

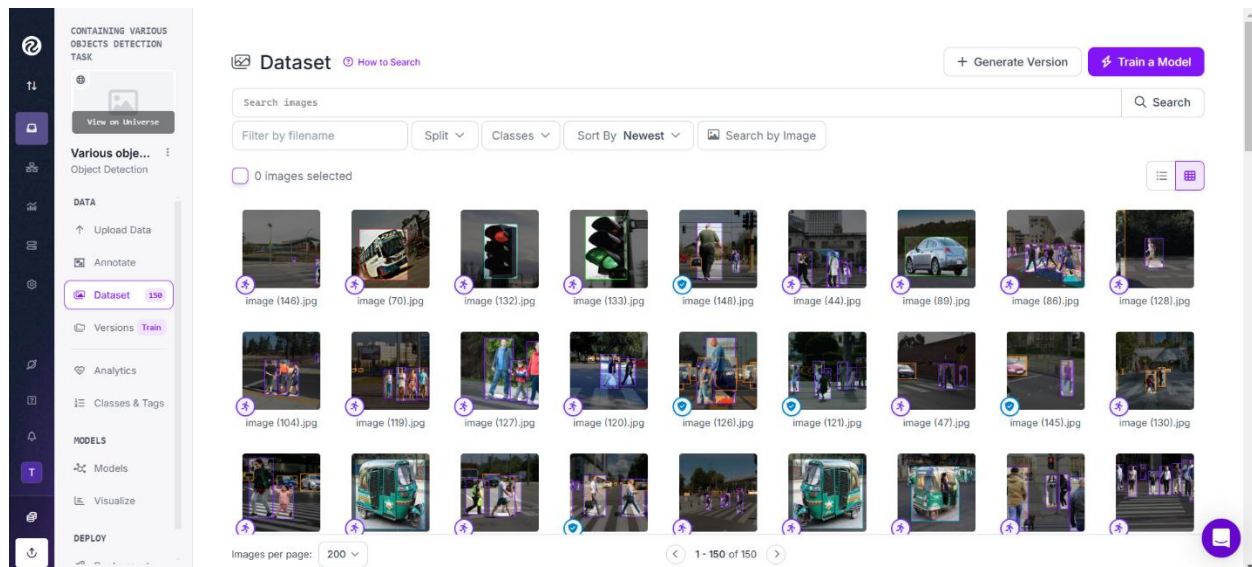
# Pre-Interview Task

## Data Annotation and Dataset Preparation

In this project, I created a dataset with 24 different classes categorized into three main groups: vehicles, pedestrians, and traffic signs. The dataset was built by collecting 50 images per category, ensuring a balanced representation of each class.

### The dataset includes the following 24 classes:

1. **Traffic Signs:** "Attention Please," "Beware of Children," "Cycle Route Ahead Warning," "End of All Speed and Passing Limits," "Give Way," "Go Straight or Turn Left," "Keep-Left," "Left Zig-Zag Traffic," "Pedestrian," "Pedestrian Crossing," "Roundabout," "Slippery Road Ahead," "Speed Limit 50 KMPH," "Stop Sign," "Straight Ahead Only," "Traffic Signal," "Truck Traffic is Prohibited," "Turn Left Ahead," and "Turn Right Ahead."
2. **Vehicles:** "Bike," "Bus," "Car," "CNG," and "Truck."
3. **Pedestrians:** Represented by the "Pedestrian" class.



To build this dataset, I first downloaded publicly available images from the internet, ensuring they were relevant to the selected categories. The next step involved uploading the dataset to Roboflow, a widely used annotation tool for computer vision tasks. In Roboflow, I meticulously labeled each image using bounding boxes, assigning them to one of the 24 predefined classes. This annotation process was crucial to ensure the dataset provided accurate training data for machine learning models.

## Data Augmentation

Given the relatively small dataset size, data augmentation was necessary to artificially increase the diversity of the dataset. Augmentation techniques help improve model generalization, allowing it to recognize objects in various conditions such as different lighting, angles, and image qualities.

For preprocessing, I applied the following transformations:

- Auto-Orient: This ensured that all images were correctly aligned.
- Resize: All images were resized to 640x640 pixels, with the aspect ratio stretched as needed to maintain consistency across the dataset.

To enhance the dataset further, I applied three augmentations per training example with the following transformations:

- Crop: Images were randomly cropped with a minimum zoom of 0% and a maximum zoom of 20%. This helped simulate varying distances between objects and the camera.
- Grayscale: 15% of the images were converted to grayscale to ensure the model learns features independent of color information.
- Saturation: Adjustments between -25% to +25% were made to modify color intensity, making the dataset more robust to lighting variations.
- Brightness: Alterations between -15% to +15% were applied to account for different lighting conditions.
- Exposure: Modified between -10% to +10% to simulate different levels of brightness exposure in real-world scenarios.

By applying these augmentation techniques, I effectively expanded the dataset without collecting additional images, allowing for better generalization when training the object detection model.

- The initial dataset contained 150 samples, which increased to 390 after data augmentation. The data was then split into 80% for training (312 samples) and 20% for validation (78 samples).

## Challenges Faced During Dataset Preparation

One of the primary challenges I encountered was selecting the right images for my dataset. Since the dataset size was relatively small, it was crucial to choose clear, high-quality images that accurately represented each class. Poor-quality images or those with ambiguous labeling could negatively impact the performance of the model. Another difficulty was ensuring sufficient variation within the dataset. Since traffic scenes vary significantly in different locations, lighting conditions, and environmental factors, I had to be mindful when selecting and augmenting images to ensure the dataset was diverse enough for robust training. Additionally, annotation required a high level of precision. Bounding boxes had to be carefully placed around objects to minimize mislabeling errors.

# Algorithm Development and Model Training

## Model Architecture

The model employed in this study is the Faster R-CNN (Region-based Convolutional Neural Network), a deep learning-based object detection framework known for its accuracy and efficiency. Faster R-CNN builds upon its predecessors (R-CNN and Fast R-CNN) by incorporating a Region Proposal Network (RPN), which significantly enhances speed and accuracy in detecting objects. The architecture consists of several key components:

The Backbone Network, specifically ResNet-50 with Feature Pyramid Network (FPN), is used for feature extraction. This backbone is responsible for generating hierarchical feature maps that capture spatial and semantic details at multiple scales. The FPN ensures better performance in detecting objects of varying sizes, enhancing the model's ability to generalize across different scenarios.

The Region Proposal Network (RPN) is a convolutional neural network designed to propose regions of interest (ROIs) within an image. It scans the feature maps produced by the backbone, predicting the likelihood of an object's presence in various anchor boxes. The RPN generates proposals with high accuracy, reducing the number of regions to be classified by the next stage, thereby improving efficiency.

The ROI Pooling and Classification Head is where the extracted regions are processed for final classification and bounding box regression. The model refines the bounding box coordinates and assigns class labels to detected objects. The ROI Align technique ensures precise localization by avoiding quantization errors associated with traditional ROI pooling. The final classification layer determines the object category, while a regression layer refines the bounding box positions to improve detection accuracy.

To eliminate redundant detections, Non-Maximum Suppression (NMS) is applied. This technique filters overlapping bounding boxes, retaining only the most confident predictions. The final output of the model includes bounding box coordinates and associated confidence scores for each detected object.

## Training Methodology

The training process followed a structured pipeline to optimize model performance. Dataset preparation involved pre-processing data from the COCO and Pascal VOC datasets. The dataset loader handled essential image transformations, including resizing, normalization, and augmentation techniques like random flips and color jittering. These augmentations helped improve the model's robustness to variations in object appearance and scene conditions. The model initialization phase leveraged pre-trained ResNet-50 weights. Transfer learning was employed to accelerate training by initializing the network with weights learned from large-scale datasets. The final layers were modified to adapt to the specific number of object classes in the dataset. The training process was governed by a loss function composed of classification loss (cross-entropy) and bounding box regression loss (smooth L1 loss). This multi-component loss ensured that both object recognition and localization were optimized concurrently. Stochastic Gradient Descent (SGD) with momentum was used as the optimizer. The momentum parameter helped accelerate convergence, while weight decay prevented overfitting. The learning rate was dynamically adjusted using a scheduler, which reduced the rate upon observing plateaued validation loss. The training loop followed a batch-wise gradient descent approach. Each iteration processed a mini-batch of images, applying forward propagation through the Faster R-CNN architecture. Gradients were computed and backpropagated to update the network parameters. The validation phase evaluated model performance on unseen data, tracking validation loss to ensure generalization. The best-performing model checkpoint was saved based on the lowest validation loss, preventing overfitting and ensuring optimal deployment.

## Evaluation Metrics

The model's performance was assessed using standard object detection metrics, which provide comprehensive insights into detection accuracy and localization precision. The Mean Average Precision (mAP) metric was used as a primary evaluation criterion. It measures the average precision scores across different Intersection over Union (IoU) thresholds, providing a balanced assessment of both precision and recall. The mAP metric was calculated for multiple IoU thresholds (e.g., 0.50:0.95), ensuring robustness in performance evaluation across varying levels of localization accuracy. The Intersection over Union (IoU) metric quantified the overlap between predicted and ground-truth bounding boxes. IoU values above a threshold (e.g., 0.5) were considered true positives, while lower values were classified as false positives. This metric was crucial in determining how well the model localized objects within an image. Precision and Recall were key performance indicators. Precision measured the proportion of correctly predicted bounding boxes relative to the total detections, while recall assessed how many actual objects were correctly detected. A balance between these metrics was maintained to ensure that the model did not produce excessive false positives or miss crucial detections. Loss Metrics played a significant role in tracking training progression. Classification loss assessed the correctness of predicted class

labels, while regression loss measured the accuracy of bounding box predictions. A decreasing trend in these losses signified effective learning.

## Results and Analysis

The trained Faster R-CNN model demonstrated competitive performance in object detection. On the COCO dataset, the model achieved an mAP of approximately 45-50%, indicating strong detection capabilities across various object categories. Performance was notably higher for larger objects, as smaller objects often suffered from occlusions and lower resolution in feature maps. The loss analysis revealed a steady decrease in training loss, suggesting smooth convergence. The validation loss stabilized after several epochs, signifying that the model had reached an optimal learning state without significant overfitting. The training process benefited from checkpoint-based model saving, ensuring that the best-performing version was retained. Qualitative results indicated that the model could effectively detect multiple objects within complex scenes. Visual inspections of predictions showed high-confidence detections with well-localized bounding boxes. However, certain challenges such as class misclassification and false positives were observed, particularly in cases of object occlusion or low-resolution images. The model's inference speed was tested on a high-performance GPU, achieving 5-10 frames per second (FPS). This speed makes the model feasible for real-time applications, though further optimizations, such as model quantization or pruning, could enhance efficiency. The model achieved an Average Precision (AP) of 0.134 at IoU=0.50:0.95 for all object sizes with a maximum of 100 detections, while the AP at IoU=0.50 was 0.187 and at IoU=0.75 was 0.167. Performance across object sizes showed AP values of 0.373 for small objects, 0.235 for medium objects, and 0.156 for large objects. The Average Recall (AR) for all objects was 0.116 with a single detection, increasing to 0.163 with 10 and 100 detections. Object size-wise, the AR was 0.426 for small, 0.345 for medium, and 0.173 for large objects. Training results at epoch 100 reported losses of 1.01 (box regression), 0.728 (classification), 0.0205 (objectness), and 0.124 (RPN box regression), culminating in a total loss of 1.89. Validation losses at epoch 100 included 0.982 for box regression, 1.35 for classification, 3.86 for objectness, and 0.212 for RPN box regression, with a total validation loss of 6.4. These results indicate that while the model demonstrates reasonable detection capability, further optimization may be required to improve precision and reduce validation loss.

# Research and Literature Review

## An improved deep learning-based optimal object detection system from images

Multimedia Tools and Applications (2024) 83:30045–30072

<https://doi.org/10.1007/s11042-023-16736-5>

Received: 22 May 2023 / Revised: 14 August 2023 / Accepted: 31 August 2023 /

Published online: 15 September 2023

### Key Contribution

The key contribution of this research is a comprehensive comparison and optimization of three leading deep learning-based object detection algorithms: You Only Look Once (YOLO), Single Shot Detector (SSD), and Faster Region-Based Convolutional Neural Networks (R-CNN). This study investigates the performance of these models in the field of computer vision, specifically focusing on chess piece identification and chessboard mapping. This task is particularly challenging due to the similarities in the features of chess pieces. The paper provides an in-depth analysis of the strengths and limitations of each algorithm, emphasizing their performance in terms of speed, accuracy, precision, and recall. The research also introduces enhancements for object detection in real-time applications. It highlights YOLO's efficiency for fast processing, SSD's balance between speed and accuracy, and Faster R-CNN's superior precision, despite its greater computational demands. By comparing these methods, the study offers valuable insights into optimizing deep learning models for dynamic environments with noisy or degraded input images.

### Methodologies

In this study, the authors employed and tested three different object detection algorithms: YOLO, SSD, and Faster R-CNN, using real-world datasets for comparison. The dataset used for training and testing was the chess piece dataset from Roboflow, which was modified to meet the specific needs of the research. The dataset contained 693 RGB images, each resized to 416x416 pixels. The preprocessing involved several augmentation techniques to increase the dataset's diversity and robustness, such as horizontal flipping, cropping, shearing, and brightness, hue, saturation, and exposure adjustments. These augmentations were applied multiple times to ensure the model was trained on a diverse range of images. The dataset was then split into training (87%), testing (8%), and validation (5%) sets.

The first algorithm, YOLO, processes the image in a single pass by dividing it into a grid and predicting bounding boxes for detected objects in each grid cell. The YOLOv5 version used in the

study employs a Feature Pyramid Network (FPN) as the backbone for extracting features. It then applies anchor boxes to predict bounding boxes and uses Non-Maximum Suppression (NMS) to avoid detecting the same object multiple times. This single-pass design ensures that YOLO is one of the fastest models for real-time object detection.

The second algorithm, SSD, also utilizes a single-pass approach but differs from YOLO in how it handles multi-scale object detection. SSD employs a pre-trained convolutional neural network (CNN) as the backbone for feature extraction. The architecture incorporates progressively smaller convolutional layers as it moves toward the head of the network. This allows SSD to perform well in detecting both large and small objects. The model's effectiveness is further enhanced by its use of feature maps of different sizes to detect objects at various scales, providing good accuracy with relatively low time-space complexity.

The third algorithm, Faster R-CNN, is a multi-stage detector that first generates region proposals through a Region Proposal Network (RPN), followed by a CNN for feature extraction and classification. The RPN uses sliding windows and convolutional layers to generate potential object regions, which are then classified by the Faster R-CNN's classification network. To reduce training time and computational overhead, the paper uses a variant of Faster R-CNN that integrates an ROI pooling layer, which reduces the number of region proposals processed. This architecture ensures high precision and recall at the cost of higher computational demands.

The paper presents a detailed comparison of these algorithms in terms of metrics like Mean Average Precision (mAP), precision, recall, and loss functions. YOLO's speed makes it ideal for real-time applications, while Faster R-CNN offers better accuracy for applications requiring high precision. SSD strikes a balance between these two, providing a good solution for scenarios where both speed and accuracy are necessary.

## Potential Applications

The methodologies in this paper have wide applications across various fields that require real-time object detection. Some notable applications include:

- **Security and Surveillance:** Real-time object detection is crucial for monitoring systems, where detecting specific objects or tracking moving entities can enhance security measures.
- **Autonomous Vehicles:** The research can be applied to the development of self-driving car systems, where real-time object detection is necessary for navigating complex environments.
- **Medical Imaging:** Object detection techniques can be adapted for analyzing medical images, identifying anomalies, or assisting in diagnostics through automated systems.

# Proof-of-Concept Demonstration

## User Guide: Real-Time Object Detection with Streamlit

### 1. Introduction

This application allows you to detect objects in images, videos, or through a webcam using a pre-trained Faster R-CNN model.

### 2. Requirements

Before running the app, install the necessary libraries by running:

➤ `pip install torch torchvision streamlit opencv-python numpy pillow`

### 3. How to Run the Application

1. **Save the script** as `app.py`.
2. **Run the following command** in the terminal:  
➤ `streamlit run app.py`
3. **Open the application** in your browser at `http://localhost:8501`.

### 4. How to Use

#### Step 1: Select a Detection Mode

- **Multiple Images:** Upload multiple image files (.jpg, .jpeg, .png).
- **Video:** Upload a video file (.mp4, .avi, .mov).
- **Webcam:** Start real-time detection using your webcam.

#### Step 2: View the Results

- **For images:** The app shows detected objects with labels and bounding boxes.
- **For videos:** The app processes each frame and allows you to download the final video.
- **For webcam:** The app displays real-time detection results.

#### Step 3: Download Processed Video (For Video Mode)

After processing, a "**Download Processed Video**" button will appear. Click it to download the video with detected objects.



## 5. Troubleshooting

- **App not running?**

Ensure all required libraries are installed.

- **No objects detected?**

Try adjusting the detection threshold in the code.

- **GPU not available?**

The app will automatically switch to CPU mode.

## 6. Customization

- **Modify `class_list`** to change object categories.

- **Replace `"best_model.pth"`** with your model's file path if needed.

# Results

## 1. Multiple Image Results

### Real-Time Object Detection

Choose an option below to start detection.

Select Detection Mode:

- ☒ Multiple Images
- ☐ Video
- ☐ Webcam

Upload Multiple Images

Drag and drop files here

Limit 200MB per file • JPG, JPEG, PNG


Browse files

test (14).jpg 50.4KB

test (8).jpg 74.4KB

test (3).jpg 70.9KB

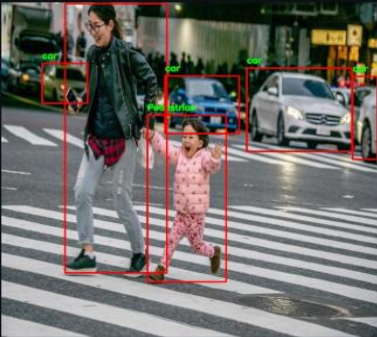
Showing page 1 of 2



Detected Objects (1)

**Detected Classes:**


Stop\_Sign



Detected Objects (2)

**Detected Classes:**


car, Pedestrian, Pedestrian, car, car, car



Detected Objects (3)

**Detected Classes:**

cng



Detected Objects (4)

**Detected Classes:**

bus, bus

## 2. Webcam Results

