OBJECT DETECTION

(To Detect a Require Object)

Summer Internship Report Submitted in partial fulfilment of the requirement for undergraduate degree of

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING

By

Doddi Ramesh

HU21CSEN0300455

Under the Guidance of

Dr. Sreerama Murty M

Assistant Professor

Department Of Computer Science and Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

August 2024

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(GITAM)

(Declared as Deemed-to-be-University u/s 3 of UGC Act 1956)
HYDERABAD CAMPUS

DECLARATION

I submit this industrial training work entitled **"To Detect a Require Object"** to GITAM (Deemed to Be University), Hyderabad in partial fulfilment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science and Engineering**". I declare that it was carried out independently by me under the guidance of **Dr. Sreerama Murty M,** Asst. Professor, GITAM (Deemed to Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD	Name: Doddi Ramesh
Date: 01-08-2024	Student Roll No: HU21CSEN0300455

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY (GITAM)

(DEEMED TO BE UNIVERSITY)

HYDERABAD CAMPUS

GITAM
DEEMED TO BE UNIVERSITY

CERTIFICATE

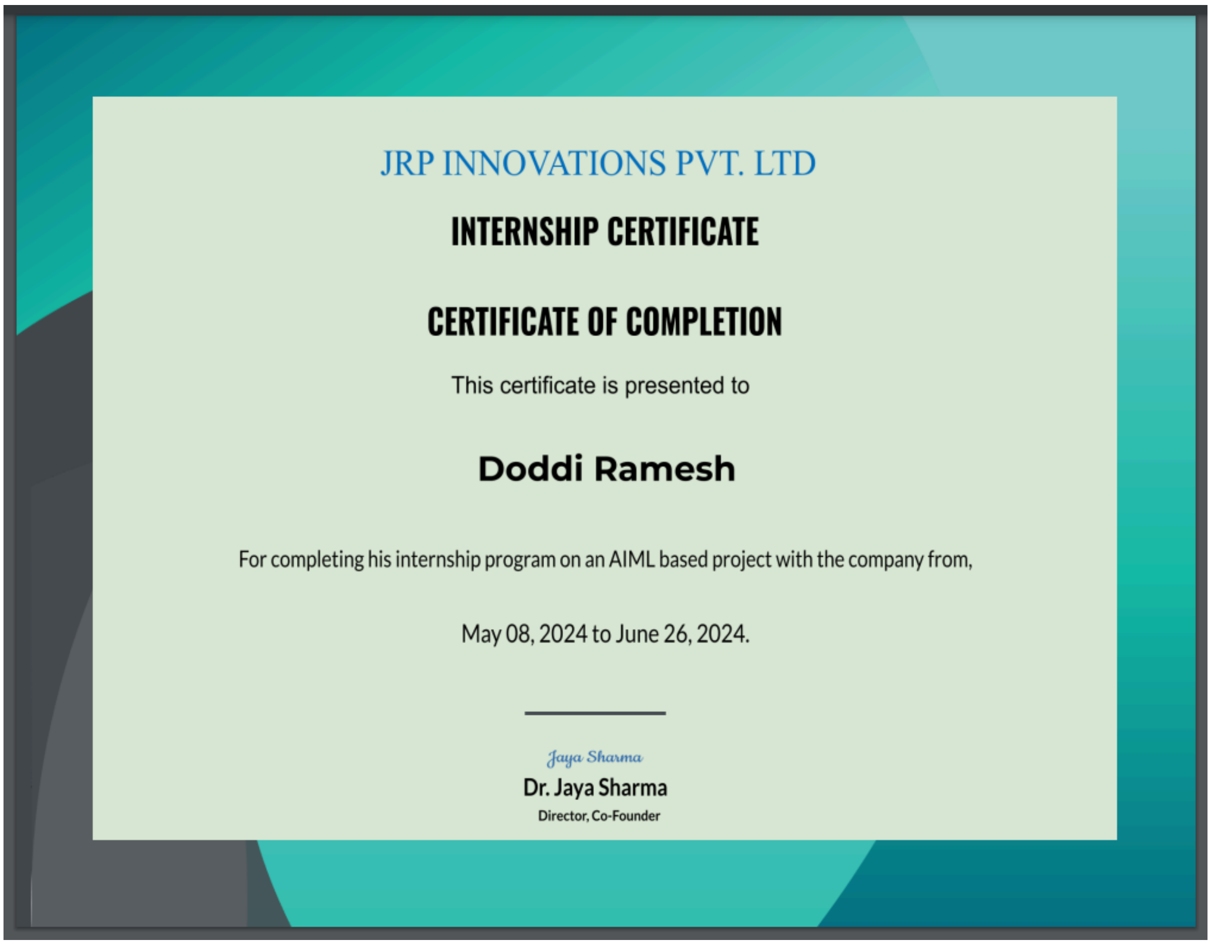
This is to certify that the Industrial Training Report entitled **"To Detect a Require Object"** is being submitted by Doddi Ramesh (HU21CSEN0300455) in partial fulfilment of the requirement for the award of **Bachelor of Technology in Computer Science And Engineering** at GITAM (Deemed to Be University), Hyderabad during the academic year 2024-2025.

It is faithful record work carried out by him at the **Computer Science and Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

[illegible]

Dr. Sreerama Murty M	**Dr. Mahaboob Basha Shaik**
Assistant Professor	Professor and HOD
Department of CSE	Department of CSE

CERTIFICATE OF COMPLETION



JRP INNOVATIONS PVT. LTD

INTERNSHIP CERTIFICATE

CERTIFICATE OF COMPLETION

This certificate is presented to

Doddi Ramesh

For completing his internship program on an AIML based project with the company from,

May 08, 2024 to June 26, 2024.

Jaya Sharma

Dr. Jaya Sharma

Director, Co-Founder

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. D S Rao**, Pro Vice Chancellor, GITAM Hyderabad and **Dr. N. Seetharamaiah**, Principal, GITAM Hyderabad.

I would like to thank respected **Dr. Mahaboob Basha Shaik,** Head of the Computer Science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Dr. Sreerama Murty M** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Doddi Ramesh

HU21CSEN0300455

ABSTRACT

This report presents a comprehensive analysis and evaluation of the YOLOv8 Object Detection Model along with various other available Object Detection Models. The process encompasses various stages, starting with collection of various information related to the Models, data preparation and extending through model training and performance assessment. Initially, the dataset is meticulously split into training and validation sets, ensuring a balanced distribution that facilitates effective learning and thorough evaluation.

The environment setup is a critical step, the YOLOv8 model is loaded, marking the commencement of the training phase. During training, the model is fine-tuned with specific parameters, including the number of epochs, batch size, and image size. These parameters are carefully chosen to optimize the model's learning process and enhance its object detection capabilities.

Following the training phase, the model undergoes a rigorous evaluation using key performance metrics such as precision, recall, mean Average Precision (mAP), and F1-score. These metrics are instrumental in providing detailed insights into the model's accuracy and overall effectiveness in detecting objects within images.

The report concludes that the YOLOv8 model, through systematic training and evaluation, demonstrates robust performance in object detection tasks. This robust performance highlights its potential for further enhancements and applications across various domains, paving the way for future advancements in the field.

Table of Contents

List of Figures

1. Object Detection Models

1.1 Yolo v8

1.1.1 Introduction:

YOLOv8, released in January 2023 by ultralytics , is the latest iteration in the You Only Look Once (YOLO) family of object detection models. YOLO models are known for their speed and accuracy, making them well-suited for real-time applications. YOLOv8 builds upon the success of previous versions by introducing new features and improvements for enhanced performance, flexibility, and efficiency.

1.1.2 Architecture:

YOLOv8 utilizes a single-stage detection architecture, meaning it predicts bounding boxes and class probabilities directly from the input image in a single forward pass.

Here's a breakdown of its key components:

o Backbone Network: YOLOv8 employs a custom convolutional neural network (CNN) architecture as its backbone. This network extracts feature maps from the input image, capturing essential information about objects and their locations.

o Focus (Path Aggregation Network): This novel component aggregates features from different depths within the backbone, allowing the model to better focus on relevant regions for object detection.

o Head Network: The head network takes the feature maps from the backbone and Focus module and generates bounding box predictions and class probabilities for each object in the image.

o Loss Function: YOLOv8 employs a composite loss function that combines classification loss (for object classes) with bounding box loss (for box localization accuracy).

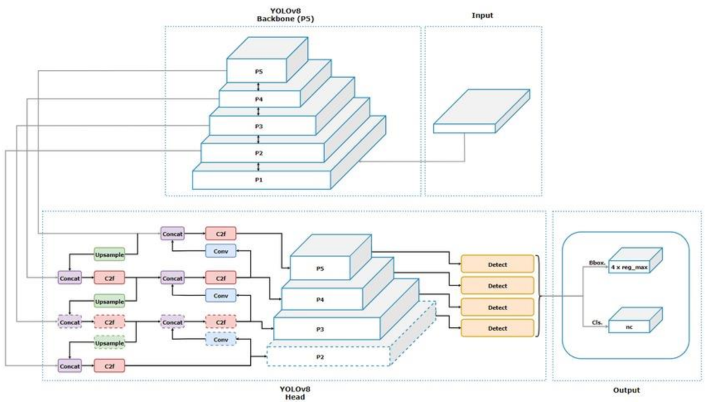


Fig 1.1. YOLO V8

1.1.3 Strengths:

o Accuracy: YOLOv8 achieves state-of-the-art performance on benchmark datasets like COCO, with variants like YOLOv8 (medium) reaching a mean Average Precision (mAP) of 50.2 at 1.83 milliseconds inference time on a Tesla A100 with TensorRT .

o Speed: YOLOv8 is known for its fast inference speed, making it suitable for real-time applications where processing delays need to be minimized.

o Flexibility: YOLOv8 supports various vision tasks beyond object detection, including image classification, pose estimation, and instance segmentation. This versatility makes it a powerful tool for diverse computer vision projects.

o Ease of Use: YOLOv8 comes with a Python package and a command-line interface (CLI) for easy deployment and customization.

1.1.4 Limitations:

o Accuracy Trade-off for Speed: While faster than some competitors, YOLOv8 might have slightly lower accuracy on specific datasets compared to models optimized purely for accuracy.

o Limited Customization: Compared to highly configurable models, YOLOv8 offers a more streamlined architecture with less room for extensive customization.

o Newer YOLO Versions: YOLOv9 and YOLOv10 were released in February and May 2024, respectively, potentially offering further advancements (although you might want to consider these newer versions depending on the report's timeframe).

1.2 Single-Shot Detector (SSD)

1.2.1 Introduction:

The Single-Shot Detector (SSD), introduced in 2015 by Wei Liu et al. is a foundational object detection model known for its balance between accuracy and speed. Unlike two-stage detectors that require separate proposal and classification stages, SSD performs both tasks in a single forward pass, making it significantly faster.

1.2.2 Architecture:

• Backbone Network:
SSD typically utilizes a pre-trained image classification network like VGG or ResNet as its backbone. This network extracts features from the input image, capturing information about objects and their locations.

- SSD Head:

This module operates on top of the feature maps extracted by the backbone. It consists of multiple convolutional layers that predict bounding boxes and class probabilities for objects at different scales and locations within the image. This is achieved by using a set of predefined anchor boxes with various sizes and aspect ratios. The SSD head refines these anchors to fit the actual objects in the image.

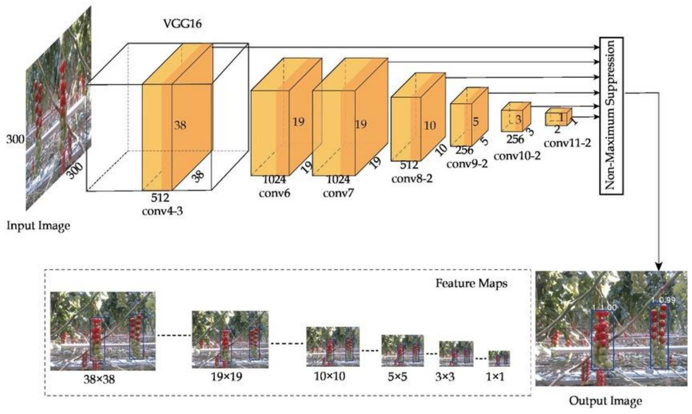


Fig 1.2. Single-Shot Detector

1.2.3 Strengths:

o Speed: SSD is significantly faster than two-stage detectors like R-CNN, making it suitable for real-time applications.

o Accuracy: SSD achieves competitive accuracy on object detection benchmarks, offering a good balance between speed and performance.

o Simplicity: The single-stage architecture makes SSD relatively simpler to train and deploy compared to two-stage detectors.

1.2.4 Limitations:

o Accuracy Trade-off: While faster than some models, SSD might have slightly lower accuracy on specific datasets compared to models optimized purely for accuracy.

o Fixed Anchor Boxes: The use of predefined anchor boxes can limit the model's ability to detect objects with unusual shapes or sizes.

o Resource Consumption: While faster than two-stage detectors, SSD still requires significant computational resources for training and inference.

Note:

Several variations of SSD exist, such as SSD MobileNet for mobile applications and RetinaNet for improved accuracy.

1.3 Faster R-CNN

1.3.1 Introduction:

Faster R-CNN, introduced in 2015 by Shaoqing Ren et al., is a significant advancement in the R-CNN object detection family. It addresses a key bottleneck in previous R-CNN models - the slow proposal generation stage. Faster R-CNN achieves high accuracy while offering a significant speed improvement, making it a popular choice for various object detection tasks.

1.3.2 Architecture:

Faster R-CNN combines two key components:

- Region Proposal Network (RPN):

This is a novel addition compared to previous R-CNN models. It's a small fully convolutional network that shares convolutional features with the main detection network. The RPN efficiently proposes candidate regions (bounding boxes) likely to contain objects in the image.

- Fast R-CNN Detector:

This builds upon the Fast R-CNN architecture, taking the proposed regions from the RPN and refining them to generate final object detections with class probabilities.

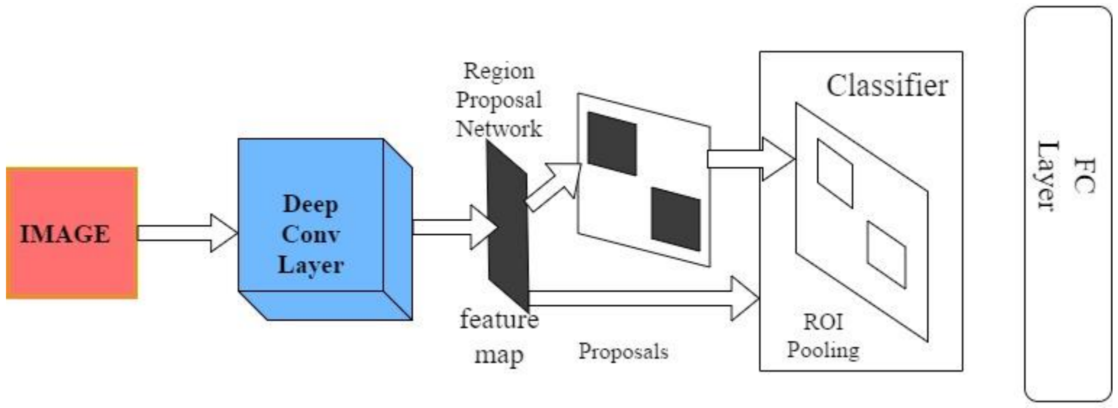


Fig 1.3. Faster R-CNN

1.3.3 Strengths:

o Accuracy: Faster R-CNN achieves state-of-the-art accuracy on benchmark datasets like PASCAL VOC, making it a highly reliable object detector.

o Speed: Compared to previous R-CNN models, Faster R-CNN significantly reduces proposal generation time, leading to faster overall detection speed.

o Shared Features: Sharing convolutional features between the RPN and the detection network improves efficiency and reduces training complexity.

1.3.4 Limitations:

o Computational Cost: While faster than previous R-CNN models, Faster R-CNN still requires more computational resources than single-stage detectors like SSD.

o Training Complexity: Training Faster R-CNN can be more complex compared to simpler models due to the two-stage architecture (RPN and detection network).

Note:

Faster R-CNN has spawned several successful variants, including Mask R-CNN for instance segmentation and FPN (Feature Pyramid Network) for improved multi-scale feature representation.

1.4 MobileNet

1.4.1 Introduction:

MobileNet, developed by Andrew Howard et al. in 2017 , is a family of lightweight convolutional neural networks (CNNs) designed specifically for mobile and embedded devices. These devices often have limited computational power and memory resources, making traditional CNNs impractical. MobileNet achieves good accuracy for image classification and object detection tasks while requiring significantly fewer parameters and computations.

1.4.2 Architecture:

The core concept of MobileNet lies in its use of depthwise separable convolutions. These convolutions break down the standard convolution operation into two separate steps:

o Depthwise Convolution: This applies a single filter to each input channel, extracting feature maps for each channel independently.

o Pointwise Convolution (1x1 Convolution): This uses a 1x1 convolution to combine the features from the depthwise convolution, reducing the number of parameters compared to a standard convolution.

By utilizing depthwise separable convolutions, MobileNet achieves significant reductions in model size and computational complexity while maintaining reasonable accuracy.

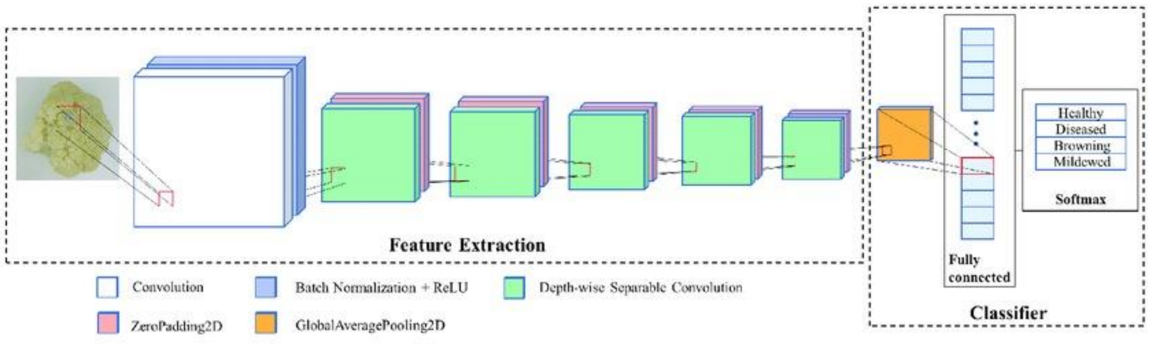


Fig 1.4. MobileNet

1.4.3 Strengths:

o Efficiency: MobileNet requires far fewer parameters and computations compared to traditional CNNs, making it ideal for deployment on resource-constrained devices.

o Accuracy: Despite its lightweight nature, MobileNet achieves competitive accuracy on various tasks, including image classification and object detection.

o Flexibility: MobileNet serves as a base network for various applications, such as MobileNet SSD for object detection and MobileNetV2 for further efficiency improvements.

1.4.4 Limitations:

o Accuracy Trade-off: While efficient, MobileNet may not achieve the same level of accuracy as larger, more complex models on specific tasks.

o Training Challenges: Training MobileNet can be more challenging than training standard CNNs due to its unique architecture.

Note:

Several variations of MobileNet exist, such as MobileNetV2 and MobileNetV3, offering further improvements in efficiency and accuracy.

1.5 ResNet (Residual Network)

1.5.1 Introduction:

ResNet, introduced by Kaiming He et al. in 2015, is a groundbreaking architecture for deep convolutional neural networks (CNNs). It addresses the vanishing/exploding gradient problem, a significant challenge in training very deep CNNs.

This problem hinders the network's ability to learn effectively as the signal from earlier layers diminishes or explodes when propagated through many layers. ResNet's innovative design allows for training much deeper networks compared to previous architectures, leading to significant improvements in accuracy on various computer vision tasks.

1.5.2 Architecture:

The core concept of ResNet lies in its use of residual blocks. These blocks are the building blocks of the network and introduce a shortcut connection that allows the gradients to flow directly through the network, bypassing some of the layers. This helps to preserve the information from earlier layers and allows for efficient training of very deep networks.

A typical residual block consists of:

- Batch Normalization (optional): Normalizes the activations of the previous layer for improved stability during training.

- Non-linear Activation (e.g., ReLU): Introduces non-linearity to the network's decision-making process.

- Convolutional Layers: These layers perform the core feature extraction tasks within the network.

- Shortcut Connection: This connection adds the input to the block directly to the output, bypassing the convolutional layers.

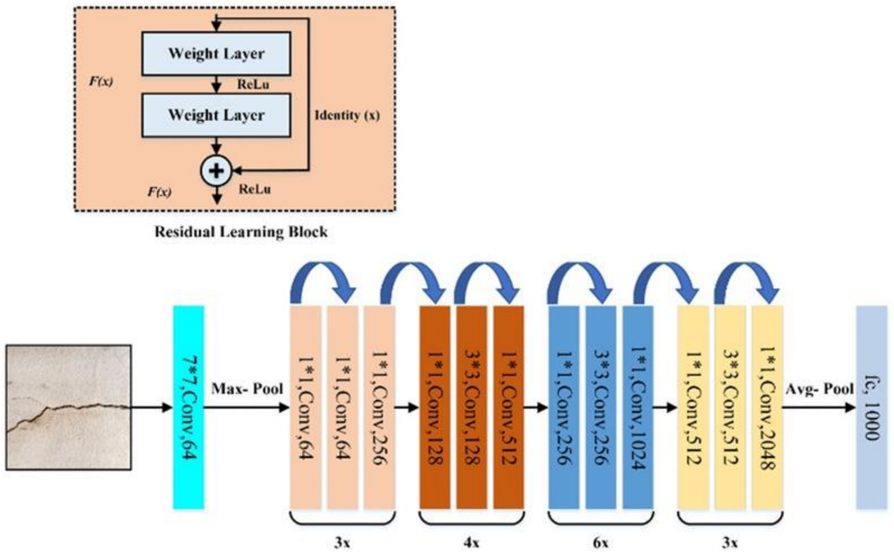


Fig 1.5. ResNet

1.5.3 Strengths:

○ Accuracy: ResNet architectures have achieved state-of-the-art performance on various image recognition and object detection tasks, demonstrating the effectiveness of deep learning with residual connections.

o Depth: ResNets can be trained with hundreds or even thousands of layers, capturing complex relationships within the data that shallower networks might miss.

o Efficiency: The shortcut connections in residual blocks help to alleviate the vanishing/exploding gradient problem, allowing for more efficient training of deep networks.

1.5.4 Limitations:

o Complexity: Designing and training ResNets can be more complex compared to simpler architectures due to the additional layers and shortcut connections.

o Computational Cost: While efficient for training, ResNets can still be computationally expensive for inference on resource-constrained devices.

o Variations: Numerous ResNet variants exist, with different depths and configurations, making it crucial to choose the right variant for the specific task.

Note:

Popular ResNet variants include ResNet-50, ResNet-101, and ResNet-152, which differ in the number of layers.
ResNets have been widely adopted as backbone networks for various computer vision tasks, including object detection (e.g., Faster R-CNN with ResNet) and semantic segmentation.

1.6 RetinaNet

1.6.1 Introduction:

RetinaNet, introduced in 2017 by Joseph Redmon and Ali Farhadi, is a single-stage object detection model designed to address challenges faced by previous models, particularly the class imbalance problem.

It builds upon the success of Fully Convolutional Networks (FCNs) and Feature Pyramid Networks (FPNs) while introducing a novel loss function to improve accuracy, especially for detecting small objects.

1.6.2 Architecture:

o Backbone Network: Similar to many object detection models, RetinaNet utilizes a pre-trained convolutional neural network (CNN) like ResNet or ResNeXt as its backbone. This network extracts feature maps from the input image, capturing essential information about objects and their locations.

o RetinaNet Subnetworks: On top of the feature maps from the backbone, RetinaNet employs two subnetworks:

o Classification Subnetwork: This predicts the probability of each object belonging to a specific class (e.g., car, person, dog).

o Regression Subnetwork: This predicts bounding boxes for the objects detected by the classification subnetwork.

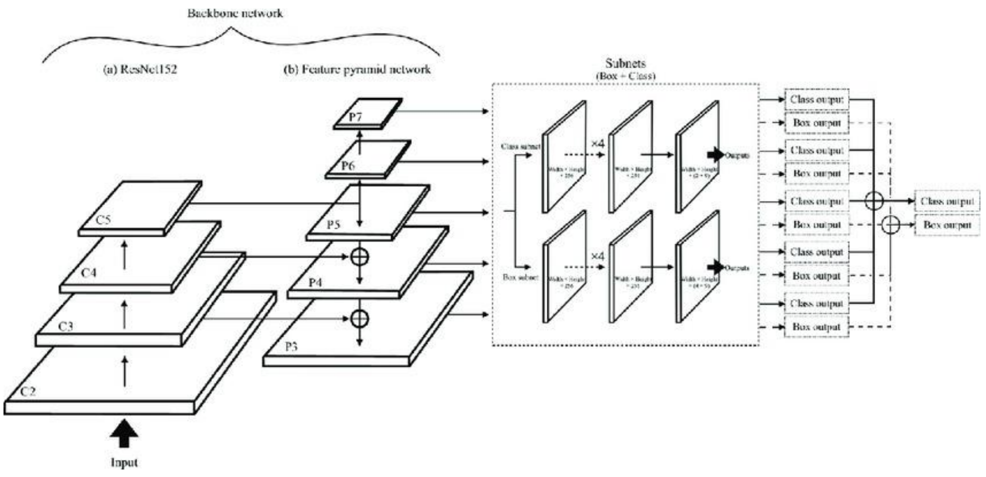


Fig 1.6. RetinaNet

1.6.3 Strengths:

o Accuracy: RetinaNet achieves state-of-the-art performance on object detection benchmarks, particularly for detecting small objects, thanks to its FPN and focal loss function.

o Speed: Compared to two-stage detectors, RetinaNet offers faster inference speed due to its single-stage architecture.

o Focus on Class Imbalance: The focal loss function specifically addresses the issue of class imbalance, a common challenge in object detection datasets where some classes have significantly fewer examples than others.

1.6.4 Limitations:

o Accuracy Trade-off: While faster than some models, RetinaNet might have slightly lower accuracy on specific datasets compared to models optimized purely for accuracy.

o Computational Cost: Training RetinaNet can require more computational resources than simpler models due to the additional subnetworks and loss function.

o Hyperparameter Tuning: Tuning the hyperparameters of the focal loss function can be crucial for optimal performance.

Note:

RetinaNet is a versatile model and can be fine-tuned for various object detection tasks beyond generic object categories.

Several variations of RetinaNet exist, such as RetinaNet-NAS for improved efficiency and performance.

1.7 Conclusion

1.7.1 A Strong Contender for Object Detection Needs

This report explored YOLOv8, a recent advancement in the You Only Look Once (YOLO) family of object detection models. Here's a breakdown of why YOLOv8 emerges as a compelling choice:

o Strong Performance: YOLOv8 achieves state-of-the-art accuracy on benchmark datasets, making it a reliable tool for real-world object detection tasks.
o Speed and Efficiency: Compared to some object detection models, YOLOv8 offers excellent inference speed, crucial for real-time applications where processing delays are critical.

o Flexibility: YOLOv8 extends beyond just object detection and supports various vision tasks, potentially broadening the scope of an object detection project.

o Ease of Use: The availability of a Python package and a CLI simplifies deployment and customization, making YOLOv8 accessible for experimentation and integration into a project.

1.7.2 Highlighting Specific Strengths:

If the internship focused on a specific aspect of object detection (e.g., real-time object tracking), emphasize how YOLOv8's speed benefits the application. For instance, you could say:

"YOLOv8's fast inference speed makes it well-suited for real-time object tracking applications, minimizing processing delays and enabling near-instantaneous detection."

1.7.3 Future Work and Advancements:

Briefly mention newer YOLO versions (YOLOv9 and YOLOv10) if they were released after the internship period. You can acknowledge their potential for further improvements while still highlighting the strengths of YOLOv8 for the timeframe of the work.

For example:
"While newer YOLO versions like YOLOv9 and YOLOv10 offer potential advancements, YOLOv8 remains a powerful choice for object detection tasks due to its accuracy, speed, and ease of use during the timeframe of this internship."

1.7.4 Overall Value Proposition:

Summarize why YOLOv8 stands out as a valuable choice for object detection tasks, considering its strengths, efficiency, and ease of use.

"In conclusion, YOLOv8 presents itself as a strong contender for various object detection needs. Its combination of high accuracy, fast inference speed, and user-friendly implementation makes it a compelling choice for researchers and developers working on real-world object detection applications."

2. Data Annotation

2.1 Introduction:

Data annotation is the process of labelling data available in a video, image or text. The data is labelled, so that models can easily comprehend a given data source and recognize certain formats, objects, information, or patterns in the future.

As part of the internship we had given an opportunity to work on annotations of the image data part of the project taken up by the company.

Out of various annotation tools available out there, Roboflow and labelimg turned out to be most promising ones for us. And we went with the LabelImg tool for annotation.

2.2 LabelImg Tool:

LabelImg is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. Besides, it also supports YOLO and CreateML formats. YOLO format here is the TXT file format.

The annotations in these files are saved with the co ordinates and index values for each annotations from the image.

2.3 Installation:

Steps to install the Tool.

2.3.1 Install Python 3:

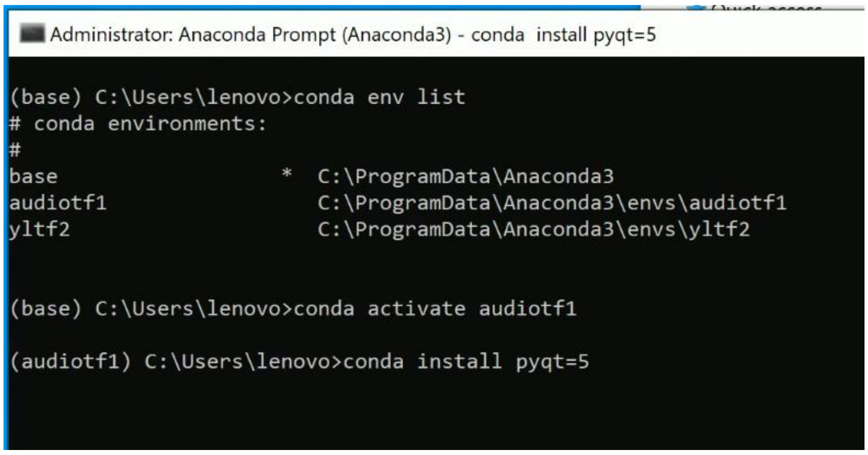
Make sure Python 3 is installed on your system. You can download it from python.org.

2.3.2 Install pip:

Ensure pip is installed. You can check with pip --version. If not installed, follow the instructions on pip's website.

2.3.3 Install PyQt5 and lxml:

Run pip3 install pyqt5 lxml to install the required libraries.



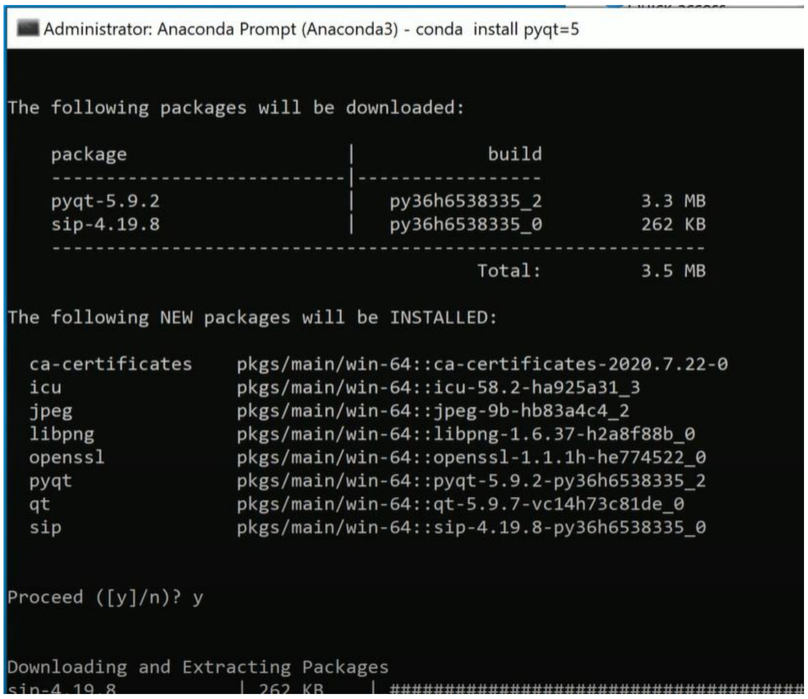
```
Administrator: Anaconda Prompt (Anaconda3) - conda  install pyqt=5

(base) C:\Users\lenovo>conda env list
# conda environments:
#
base                  *  C:\ProgramData\Anaconda3
audiotf1                 C:\ProgramData\Anaconda3\envs\audiotf1
yltf2                    C:\ProgramData\Anaconda3\envs\yltf2


(base) C:\Users\lenovo>conda activate audiotf1

(audiotf1) C:\Users\lenovo>conda install pyqt=5
```

Fig 2.1 Install pyqt5



```
Administrator: Anaconda Prompt (Anaconda3) - conda  install pyqt=5

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    pyqt-5.9.2                 |   py36h6538335_2         3.3 MB
    sip-4.19.8                 |   py36h6538335_0         262 KB
    ------------------------------------------------------------
                                           Total:         3.5 MB

The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2020.7.22-0
  icu                pkgs/main/win-64::icu-58.2-ha925a31_3
  jpeg               pkgs/main/win-64::jpeg-9b-hb83a4c4_2
  libpng             pkgs/main/win-64::libpng-1.6.37-h2a8f88b_0
  openssl            pkgs/main/win-64::openssl-1.1.1h-he774522_0
  pyqt               pkgs/main/win-64::pyqt-5.9.2-py36h6538335_2
  qt                 pkgs/main/win-64::qt-5.9.7-vc14h73c81de_0
  sip                pkgs/main/win-64::sip-4.19.8-py36h6538335_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
sip-4.19.8           | 262 KB    | ########################################
```

Fig 2.2 Install pyqt packages

2.3.4 Install labelImg:

Use pip3 install labelImg to install the LabelImg tool.

2.3.5 Verify Installation:

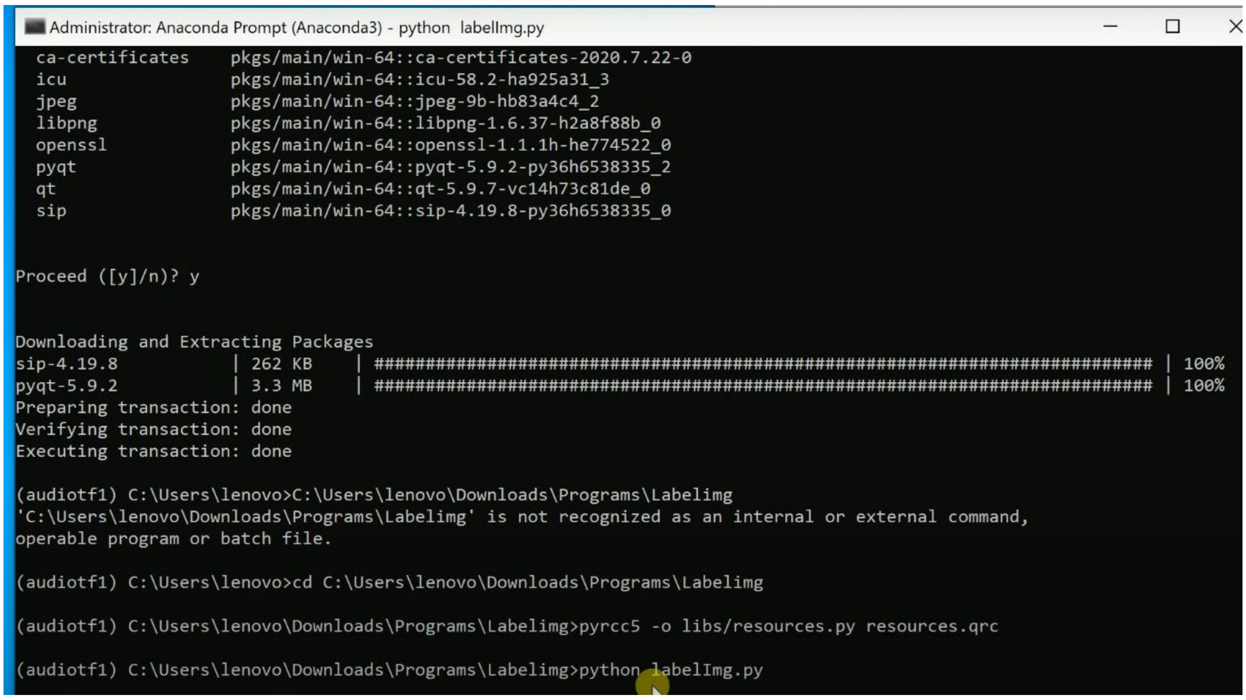
Run labelImg to verify the installation.

2.3.6 Running LabelImg with an Image:
Execute labelImg [IMAGE_PATH] [PRE-DEFINED CLASS FILE] to open LabelImg with a specific image and class file.

2.3.7 Navigate to the directory:
cd labelImg.

2.3.8 Compile Resources:
Run make qt5py3 to compile the resources.

2.3.9 Run LabelImg from Source:
Execute python3 labelImg.py to start LabelImg.
This will get you up and running with LabelImg on your system.


Administrator: Anaconda Prompt (Anaconda3) - python labelImg.py

```
  ca-certificates    pkgs/main/win-64::ca-certificates-2020.7.22-0
  icu                pkgs/main/win-64::icu-58.2-ha925a31_3
  jpeg               pkgs/main/win-64::jpeg-9b-hb83a4c4_2
  libpng             pkgs/main/win-64::libpng-1.6.37-h2a8f88b_0
  openssl            pkgs/main/win-64::openssl-1.1.1h-he774522_0
  pyqt               pkgs/main/win-64::pyqt-5.9.2-py36h6538335_2
  qt                 pkgs/main/win-64::qt-5.9.7-vc14h73c81de_0
  sip                pkgs/main/win-64::sip-4.19.8-py36h6538335_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
sip-4.19.8           | 262 KB    | ############################################################################ | 100%
pyqt-5.9.2           | 3.3 MB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(audiotf1) C:\Users\lenovo>C:\Users\lenovo\Downloads\Programs\Labeling
'C:\Users\lenovo\Downloads\Programs\Labeling' is not recognized as an internal or external command,
operable program or batch file.

(audiotf1) C:\Users\lenovo>cd C:\Users\lenovo\Downloads\Programs\Labeling

(audiotf1) C:\Users\lenovo\Downloads\Programs\Labeling>pyrcc5 -o libs/resources.py resources.qrc

(audiotf1) C:\Users\lenovo\Downloads\Programs\Labeling>python labelImg.py
```

Fig 2.3 Run LabelImg

2.4 working:

LabelImg is a graphical image annotation tool that allows users to label images for machine learning datasets. Here's how to use it:

Basic Workflow of LabelImg:

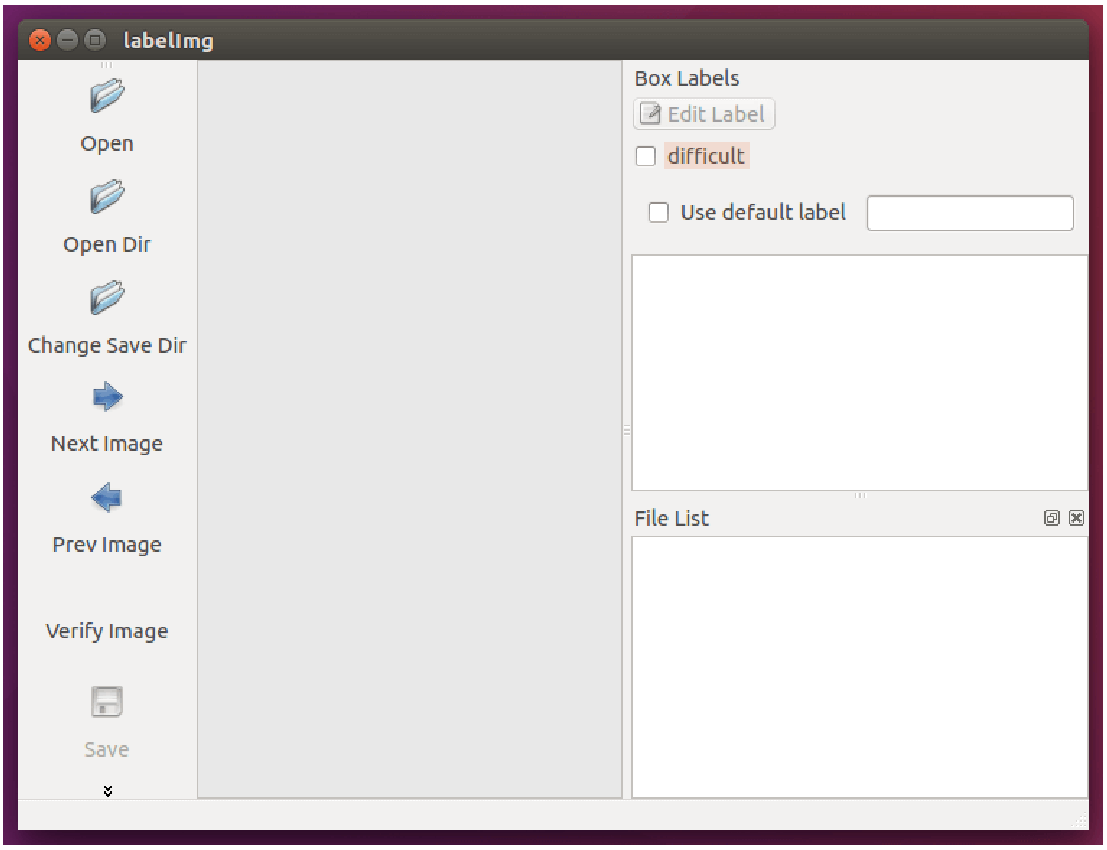


Fig 2.4 LabelImg Interface

2.4.1 Launch LabelImg:

Open LabelImg by running labelImg in your terminal.

2.4.2 Set Annotation Directory:

Go to Menu > File > Change default saved annotation folder to set the directory where your annotations will be saved.

2.4.3 Open Directory with Images:

Click Open Dir in the toolbar or go to Menu > Open Dir and select the directory containing the images you want to annotate.

2.4.4 Create RectBox (Annotation):

Click Create RectBox in the toolbar or press w.
Click and drag your mouse over the image to draw a bounding box around the object you want to annotate.

2.4.5 Label the Bounding Box:

After drawing a bounding box, a dialog will appear asking for the label.
Enter the label name and click OK.

2.4.6 Adjust Bounding Box:

You can resize and move the bounding box using your mouse.

Use arrow keys for fine adjustments.

2.4.7 Save Annotations:

Click Save in the toolbar or press Ctrl + S to save the annotations.
An XML file (for PASCAL VOC) or a text file (for YOLO) will be created in the specified annotation directory.

2.4.8 Navigate Through Images:

Use d to move to the next image and a to move to the previous image.
Ensure you save your annotations before moving to another image.

2.4.9 Edit Pre-defined Classes:

You can create a file named predefined_classes.txt in the data directory with a list of class names, each on a new line. This will populate the label dialog with predefined classes.

2.4.10 Switch Annotation Format:

By default, annotations are saved in PASCAL VOC format. To switch to YOLO format, click the PascalVOC button in the toolbar to toggle to YOLO.

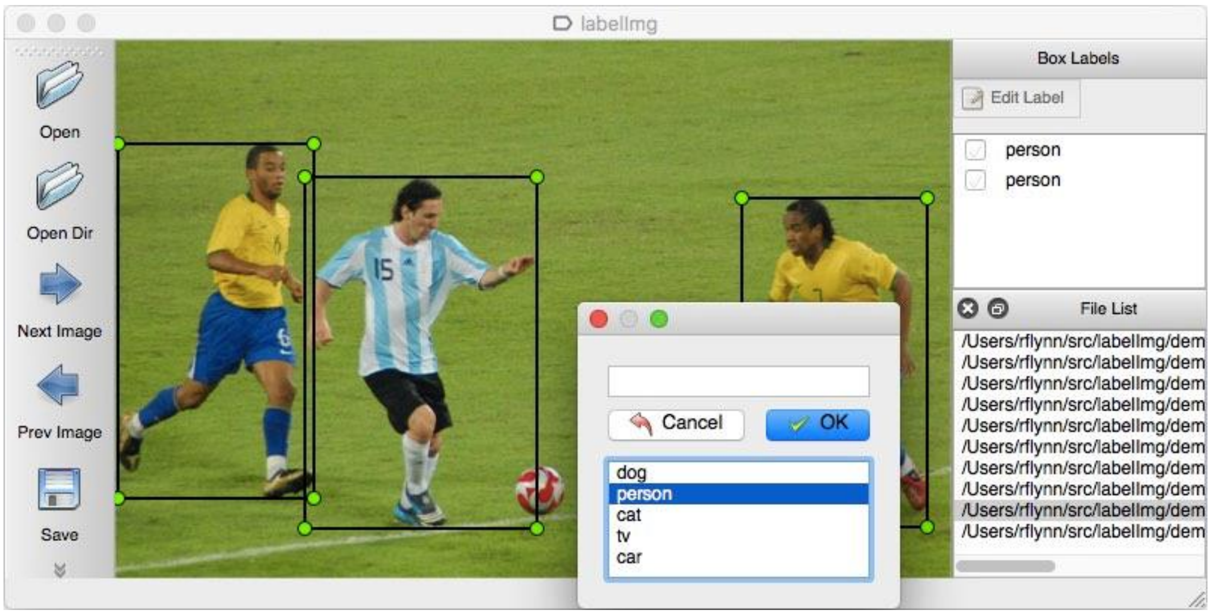
2.4.11 Additional Features:

2.4.11.1 Verify Images:

Press Space to flag the current image as verified, which helps in reviewing the dataset.

2.4.11.2 Hotkeys:

Use hotkeys to speed up the workflow, such as Ctrl + r to change the annotation directory, Ctrl + d to copy the current label and box, and del to delete the selected box.



/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem
/Users/rflynn/src/labelImg/dem

Fig 2.5 Annotation Example

3. Model Building and Evaluation

3.1 Introduction:

This report details the process of building, training, and evaluating a YOLOv8 model for object detection. The notebook provided consists of code cells that handle data splitting, environment setup, model loading, model training, and evaluation.

3.2 Data Splitting:

The initial step involves splitting the dataset into training and validation sets. The function `split_dataset`:
- Takes paths to the image and label directories.
- Shuffles the dataset.
- Splits it based on a specified training ratio (default is 80% training and 20% validation).
- Copies the files into newly created subdirectories for training and validation.

```python
import os
import random
import shutil
def split_dataset(image_dir, label_dir, train_ratio=0.8):
    image_dir = os.path.abspath(image_dir)
    label_dir = os.path.abspath(label_dir)
    images = os.listdir(image_dir)
    labels = os.listdir(label_dir)
    dataset = list(zip(images, labels))
    random.shuffle(dataset)
    split_index = int(len(dataset) * train_ratio)
    train_set = dataset[:split_index]
    val_set = dataset[split_index:]
    train_image_dir = os.path.join(image_dir, 'train')
    train_label_dir = os.path.join(label_dir, 'train')
    val_image_dir = os.path.join(image_dir, 'val')
    val_label_dir = os.path.join(label_dir, 'val')
    os.makedirs(train_image_dir, exist_ok=True)
    os.makedirs(train_label_dir, exist_ok=True)
    os.makedirs(val_image_dir, exist_ok=True)
    for image, label in train_set:
        shutil.copy(os.path.join(image_dir, image), train_image_dir)
        shutil.copy(os.path.join(label_dir, label), train_label_dir)
    for image, label in val_set:
        shutil.copy(os.path.join(image_dir, image), val_image_dir)
        shutil.copy(os.path.join(label_dir, label), val_label_dir)
    print(f'Training set: {len(train_set)} images')
    print(f'Validation set: {len(val_set)} images')
# Usage
split_dataset('/path/to/images', '/path/to/labels')
```

3.3 Environment Setup:

The notebook checks the PyTorch installation and CUDA availability to ensure the environment is properly configured. It then loads the YOLOv8 model.

```python
import torch from ultralytics import YOLO
# Check PyTorch installation
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
# Load YOLOv8 model
model = YOLO('yolov8n.pt')
```

3.4 Model Training:

The function `train_yolo_model` is defined to train the YOLOv8 model. This function:
- Clears the GPU cache to free up memory.
- Loads the YOLOv8 model.
- Sets training parameters, including epochs, batch size, and image size.
- Defines paths to the training and validation datasets.
- Initiates the training process.

~~~
```python
import os
import torch from ultralytics import YOLO
def train_yolo_model():
    try:
        # Clear GPU cache
        torch.cuda.empty_cache()
        # Load the YOLOv8 model
        print("Loading YOLOv8 model...")
        model = YOLO('yolov8n.pt')
        # Set training parameters
        epochs = 100
        batch_size = 16
        img_size = 640
       # Define dataset paths
        data = {
            'train': '/path/to/train/images',
            'val': '/path/to/val/images'
        }
# Start training
        model.train(data=data, epochs=epochs, batch_size=batch_size, img_size=img_size)
    except Exception as e:
        print(f"An error occurred during training: {e}")
# Usage
train_yolo_model()
```
~~~

3.5 Model Evaluation

After training the model, it is essential to evaluate its performance on the validation set. The notebook includes steps to run the model on the validation set and calculate the performance metrics such as precision, recall, mAP (mean Average Precision), and F1-score.

~~~
```python
# Evaluate the model on the validation set
metrics = model.val(data='/path/to/val/images')
print(metrics)
```
~~~

3.6 Output and Accuracy:

The evaluation metrics provide insights into the model's performance.
The key metrics include:

- Precision: The ratio of correctly predicted positive observations to the total predicted positives.

- Recall: The ratio of correctly predicted positive observations to all observations in the actual class.

- mAP (mean Average Precision): The mean of the average precision scores for each class.

- F1-score: The weighted average of Precision and Recall.

The evaluation output will include these metrics, allowing for an assessment of the model's accuracy and overall performance.

CONCLUSION

This report provides an overview of the process of building, training, and evaluating a YOLOv8 model. The notebook includes functions for data splitting, environment setup, model training, and evaluation. The model's performance metrics indicate its effectiveness in object detection tasks.

For further analysis, additional steps such as hyperparameter tuning, data augmentation, and model optimization can be performed to enhance the model's performance. The provided functions and evaluation metrics form a comprehensive approach to developing a robust object detection model using YOLOv8.

REFERENCE LINKS

Faster RCNN - https://arxiv.org/pdf/1506.01497

Yolo V8 - https://arxiv.org/abs/1506.02640

SSD - https://arxiv.org/pdf/1512.02325

Annotation Tool - https://github.com/HumanSignal/labelImg

Yolo V8 Code - https://github.com/pra-sh-ant/tank_rec