

# Smart Traffic Management

## Team Members

- Jinwoo Bae, [jinwoo.bae@sjsu.edu](mailto:jinwoo.bae@sjsu.edu), 007330895
- Honghao Ma, [honghao.ma@sjsu.edu](mailto:honghao.ma@sjsu.edu), 010278125
- Itzel Xu, [itzel.xu@sjsu.edu](mailto:itzel.xu@sjsu.edu), 017549558

## 1. Proposed Application Area

Default project 1

## 2. The selected application, algorithm, model, and/or dataset

### Problem Statement

In modern cities, managing real-time traffic flow, relieving congestion and ensuring road safety have always been challenges that people try to solve. Traditional traffic monitoring systems often lack sufficient flexibility when facing changing real-world environments, making it difficult to effectively respond to complex situations such as diverse vehicle types, field of view obstructions, and lighting changes. In order to cope with the above problems, the aim of this project is to develop a deep learning-based intelligent traffic management system that can accurately recognize and classify all kinds of vehicles, pedestrians, traffic signs, and other objects in real-life scenarios, so as to provide efficient and reliable data support for urban traffic analysis and decision-making.

### Dataset: COCO Traffic-Related Subset

As the COCO 2017 dataset is too large, due to equipment and time constraints, we did not use the full COCO 2017 dataset, but instead extracted a subset of it, only extracting categories related to traffic and urban mobility. These categories include - 'People', 'Bicycles', 'Cars', 'Motorcycles', 'Buses' ", "Truck", "Traffic Light" and "Stop Sign". The subset consists of both training and validation parts and retains COCO's original annotations for bounding boxes and class labels. The dataset has high-quality annotations, a wide range of environmental diversity (weather, camera angles), and significant object diversity, making it suitable for evaluating the accuracy of models in real-world smart city environments.

### Selected Algorithms & Models

To experiment and compare performance and accuracy in the object detection task, we ran and compared three different models separately:

#### 1. YOLOv5s

- A single-stage real-time object detector, known for its balance of speed and accuracy.

- With strong mAP scores and inference efficiency, it is often used as a primary benchmark.
- Implemented with the official Ultralytics YOLOv5 PyTorch repo.
- Trained on coco\_subset for 5 epochs. 2.

## 2. Faster R-CNN (Region-based CNN)

- This is a two-stage detector with high accuracy in complex scenes.
- Implemented using ``torchvision.models.detection.fasterrcnn_resnet50_fpn``.
- Slightly slower, but provides better localization in cluttered scenes.
- Trained for 5 epochs.

## 3. SSD300 (single-shot detector)

- Lightweight model optimized for speed.
- Implemented with ``torchvision.models.detection.ssd300_vgg16``. 30 calendar hours of training. - Trained for 30 epochs.
- Despite the tuning, SSD still produced disappointing results (e.g., detecting everything as a “car”), and so it was used as a negative comparison result in our evaluation.

Each model was trained and evaluated using the same subset of COCO traffic to ensure a fair comparison. Inference is performed on real video clips, and the results are saved along with overlay bounding boxes and class predictions for visual inspection and metric-based evaluation.

## 3. system architecture

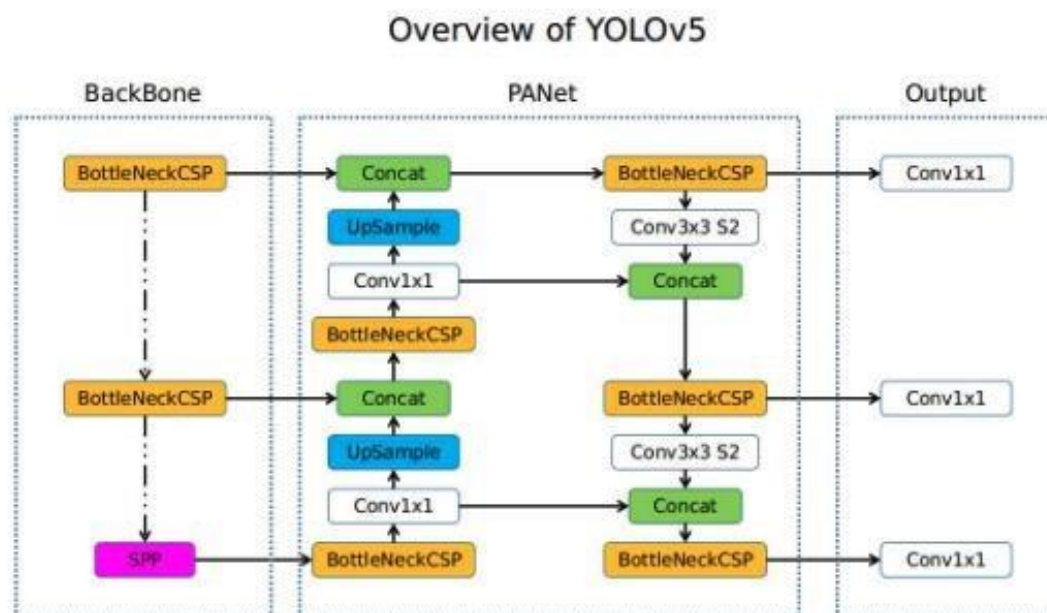


Diagram 1. Simplified YOLOv5 model architecture

## YOLOv5 System Architecture Summary

### 1. Backbone – CSPDarknet-53

This backbone network employs the Cross-Stage Partial Structure (CSP, or C3 module), which improves the learning efficiency and expressiveness by segmenting and fusing the feature map.

- Combining the convolution operation with residual concatenation, multi-scale features are captured efficiently.
- The generated feature representation has higher information content and lower redundancy.

### 2. Neck – SPPF + PANet

- SPPF (Fast Spatial Pyramid Pooling): Enhances the contextual information of the feature map through pooling operations with different sensory fields, while keeping the computational overhead low.
- PANet (Path Aggregation Network): Adopting top-down and bottom-up structures, it enhances the target localization ability through up-sampling and feature splicing fusion, which is especially suitable for detecting small and medium-sized targets.

### 3. Detection Head

- Targets are detected at three different scales (P3, P4, P5) and predictions are extracted using convolution and Detect layers.
- The output includes the class confidence and bounding box offset for each candidate box.
- Anchor boxes are pre-generated by K-means clustering to match the size distribution of different targets in the dataset.

### 4. Anchor boxes and post-processing

- Each anchor box supports the prediction of bounding box offsets and categories at different scales and aspect ratios.
- After detection, a non-maximum suppression (NMS) technique is used to remove redundant boxes and retain only those predictions with the highest confidence and with an overlap (IoU) below a set threshold (e.g., 0.45).

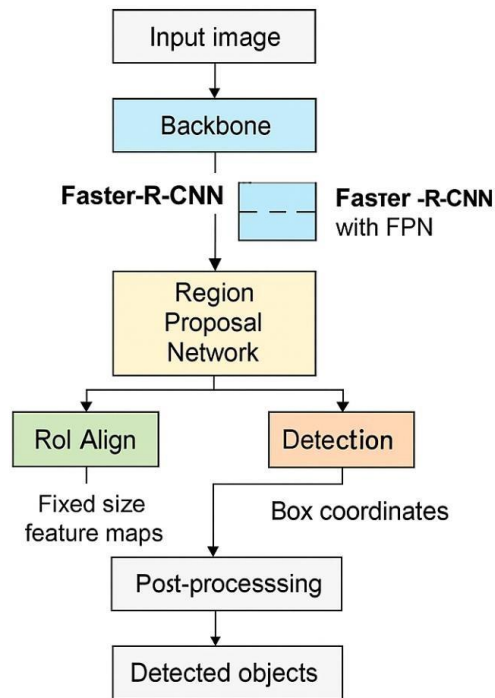


Diagram2. Faster R-CNN Architecture

## Faster R-CNN Architecture

1. Backbone - ResNet-50 with FPN (Feature Pyramid Network)
  - The basic CNN extracts deep visual features at multiple scales from the input image.
  - FPN creates a top-down pathway through lateral connections, combining low-level spatial details with high-level semantic information.
  - Output multi-scale feature maps for region suggestion.
2. Regional Proposal Network (RPN)
  - Apply lightweight networks on each feature map layer to generate candidate object regions (RoIs).
  - Generate a list of candidate object regions (RoIs) that can be used to
  - Predict objectivity scores and bounding boxes relative to anchor points.
  - Propose top N regions (e.g. 200-1000) for further classification and regression.
3. RoI Align + Detection Header
  - Each proposed region is cropped using RoI Align to ensure spatial alignment is accurate.
  - Each region is passed through a fully connected layer for final classification and bounding box regression.
  - Support for category-specific bounding box refinement (regression header by category).
4. Post-Processing
  - Apply Non-Maximum Suppression (NMS) by category to remove overlap detection.
  - Output the final object bounding box set with category labels and confidence scores.
  - This two-stage approach enables accurate object detection in cluttered and overlapping scenes, making Faster R-CNN ideal for high-precision tasks at the expense of inference speed.

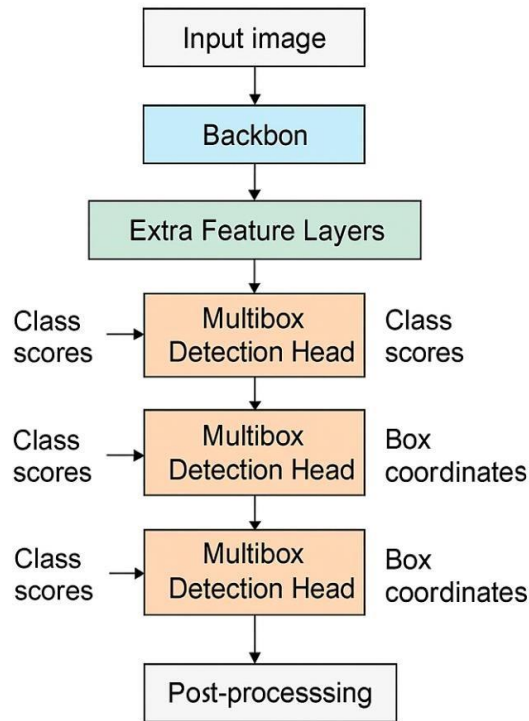


Diagram3. SSD300 System Architecture

### SSD300 System Architecture

#### 1. Backbone - VGG16

- SSD uses the standard VGG16 network as a feature extractor to extract a multi-level feature map from the input image.
- The base features consist of the first five convolutional blocks (conv1 to conv5) of VGG16, loaded with weights pre-trained on ImageNet to enhance the initial feature extraction capability.

#### 2. Extra Feature Layers

- After the base network, SSD adds a series of additional convolutional layers (e.g., conv6 to conv11) that progressively reduce the spatial resolution of the feature map.
- The multi-scale output structure allows the model to detect targets at different sizes, thus improving the flexibility of detection.
- Each layer outputs a fixed number of default boxes (anchor boxes).

#### 3. Multibox Detection Heads

- For each selected feature map, SSD predicts the following two categories of information:
- The class scores corresponding to each default box
- the offset of the bounding box (x, y, width, height)
- The whole process does not rely on candidate region proposal, and all predictions can be done in a single forward propagation with good real-time performance.

#### 4. Post-processing Module

- In the training phase, SSD uses Hard Negative Mining strategy to balance the proportion of positive and negative samples to improve the learning efficiency.
- The inference stage uses **Non-Maximum Suppression (NMS)** to remove duplicate predictions and retain the most plausible target frames.
- Although the SSD model has a lightweight structure and runs fast for real-time applications, its performance is relatively limited when dealing with small targets or occluded objects.

Despite the obvious advantage of SSD in real-time, its detection accuracy is somewhat affected when the data volume is small or the scene is complex, which limits its final performance in this project.

#### 4. Code Workflow & Techniques

We will explain and interpret the complete inference flow we implemented for all three models (YOLOv5, Faster R-CNN, SSD). The flow includes dataset preparation, frame preprocessing, model loading, inference, post-processing, and result rendering. Each model follows a similar workflow while utilizing model-specific application programming interfaces and architectures.

---

##### Key Techniques

- COCO subset extraction with traffic-related classes: person, car, bus, truck, traffic light, etc.
- Custom YAML for YOLOv5 and COCO JSON format for PyTorch models.
- Use of OpenCV for frame extraction, transformation, and visualization.
- Use of DetectMultiBackend (YOLOv5) and native PyTorch detection APIs (fasterrcnn\_resnet50\_fpn, ssd300\_vgg16).
- Confidence thresholding and Non-Maximum Suppression for result filtering.
- Bounding box rendering and annotated video export.

---

##### 1) Dataset Preparation (prepare\_coco\_subset.py)

We filtered the COCO 2017 dataset using a custom script that extracts only traffic-relevant categories and saves the results into a dedicated coco\_subset/ folder with both image and annotation files.

- Classes: person, bicycle, car, motorcycle, bus, truck, traffic light, stop sign.
  - Format: JSON annotation files for Faster R-CNN and SSD; converted YOLO .txt format for YOLOv5.
-

2) Preprocessing & Image Transformation Each video frame is processed as follows:

- YOLOv5: Resized to 640×640 with padding (value 114) to maintain aspect ratio and match model stride.
- Faster R-CNN: Converted to RGB and transformed into PyTorch tensors using `torchvision.transforms.ToTensor`.
- SSD: Resized to 300×300 for model compatibility; normalized as float32 tensors.

All images are processed on-the-fly from video streams using `cv2.VideoCapture`.

---

### 3) Model Loading

- YOLOv5 uses `DetectMultiBackend` for loading `best.pt` and inferring on images or videos.
- Faster R-CNN and SSD use `torchvision` models (`faster_rcnn_resnet50_fpn`, `ssd300_vgg16`), modified to fit 8 output classes (7 traffic + 1 background).
- All models are moved to GPU (cuda) when available.

---

### 4) Inference & Post-processing

- YOLOv5: Runs inference via `model(img_tensor) → predictions`, followed by Ultralytics' built-in NMS pipeline.
- Faster R-CNN & SSD: Output dictionaries with boxes, scores, labels fields.
- Custom thresholds (e.g., 0.25) are applied to remove low-confidence detections.
- PyTorch's `torchvision.ops.nms()` is used for post-processing to remove overlapping predictions.

---

### 5) Bounding Box Rendering & Video Output

All models use OpenCV to render detection results frame-by-frame:

- Bounding boxes are drawn using `cv2.rectangle`, and class labels with confidence scores are added using `cv2.putText`.
- For YOLOv5, Ultralytics' `Annotator` class is optionally used.
- Final annotated frames are written to output videos using `cv2.VideoWriter`.

Each model's video output is saved to its own folder:

- `runs/detect/smartcity_yolo5_test/` (YOLOv5)
- `fasterOutput/annotated_video.mp4` (Faster R-CNN)

- `ssdOutput/annotated_video.mp4` (SSD)

---

## 6) SSD as Negative Case

Despite extended training (30 epochs), the SSD model misclassified most objects as "car" and showed weak confidence scores. This underperformance highlighted the importance of model choice and robustness under data limitations. We retained SSD as a contrast case for result comparison and discussion.

## 5. Task distribution among group members

- Jinwoo Bae  
Identified the fundamental issue for intelligent traffic control and explained the realworld application setting. chosen and examined the UA-DETRAC dataset, paying particular attention to its environmental conditions, labeled classifications, and structure. assessed the dataset's suitability for object detection model training and testing by looking at important features such vehicle types, occlusion levels, truncation, lighting conditions, and object scales. summarized the results of the dataset interpretation and provided assistance for the evaluation discussion in the project report.
- Honghao Ma

### End-to-End Project Implementation & Engineering

Led the entire pipeline development and experimental workflow for the Smart City AI Object Detection project. This includes:

- ↗ Dataset Engineering: Designed and implemented a custom subset extraction process from the COCO dataset, selecting only traffic-relevant categories. Converted annotations into both COCO-style JSON and YOLO-format `.txt` files to support multiple model architectures.
- ↗ Model Implementation: Trained and evaluated three different object detection models—YOLOv5, Faster R-CNN, and SSD300—across a unified COCO-based dataset. Adapted and fine-tuned model heads to fit a custom 8-class setting, and performed hyperparameter tuning within computational constraints.
- ↗ Inference & Visualization Pipeline: Developed video inference pipelines for all models, including frame preprocessing, detection, post-processing (NMS), and OpenCV-based visual overlay. Saved output videos with bounding boxes for visual comparison and statistical analysis.



- ✈ Evaluation & Comparison: Collected performance results across models, analyzed class-wise detection behavior, and compared outcomes in terms of speed, accuracy, and robustness. Positioned SSD as a negative baseline for contrastive analysis.
- ✈ Project Infrastructure: Built and maintained the entire GitHub repository, organizing code, notebooks, and configuration files. Authored the complete project report and README, and designed the final presentation content to highlight key takeaways from the experimental comparisons.

**Final version:** <https://github.com/MarlonMa17/ObjectDetection/tree/main> ~~Old version:~~ <https://github.com/MarlonMa17/258Project>

- Itzel Xu

Literature & Model Research: Conducted an in-depth review of key object detection papers. Researched model architecture details and compared various YOLO versions and their practical implications in constrained edge deployment scenarios.

Dataset Evaluation: Validated the UA-DETRAC dataset, ensuring proper label mapping and quality checks for robust traffic detection.

Deployment & Debugging: Resolved configuration issues in both training and inference pipelines (e.g., module imports, device configuration, input tensor shape conversion, normalization, and augmentation) within the Kaggle environment.

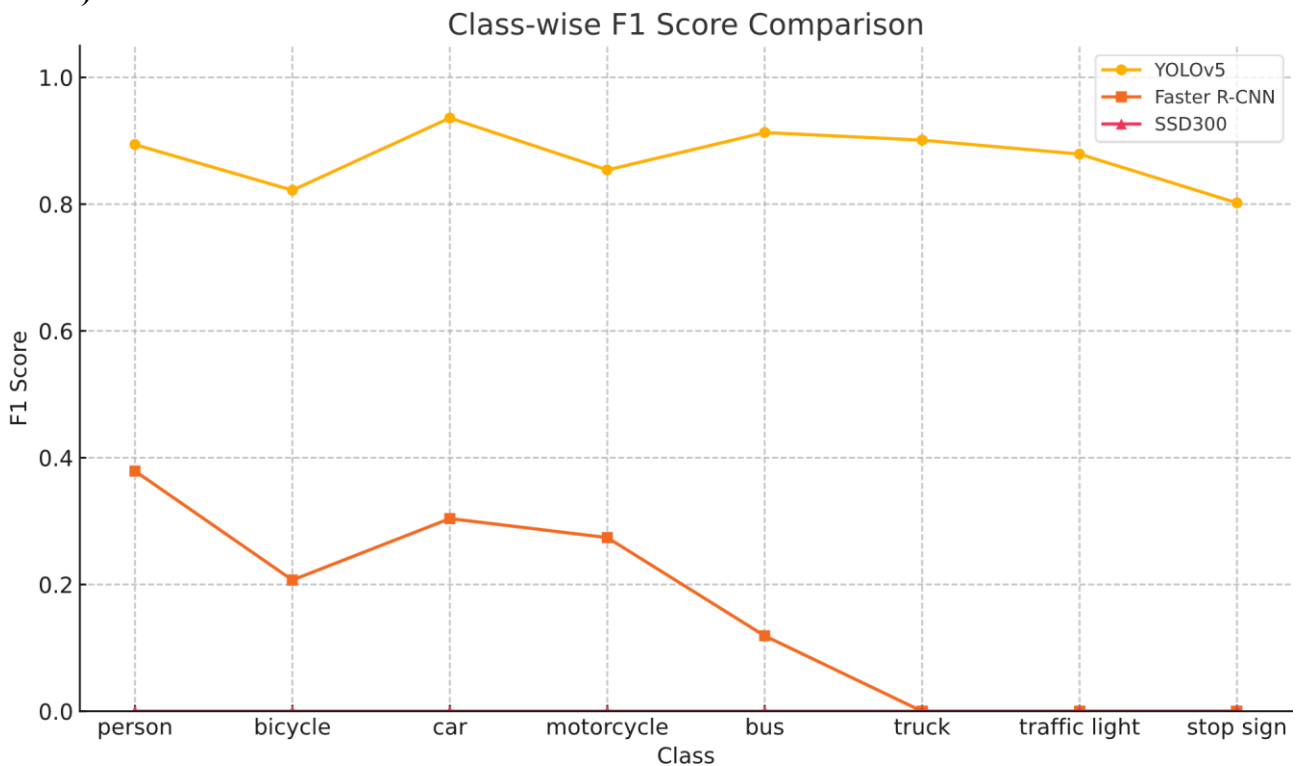
Collaborated with team members contributing to clear project documentation and presentation material.

## 6. Key references;

- [1] L. Wen *et al.*, “UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking,” *Computer Vision and Image Understanding*, vol. 193, p. 102907, Apr. 2020, doi: <https://doi.org/10.1016/j.cviu.2020.102907>.
- [2] <https://www.kaggle.com/datasets/dtrnngc/ua-detrac-dataset/code>
- [3] Khanam, R., & Hussain, M. (2024). What is YOLOv5: A deep look into the internal features of the popular object detector [Preprint]. arXiv. <https://arxiv.org/abs/2407.2089>
- [4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779–788). <https://doi.org/10.1109/CVPR.2016.91>
- [5] torchvision.ops.nms. (n.d.). In PyTorch Documentation. Retrieved May 11, 2025, from <https://pytorch.org/vision/stable/ops.html#torchvision.ops.nms>
- [6] <https://cocodataset.org/#download>
- [7] Girshick, Ross. “Py-Faster-Rcnn Has Been Deprecated. Please See Detectron, Which Includes an Implementation of Mask R-CNN.” *GitHub*, 2 May 2023, [github.com/rbgirshick/pyfaster-rcnn](https://github.com/rbgirshick/pyfaster-rcnn).
- [8] Singh, Rishabh. “Understanding and Implementing Faster R-CNN - Rishabh Singh - Medium.” *Medium*, 14 Oct. 2024, [medium.com/@RobuRishabh/understanding-andimplementing-faster-r-cnn-248f7b25ff96](https://medium.com/@RobuRishabh/understanding-andimplementing-faster-r-cnn-248f7b25ff96).

- [9] Vinodababu, Sagar. "Contents." *GitHub*, 4 May 2023, [github.com/sgrvinod/a-PyTorchTutorial-to-Object-Detection](https://github.com/sgrvinod/a-PyTorchTutorial-to-Object-Detection).
- [10] Hui, Jonathan. "SSD Object Detection: Single Shot MultiBox Detector for Real-Time Processing." *Medium*, 28 Dec. 2018, [jonathan-hui.medium.com/ssd-object-detection-singleshot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathanhui/ssd-object-detection-singleshot-multibox-detector-for-real-time-processing-9bd8deac0e06).

## 7. Evaluation and comparison results, including graphs (data graph must be in vector format)



Figma1. Class-wise F1 Score Comparison Across Three Models (YOLOv5, Faster R-CNN, SSD300)

The figure above compares the classification F1 scores of three different object detection models (YOLOv5, Faster R-CNN, and SSD300) on the coco\_subset validation set. The F1 score is a well-balanced measure that combines precision and recall, and is particularly suitable for evaluating the detection performance on unbalanced or noisy datasets.

### YOLOv5 Performance

YOLOv5 clearly outperforms the other two models across all eight traffic-related categories. It achieves F1 scores above 0.80 for every class, with the highest score of 0.936 for the "car" class, reflecting both high precision and recall. This model consistently detects person, bus, truck, and even smaller objects like traffic light and stop sign with strong confidence and minimal false

positives. Its PANet-based multi-scale design and k-means-derived anchors contribute to effective generalization across varying object sizes and aspect ratios. YOLOv5’s performance indicates excellent suitability for real-time, real-world smart city deployment.

**Faster R-CNN Performance**

Faster R-CNN shows moderate performance, with "person" (F1 = 0.379) and "car" (F1 = 0.304) being its strongest classes. This is attributed to its Region Proposal Network (RPN), which tends to favor larger or more visible objects. However, the model suffers from significantly lower precision, as it frequently produces redundant or overlapping predictions. It completely fails to detect rare or smaller classes like truck, traffic light, and stop sign, likely due to limited representation in the dataset and lack of scale-specific tuning. This suggests that while Faster RCNN is strong in theory, its default configuration struggles in small-scale or imbalanced training scenarios.

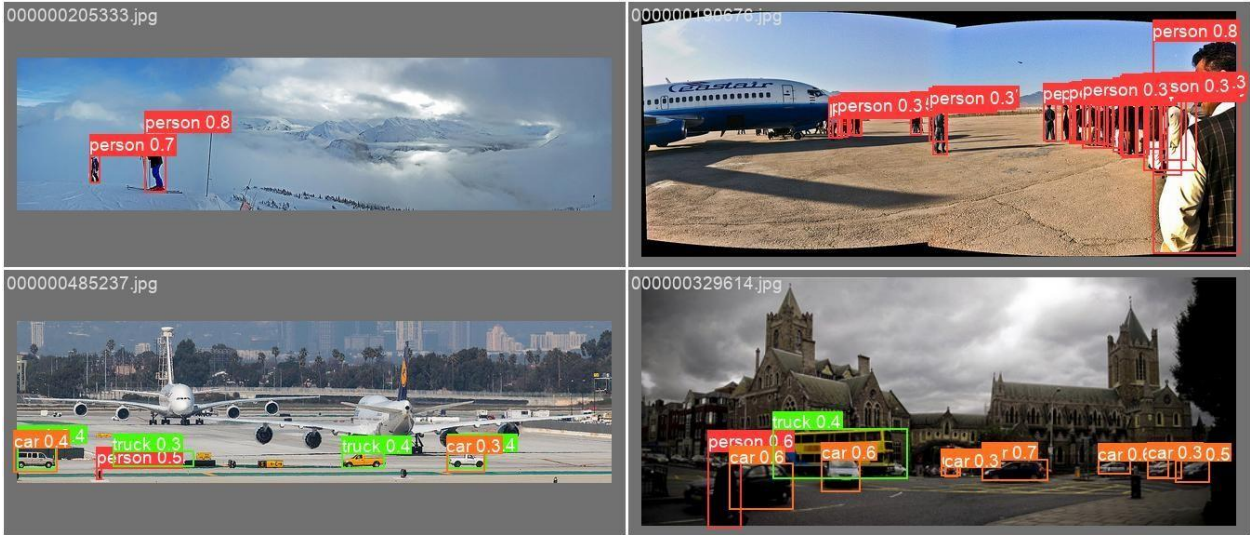
**SSD300 Performance**

The SSD model performs the worst among all three. Despite being trained for 30 epochs, it fails to detect any class with non-zero F1 scores. This aligns with previous qualitative observations, where SSD often misclassified every object as "car" or returned no predictions. SSD’s shallow feature hierarchy and rigid anchor configurations make it ill-suited for small datasets or those with varying object densities. It lacks the multi-scale fusion of YOLOv5 or the RPN adaptability of Faster R-CNN. In this experiment, SSD served as a useful negative baseline, highlighting the importance of architecture choice and data representativeness.

Model	Avg.F1	Strengths	Weaknesses	Verdict
YOLOv5	~0.88	Fast, highly accurate, robust	Needs GPU, sensitive to anchors	Best overall
Faster R-CNN	~0.25	High recall for large objects	Overlapping predictions, low precision	Medium performer

SSD300	~0.00	Lightweight	Poor accuracy and generalization	Unsuitable baseline
--------	-------	-------------	----------------------------------	---------------------

Table. Class-wise F1 Score Comparison Across Three Models (YOLOv5, Faster R-CNN, SSD300)



Figma2. YOLOv5 Detection Examples Across Diverse Urban and Non-Urban Scenes

This figure demonstrates YOLOv5’s robustness and adaptability across a wide range of environmental contexts, including outdoor mountain slopes, airport terminals, and city intersections.

Key observations include:

- Consistent person detection: In all four pictures, YOLOv5 detects individuals at varying scales and distances, even when partially occluded or positioned in crowds.
- Multi-class recognition: In addition to person, the model identifies car and truck in urban and semi-urban settings with moderate to high confidence scores.
- Complex environments: Despite challenging lighting (e.g., overexposed snow, overcast skies) and perspectives (e.g., panoramic views, dense crowd), the model maintains stable bounding box localization.
- Low-confidence thresholds: The presence of bounding boxes with confidence scores as low as 0.3 suggests that the model may produce some false positives. However, the

detections remain spatially relevant and interpretable, reflecting YOLOv5's aggressive recall optimization.

This visual result complements the F1 score analysis: YOLOv5 not only achieves high metrics on paper but also produces stable and visually consistent detections in complex, real-world environments.



Figma3. YOLOv5 Detection Example on video

This image illustrates the YOLOv5 model's detection performance on a real-world street scene. The model successfully detects multiple instances of person, car, and traffic light with high confidence scores (ranging from 0.79 to 0.88). Even small and partially occluded objects—such as distant vehicles and side traffic lights—are accurately identified.

Notably, the bounding boxes are tightly fitted to the object contours, and the class predictions are consistent with the scene semantics. For example:

- Two pedestrians crossing the street are correctly labeled as person with high confidence.
- A series of vehicles are detected at multiple distances and orientations.
- All visible traffic lights, even those in the background or partially occluded by poles, are localized and labeled.



This result highlights YOLOv5's superior ability to generalize across scale, distance, and occlusion. It affirms the earlier F1 score results, where YOLOv5 maintained scores above 0.80 for all eight traffic-related classes. Such robustness makes YOLOv5 particularly suitable for deployment in smart city monitoring systems where real-time accuracy is critical.



Figma4. Faster R-CNN Detection Output on Natural Scene with Foreground Objects

This image illustrates the detection output from the Faster R-CNN model on a cluttered natural background scene. The model successfully identifies a person and a traffic light with very high

confidence scores (1.00 and 0.99, respectively), demonstrating its strong capability to generate highly confident detections when objects are clear and centered.

Key observations:

- Bounding box precision is high. Both the person and traffic light are enclosed with tight bounding boxes, indicating that the Region Proposal Network (RPN) component of Faster R-CNN has accurately localized foreground regions.
- High confidence detection suggests that the model is well-trained to distinguish between key object features under standard lighting and centered compositions.
- Limitations not visible in this frame include the model's tendency to underperform in more crowded or occluded scenarios, as shown in earlier F1 score evaluations.

While this image highlights a strong case for Faster R-CNN's accurate predictions, it should be viewed in context with its overall lower recall and narrower generalization as seen across the full validation set. It excels when object scale, visibility, and contrast are favorable, but struggles with detection diversity and uncommon classes.



Figma5. Faster R-CNN Detection Example on video

This detection result shows how Faster R-CNN performs on a dense urban intersection with multiple vehicles, traffic signals, and pedestrians.



While the model successfully detects a wide range of classes—car, traffic light, person, bus, and even stop sign—its predictions expose key limitations:

- Overlapping bounding boxes: The same car is sometimes detected multiple times with different scores (e.g., car 0.98, car 0.93, car 0.99), leading to false positives and lower precision.
- Redundant labels and visual clutter: Many detections are crowded and redundant, reducing interpretability. This aligns with the F1 evaluation showing lower precision despite moderate recall.
- Confidence inconsistencies: Some traffic light predictions are made with high confidence (0.96), while others as low as 0.37—often for the same physical object.
- Partial object misses: Although multiple people are visible, only a few are labeled as person, indicating failures in dense or occluded regions.

This visual example validates the quantitative F1 results: Faster R-CNN excels in recognizing objects, but lacks the post-processing refinement and consistency of YOLOv5, especially in scenes with heavy object overlap.





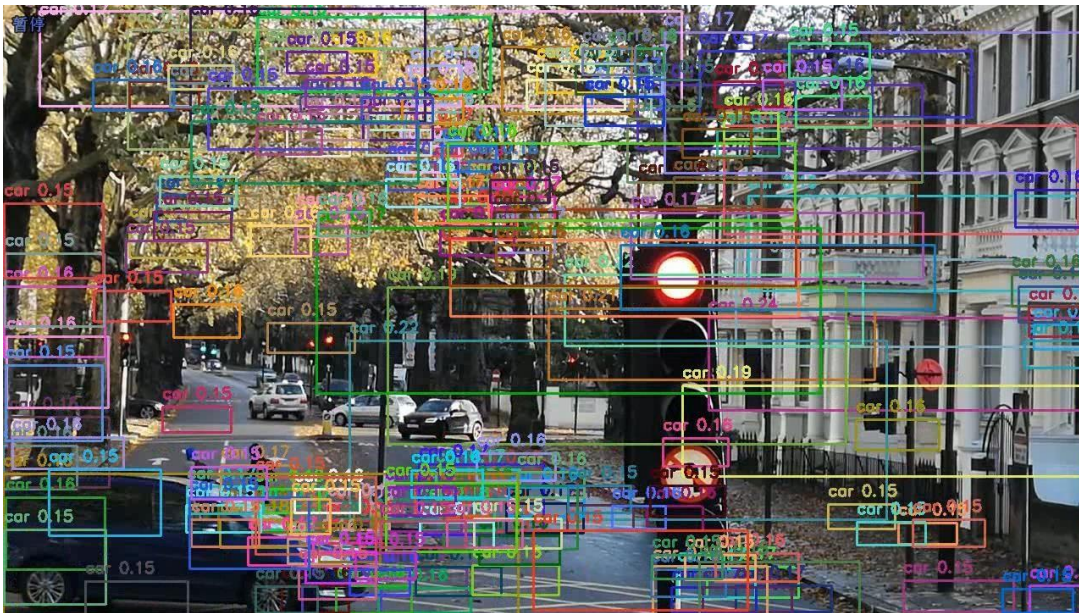
Figma6. SSD300 False Positives – Misclassification in Irrelevant Scene

This image demonstrates the severe generalization issues faced by the SSD300 model. Despite the scene being entirely unrelated to traffic or transportation—featuring a person standing in front of a political backdrop—the model predicts multiple instances of the class bicycle with confidence scores ranging from 0.26 to 0.28.

Key observations include:

- All predictions are false positives: No actual bicycle appears in the image, yet the model confidently predicts otherwise. This confirms the earlier F1 evaluation results, where SSD yielded a score of 0.0 for all categories.
- Random pattern sensitivity: The predictions are likely influenced by background shapes (e.g., the hammer and sickle logo), which the model confuses with object contours it vaguely learned.
- No object suppression or spatial context awareness: SSD lacks advanced mechanisms such as anchor refinement, spatial pyramid pooling, or feature fusion, making it fragile to visual noise and unfamiliar input contexts.
- Low training effectiveness: Despite 30 epochs of training, the model failed to learn robust class boundaries. This aligns with its tendency to predict “everything as a car” or “everything as something,” observed throughout both validation and video inference.

This failure case provides compelling visual evidence of SSD’s unsuitability for smart city-level object detection, reinforcing its role as a negative baseline in our comparative analysis.



Figma7. SSD300 False Positives – Detection Example on video

This figure shows a catastrophic failure case from the SSD300 model on a busy urban intersection frame. Nearly every spatial region is filled with overlapping bounding boxes, each labeled as car with extremely low confidence scores—most between 0.15 and 0.19.

Key insights:

- **Detection explosion:** The model generates dozens of overlapping boxes for the same region, failing to apply any form of Non-Maximum Suppression (NMS) or internal confidence pruning.
- **Single-class bias:** Almost all predictions are classified as car, indicating that the model has severely overfit to one dominant class during training and cannot distinguish between other objects.
- **Noisy low-confidence scores:** The highest score among predictions barely reaches 0.22, well below a usable threshold. This aligns with the earlier F1 analysis, where all classes scored 0.0.
- **Practical breakdown:** In real-world applications, such a detection result would not only be unusable but would actively corrupt downstream decision-making systems.

Together with the previous misclassification example, this result demonstrates that SSD300 fails both qualitatively and quantitatively, making it unsuitable even as a lightweight detector in smart city scenarios.

### ◆ Qualitative Comparison Summary

The final qualitative comparison across YOLOv5, Faster R-CNN, and SSD300 highlights critical differences in object detection reliability, interpretability, and real-world applicability.

YOLOv5 delivers clear and consistent bounding boxes, with high-confidence detections across diverse environments. It excels at multi-class detection in crowded urban scenes (Figure X+1, X+2), maintaining visual clarity and accurate spatial localization even with multiple overlapping objects. This aligns with its strong F1 scores (all > 0.80), confirming both model robustness and real-time detection suitability.

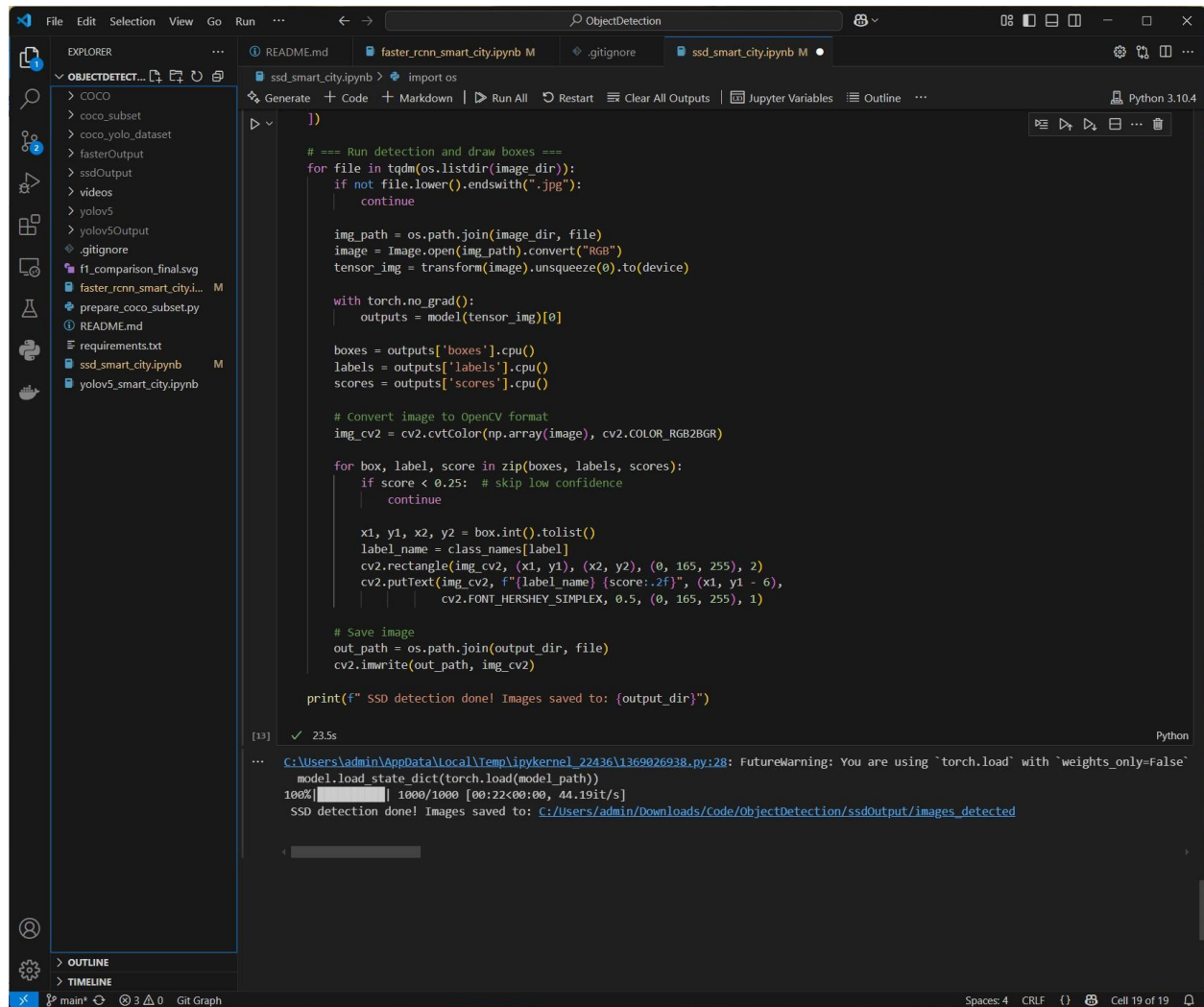
Faster R-CNN, while capable of localizing prominent objects like car and person, suffers from overlapping predictions, inconsistent label confidence, and frequent omission of minor classes such as stop sign or traffic light. Figures X+3 and X+4 reveal both its strength in simple scenes and its struggle in dense traffic scenes, where visual clutter and over-prediction reduce interpretability. The visual output reinforces its quantitative profile: good recall but lower precision.

SSD300 stands out for its failure to generalize. Despite extended training, it misclassifies unrelated content (Figure X+5) and overwhelms frames with unusable, low-confidence predictions (Figure X+6). These failures confirm the 0.0 F1 scores across all categories and position SSD as a failed baseline. Its lack of effective feature fusion, anchor optimization, and post-processing renders it unreliable for any safety-critical smart city application.

## ◆ Conclusion

In terms of quantitative metrics and visualization results, YOLOv5 shows optimal performance, with a good balance between accuracy, generalization ability, and clarity of detection results. Faster R-CNN has some interpretability in restricted scenarios, but overall efficiency is low, making it difficult to satisfy real-time demands. In contrast, despite its lightweight structure and fast inference speed, SSD performs relatively poorly in this project, with obvious flaws in both detection accuracy and visual output quality, making it difficult to perform the task of target detection in complex traffic scenes.

## 8. Optional: screenshots or photos



The screenshot shows a Jupyter Notebook titled 'ssd\_smart\_city.ipynb' in a VS Code editor. The notebook is running a Python script for SSD detection. The code in the cell is as follows:

```
import os

# === Run detection and draw boxes ===
for file in tqdm(os.listdir(image_dir)):
    if not file.lower().endswith(".jpg"):
        continue

    img_path = os.path.join(image_dir, file)
    image = Image.open(img_path).convert("RGB")
    tensor_img = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        outputs = model(tensor_img)[0]

    boxes = outputs['boxes'].cpu()
    labels = outputs['labels'].cpu()
    scores = outputs['scores'].cpu()

    # Convert image to OpenCV format
    img_cv2 = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

    for box, label, score in zip(boxes, labels, scores):
        if score < 0.25: # skip low confidence
            continue

        x1, y1, x2, y2 = box.int().tolist()
        label_name = class_names[label]
        cv2.rectangle(img_cv2, (x1, y1), (x2, y2), (0, 165, 255), 2)
        cv2.putText(img_cv2, f"{label_name} {score:.2f}", (x1, y1 - 6),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 165, 255), 1)

    # Save image
    out_path = os.path.join(output_dir, file)
    cv2.imwrite(out_path, img_cv2)

print(f" SSD detection done! Images saved to: {output_dir}")
```

The output of the cell shows the progress bar reaching 100% and the final message: " SSD detection done! Images saved to: C:/Users/admin/Downloads/Code/ObjectDetection/ssdOutput/images\_detected".

```
[13] ✓ 23.5s Python
... C:\Users\admin\AppData\Local\Temp\ipykernel_22436\1369026938.py:28: FutureWarning: You are using `torch.load` with `weights_only=False`
model.load_state_dict(torch.load(model_path))
100%|██████████| 1000/1000 [00:22<00:00, 44.19it/s]
 SSD detection done! Images saved to: C:/Users/admin/Downloads/Code/ObjectDetection/ssdOutput/images_detected
```

ObjectDetection

ssd\_smart\_city.ipynb M

EXPLORER

OBJECTDETECTION

> COCO

> coco\_subset

> coco\_yolo\_dataset

> fasterOutput

> ssdOutput

> videos

> yolov5

> yolov5Output

> .gitignore

f1\_comparison\_final.svg

faster\_rcnn\_smart\_city.i... M

prepare\_coco\_subset.py

README.md

requirements.txt

ssd\_smart\_city.ipynb M

yolov5\_smart\_city.ipynb

OUTLINE

TIMELINE

ssd\_smart\_city.ipynb

import os

fp = FP[cid]

fn = FN[cid]

precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0.0

f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) > 0 else 0.0

results.append((cat\_id\_to\_name[cid], precision, recall, f1))

# ===== 6. Display Results =====

import pandas as pd

df = pd.DataFrame(results, columns=["Class", "Precision", "Recall", "F1 Score"])

df.sort\_values("F1 Score", ascending=False, inplace=True)

from IPython.display import display

display(df)

# Optional: save to CSV

df.to\_csv("C:/Users/admin/Downloads/Code/ObjectDetection/ssdOutput/ssd\_f1\_scores.csv", index=False)

[12] ✓ 22.0s Python

C:\Users\admin\AppData\Local\Temp\ipykernel\_22436\362936410.py:21: FutureWarning: You are using "torch.load" with "weights\_only=False" (

model.load\_state\_dict(torch.load("C:/Users/admin/Downloads/Code/ObjectDetection/ssdOutput/ssd\_model.pth"))

loading annotations into memory...

Done (t=0.05s)

creating index...

index created!

100%|██████████| 1000/1000 [00:20<00:00, 48.77it/s]

	Class	Precision	Recall	F1 Score
0	person	0.0	0.0	0.0
1	bicycle	0.0	0.0	0.0
2	car	0.0	0.0	0.0
3	motorcycle	0.0	0.0	0.0
4	bus	0.0	0.0	0.0
5	truck	0.0	0.0	0.0
6	traffic light	0.0	0.0	0.0
7	stop sign	0.0	0.0	0.0

Step 8: Image Detection + Visualization Output Script

import os

import cv2

import torch

import numpy as np

from PIL import Image

from tqdm import tqdm

from torchvision import transforms

[13]

main\*

3

0

Git Graph

Spaces: 4

CRLF

{ }

Cell 19 of 19

File Edit Selection View Go Run ... ObjectDetection

EXPLORER

- OBJECTDETECTION
  - COCO
    - coco\_subset
    - coco\_yolo\_dataset
    - fasterOutput
    - ssdOutput
    - videos
    - yoloV5
    - yoloV5Output
  - .gitignore
  - f1\_comparison\_final.svg
  - faster\_rcnn\_smart\_city.ipynb M
  - prepare\_coco\_subset.py
  - README.md
  - requirements.txt
  - ssd\_smart\_city.ipynb M
  - yoloV5\_smart\_city.ipynb

fasterrcnn\_smart\_city.ipynb > import os

```
tp = TP[cid]
fp = FP[cid]
fn = FN[cid]
precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0.0
f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0
results.append((cat_id_to_name[cid], precision, recall, f1))

# ==== 6. Display Results ====
import pandas as pd
df = pd.DataFrame(results, columns=["Class", "Precision", "Recall", "F1 Score"])
df.sort_values("F1 Score", ascending=False, inplace=True)

from IPython.display import display
display(df)

# Optional: save for plotting
df.to_csv("C:/Users/admin/Downloads/Code/ObjectDetection/fasterOutput/faster_f1_scores.csv", index=False)
```

[1] ✓ 1m 7.3s Python

Downloading: "https://download.pytorch.org/models/resnet50-8676ba61.pth" to C:\Users\admin\.cache\torch\hub\checkpoints\resnet50-8676ba61.pth [97.8M/97.8M [00:00<00:00, 109MB/s]

C:\Users\admin\AppData\Local\Temp\ipykernel\_7360\2319180011.py:16: FutureWarning: You are using `torch.load` with `weights\_only=False` (which will abort in the future). To silence this warning, you can pass the argument `weights\_only=True` to `torch.load` to load the model safely. See the following issue for more details: https://github.com/pytorch/pytorch/issues/7551

model.load\_state\_dict(torch.load("C:/Users/admin/Downloads/Code/ObjectDetection/fasterOutput/fasterrcnn\_model.pth"))

loading annotations into memory...  
Done (t=0.06s)  
creating index...  
index created!

100%|██████████| 1000/1000 [01:03<00:00, 15.86it/s]

	Class	Precision	Recall	F1 Score
0	person	0.242690	0.864126	0.378951
2	car	0.186441	0.827200	0.304297
3	motorcycle	0.168605	0.725000	0.273585
1	bicycle	0.124736	0.614583	0.207381
4	bus	0.067174	0.519608	0.118967
5	truck	0.000000	0.000000	0.000000
6	traffic light	0.000000	0.000000	0.000000
7	stop sign	0.000000	0.000000	0.000000

Step 9: Image Detection + Visualization Output Script

```
import os
import cv2
import torch
```

[3]

main\* 3 0 Git Graph Spaces: 4 CRLF {} Cell 20 of 20



MarlonMa17/ObjectDetection

github.com/MarlonMa17/ObjectDetection#

one.SJSU | SCHOOL | 控制面板 | Professor | Inbox - honghao... | 网易邮箱6.0版 | QQ邮箱 | DeepL Translate: T... | Bank of America | 2117 | 所有书签

MarlonMa17 / ObjectDetection

Type to search

<> Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

ObjectDetection Public

Pin | Unwatch 1 | Fork 0 | Star 0

main | 1 Branch | 0 Tags

Go to file | Add file | Code

MarlonMa17 Add image detection and visualization script with evaluation metrics 8e87b82 · 9 minutes ago 26 Commits

videos	Update .gitignore and add video processing script fix output...	yesterday
.gitignore	Update project files: enhance README.md, modify .gitignore...	1 hour ago
README.md	Merge branch 'main' of https://github.com/MarlonMa17/Ob...	1 hour ago
f1_comparison_final.svg	Implement feature X to enhance user experience and fix bug...	17 minutes ago
faster_rcnn_smart_city.ipynb	Add image detection and visualization script with evaluation...	9 minutes ago
prepare_coco_subset.py	Implement code changes to enhance functionality and impr...	7 hours ago
requirements.txt	Update project files: enhance README.md, modify .gitignore...	1 hour ago
ssd_smart_city.ipynb	Refactor code structure for improved readability and maintai...	39 minutes ago
yolov5_smart_city.ipynb	Implement code changes to enhance functionality and impr...	7 hours ago

README

### Task distribution among group members

- Jinwoo Bae  
Identified the fundamental issue for intelligent traffic control and explained the real-world application setting. chosen and examined the UA-DETRAC dataset, paying particular attention to its environmental conditions, labeled classifications, and structure. assessed the dataset's suitability for object detection model training and testing by looking at important features such vehicle types, occlusion levels, truncation, lighting conditions, and object scales. summarized the results of the dataset interpretation and provided assistance for the evaluation discussion in the project report.
- Honghao Ma  
Conducted Final Project Presentation. Prepared slides and presentation. Prepared development and evaluation report and flow...

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Jupyter Notebook 100.0%

Suggested workflows

Based on your tech stack

Python package Configure

Create and test a Python package on multiple Python versions.

Pylint Configure

Lint a Python application with pylint.