Deep Learning

Object Detection

Learning Objectives

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

- Examine the object detection algorithm YOLOv3 in detail
- Apply the knowledge of dataset preparation techniques to prepare a dataset specifically for YOLOv3
- Apply YOLOv3 for object detection
- Evaluate the impact of the TensorFlow Lite conversion on the model's performance and deployment efficiency

Business Scenario

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

The company aims to recognize the products that customers are selecting 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 be used to deploy the algorithm on store employees' mobile devices.

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

Introduction to Object Detection

Discussion

Discussion: Object Detection

Duration: 10 minutes

- How does object detection work?
- What are some common methods used for object detection?

Object Detection

Object detection is the process of identifying and locating specific objects, such as a person, dog, car, or truck, in an image or video.

It determines and acknowledges the objects depicted in an image using bounding boxes.

Object detection is used in autonomous vehicles, surveillance systems, and retail analytics, among others.

Object Detection

Following are some of the benefits and practical uses of object detection:

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

It facilitates real-time processing and finds applications 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 would allow a computer to operate the vehicle autonomously.

The system's navigation can be simplistically developed by:

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

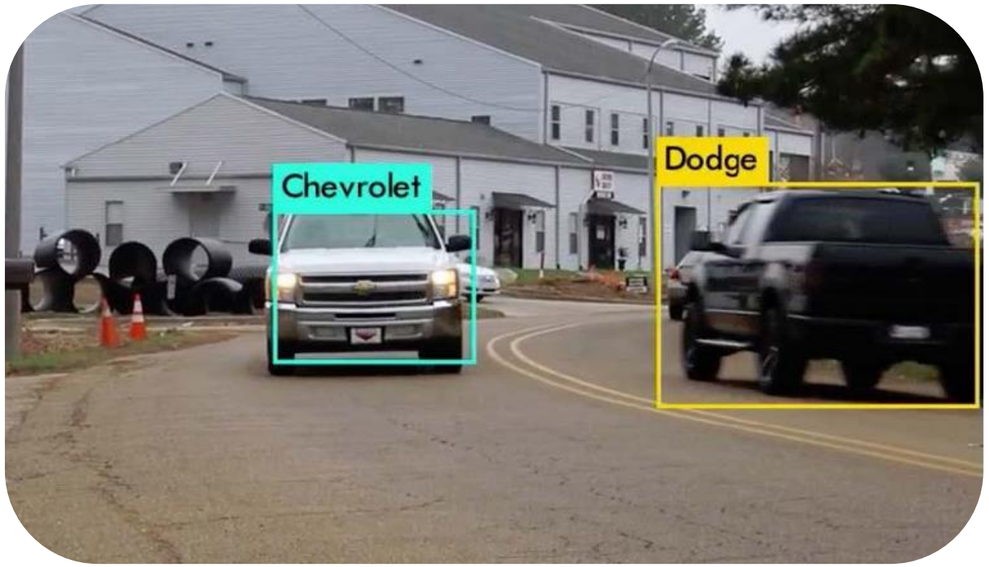
Object Detection: Example

To develop a system's navigation, use Google open images dataset V6 and consider the following classes:

- Car
- Person
- Building
- Trees

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 whether they are stationary or moving.

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

Object Detection: Example

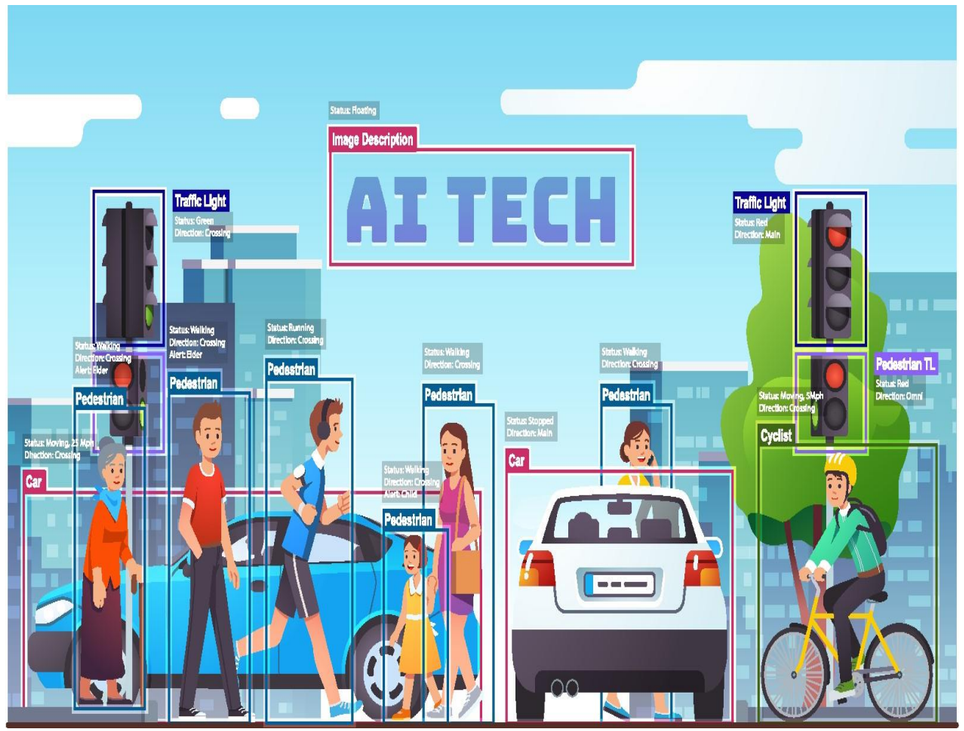
Consider the following image with a traffic signal and a traffic sign board

The system is useful for understanding what lies ahead and taking action based on the color of the traffic lights.

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

Object Detection in Computer Vision

Object detection in computer vision involves the following two-steps:



Object classification

Object localization

Object Detection Modes

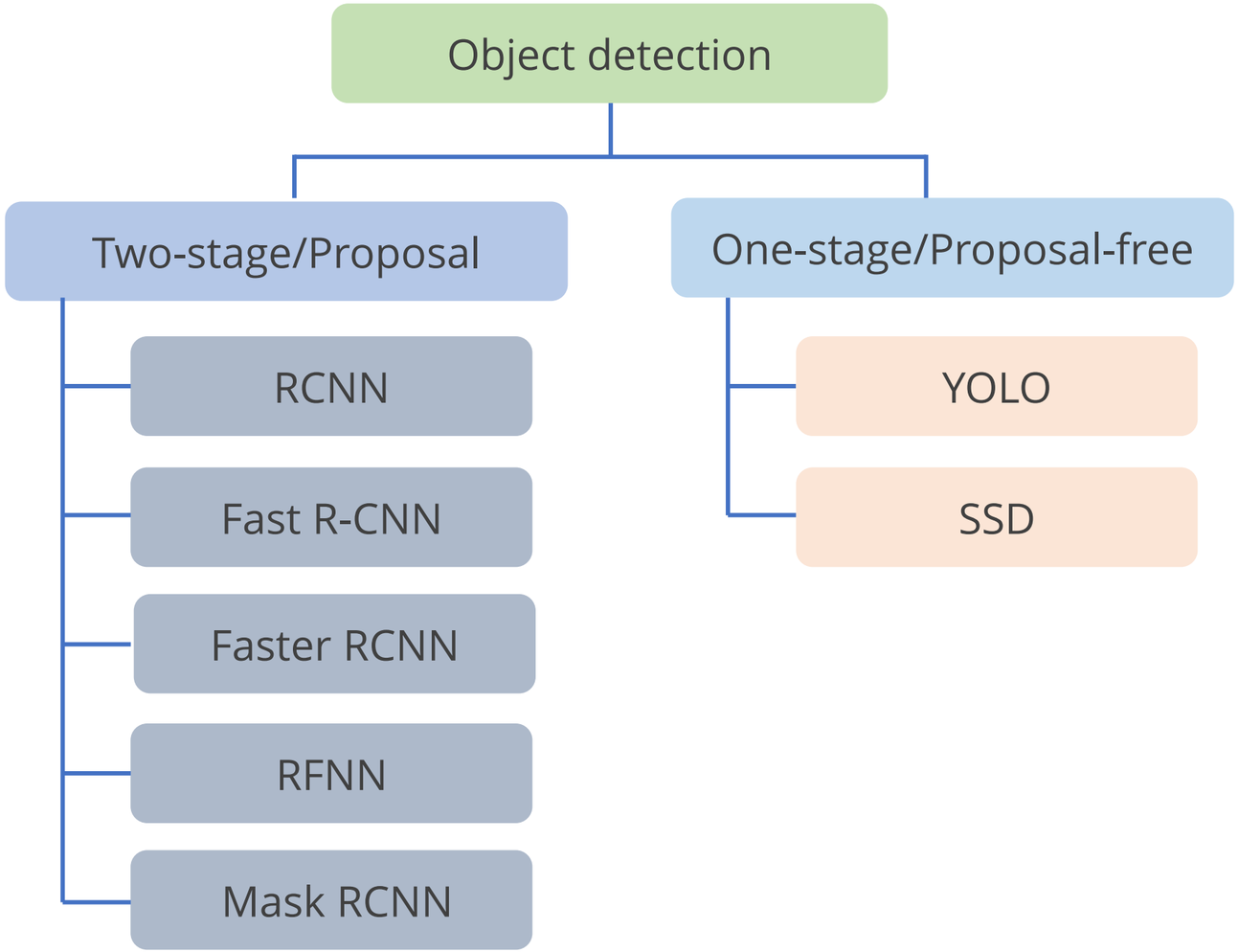
The two modes of object detection are:

One-stage/Proposal-free → It is an object detection mode that simultaneously predicts the class and bounding box of objects in an image, without the need for an initial proposal stage.

Two-stage/Proposal → It is an object detection mode that first generates region proposals where objects might exist and then classify those regions in a separate step.

Object Detection Modes

The classification of object detection modes is as follows:



Object Detection Modes: Two-Stage or Proposal-Based

RCNN	It uses the selective search for region extraction and applies CNNs for accurate classification and bounding box refinement.
Fast R-CNN	It improved upon R-CNN by sharing convolutional features and using RoI pooling for efficient region proposal processing.
Faster RCNN	It introduced a Region Proposal Network (RPN) for direct region proposal generation, optimizing training and computation.
RFPNN	It introduced a Region Proposal Network (RPN) for efficient region proposal generation, enhancing training and computation.
Mask RCNN	It enhances Faster R-CNN with pixel-level segmentation, enabling instance segmentation within the R-CNN family.

Object Detection Modes: One-Stage or Proposal-Free

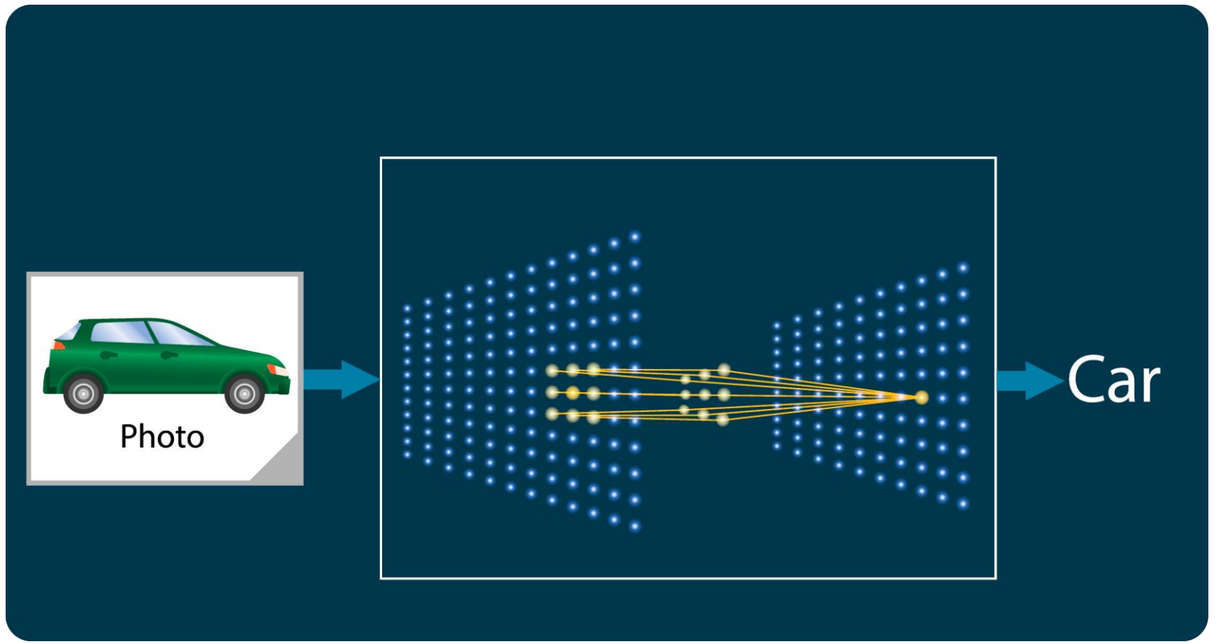
YOLO and SSD are real-time object detection algorithms.

YOLO	It is a fast and efficient algorithm that predicts bounding boxes and class probabilities directly, making it ideal for video analysis and robotics.
SSD	It enhances object detection accuracy by using default bounding boxes of varied scales, proving particularly beneficial for small objects.

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

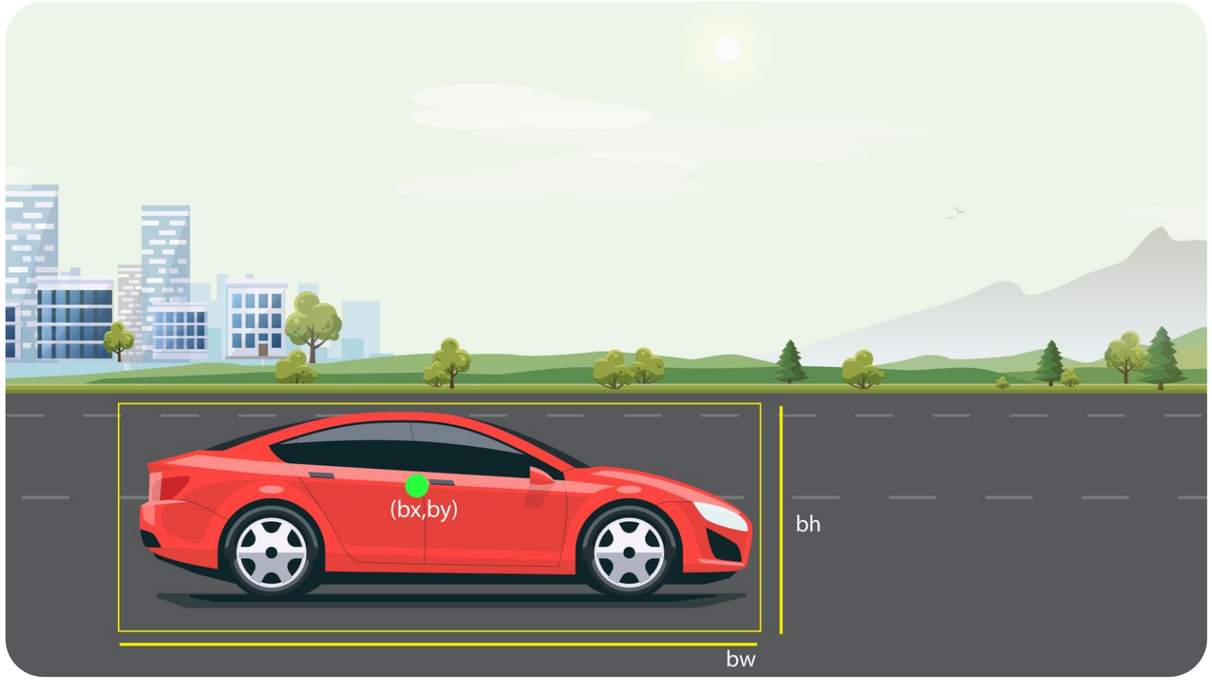
Object Classification

The algorithm identifies the detected object's class and location based on the 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

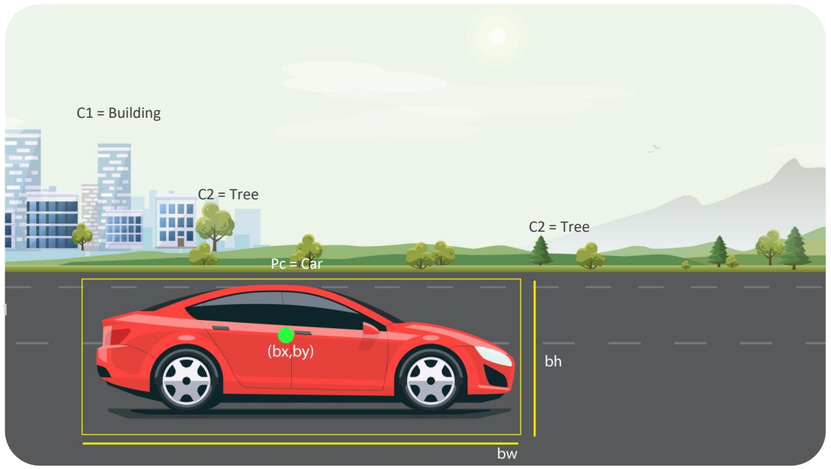
bw is the width of the bounding box with respect to the image width

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

Object Detection for Multiple Objects

Object Detection for Multiple Objects

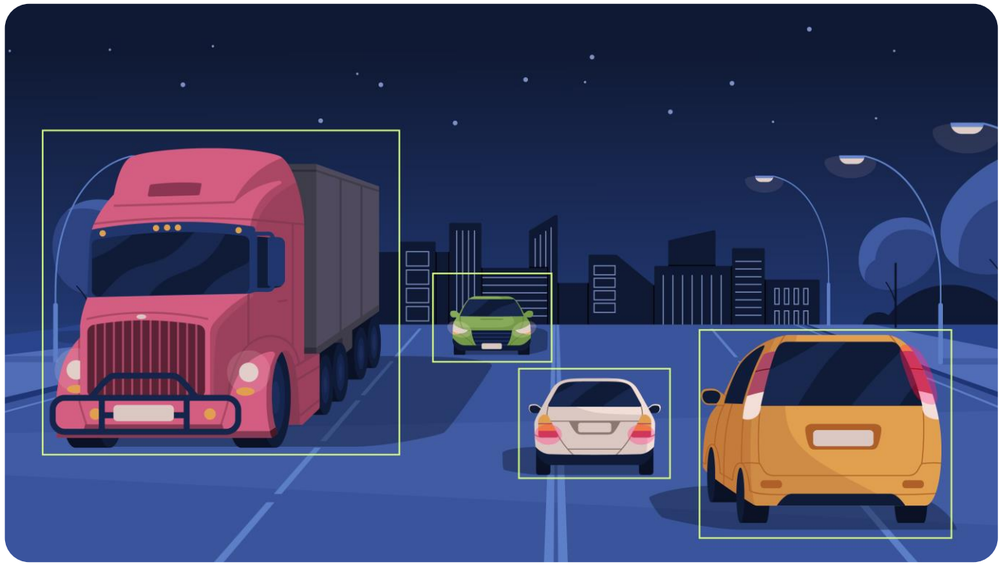
For any given image, an object detection algorithm will generally produce a vector.



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 represent 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.

Estimating the Bounding Boxes

The steps for estimating bounding boxes for multiple objects are:

Step 1:
Grid division

Step 2:
Object search and probability calculation

Step 4:
Neural network optimization

Step 3:
Vector generation

Estimating the Bounding Boxes

The steps for estimating bounding boxes for multiple objects are:

Step 1:
Grid division

The image is partitioned into a 4x4 grid, resulting in smaller regions of equal size 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

The steps for estimating bounding boxes for multiple objects are:

Step 3: Vector generation

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

Step 4: Neural network optimization

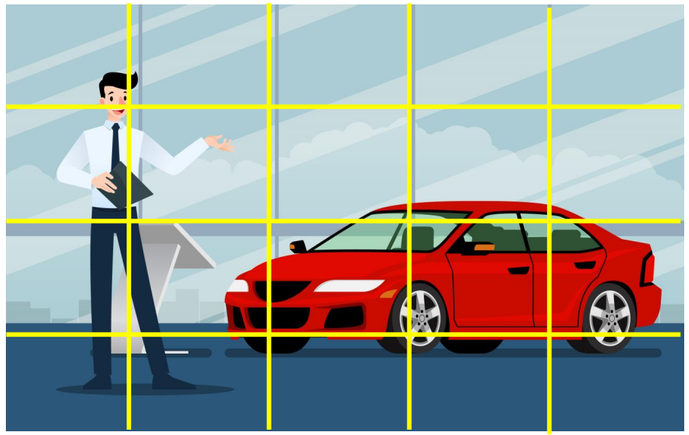
A neural network enhances bounding box estimation accuracy by optimizing the refinement process with generated vectors.

Estimating Bounding Boxes: Example

Consider the following image

Estimating Bounding Boxes: Example

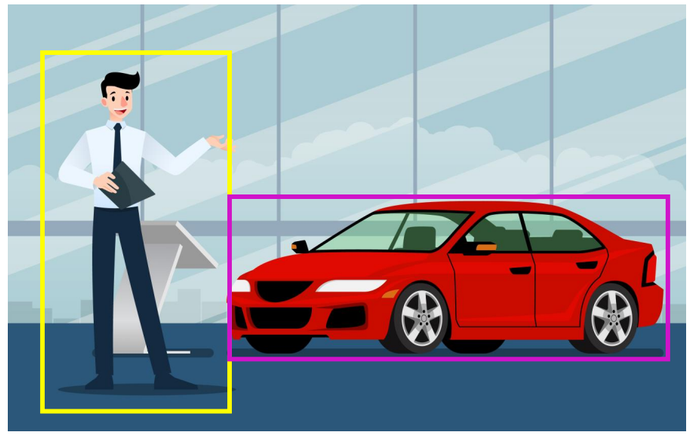
Object detection uses grid-based division and deep learning models, such as CNNs, to generate feature vectors for predicting bounding boxes, object scores, and class probabilities.



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

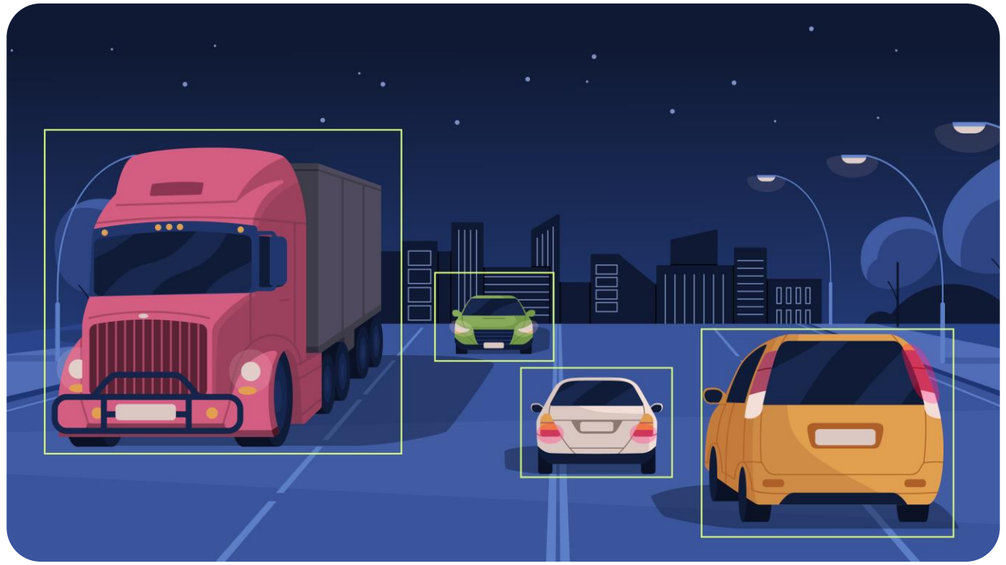
CNN is crucial in object detection, identifying object locations, and estimating bounding boxes accurately.

- 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.

Issues in Multiple Object Detection

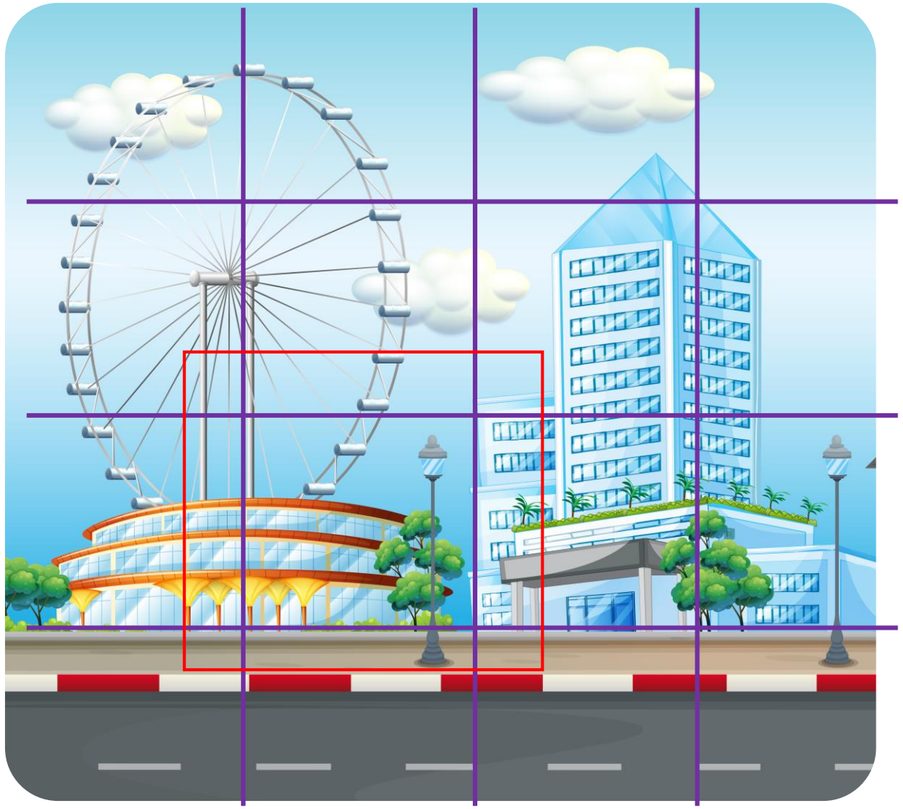
The same object may have several bounding boxes.



Solution: Use Intersection over Union (IoU)

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.

Multiple Object

Single Class

Multiple Object

Multiple Classes

Discussion: Object Detection

Duration: 10 minutes

- How does object detection work?

Answer: Object detection works by applying a trained model to an input image or video. The model analyzes the image and predicts bounding boxes around objects of interest, along with their corresponding class labels.

- What are some common methods used for object detection?

Answer: Some common methods used for object detection include the R-CNN family of algorithms (including Fast R-CNN and Faster R-CNN), Single Shot MultiBox Detector (SSD), and You Only Look Once (YOLO).

High-Level Overview of the YOLOv3 Algorithm

RCNN and Faster RCNN

Both RCNN (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 RCNN approach.

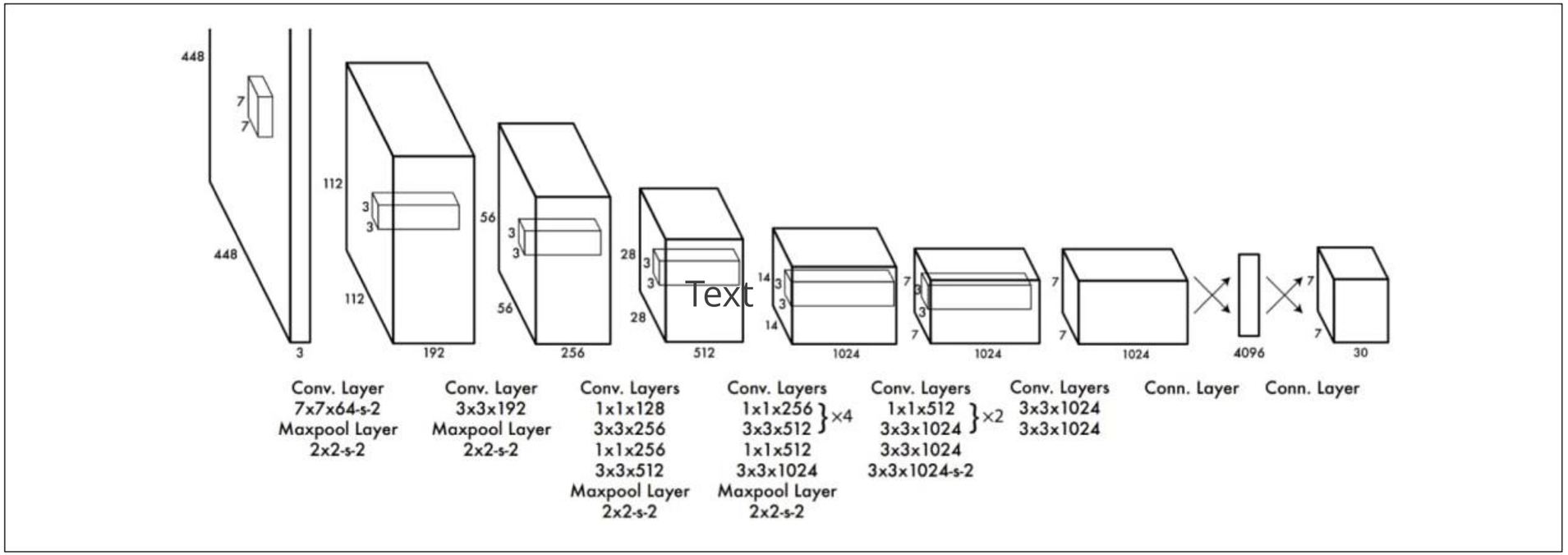
RCNN uses region proposals, a CNN for feature extraction, and separate classifiers for object classification.

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

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

Why YOLOv3 over RCNN?

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



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

Image source: https://assets-global.websitefiles.com/5d7b77b063a9066d83e1209c/63c697fd4ef3d83d2e35a8c2_YOLO%20architecture-min.jpg

RCNN Algorithm Overview

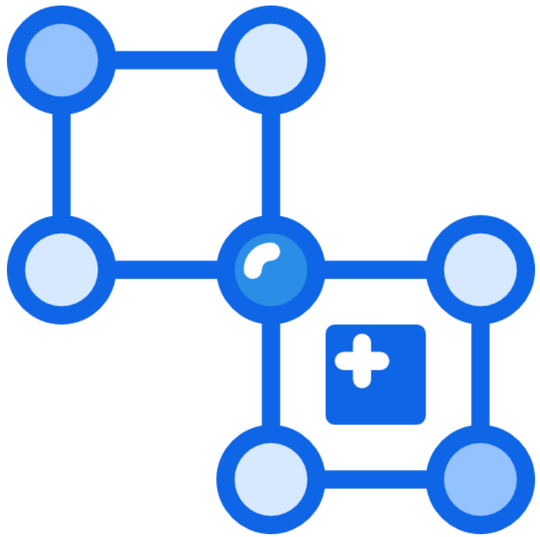
RCNN and regions of interest (ROIs) are versatile techniques used in computer vision for object-related tasks such as:

- Localization
- Instance segmentation
- Classification
- Detection

Algorithms such as RCNN 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

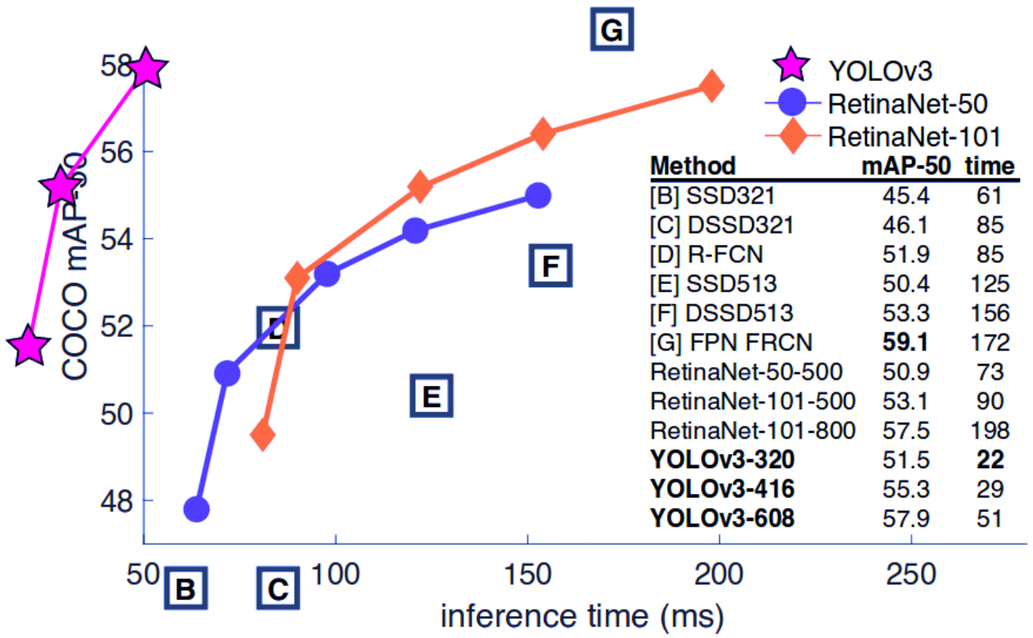
The YOLO algorithm predicts classes and bounding boxes for the image based on regression rather than selecting ROIs.



The bounding box prediction step is improved, and three different scales are used to extract features.

YOLOv3 Algorithm Overview

The graph below displays a comparison of the inference times of different methods implemented on the widely used COCO (Common Objects in Context) dataset.

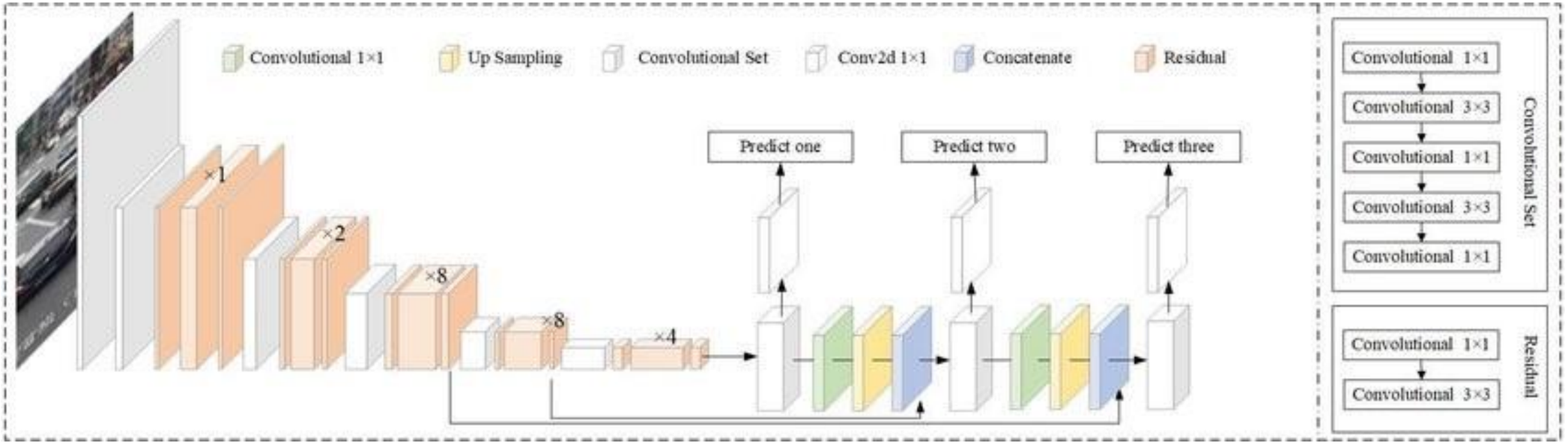


Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

COCO dataset contains labeled images for object detection, segmentation, and captioning tasks in computer vision.

YOLOv3 Algorithm Overview

Yolo performs best because it has the highest speed and ability to handle large amounts of data.

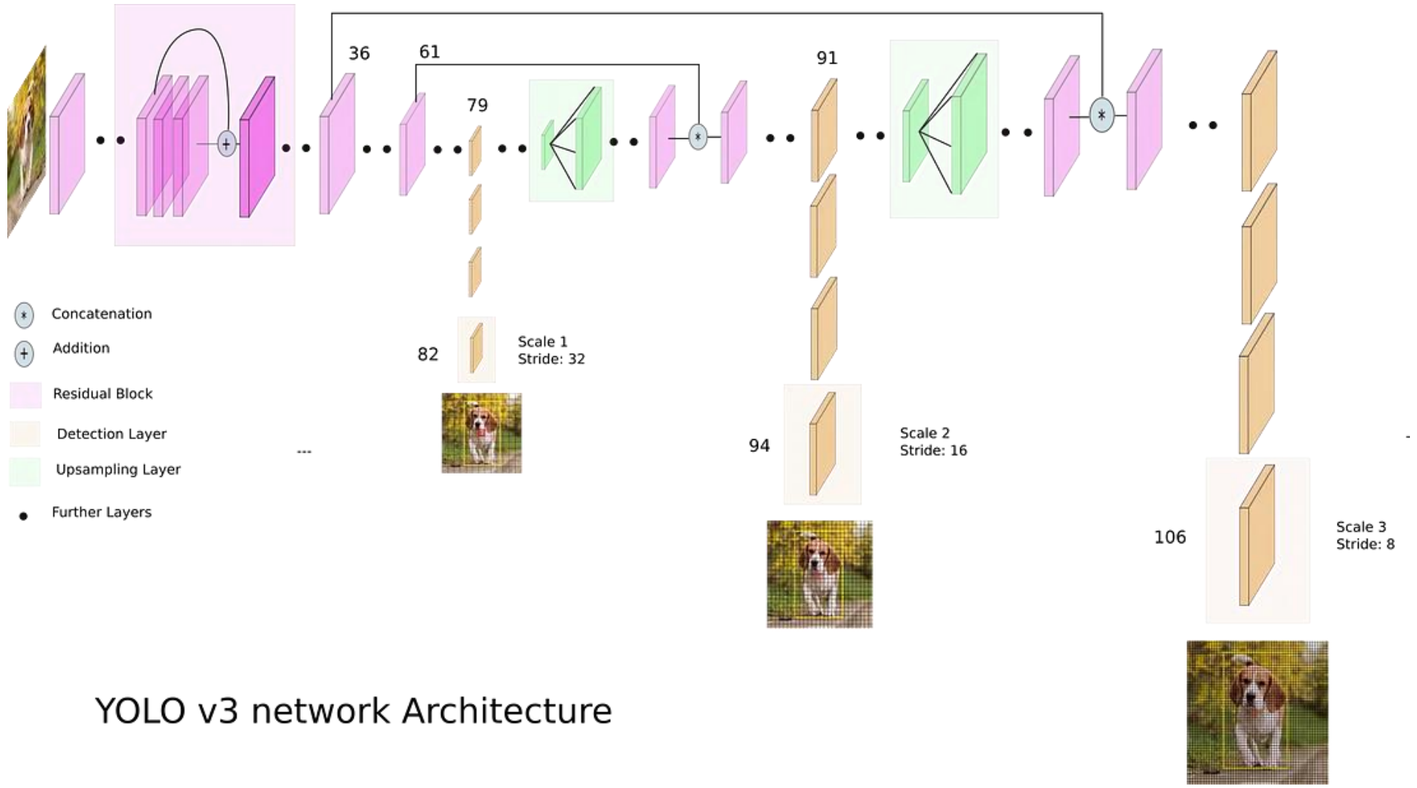


YOLOv3 has 53 convolutional layers called Darknet-53 that are mostly made up of residual connections. Darknet-53 achieves improved speed and effectiveness by leveraging the GPU more efficiently.

Image source: https://www.researchgate.net/figure/Structure-detail-of-YOLOv3It-uses-Darknet-53-as-the-backbone-network-and-uses-three_fig1_335865923

YOLOv3 Algorithm Overview

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



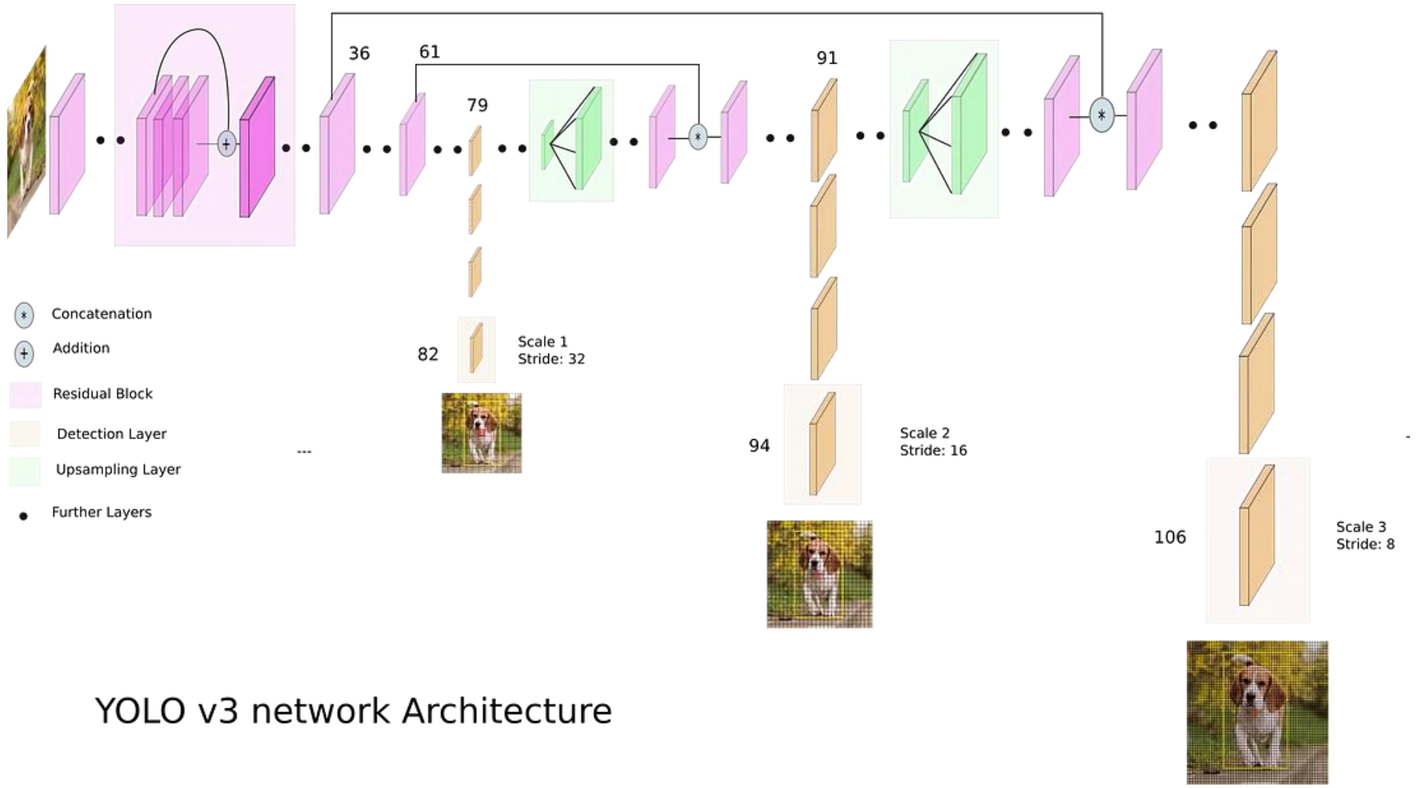
YOLO v3 network Architecture

For example, when a 512x512 input image is processed, it generates three distinct output shape tensors, each having its own size.

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

YOLOv3 Algorithm Overview

The strides help in detecting fine-grained small objects in the image easily.



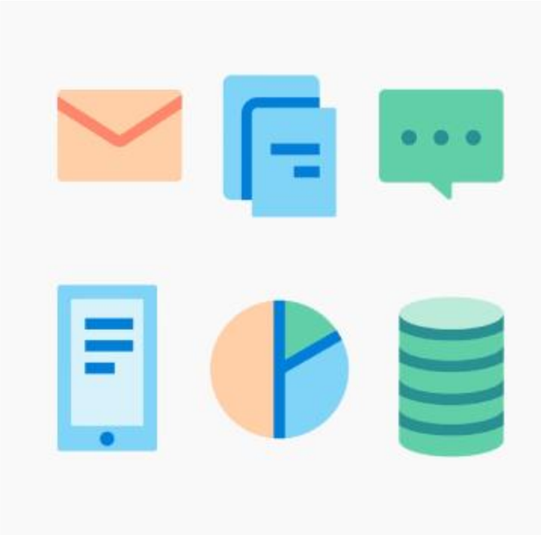
YOLO v3 network Architecture

The use of 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 is a little complex but easy to follow..



To begin data preparation, the first step is dataset generation, followed by data annotation.

Consider using V5, the Google Open Image dataset, which has 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

Use the following git clone command to clone the existing OIDv4 Toolkit repository:

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

Run the script as shown below:

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

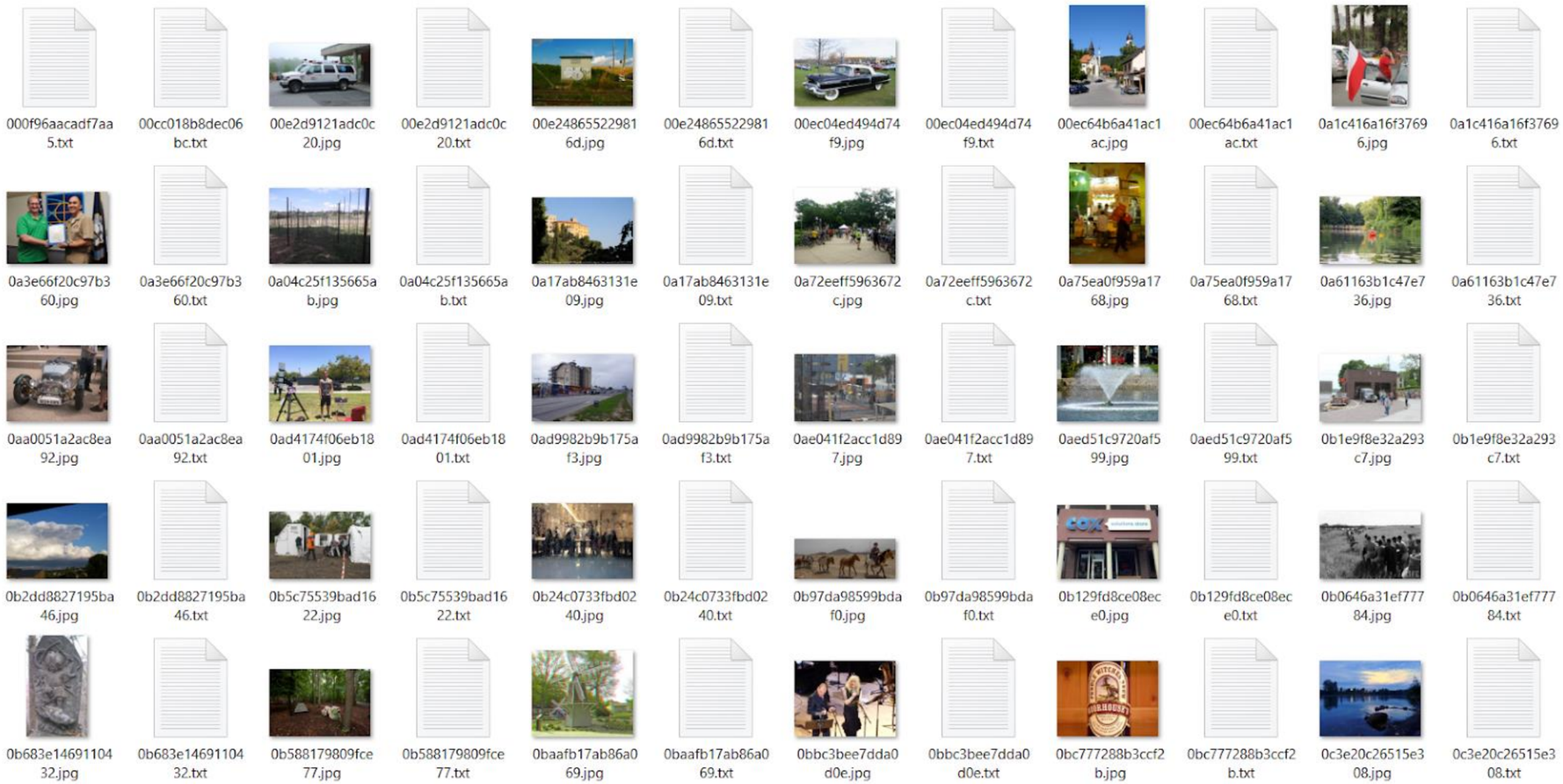
Dataset Preparation: Steps

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 are named after their corresponding image file names and contain the image labels along with the bounding box coordinates.

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

Example:
2 0.087402 0.911679 0.094727 0.173723

Dataset Preparation: Steps

The steps for data preparation are as follows:

Step 1 Open a text editor or code editor of your choice.

Step 2 Create a new file and save it as "custom_data.yaml".

Step 3 Copy the following content and paste it into the "custom_data.yaml" file.

Step 4 Save the file.

Dataset Preparation: Steps

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

Dataset/images

Dataset/labels

Dataset Preparation: Steps

Clone the repository for training using the following command:

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

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 pre-trained model.

Syntax:

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

Assisted Practice

Let’s understand the concept of object detection with YOLOv3 using Jupyter Notebooks.

- 9.07_Object Detection with YOLOv3

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)

It is a cross-platform framework for efficiently deploying machine learning models on mobile devices and embedded systems.

TFLite models are lightweight models used to get inferences on edge devices like:

- 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 mute people

Step 1

Collect a dataset comprising diverse hand gesture

Step 2

Train the dataset using a CNN classifier and save the trained model for future use

Step 3

Convert the saved model to TF Lite to easily make inferences from mobile devices

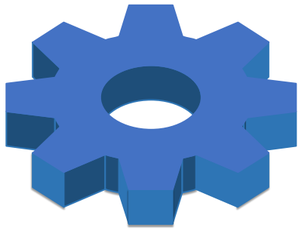
TensorFlow Lite: Example

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

- Optimizes model size and speeds up inference, addressing resource limitations
- Is designed for resource-constrained devices like smartphones, Arduino platforms, and microcontrollers
- Provides compatibility with various hardware architectures, maximizing performance
- Enables improved performance and efficient deployment of hand gesture detection systems

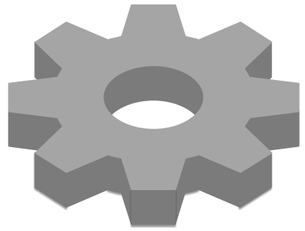
Constraints in Edge Devices

The challenges faced on edge devices are:



Low memory

Limited storage capacity



High output latency

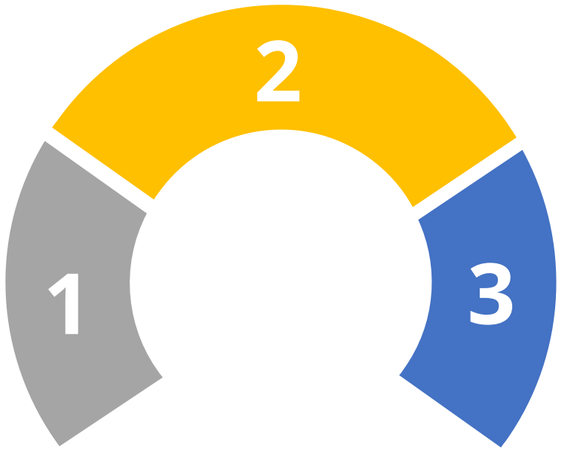
TF Lite enables the deployment of lightweight models with low latency for efficient inference, ensuring optimal performance on edge devices.

TensorFlow Lite: Advantages

TF Lite provides the following advantages:

As predictions are performed on the devices themselves, user privacy is upheld since there is no data transfer between the device and external servers.

Due to the extremely low latency, the predictions made by the model are exceptionally fast.



TensorFlow Hub offers a wide range of pre-trained models that can be readily utilized for various tasks and applications.

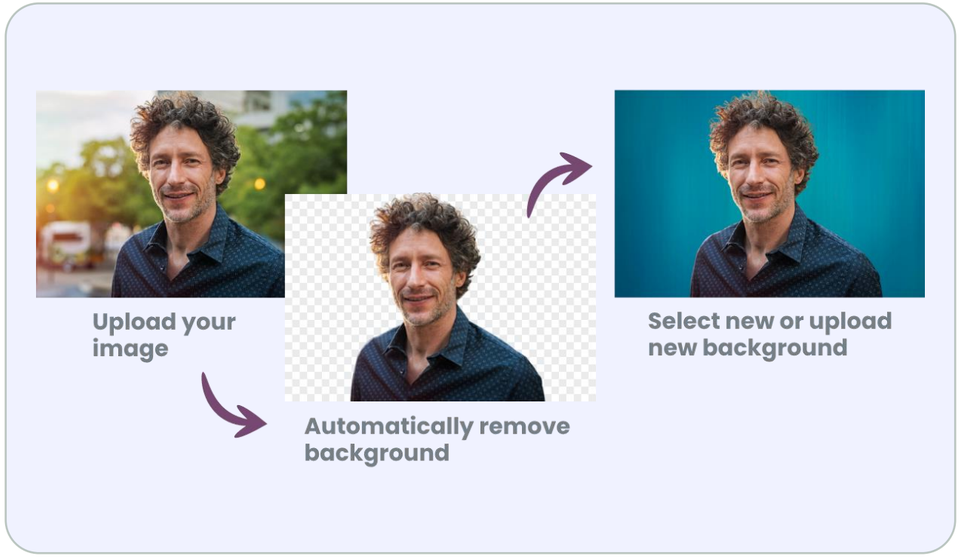
Apps Built Using TensorFlow Lite

Google Translate

It 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 have implemented this feature.

Most of these apps utilize TF Lite models to identify and isolate the person in the foreground while altering the background.

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

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

To convert a TensorFlow model into a TensorFlow Lite (TF Lite) model, you can use the TensorFlow Lite Converter. Here are the steps to follow:

Step 1	Install the required packages
Step 2	Save the TensorFlow model
Step 3	Convert the Saved model to TensorFlow Lite

Assisted Practice

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

- 9.09_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

ASSISTED PRACTICE

Key Takeaways

- Object detection determines and acknowledges the objects depicted in an image using bounding boxes.
- Data preparation involves data collection and data annotation.
- The YOLOv3 algorithm detects objects in images and videos.
- TensorFlow Lite Converter is used to convert a TensorFlow model into a TensorFlow Lite (TF Lite) model.

Knowledge Check

Knowledge Check

1

What is the first step in object detection?

A. Object localization

B. Object classification

C. Object segmentation

D. Object detection

Knowledge Check

1

What is the first step in object detection?

A. Object localization

B. Object classification

C. Object segmentation

D. Object detection

The correct answer is **B**

The first step in object detection is object classification, where the algorithm initially classifies the detected objects based on the given features.

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!