Deep Learning with Keras and TensorFlow

Object Detection

Learning Objectives

By the end of this lesson, you will be able to:

- Implement and analyze the YOLOv3 object detection algorithm to effectively detect and localize objects in images and videos
- Apply dataset preparation techniques to prepare a dataset specifically for YOLOv3 to enhance the accuracy of DL models
- Estimate bounding boxes for multiple objects within an image to accurately determine their positions and dimensions
- Differentiate between two-stage and one-stage object detection modes to select the appropriate method for various applications

Business Scenario

ABC, a retail company, wants to enhance its in-store customer experience by implementing an object detection system.

The company aims to recognize the products that customers select and provide them with relevant information, such as product details and reviews.

To achieve this, they plan to use YOLO, a state-of-the-art algorithm for object detection. TensorFlow Lite, a cross-platform framework, will deploy the algorithm on the mobile devices of the store employees.

By implementing this system, ABC hopes to provide its customers with a personalized shopping experience and streamline in-store operations with real-time insights into customer behavior.

Introduction to Object Detection

Object Detection

Object detection is the process of identifying and locating specific objects in an image or a video.

It identifies objects in an image or a video using bounding boxes.

Autonomous vehicles, surveillance systems, and retail analytics use object detection among other applications.

Object Detection: Benefits and Uses

It enables precise object localization and is useful in autonomous navigation, surveillance systems, and augmented reality.

It facilitates real-time processing, making it valuable in video surveillance, live analytics, and interactive systems.

It enables simultaneous detection of multiple objects, useful for people counting, object tracking, and retail applications.

Object Detection: Example

Consider developing a self-driving car system that allows a computer to operate the vehicle independently

The system’s navigation can be developed by:

- Detecting the environment
- Categorizing every object around the car that it may encounter

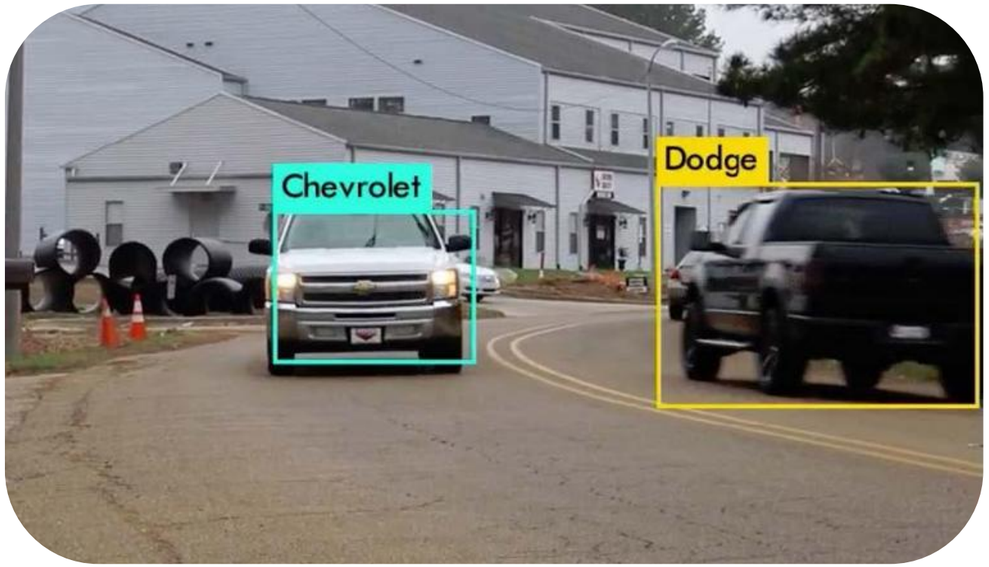
Object Detection: Example

To develop the system's navigation, use the Google Open Images Dataset V6 and consider the following classes:

- Car
- Person
- Building
- Tree

Object Detection: Example

Consider the following image with two cars:

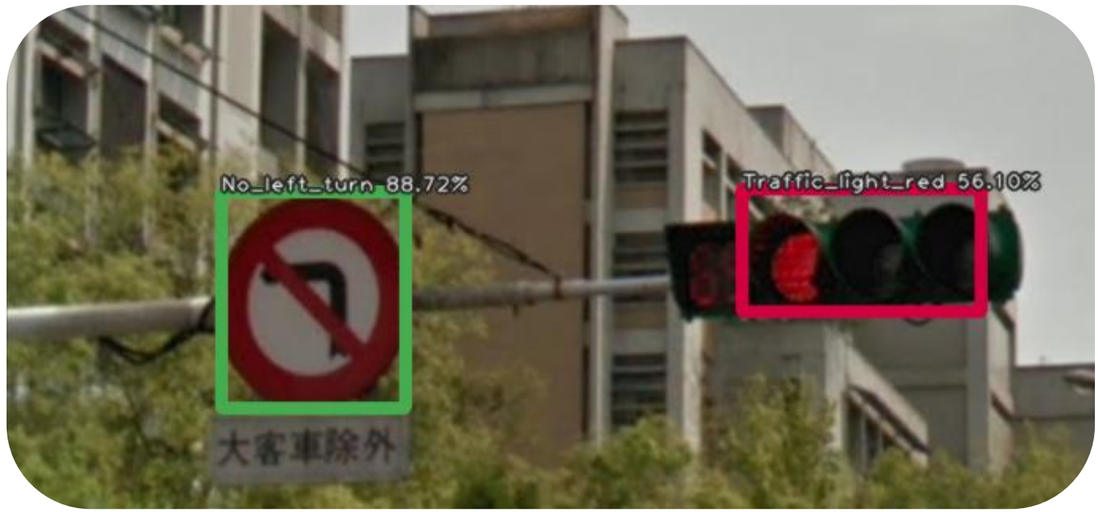


The system can be further developed to understand the sizes and types of cars and determine if they are stationary or moving.

Image Source: https://storage.googleapis.com/openimages/web/index.html)

Object Detection: Example

Consider the following image, which shows a traffic signal and traffic sign.



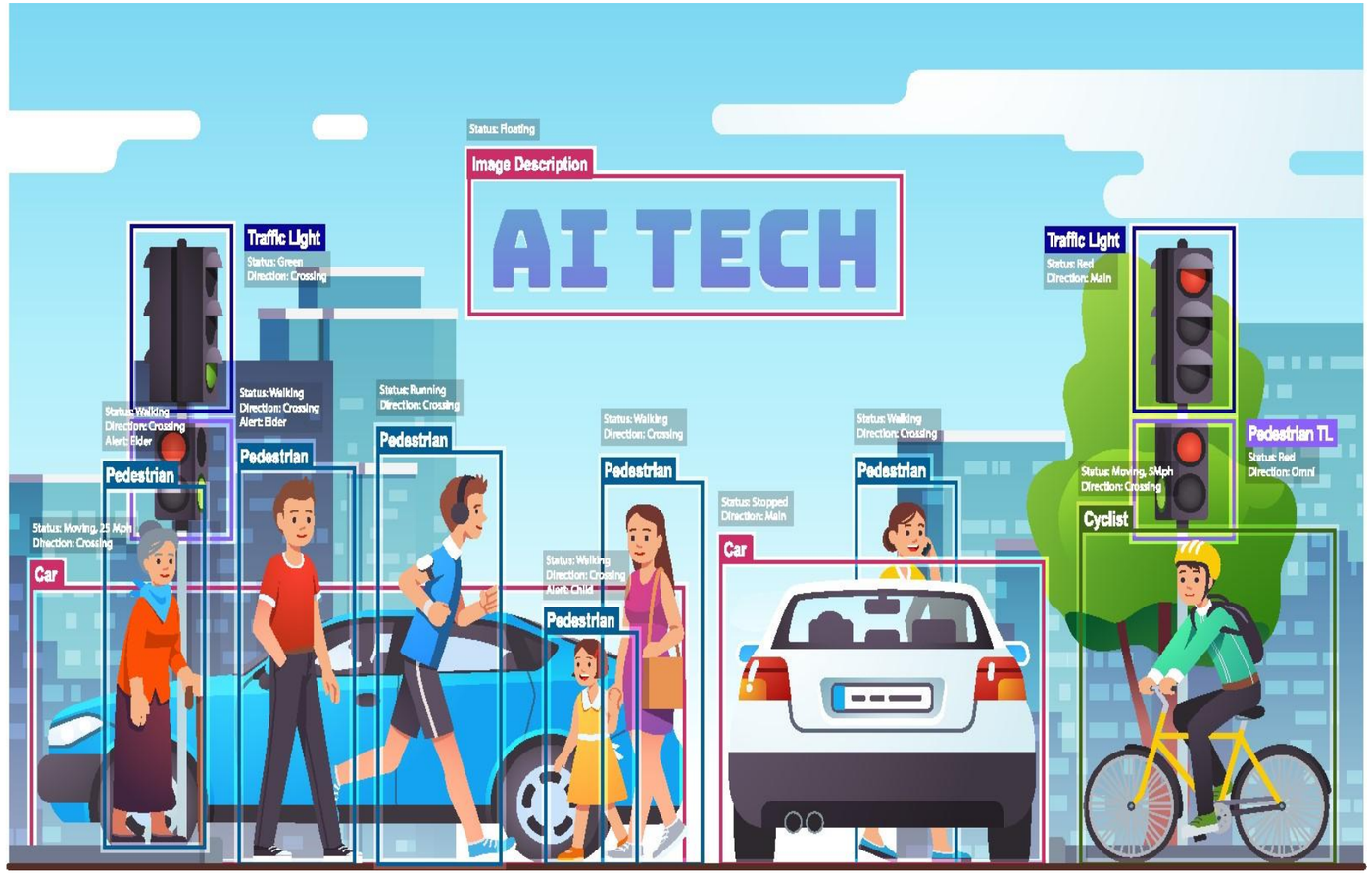
The system helps understand what lies ahead and take action based on the traffic lights.

Image source: https://storage.googleapis.com/openimages/web/index.html)

Object Detection Techniques in Computer Vision

Object Detection in Computer Vision

Object detection in computer vision involves the following two steps:

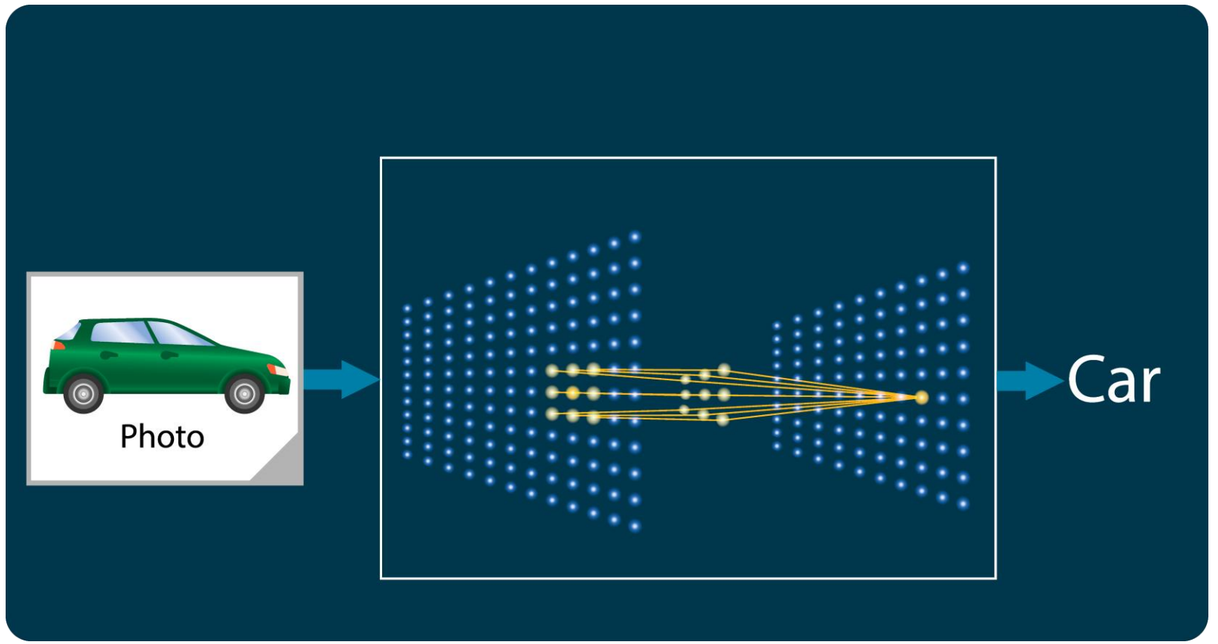


Object classification

Object localization

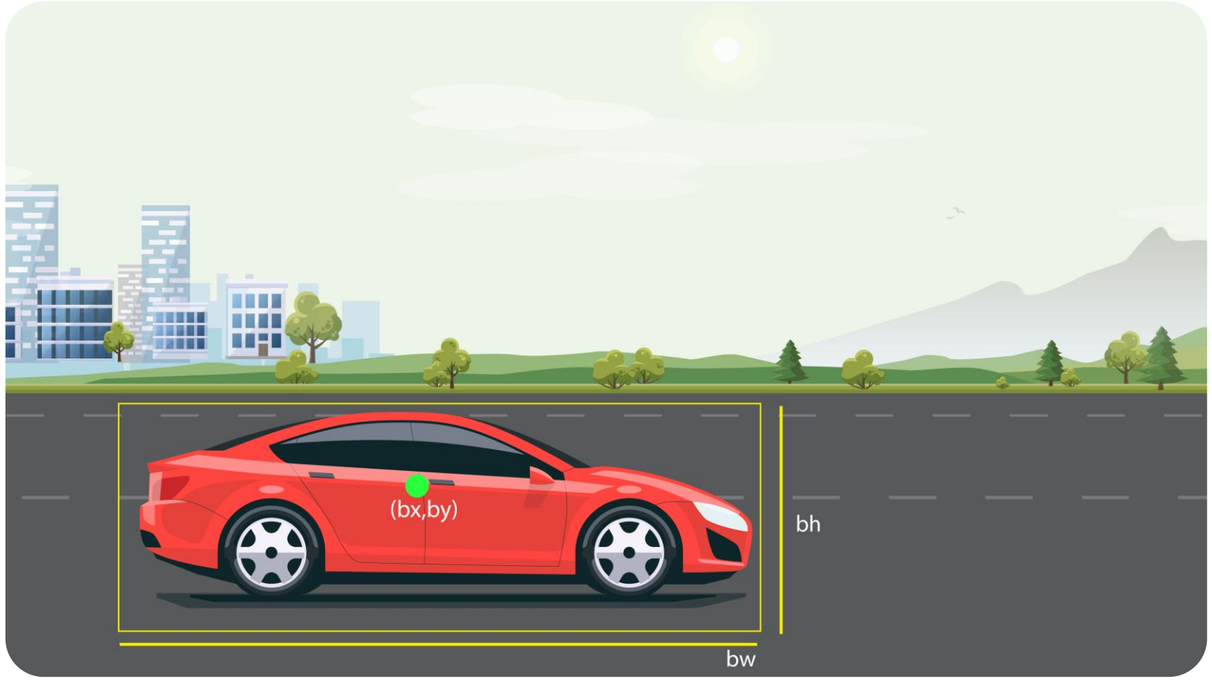
Object Classification

The algorithm identifies the detected object's class and location based on its specified features.



Object Localization

The algorithm predicts an object's boundaries and exact location in an image.



bx, by are the coordinates of the center of the bounding box.

bw is the width of the bounding box with respect to the image width.

bh is the height of the bounding box with respect to the image height.

Object Detection: Modes

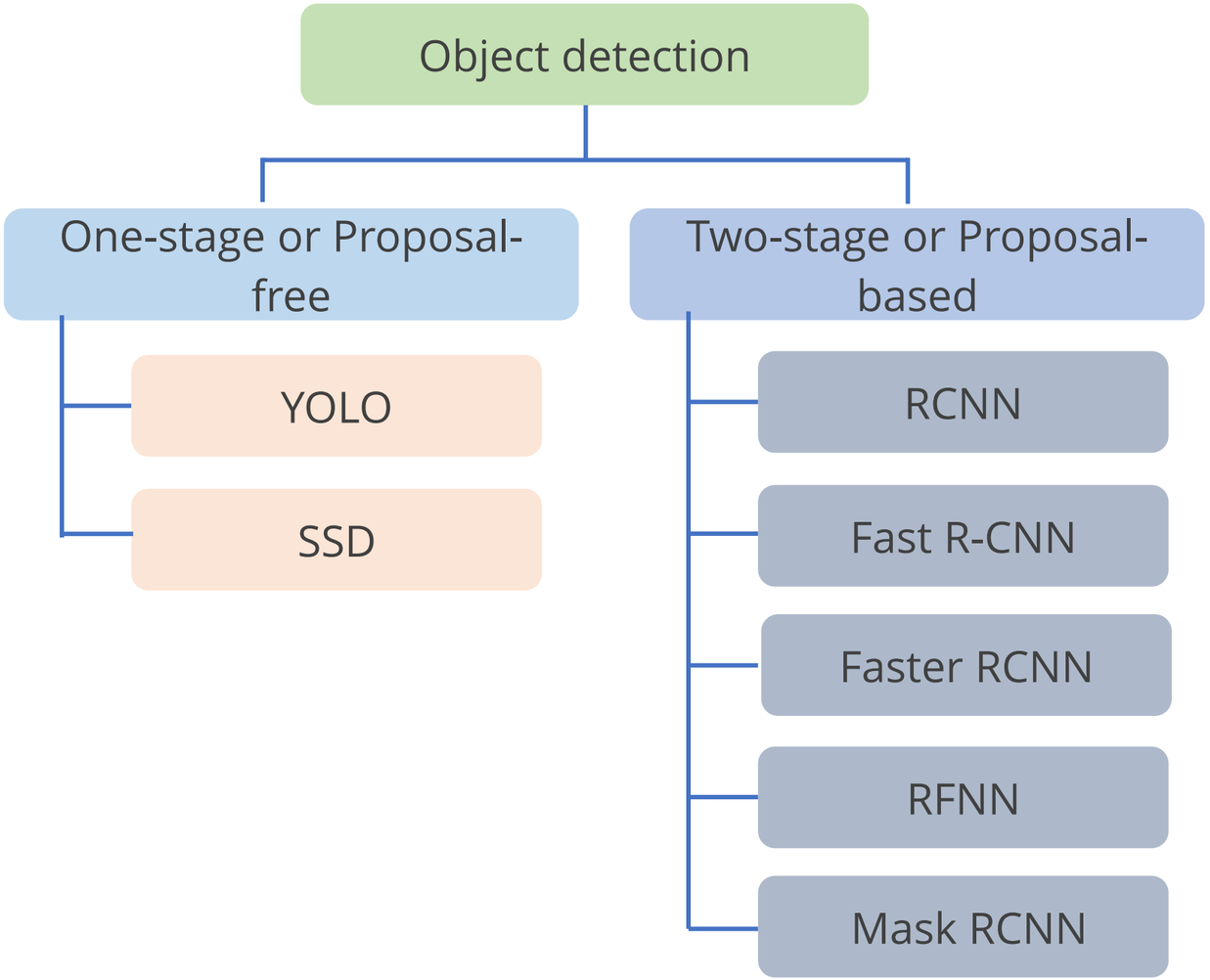
The two modes of object detection are:

One-stage or Proposal-free → This mode simultaneously predicts the class and bounding box of objects in an image without needing an initial proposal stage.

Two-stage or Proposal-based → This mode first generates region proposals where objects might exist and then classifies those regions in a separate step.

Object Detection: Modes

Classifying object detection modes:



Object Detection Modes: One-Stage or Proposal-Free

YOLO and SSD are the real-time object detection algorithms.

YOLO	Is a fast and efficient algorithm that predicts bounding boxes and class probabilities directly, making it ideal for video analysis and robotics
SSD	Enhances object detection accuracy by using default bounding boxes of varied scales, which are particularly beneficial for small objects

Their effectiveness and efficiency make them popular choices for real-time object detection tasks.

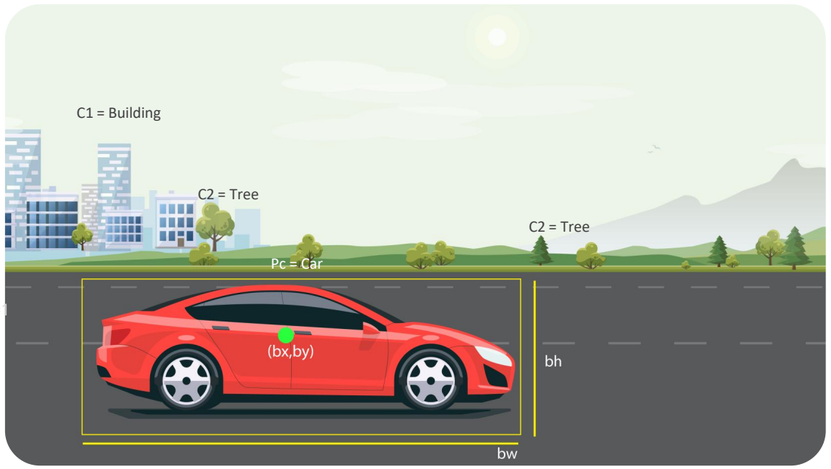
Object Detection Modes: Two-Stage or Proposal-Based

R-CNN	It uses selective search for region extraction and applies CNNs for accurate classification and bounding box refinement.
Fast R-CNN	It improves upon R-CNN by sharing convolutional features and using RoI pooling for efficient region proposal processing.
Faster R-CNN	It uses a Region Proposal Network (RPN) to quickly identify potential object regions in images.
RFPNN	It introduces a Region Proposal Network (RPN) for efficient region proposal generation, enhancing training and computation.
Mask R-CNN	It enhances Faster R-CNN with pixel-level segmentation, enabling instance segmentation within the R-CNN family.

Object Detection for Multiple Objects

Object Detection for Multiple Objects

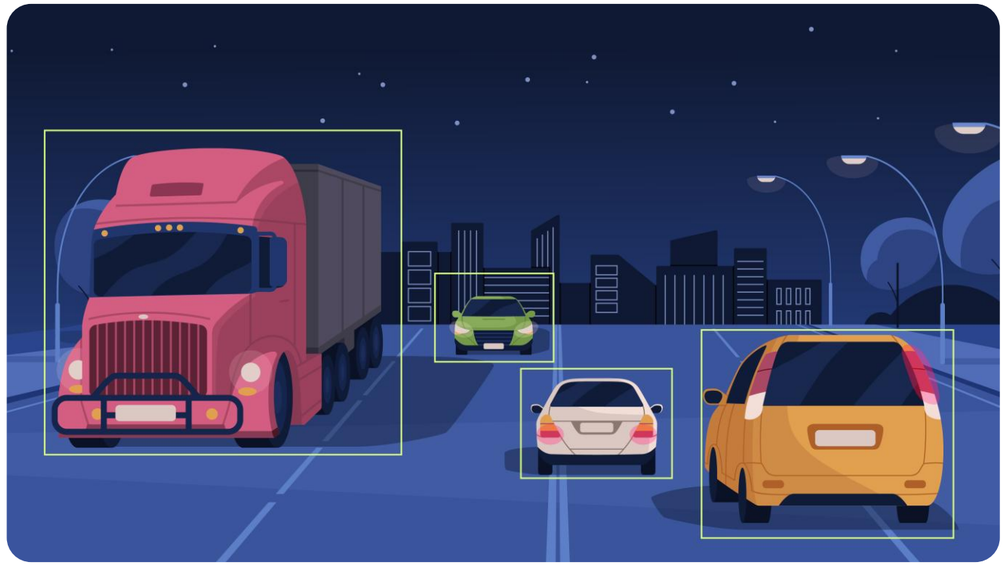
For any given image, an object detection algorithm will generally produce a vector.



Pc	→	Confidence in object presence within the bounding box
bx	→	x coordinate of the center of the bounding box
by	→	y coordinate of the center of the bounding box
bw	→	Width of the bounding box
bh	→	Height of the bounding box
C_1... Cn	→	Probabilities for each class representing the model's confidence in object classification

Object Detection for Multiple Objects

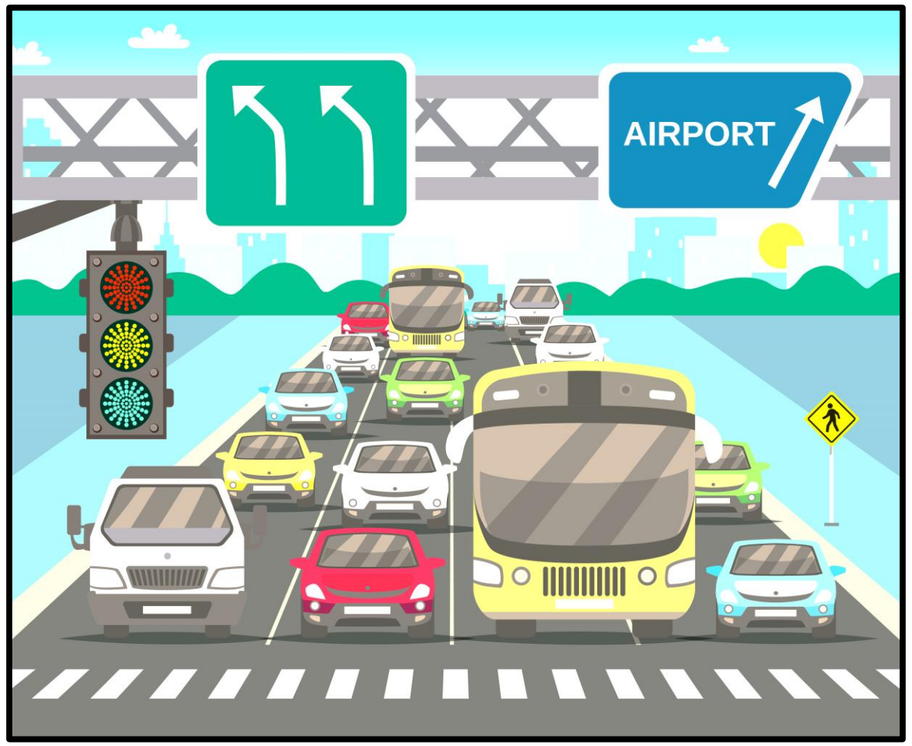
Detection of a single object in an image involves a sophisticated process.



- An object detection algorithm divides the image into a grid and predicts bounding boxes with confidence scores for each grid cell.
- The algorithm determines the presence of an object and calculates the precise coordinates, width, and height of the detected object's bounding box.
- Additionally, it predicts the probability of an object's presence within the predicted bounding boxes.

Estimating the Bounding Boxes

Consider the following image with multiple objects in traffic.



Creating advanced systems like self-driving cars can be quite difficult considering it has to detect multiple objects before taking a decision.

Estimating the Bounding Boxes

Steps for estimating bounding boxes for multiple objects:

Step 1:
Grid division

Step 2:
Object search and probability calculation

Step 4:
Neural network optimization

Step 3:
Vector generation

Estimating the Bounding Boxes

Steps for estimating bounding boxes for multiple objects:

Step 1:
Grid division

The image is partitioned into a 4x4 grid, creating equally sized small regions within the image.

Step 2:
Object search and probability calculation

The algorithm systematically analyzes each grid box to detect objects in the image, calculating probabilities for the presence of an object in each region.

Estimating the Bounding Boxes

Steps for estimating bounding boxes for multiple objects:

Step 3:
Vector generation

Each grid cell generates a vector of values, containing probability scores, class labels, confidence scores, and other relevant object information.

Step 4:
Neural network optimization

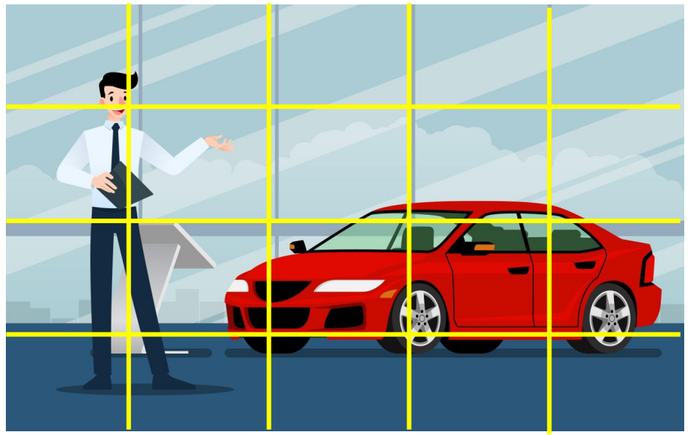
A neural network enhances the bounding box estimation accuracy by optimizing the refinement process based on the generated vectors.

Estimating Bounding Boxes: Example

Consider the following image

Estimating Bounding Boxes: Example

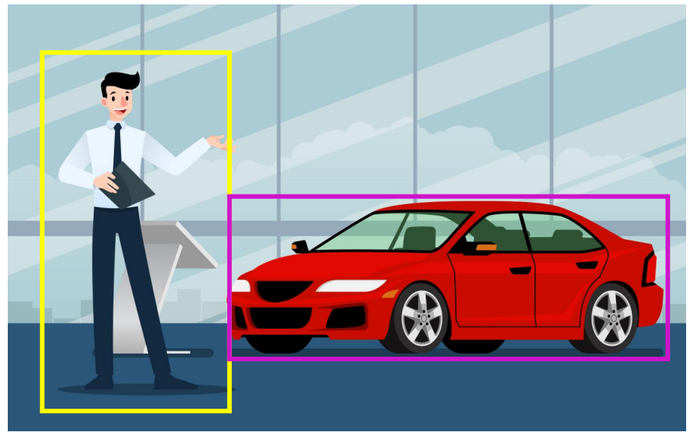
Object detection uses grid-based division and deep learning models, such as CNNs, to generate feature vectors for predicting bounding boxes, object scores, and class probabilities.



Input image

Extract vector values

Iterate



Locate objects

Through iterative training, accuracy is enhanced, with methodologies varying based on the chosen object detection architecture.

Estimating Bounding Boxes: Example

CNN is crucial in detecting objects, identifying object locations, and estimating bounding boxes accurately.

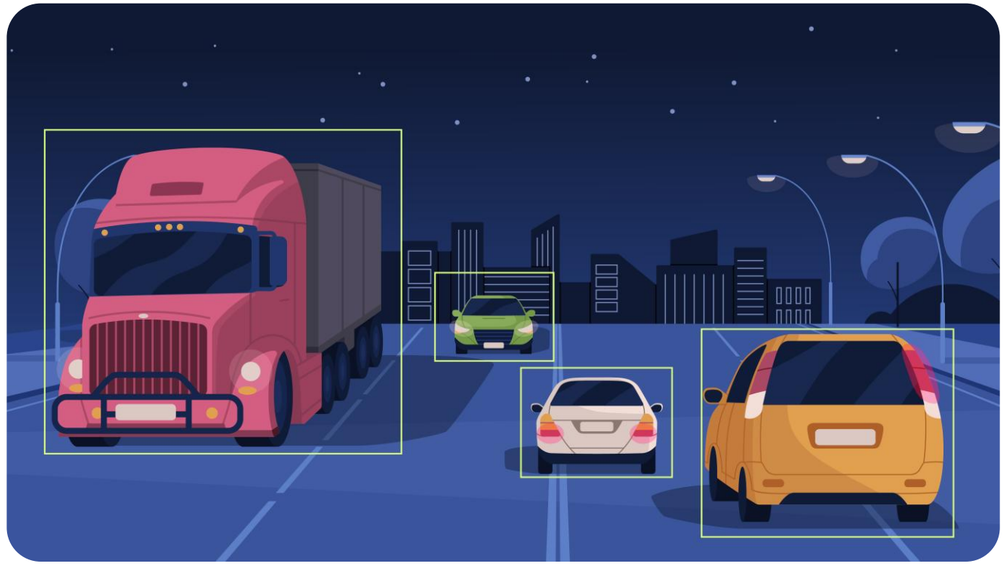
- By analyzing the input image, CNN effectively detects objects using powerful feature extraction capabilities.
- The image is represented as a 3D tensor (4x4x7), divided into a grid to capture information across spatial regions.
- The tensor is divided into a 4x4 grid, with each grid cell containing a vector of length 7.
- Each vector (length 7) represents features like color, texture, or object characteristics.

This allows accurate predictions about objects present in the image.

Challenges in Object Detection

Issues in Multiple Object Detection

The same object may have several bounding boxes.

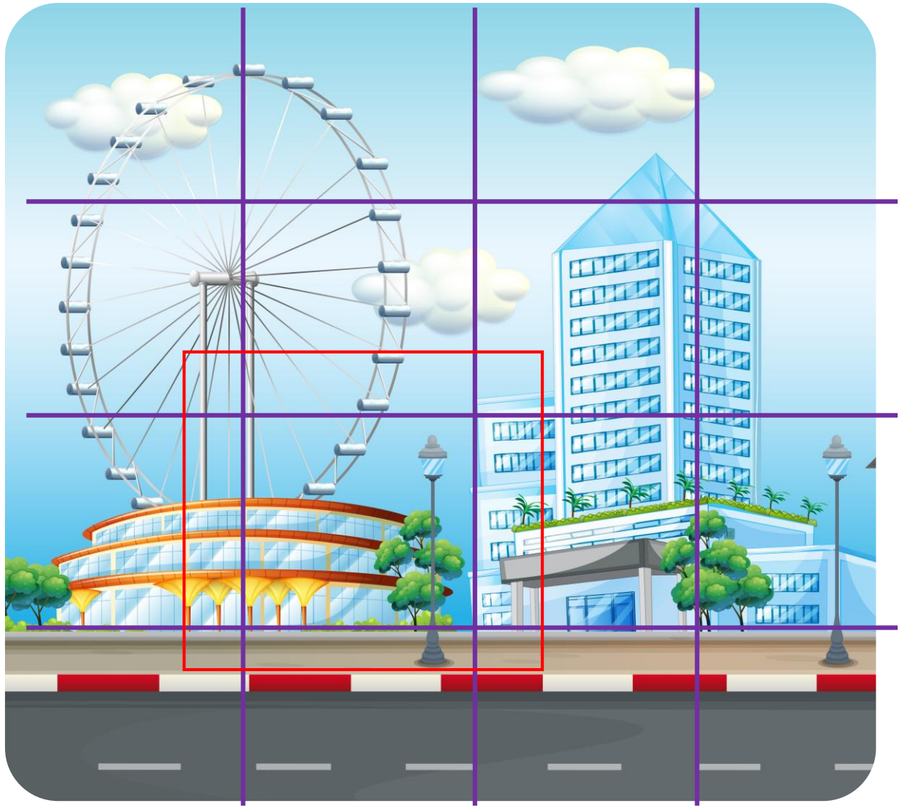


Solution: Use Intersection over Union (IoU)

Intersection over Union (IoU) is a metric used to evaluate the accuracy of an object detector by comparing the overlap between the predicted bounding box and the ground truth bounding box. It is calculated as the area of the intersection divided by the area of the union of the two boxes.

Issues in Multiple Object Detection

Two centers of objects could be sharing the same grid cell.



Solution: Concatenate the grid vectors

The objects could be any entities or elements that are being represented or detected within the grid cells.

Single vs. Multiple Object Detection

Single class:
In a single-class scenario, the task is to detect and localize objects belonging to a specific class or category.

Multiple classes:
In a multiple-class scenario, the task involves detecting and classifying objects from multiple distinct classes or categories.

High-Level Overview of the YOLOv3 Algorithm

R-CNN and Faster R-CNN

Both R-CNN (Region-Based Convolutional Neural Network) and Faster R-CNN are object detection algorithms, but Faster R-CNN is an improved version that builds upon the original R-CNN approach.

R-CNN uses region proposals, a CNN for feature extraction, and separate classifiers for object classification.

Faster R-CNN introduces a region proposal network, making the detection process faster and efficient than R-CNN.

Faster R-CNN strikes a balance between accuracy and speed and has become widely adopted in computer vision.

Why Choose YOLOv3 over R-CNN?

The YOLO (You Only Look Once) algorithm has set the standard for implementing object detection.

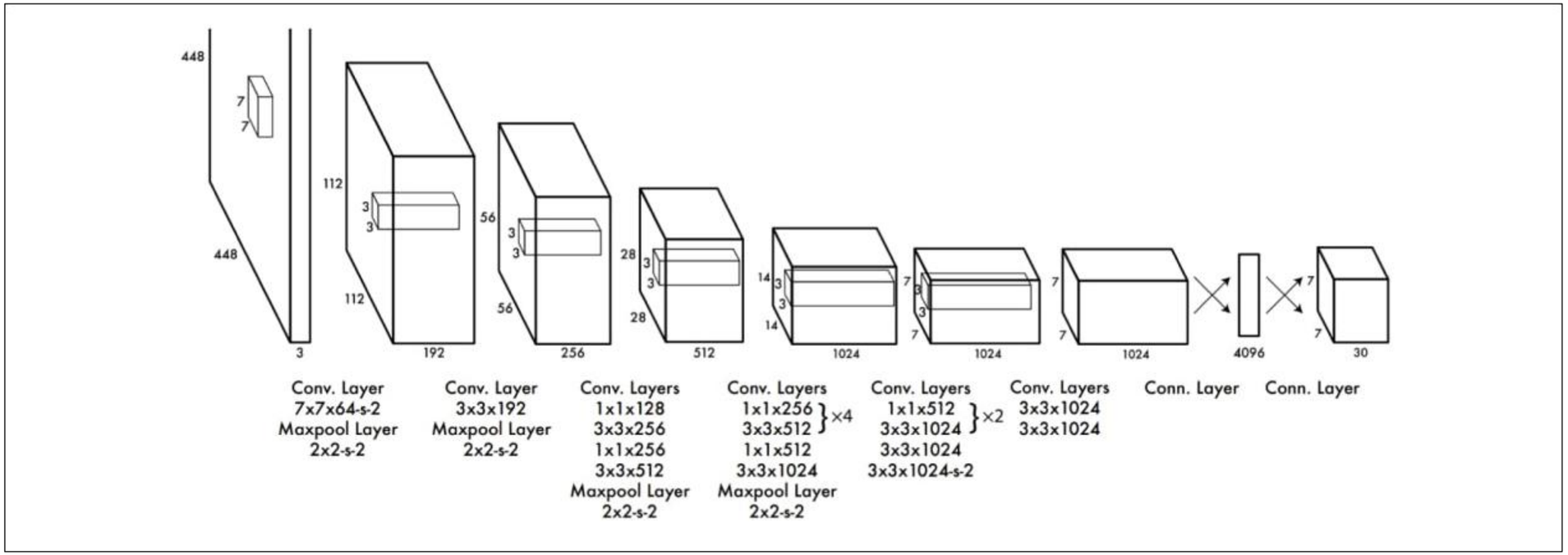


Image Reference: Kwan, C., Gribben, D., Chou, B., Budavari, B., Larkin, J., Rangamani, A., Tran, T., Zhang, J., & Etienne-Cummings, R. (n.d.). Real-Time and Deep Learning Based Vehicle Detection and Classification Using Pixel-Wise Code Exposure Measurements. *Electronics*, *9*(6), 1014. https://doi.org/10.3390/electronics9061014

It has effectively surpassed previous state-of-the-art algorithms like R-CNN and Faster R-CNN.

R-CNN Algorithm Overview

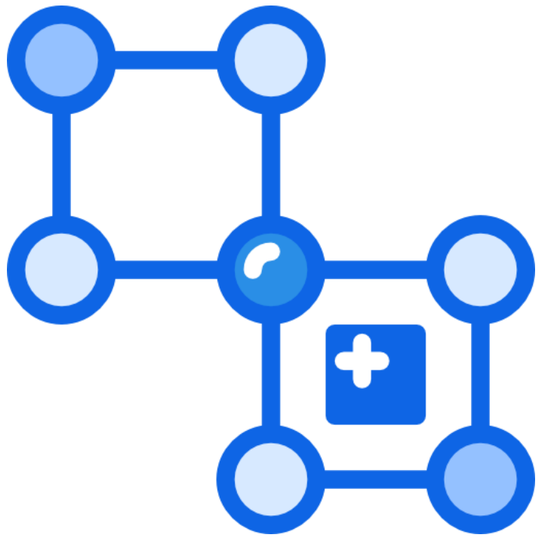
R-CNN and Regions of Interest (ROIs) are versatile techniques used in computer vision for object-related tasks such as:

- Localization
- Instance segmentation
- Classification
- Detection

Algorithms such as R-CNN focus on image ROIs, which are then classified using CNNs. It is a time-consuming process, because the predictions must be run for each region.

YOLOv3 Algorithm Overview

The YOLO (You Only Look Once) algorithm predicts classes and bounding boxes for the image based on regression, rather than selecting Regions of Interest (ROIs).



The bounding box prediction step is improved with YOLOv3, using three different scales to extract features more effectively.

YOLOv3 Algorithm Overview

YOLO performs best, because it has the highest speed and ability to handle huge volumes of data.

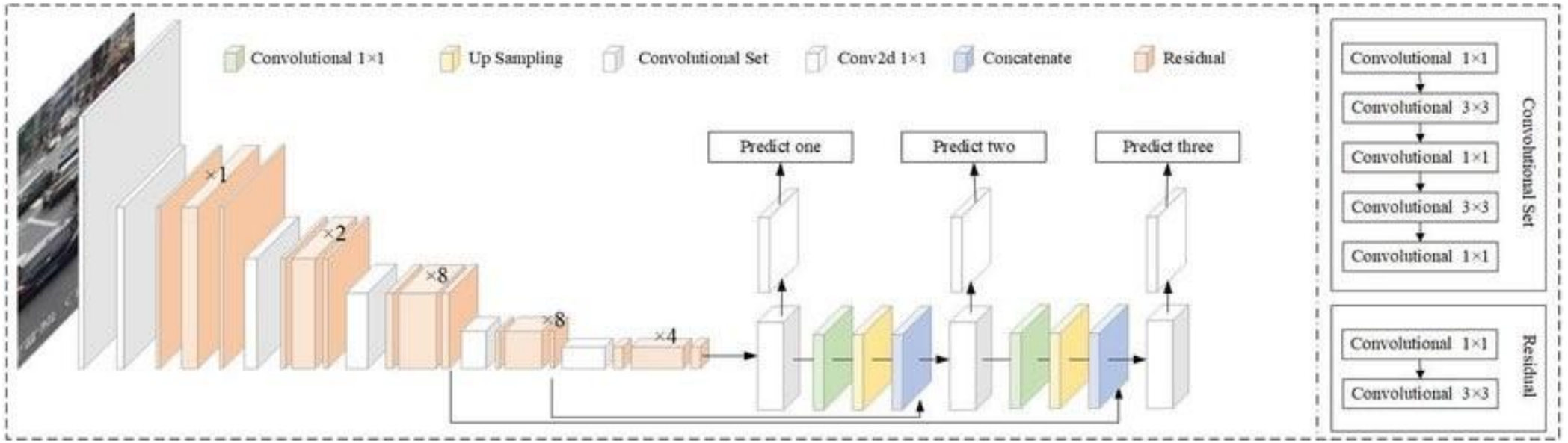
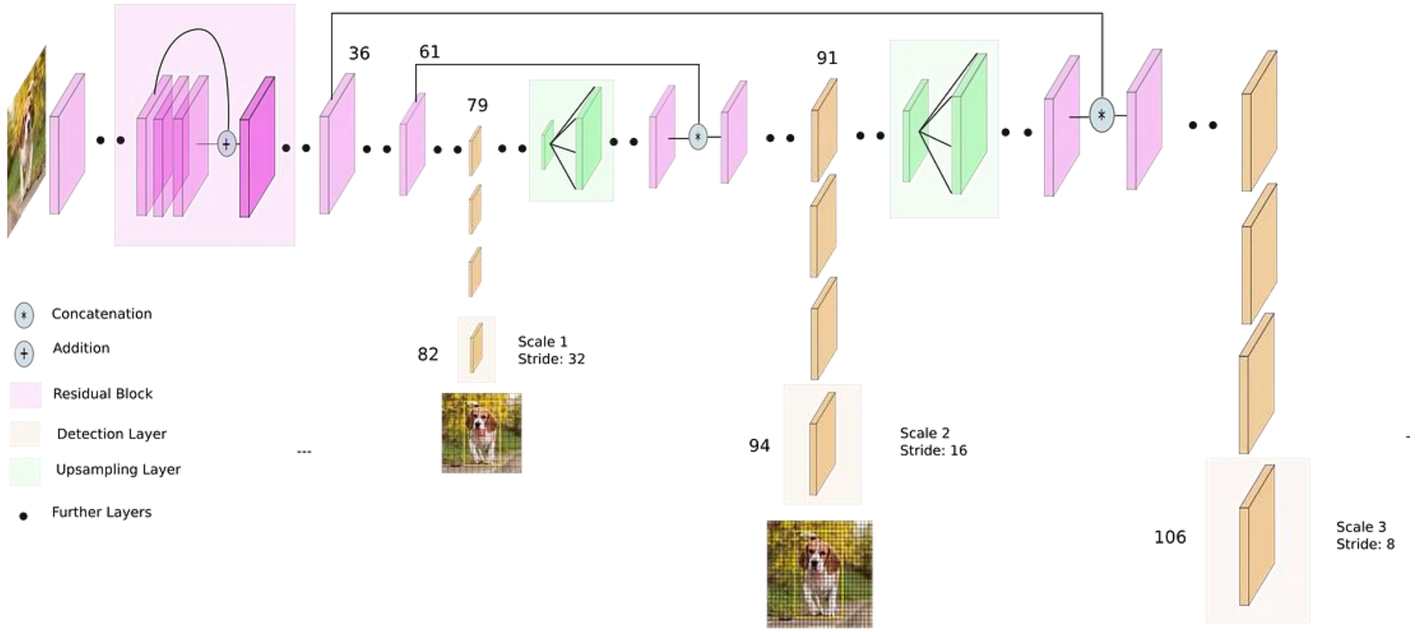


Image Reference: Mao, Q., Sun, H., Liu, Y., & Jia, R. (2019). Mini-YOLOV3: Real-Time Object Detector for embedded Applications. *IEEE Access*, *7*, 133529–133538. https://doi.org/10.1109/access.2019.2941547

YOLOv3 has 53 convolutional layers called Darknet-53 that are mostly made up of residual connections. Darknet-53 achieves improved speed and effectiveness by leveraging the GPU more efficiently.

YOLOv3 Algorithm Overview

YOLOv3 detects objects of varying sizes with three different strides: 32, 16, and 8.



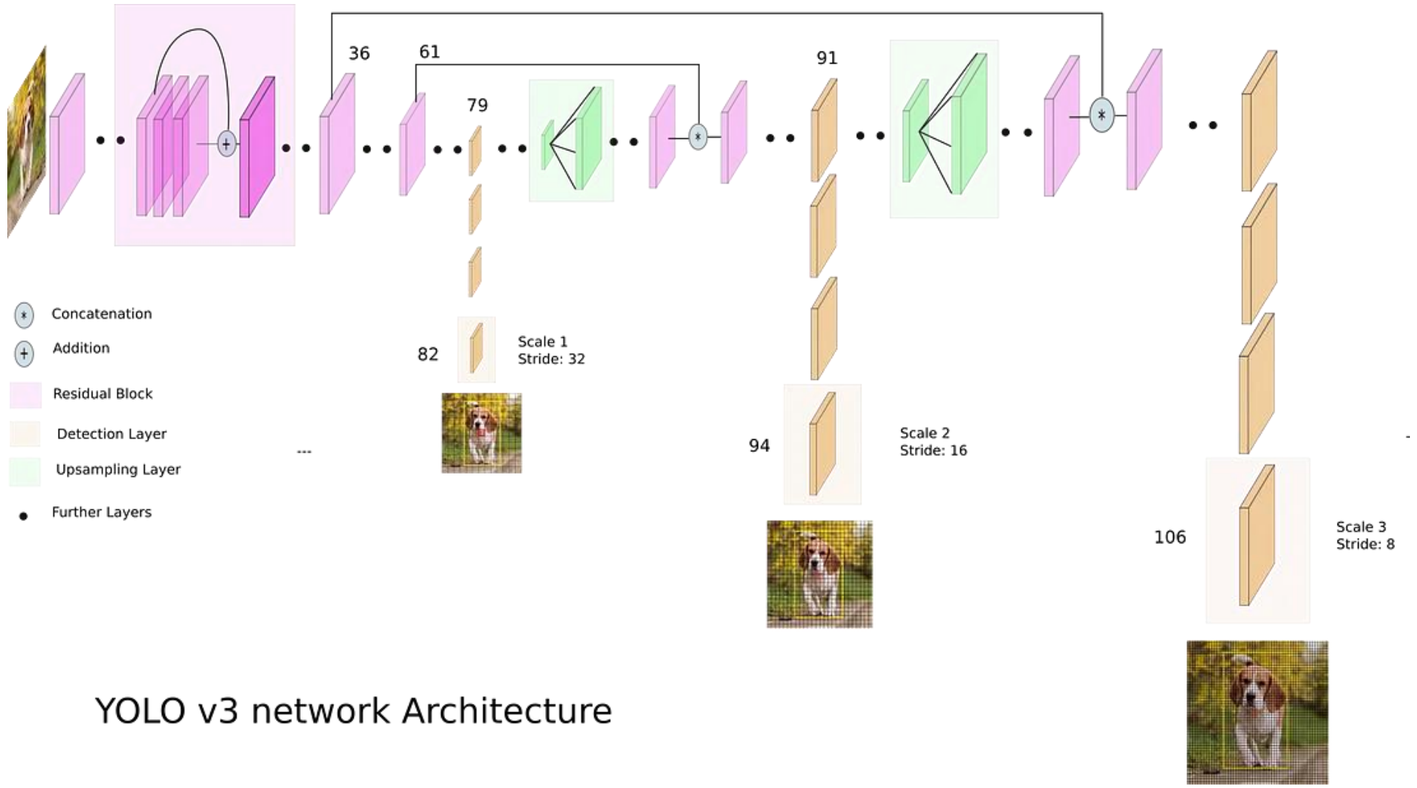
YOLO v3 network Architecture

Example: When a 512x512 input image is processed, it generates three distinct output shape tensors, each having its own size. The three scales allow YOLOv3 to detect small, medium, and large objects.

Image source: https://miro.medium.com/v2/resize:fit:1100/format:webp/1*d4Eg17IVJ0L41e7CTWLLSg.png

YOLOv3 Algorithm Overview

YOLOv3 detects objects of varying sizes with three different strides: 32, 16, and 8.



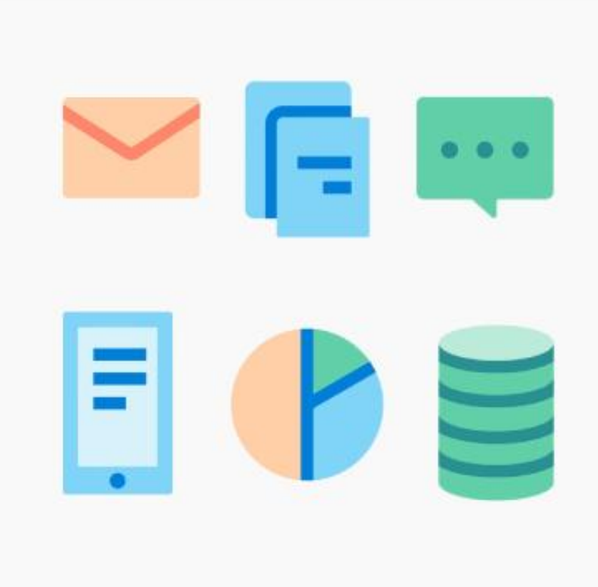
YOLO v3 network Architecture

A residual network provides a solution to mitigate the problem of gradient vanishing that arises with deeper neural networks.

Dataset Preparation for YOLOv3

Dataset Preparation for YOLOv3

The dataset preparation for YOLOv3 may seem complex, but it is quite easy to follow.



1. **Dataset Generation:**
   - Collect and organize images for training
2. **Data Annotation:**
   - Annotate the images with bounding boxes and labels

Consider using the Google Open Images dataset (V6), which contains over 80 million annotations.

Dataset Preparation for YOLOv3

Extract labeled images from the dataset

Person

Car

Building

Tree

The OIDv4 Toolkit facilitates the extraction of the specified data along with their bounding boxes, which are stored in an XML format.

Dataset Preparation: Steps

Follow these steps to prepare the dataset for YOLOv3 using the OIDv4 Toolkit:

1. Open a terminal on a machine with Python and Git installed and run the following command to clone the repository:

```
git clone https://github.com/EscVM/OIDv4_ToolKit
```

2. Download the dataset and navigate to the cloned repository directory

```
cd OIDv4_ToolKit
```

Dataset Preparation: Steps

3. Run the script to download the specified classes (Car, Building, Tree, and Person):

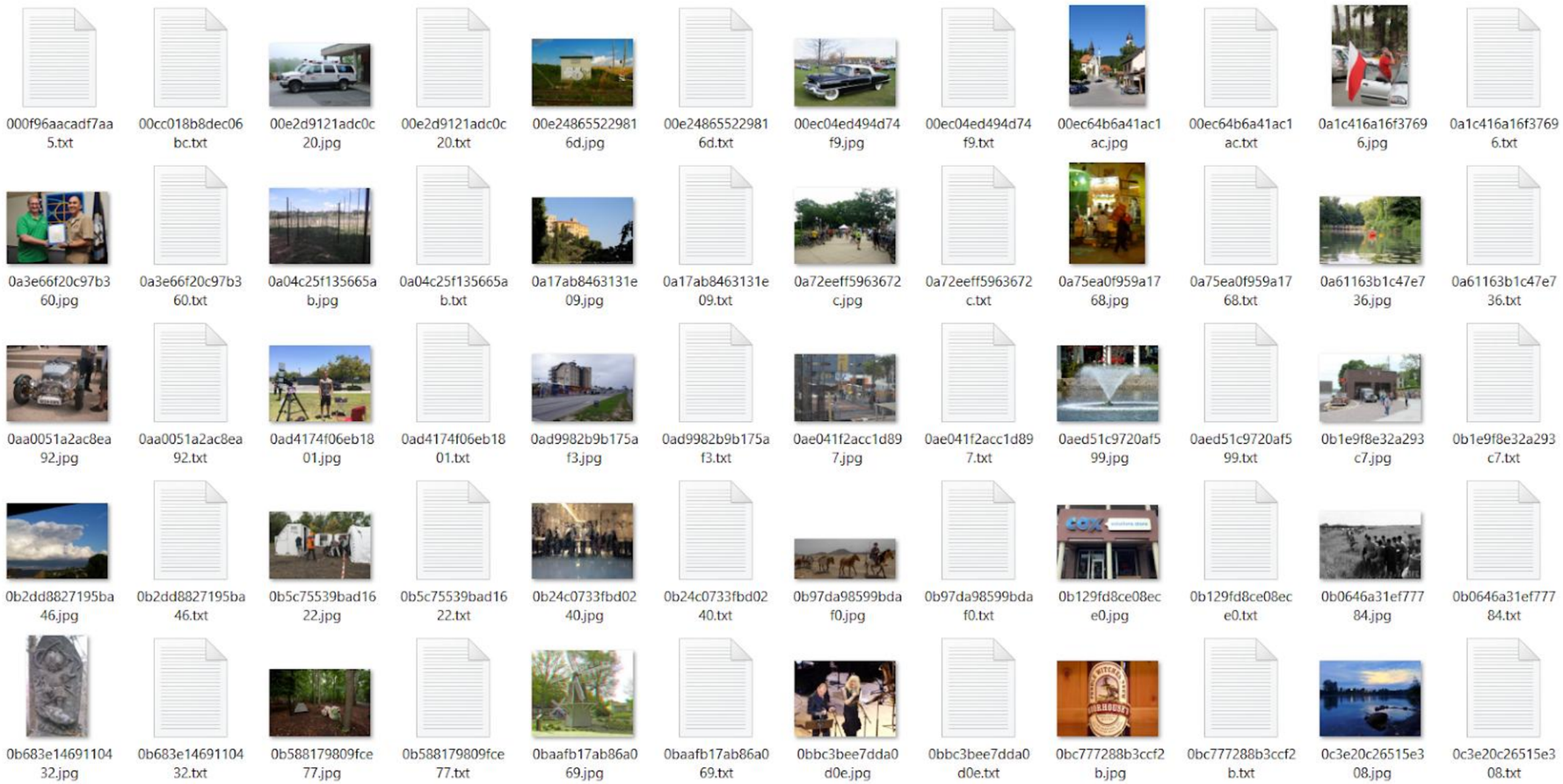
```
python3 main.py downloader --classes Car Building Tree Person --type_csv train --multiclasses 1 --limit 600
```

4. The designated directory will contain both the images and their corresponding label files.

```
OID/Dataset/car_building_tree_person
```

Dataset Preparation: Steps

The images in the directory will be displayed as follows:



Dataset Preparation: Steps

The text files generated during the dataset preparation are named after their corresponding image files. These text files contain the image labels along with the bounding box coordinates.

Example:

<label><Bbox x><Bbox y><Bbox w><Bbox h>

2 0.087402 0.911679 0.094727 0.173723

This format helps in organizing the data and ensuring that each image has its associated labels and bounding box coordinates, which are essential for training the YOLOv3 model.

Dataset Preparation: Steps

Organize images in the *images* folder and text files in the *labels* folder, both located within the *dataset* folder.

The *labels* folder should contain text files with the same names as the corresponding images. Each text file contains the labels and bounding box coordinates for the objects in the image.

Dataset Preparation: Steps

1. Clone the repository for training using the following command:

```
git clone https://github.com/ultralyatics/YOLOv3
```

This repository contains the code and structure needed for training the YOLOv3 model.

2. Create a file named custom_data.yaml inside YOLOv3/data.

Dataset Preparation: Steps

Include the following code within the custom_data.yaml file

Syntax:

```
path: ../dataset # dataset root dir
train: images # train images (relative to 'path') 128 images
val: images # val images (relative to 'path') 128 images
test: # test images (optional)

# Classes
nc: 4 # number of classes
names: ['car', 'building', 'tree', 'person'] # class names
```

Dataset Preparation: Steps

To expedite the training process, execute the training script and load a pretrained model.

Syntax:

```
python3 train.py --img 416 --batch 16 --epochs 5 --data custom_data.yaml --weights YOLOv3.pt
```

Assisted Practice

Let us understand the concept of object detection with YOLO using Jupyter Notebooks.

- 10.07_Object Detection with YOLO

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Introduction to TensorFlow Lite

TensorFlow Lite (TFLite)

TensorFlow Lite (TFLite) is a cross-platform framework designed for efficiently deploying machine learning models on mobile devices and embedded systems.

TFLite models are lightweight and optimized for inference on edge devices such as:

- Mobile phones
- Microcontrollers

They are well-suited for integrating machine learning into edge devices.

TensorFlow Lite: Example

Consider building a system to detect the hand gestures of individuals who are unable to speak

Step 1

Collect a dataset with diverse hand gestures

Step 2

Train the dataset using a CNN classifier and save the trained model

Step 3

Convert the saved model to TensorFlow Lite (TFLite) to enable efficient inference on mobile devices

TensorFlow Lite: Features

Convert the model to TensorFlow Lite (TFLite) format to overcome constraints for deploying hand gesture detection on edge devices

Key features:

- Optimizes model size and speeds up inference
- Designed for resource-constrained devices like smartphones, Arduino platforms, and microcontrollers
- Compatible with various hardware architectures, maximizing performance
- Enables efficient deployment of hand gesture detection systems

Constraints in Edge Devices

Challenges faced by edge devices:

Low memory

Limited storage capacity

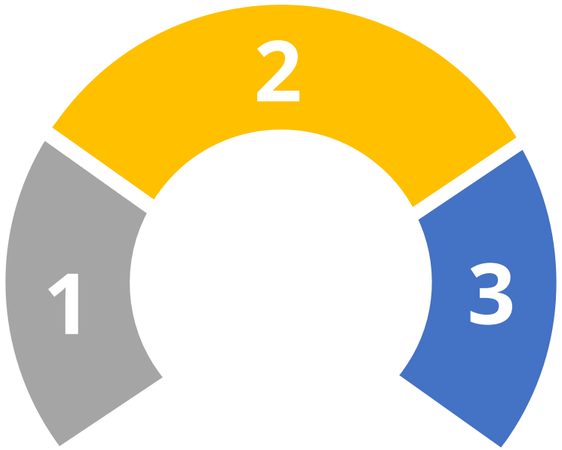
High output latency

TFLite enables the deployment of lightweight models with low latency for efficient inference, ensuring optimal performance on edge devices.

TensorFlow Lite: Advantages

Low Latency: Predictions are exceptionally fast due to minimal latency.

User Privacy: Predictions are performed on the device, ensuring no data transfer to external servers.



Pretrained Models: TensorFlow Hub offers a wide range of pretrained models for various tasks and applications.

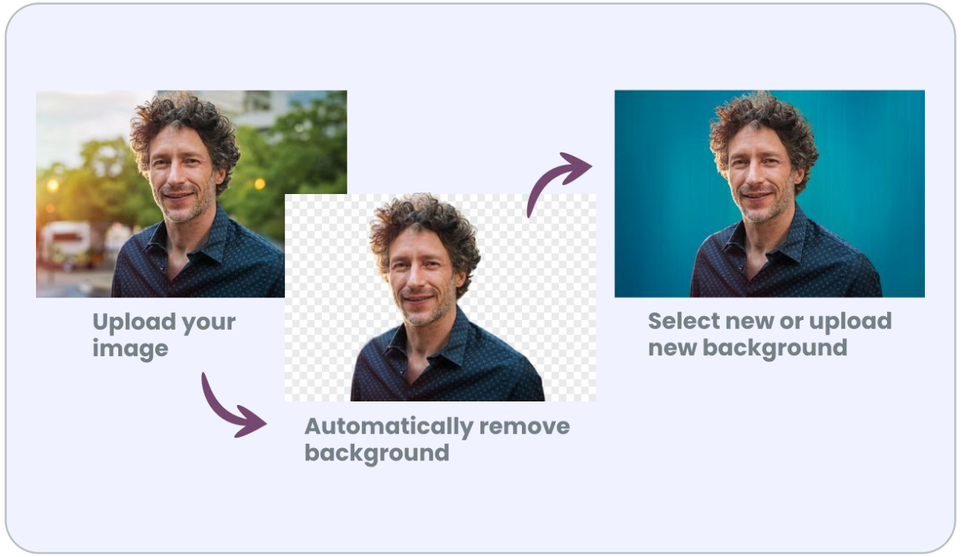
Apps Built Using TensorFlow Lite

Google Translate

Google Translate can capture and translate text from any language in real time, even without an internet connection.

Apps Built Using TensorFlow Lite

Background-changing apps



Apps like Zoom and Google Meet use TFLite models to identify and isolate the person in the foreground while changing the background.

Converting a TensorFlow Model into a TensorFlow Lite (TFLite) Model

Converting a TensorFlow Model into a TensorFlow Lite (TfLite) Model

To convert a TensorFlow model into a TensorFlow Lite (TFLite) model, you can use the TensorFlow Lite Converter. Steps to follow:

Step 1	Install the required packages, for example, TensorFlow and TFLite Converter
Step 2	Save the trained TensorFlow model, for example, model.save('model.h5')
Step 3	Convert the saved model to TensorFlow Lite using the TFLite Converter

Assisted Practice

Let us understand the concept of converting TF model into TF Lite model using Jupyter Notebooks.

- 10.10_Converting_TF_Model_into_TF_Lite_Model

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Key Takeaways

- Object detection identifies and localizes objects in an image using bounding boxes.
- Data preparation includes data collection and annotation.
- The YOLOv3 algorithm efficiently detects objects in images and videos.
- The TensorFlow Lite Converter transforms a TensorFlow model into a TensorFlow Lite (TFLite) model.

Knowledge Check

Knowledge Check 1

Which of the following applications benefits the most from real-time object detection?

A. Image archiving systems

B. Autonomous vehicles

C. Historical data analysis

D. Document scanning

Knowledge Check 1

Which of the following applications benefits the most from real-time object detection?

A. Image archiving systems

B. Autonomous vehicles

C. Historical data analysis

D. Document scanning

The correct answer is **B**

Real-time object detection allows the system to identify and locate objects around the vehicle instantly. This capability is essential for navigating environments safely and making real-time driving decisions.

Knowledge Check

2

What is Darknet-53?

A. A backbone network used in YOLOv3

B. A dataset for object detection

C. A neural network architecture for object segmentation

D. An algorithm for object classification

Knowledge Check 2

What is Darknet-53?

A. A backbone network used in YOLOv3

B. A dataset for object detection

C. A neural network architecture for object segmentation

D. An algorithm for object classification

The correct answer is **A**

Darknet-53 is a backbone network used in YOLOv3, which mainly consists of residual connections.

Knowledge Check

3

What are the constraints in inference on edge devices?

A. More memory

B. Limited storage capacity

C. Low output latency

D. All of the above

Knowledge Check 3

What are the constraints in inference on edge devices?

A. More memory

B. Limited storage capacity

C. Low output latency

D. All of the above

The correct answer is **D**

Constraints in inference on edge devices include limited storage capacity, less memory, and high output latency.

Thank You!