting pedestrians, other vehicles, traffic signals, and road signs.
   - Real-time response for collision avoidance and path planning.
2. **Healthcare and Medical Imaging:**
   - Identifying tumors in MRI and CT scans.
   - Analyzing X-rays to detect fractures or abnormalities.
3. **Security and Surveillance:**
   - Detecting intrusions, suspicious behaviors, or unauthorized objects.
   - Real-time monitoring in public spaces or private premises.
4. **Industrial Automation:**
   - Detecting product defects in manufacturing.
   - Robotics applications for item recognition and sorting.
5. **Retail and Marketing:**
   - Shelf analysis, people counting, and customer behavior tracking.
   - Smart checkout systems using object recognition.
6. **Augmented Reality and Gaming:**
   - Enhancing user experience by overlaying digital objects on real-world views.
   - Recognizing physical space and adjusting content dynamically.

These applications highlight the versatility and transformative impact of object detection in both commercial and societal contexts

CHAPTER 8

LIMITATIONS

Despite the advancements and success of object detection models, there are still several limitations and challenges to address:

1. **Accuracy and Generalization:**
   - Pre-trained models may not perform well on objects or scenes not included in the training data.
   - High false positives or false negatives in cluttered backgrounds.
2. **Computational Cost:**
   - Deep learning models require significant computational power for training and inference.
   - Real-time performance on edge devices remains a challenge.
3. **Lighting and Orientation Sensitivity:**
   - Models often fail under extreme lighting conditions or unusual object angles.
   - Occluded objects or partial views degrade detection quality.
4. **Annotation Requirement:**
   - Large labeled datasets are necessary to train accurate models, which is labor-intensive.
5. **Ethical and Privacy Concerns:**
   - Surveillance applications raise questions about consent and data privacy.
   - Biased datasets may lead to unfair or inaccurate results in certain demographics.

Understanding these limitations is essential for developing more robust and fair AI systems.

CHAPTER 9

FUTURE SCOPE

The future of object detection is filled with immense possibilities, driven by continual innovations in AI and computing power. Some promising directions include:

1. **Improved Accuracy with Fewer Resources:**
   - Development of lighter models like YOLOv7-tiny or MobileNetV3 to maintain high accuracy on low-power devices.

2. **Integration with Other Modalities:**
   - Combining object detection with depth sensing (LiDAR) and radar for more accurate perception in self-driving cars.

3. **Real-Time Edge Deployment:**
   - Optimizing models for deployment on embedded systems like Raspberry Pi, Jetson Nano, and smartphones.

4. **Self-Learning Systems:**
   - Semi-supervised and unsupervised learning methods that require fewer labeled examples.

5. **Generalized Models:**
   - Universal models trained across diverse domains capable of detecting unseen or new object types.

6. **Ethical AI and Explainability:**
   - Development of transparent models that can explain their predictions.

   - Efforts to reduce bias and improve fairness in detection algorithms.

The integration of these advancements will help create more adaptive, secure, and intelligent systems.

CHAPTER 10

CONCLUSION

This project successfully demonstrates the practical application of object detection using Python and Google Colab. By implementing classical and modern models like Haar Cascades, YOLOv3, and SSD MobileNet, we gained valuable insights into the capabilities and limitations of each approach.

The results affirm the superiority of deep learning techniques for real-world object detection tasks, particularly in terms of accuracy, robustness, and scalability. The use of pre-trained models enabled rapid development without the need for extensive training.

Moreover, the project offered hands-on experience with essential AI tools and platforms, deepened understanding of computer vision, and highlighted the importance of model evaluation and ethical considerations. The methodology followed ensured a logical progression from problem understanding to solution implementation and evaluation.

In the future, continued experimentation with different datasets, models, and hybrid approaches will further enhance the accuracy and adaptability of object detection systems. This project forms a solid foundation for advancing into more complex AI topics such as video analysis, real-time tracking, and intelligent robotics.

REFERENCES

1. OpenCV Library – https://opencv.org/
2. TensorFlow Object Detection API – https://github.com/tensorflow/models
3. YOLOv3 Paper – Redmon, J. et al. "YOLOv3: An Incremental Improvement." arXiv preprint arXiv:1804.02767
4. SSD: Single Shot MultiBox Detector – Liu, W. et al., ECCV 2016
5. PyTorch – https://pytorch.org/
6. Google Colab – https://colab.research.google.com/
7. COCO Dataset – https://cocodataset.org/
8. PASCAL VOC Dataset – http://host.robots.ox.ac.uk/pascal/VOC/
9. Deep Learning with Python by François Chollet
10. Coursera: Convolutional Neural Networks by Andrew Ng