

REAL TIME TRAFFIC MANAGEMENT SYSTEM USING YOLOv3 ALGORITHM

*A Project report submitted in partial fulfillment of the requirements for
the award of the degree of*

**BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted by

Ch. Sai Gowtham (319126512078)

E. Rajkamal (319126512079)

T. S. N. Bhargava Ram (320126512L11)

N. Sandeep (319126512104)

B. Nayak (319126512072)

Under the guidance of

Dr. G. Manmadha Rao

M.E, Ph.D

**Professor, Department of ECE
& Head - Training & Placements**



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)**

*(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC)
Sanghivalasa, Bheemili Mandal, Visakhapatnam Dist. (A.P)*

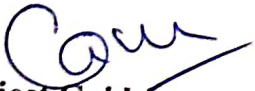
2022-2023

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)
(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with
'A' Grade)
Sanghivalasa, Bheemili Mandal, Visakhapatnam Dist. (A.P)

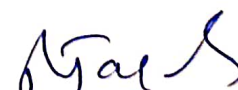


CERTIFICATE

This is to certify that the project report entitled "REAL TIME TRAFFIC MANAGEMENT SYSTEM USING YOLOv3 ALGORITHM" submitted by Ch. Sai Gowtham (319126512078), E. Rajkamal (319126512079), T. S. N. Bhargava Ram (320126512L11), N. Sandeep (319126512104), B. Nayak (319126512072) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics & Communication Engineering of Anil Neerukonda Institute of technology and Sciences (A), Visakhapatnam is a record of bona fide work carried out under my guidance and supervision.


Project Guide
Dr. G. Manmadha Rao
Professor, Department of ECE
& Head – Training & Placements
ANITS

Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sanghivalasa, Visakhapatnam-5311


Head of the Department
Dr. B. Jagadeesh
Professor & HOD
Department of ECE
ANITS

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Sciences
Sanghivalasa - 531 162

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Dr. G. Manmadha Rao**, Professor, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement.

We are grateful to **Dr. B. Jagadeesh**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sanghivalasa**, for their encouragement and cooperation in carrying out this work.

We express our thanks to all **teaching faculty** of the Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS, for providing great assistance in the accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last, but not least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

Ch. Sai Gowtham (319126512078)
E. Rajkamal (319126512079)
T. S. N. Bhargava Ram (320126512L11)
N. Sandeep (319126512104)
B. Nayak (319126512072)

ABSTRACT

Due to urbanization, there is a huge increase in vehicle usage day by day. The present automated traffic control systems are based on a fixed time concept, where signal timing is allocated equally for all lanes regardless of vehicle density in the lanes. As a result of this traffic congestion, a large amount of pollution and heat is generated, which internally affects society and the environment. To overcome these issues, we are proposing a model to monitor the density of vehicles in each lane using real-time video to allocate the dynamic time for vehicle and pedestrian passage. In addition, emergency vehicles can also be identified using this method to ensure a quick free passage through emergency lanes. This model uses YOLOv3 algorithm and Microsoft's COCO Dataset to identify the vehicle type and density in each lane. YOLO is a real-time object recognition system that can recognize multiple objects in a single frame. YOLO is based on a single Convolutional Neural Network (CNN). Yolo is a futuristic recognizer that outperforms current detectors in terms of accuracy and FPS. Real-time video can be analyzed for automatic vehicle detection using OpenCV and Python. OpenCV is a library of programming functions mainly aimed at real-time computer vision. The analyzed data also helps to minimize the traffic violators and this data is stored in the cloud and can be used for future reference to analyze traffic patterns. This model helps us to have an efficient and reliable traffic management system.

CONTENTS

LIST OF FIGURES	iv
LIST OF ABBREVIATIONS	v
LITERATURE REVIEW	1
CHAPTER 1: INTRODUCTION	3
1.1 Project Objective	3
1.2 Project Outline	3
CHAPTER 2: CONVOLUTION NEURAL NETWORKS	4
2.1 Introduction	4
2.2 Layers inside CNN	4
2.2.1 Convolution Layer	5
2.2.2 Pooling Layer	7
2.2.3 Fully Connected Layer	8
2.2.4 Non-Linearity Layers	8
2.3 Designing a Convolutional Neural Network	9
2.4 Applications of CNN	13
CHAPTER 3: METHODOLOGY	15
3.1 Introduction about YOLOv3 Algorithm	15
3.2 YOLOv3 Architecture	16
3.2.1 Description of Architecture	16
3.2.2 Architecture Diagram of YOLOv3	17
3.2.3 Details of Layers	17
3.2.4 Input Details for Model Inference	20
3.2.5 Details on Model Inference program and output	20
3.2.6 YOLOv3's Speed and Accuracy	22
CHAPTER 4: RESULTS	23
4.1 Summary of Code	23
4.2 Model creation and Training	24
4.3 Execution of Code	24
4.3.1 Procedure to run the software	24
4.4 Obtained Results	25
CHAPTER 5: TOOLS USED	27
5.1 Python	27
5.1.1 Introduction	27
5.1.2 Requirements	27
5.1.3 Features of Python	28
5.1.4 Setting Up Python	29

5.1.5 Conditional Statements in Python	30
5.1.6 Loops in Python	32
5.1.7 Python Dictionary	33
5.1.8 Functions in Python	33
5.2 Computer Vision	36
5.2.1 Open CV	36
5.2.2 Installation of Open CV on Windows	37
5.2.3 Read, Display, and write an Image using Open CV	38
5.3 COCO Dataset	41
5.3.1 Introduction to COCO	41
5.3.2 Inside COCO Dataset	42
5.3.3 MS COCO Dataset Classes	43
5.3.4 Using COCO Dataset	45
CONCLUSION	47
FUTURE SCOPE	48
REFERENCES	49
PUBLISHED PAPER STATUS	50

LIST OF FIGURES

Figure 2.2.1: Illustration of Convolution Operation	5
Figure 2.2.2: Pooling Operation	7
Figure 3.2.1: Architecture Diagram of YOLOv3	17
Figure 3.2.2: Object Detection Result using YOLOv3 Example 1	21
Figure 3.2.3: Object Detection Result using YOLOv3 Example 2	22
Figure 3.2.4: Object Detection Result using YOLOv3 Example 3	22
Figure 4.4.1: Output of Video 1	25
Figure 4.4.2: Output of Video 2	25
Figure 4.4.3: Output in Windows Terminal at the time of Video Processing	26
Figure 4.4.4: Output in Windows Terminal after Video Processing	26
Figure 5.2.1 Feature Extraction and Image Classification using Open CV	36
Figure 5.3.1: Image Classification using COCO Dataset	43
Figure 5.3.2: List of 80 classes in COCO Dataset	44
Figure 5.3.3: Graphical Representation of Classes v/s Number of images present in it.	45
Figure 5.3.4: Object Detection in Different Images	45
Figure 5.3.5: Representation of Instance Segmentation	46

LIST OF ABBREVIATIONS

YOLO	You Only Look Once
RTTMS	Real Time Traffic Management System
CNN	Convolution Neural Networks
ANN	Artificial Neural Networks
COCO	Common Objects in Context
IDE	Integrated Development Environment
RAM	Random Access Memory
GUI	Graphical User Interface
Open CV	Open Computer Vision
ReLU	Rectified Linear Units
API	Application Programming Interface

LITERATURE REVIEW

1. In the past, various methods for calculating traffic density have been used. A traffic control system using a surveillance system had been proposed in paper ^[1], but the density has been calculated using images and image processing in which the system has its own limitations. This technique requires a lot of time for calculating the traffic density.
2. In paper ^[2], the model uses Convolutional Neural Networks (CNNs), a type of deep learning algorithm, for image recognition and detection. These CNN networks were built using two data sets, MNIST and CIFAR-10. After evaluation of the performance, the results showed an accuracy of 99.6% but the speed has become a constraint for this model which can be overcome by using YOLOv3.
3. The extension to CNN's object detection algorithm is called YOLOv3. As a result, paper ^[3] provides a full explanation of CNN and deep learning. Artificial neural networks (ANNs) with several layers are referred to as deep neural networks. These layers have the capacity to manage massive volumes of data. Convolutional neural networks (CNNs), a subset of deep neural networks with several layers, have been proven to perform exceptionally well in machine learning tasks. This essay also seeks to describe and characterize the key components of CNN, how they function, and the factors that influence their effectiveness.
4. The paper ^[4] represents a co-training-based vehicle detection system for traffic monitoring and control. This system employs a Haar feature-based Classifier trained using AdaBoost algorithm to detect vehicles in low resolution cameras and scenarios where license plates are not visible. Haar feature-based classifier is a method for vehicle detection in computer vision. AdaBoost (Adaptive Boosting) is an ensemble machine learning algorithm that can be used to classify by combining multiple classifiers to form a strong classifier. The main usage of this model is to search for specific vehicles based on attributes such as color, time and date, speed, and direction of movement.
5. Based on the above papers, YOLOv3 was found to be the best algorithm for object detection. In paper ^[5], a lightweight vehicle detection and pedestrian algorithm was proposed, which will be helpful in finding the number of vehicles and pedestrians present in each lane at a particular point of time. But this algorithm has a limitation that it can only process images and these images are processed on a timely basis. Due to this

gap in time, the signal time calculation would be delayed which further causes issues in signal timings.

6. As mentioned earlier, the time calculation would be delayed if the algorithm proposed in paper ^[5], in paper ^[6], an optimization algorithm has been proposed. In this model, the YOLOv3 algorithm is implemented with the COCO dataset. This model takes input from the user, and with the help of the YOLOv3 algorithm, the model predicts the types of objects present in the given image and the accuracy gained by this model is 93%.
7. The paper ^[7] is about Microsoft's COCO Dataset. COCO stands for Common Objects in Context. As previously mentioned in paper ^[6], the YOLOv3 algorithm with the COCO data set has gained an accuracy of 93%. COCO Dataset consists of 91 object types, also called classes. These classes contain common objects such as vehicles, animals, food items etc. And in these classes, there are around 328,000 images, which are helpful in classifying multiple objects present in a single image. This data set consists of different types of vehicles such as bikes, cars, trucks, trains etc. and different types of animals such as dogs, cats, horses etc. Thus, any object detection algorithm that uses COCO Dataset will be able to categorize multiple objects though they are present in a single image.
8. The YOLO algorithm can be studied on the website mentioned in ^[8]. This website is made by the creator of the YOLO algorithm, and it also consists of YOLO configuration files and weight files which are prerequisites to implement this algorithm. This website also consists of comparisons of the YOLOv3 algorithm with other algorithms.

CHAPTER 1: INTRODUCTION

1.1 Project Objective

The objective of this project is to create a system that uses the YOLOv3 object identification algorithm to automatically detect and analyze traffic flow in real-time. The system aims to provide real-time traffic management insights to traffic authorities to help them make better decisions related to traffic flow and congestion. By improving traffic flow, reducing congestion, and minimizing delays, this system aims to enhance the overall traffic management and transportation experience for drivers and passengers alike.

1.2 Project Outline

Traffic congestion is a growing problem in urban areas around the world, leading to increased travel times, reduced productivity, and increased emissions. One approach to addressing this problem is using a real-time traffic management system (RTTMS), which can monitor traffic conditions in near real-time and adjust traffic signal timings, routing, and other parameters to improve traffic flow. There are numerous image processing and video processing algorithms that can be used in the real world as technology advances every day. One such technique is the well-liked object identification algorithm YOLOv3, which is renowned for its speed and accuracy in real-time. This document provides a summary of the most recent developments in RTTMS, including the data sources that are used to collect traffic data, the algorithms and models used to analyze and interpret this data, and the strategies and tactics used to control traffic flow. The system will use live video feeds from traffic cameras and apply the YOLOv3 algorithm to detect and classify different types of vehicles and their movements, including speed, direction, and density. The output of the system will be visualized using interactive dashboards and reports that provide real-time traffic management insights, such as the current traffic situation, congestion areas, and recommended traffic diversion routes.

CHAPTER 2: CONVOLUTION NEURAL NETWORKS

2.1 Introduction

Artificial neural networks known as convolutional neural networks (CNNs) are employed in the categorization and object detection of images. They have shown to be very successful in many computer vision applications and are inspired by the structure and operation of the visual cortex in the brain.

A CNN is fundamentally made up of numerous layers of connected nodes that process and interpret input data. When it comes to image classification, an image is the input, and the output is a label or group of labels that describe the contents of the image. Based on the convolutional neural network theory, it applies filters to the input image to extract meaningful information. Additional layers in the network subsequently process these features to classify the input image. CNNs are now often employed for a wide range of tasks, such as image and video identification, self-driving cars, medical diagnosis, and natural language processing. As a result of their great accuracy and effectiveness, they are a crucial tool for deep learning practitioners.

2.2 Layers inside CNN

Convolutional layer is the first layer of a CNN, where a series of filters (also known as kernels) are applied to the input image. Each filter slides across the image, producing a transformed version of the input based on the values of the pixels it overlaps. This process is referred to as convolution. The transformed representations produced by the filters are then passed through activation functions to produce a set of feature maps.

The feature mappings from the preceding layer are subjected to additional filtering in subsequent convolutional layers, which results in increasingly more abstract representations of the image. The spatial dimensions of the feature maps are subsequently decreased by feeding these altered representations into pooling layers. By making the network invariant to minor translations or distortions in the input image, this improves the resilience of the CNN.

The output from the previous layers is merged and processed in fully connected layers, which is where the final classification or detection result is produced, after the modified representations have been processed via numerous layers of convolution and pooling.

The capacity of CNNs to learn features from the input data is a significant feature. Based on the difference between the network's expected output and the true label during training, the weights of the filters and other network parameters are adjusted. The network gains the ability to identify and extract crucial information from the input photos over time, which it utilises to generate precise predictions.

Due to its capacity to learn features directly from the input data, handle massive volumes of data, and excel at tasks like object detection and image classification, CNNs have become the state-of-the-art in many computer vision applications. Furthermore, because convolution and pooling layers are used, CNNs are extremely parallelizable and may be used on GPUs for effective inference and training.

2.2.1 Convolution Layer

The convolution layer is the core element of the CNN. It bears most of the network's computational burden.

The kernel—a group of learnable parameters—and the limited region of the receptive field are two matrices that are combined in this layer to form a dot product. The kernel is deeper yet smaller in space than an image. This means that if an image has three (RGB) channels, the kernel height and width will be spatially tiny, but the depth will increase to include all three channels.

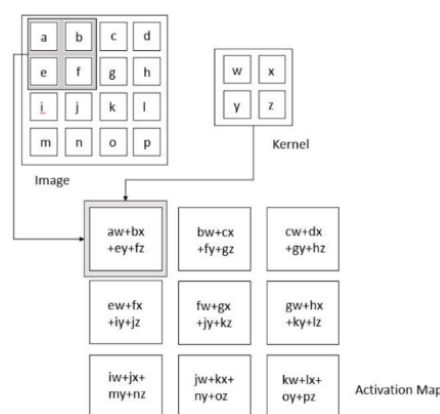


Figure 2.2.1: Illustration of Convolution Operation

During the forward pass, the kernel traverses the picture's height and width, generating an image of that receptive area. Therefore, an activation map—a two-dimensional representation of the picture—is produced, displaying the kernel's reaction at each spot in the image. The sliding size of the kernel is referred to as a stride.

If we have an input of size $W \times W \times D$, a D_{out} number of kernels with a spatial dimension of F , a stride S , and an amount of padding P , we can determine the output volume using the formula below:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Therefore, an output volume with the dimensions $W_{out} \times W_{out} \times D_{out}$ will be produced.

2.2.1.1 Motivation behind Convolution

Convolution makes advantage of the three essential concepts of sparse interaction, parameter sharing, and equivariant representation that inspired researchers in computer vision.

In simple neural network layers, matrix multiplication is used to explain how an input unit interacts with an output unit using a matrix of parameters. This suggests that all input and output devices are in communication with one another. Yet, there is little interaction between convolution neural networks. This is accomplished by making the kernel smaller than the input, so that, even when an image has millions or thousands of pixels, we are able to find important information by processing it with the kernel that is just tens or hundreds of pixels in size. This suggests that we need to store fewer parameters, which reduces the model's memory need and increases the statistical power of the model.

If a characteristic is relevant at another spatial position, it should be advantageous to calculate it at point (x_1, y_1) . (x_2, y_2) . It implies that neurons must use the same set of weights whether creating an activation map for a single two-dimensional slice or for a single activation map. Unlike to convolution networks, which employ shared parameters so that weights given to one input are also applied to other inputs and outputs, conventional neural networks only ever use a single member of the weight matrix. Due to parameter sharing, the layers of a convolution neural network will have the property of equivariance to translation. It claims that if we make specific changes to the input, the output will do the same.

Due to parameter sharing, the layers of a convolution neural network will have the property of equivariance to translation. It states that the output will change in accordance with how the input is changed.

2.2.2 Pooling Layer

By calculating an aggregate statistic from the surrounding outputs, the pooling layer serves as a stand-in for the network's output at specific locations. The representation's spatial dimension is lowered as a result, reducing the need for computation and weights. The pooling process is individually applied to each slice of the representation.

The L2 norm of the rectangular neighbourhood, the average of the rectangular neighbourhood, and one of the pooling functions are all influenced by the separation from the centre pixel. Nevertheless, the most used method is max pooling, which presents the most neighbourhood output.

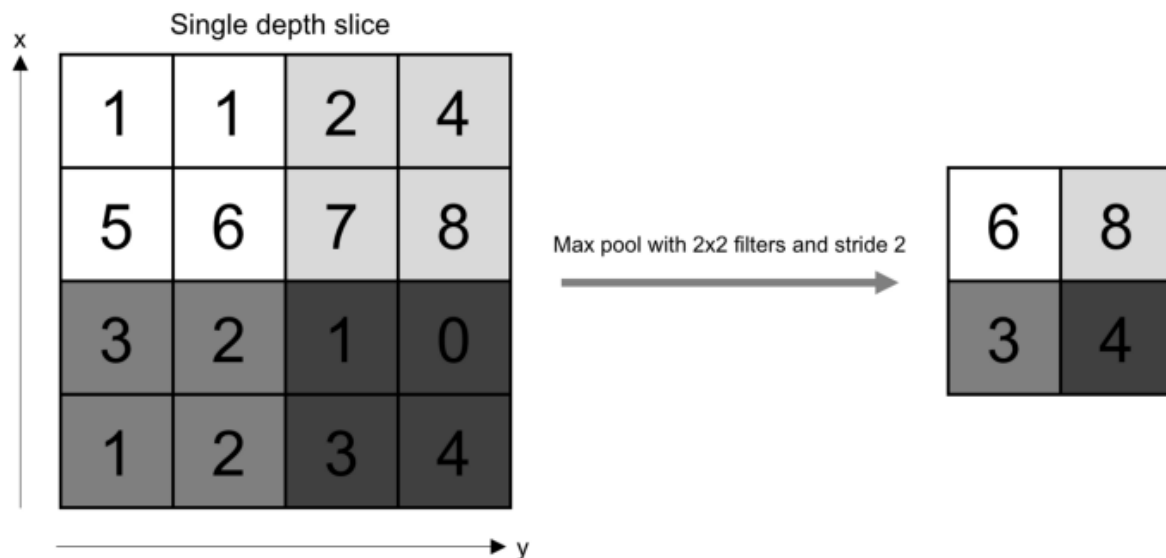


Figure 2.2.2: Pooling Operation

If we have a pooling kernel with a spatial dimension of F , an activation map with a size of $W \times W \times D$, and a stride of S , we can use the following formula to determine the size of the output volume:

$$W_{out} = \frac{W - F}{S} + 1$$

$W_{out} \times W_{out} \times D$ will be the output volume as a result.

An object can always be recognised regardless of where it is in the screen because pooling always offers some translation invariance.

2.2.3 Fully Connected Layer

Every neuron in this layer is completely linked to every other neuron in the layer before and after, just as in a traditional FCNN. Because of this, the calculation may be performed as usual utilising a matrix multiplication and bias effect.

The FC layer maps the representation between the input and the output. In a Convolutional Neural Network (CNN), a fully connected layer connects every neuron in one layer to every neuron in every other layer. These layers are often encountered after the application of multiple convolutional and pooling layers, towards the conclusion of the network. The convolutional layers' retrieved features are used by the fully connected layer to do classification. A weight is assigned to each piece of information that each neuron in the fully connected layer gets from every neuron in the layer below. These weights are altered throughout training to lessen the discrepancy in accuracy between the predicted and actual outputs. Generally, a probability distribution across all possible classes is provided by applying a SoftMax activation function to the output of the fully linked layer.

2.2.4 Non-Linearity Layers

Non-linearity layers are frequently included right after the convolutional layer to add non-linearity to the activation map because convolution is a linear process and images are anything but linear.

Non-linear operations come in a variety of forms, the most well-known being:

2.2.4.1 Sigmoid Nonlinearity

The sigmoid nonlinearity's mathematical counterpart:

$$\sigma(\kappa) = \frac{1}{1 + e^{-\kappa}}$$

The 0–1 range is "squished" to fit a real-valued number.

The gradient of a sigmoid nearly zeroes out when the activation takes place at either tail, which is a very unfavourable sigmoid property. The gradient will be effectively "killed" via backpropagation if the local gradient becomes too modest. The output of the sigmoid will either

be all positives or all negatives if the input to the neuron is always positive. This will cause a zigzag dynamic of gradient updates for weight.

2.2.4.2 Tanh Nonlinearity

A real-valued number is condensed by Tanh to the range $[-1, 1]$. The activation saturates like the sigmoid, however its output is zero-centered unlike the sigmoid neurons.

2.2.4.3 ReLU Nonlinearity

The Rectified Linear Unit (ReLU) has become quite popular recently. It does the calculation for the function $f(\kappa) = \max(0, \kappa)$. In other words, at 0 threshold, the activation just exists.

ReLU is more trustworthy than sigmoid and tanh and speeds up convergence by a factor of six.

However, ReLU could be sensitive during training, which is a disadvantage. Strong gradients that stop the neuron from ever updating further may update it. But we can make this work by selecting a suitable learning rate.

2.3 Designing a Convolutional Neural Network

Designing a Convolutional Neural Network (CNN) involves several steps, including:

1. Define the problem: Determine the problem you are trying to solve and identify the type of data you will be working with (e.g., images, audio, text).
2. Gather and pre-process the data: Collect enough data and pre-process it by scaling, normalizing, and augmenting it as necessary.
3. Choose a CNN architecture: Select an appropriate CNN architecture based on the problem and data type. Common architectures include VGG, ResNet, and Inception.
4. Set up the network: Select the quantity of layers, the size of the filters, and the strides for each layer. Choose whether ReLU or sigmoid will be used as the activation functions.

5. Train the network: Train the network by supplying it with batches of previously processed data and modifying the weights and biases of the neurons to reduce the loss function. Select a suitable optimizer, such as Stochastic Gradient Descent or Adam.
6. Assess the model: Check the trained model's accuracy on a validation set and make any necessary hyperparameter adjustments.
7. Apply the model: The model can be applied to forecast new data once it has been trained and assessed.

Each of these factors must be carefully taken into account while designing a CNN, and the fundamental ideas guiding neural networks and machine learning must also be well grasped.

Example: We can build a convolutional neural network using Fashion-MNIST, a dataset of Zalando article photographs with a training set of 60,000 samples and a test set of 10,000 cases. Each picture consists of a 28x28 grayscale graphic, and a label selected from one of 10 classes.

The following is our convolutional neural network's architecture:

```
[INPUT]
→ [CONV 1] → [BATCH NORM] → [ReLU] → [POOL 1]
→ [CONV 2] → [BATCH NORM] → [ReLU] → [POOL 2]
→ [FC LAYER] → [RESULT]
```

With a stride size of 1 and padding of 2, we will use a spatial kernel that is 5 x 5 in size. For both pooling layers, we'll use the maximum pool operation with kernel size 2, stride 2, and no padding.

Calculations:

CONV 1

Input Size ($W_1 \times H_1 \times D_1$) = $28 \times 28 \times 1$

- Requires four hyperparameter:
 - Number of kernels, $k = 16$
 - Spatial extend of each one, $F = 5$
 - Stride Size, $S = 1$
 - Amount of zero padding, $P = 2$
- Outputting volume of $W_2 \times H_2 \times D_2$
 - $W_2 = (28-5+2(2))/ 1+1 = 28$
 - $H_2 = (28-5+2 (2))/ 1+1 = 28$

- $D_2 = k$

Output of Conv 1 ($W_2 \times H_2 \times D_2$) = $28 \times 28 \times 16$

POOL 1

Input Size ($W_2 \times H_2 \times D_2$) = $28 \times 28 \times 16$

- Requires two hyperparameter:
 - Spatial extend of each one, $F = 2$
 - Stride Size, $S = 2$
- Outputting volume of $W_3 \times H_3 \times D_2$
 - $W_3 = (28-2)/ 1+1 = 14$
 - $H_3 = (28-2)/ 1+1 = 14$

Output of Pool 1 ($W_3 \times H_3 \times D_2$) = $14 \times 14 \times 16$

CONV 2

Input Size ($W_3 \times H_3 \times D_2$) = $14 \times 14 \times 16$

- Requires four hyperparameter:
 - Number of kernels, $k = 32$
 - Spatial extend of each one, $F = 5$
 - Stride Size, $S = 1$
 - Amount of zero padding, $P = 2$
- Outputting volume of $W_4 \times H_4 \times D_3$
 - $W_4 = (14-5+2(2))/ 1+1 = 14$
 - $H_4 = (14-5+2 (2))/ 1+1 = 14$
 - $D_3 = k$

Output of Conv 2 ($W_4 \times H_4 \times D_3$) = $14 \times 14 \times 32$

POOL 2

Input Size ($W_4 \times H_4 \times D_3$) = $14 \times 14 \times 32$

- Requires two hyperparameter:
 - Spatial extend of each one, $F = 2$

- Stride Size, $S = 2$
- Outputting volume of $W_5 \times H_5 \times D_3$
 - $W_5 = (14-2)/ 2+1 = 7$
 - $H_5 = (14-2)/ 2+1 = 7$

Output of Pool 2 ($W_5 \times H_5 \times D_3$) = $7 \times 7 \times 32$

FC Layer

Input Size ($W_5 \times H_5 \times D_3$) = $7 \times 7 \times 32$

Output Size (Number of Classes) = 10

Code snippet for defining the convolution network:

```
class convnet1(nn.Module):
    def __init__(self):
        super(convnet1, self).__init__()
        # Constraints for layer 1
        self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size
                               = 5, stride = 1, padding = 2)
        self.batch1 = nn.BatchNorm2d(16)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size
                                   = 2) #default stride is equivalent to the kernel_size
        # Constraints for layer 2
        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size
                               = 5, stride = 1, padding = 2)
        self.batch2 = nn.BatchNorm2d(32)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size = 2)
        # Defining the Linear layer
        self.fc = nn.Linear(32 * 7 * 7, 10)
        # defining the network flow
    def forward(self, x):
        # Conv 1
```

```

out = self.conv1(x)
out = self.batch1(out)
out = self.relu1(out)
# Max Pool 1
out = self.pool1(out)
# Conv 2
out = self.conv2(out)
out = self.batch2(out)
out = self.relu2(out)
# Max Pool 2
out = self.pool2(out)
out = out.view(out.size(0), -1)
# Linear Layer
out = self.fc(out)
return out

```

The network additionally uses batch normalisation, which expressly mandates that the network adopt a unit Gaussian distribution. This avoids improper weight matrix initialization. Cross-entropy has been used as the loss function during training, along with the Adam Optimizer and a learning rate of 0.001. For the test dataset, the model is 90% accurate after training.

2.4 Applications of CNN

Convolutional neural networks (CNNs) are frequently employed in many different applications, such as:

1. Image recognition: CNNs are important in applications like self-driving cars, security systems, and medical diagnosis because they can precisely recognise objects, people, and animals in photographs and videos.
2. Natural language processing: CNNs can be used to analyse and process text data, making them useful in applications such as sentiment analysis, language translation, and chatbots.
3. Speech recognition: CNNs can be used to analyse and classify speech signals, making them useful in applications such as voice assistants and speech-to-text transcription.

4. Recommendation systems: CNNs are valuable in applications like product suggestions and personalised marketing because they can be used to forecast customer preferences based on prior activity.
5. Robotics: CNNs can be used to process sensor data and make decisions for autonomous robots, making them useful in applications such as manufacturing, agriculture, and space exploration.
6. Medical imaging: CNNs can be used to analyse medical images and detect abnormalities, making them useful in applications such as cancer diagnosis and radiology.

Overall, CNNs have demonstrated outstanding performance in a variety of applications, making them a crucial tool for academics, engineers, and developers in a variety of industries.

CHAPTER 3: METHODOLOGY

3.1 Introduction about YOLOv3 Algorithm

The third iteration of the well-known object detection system YOLO (You Only Look Once), sometimes known as YOLOv3, is called YOLOv3. Because it is a single shot multi-box detection system, it can recognise several objects in an image during a single network pass. This makes YOLOv3 extremely efficient since, unlike previous object detection algorithms, it does not need to scan the image more than once.

The input image is divided into a grid of cells by YOLOv3, and each cell oversees determining the existence and placement of items in its region. To identify the existence and placement of objects in the image, the network generates a set of bounding boxes for each cell along with the related class probabilities. YOLOv3 gains the ability to forecast the position and dimensions of the bounding boxes as well as the class probabilities for each object in the image during the training phase.

The use of anchor boxes in YOLOv3 is one of its primary characteristics. A collection of bounding boxes called "anchor boxes" is utilised to increase the accuracy of the network's predictions. Each anchor box is given a cell in the grid by YOLOv3, which modifies the form and size of each box during training to better match the location and size of the objects in the image.

In addition to these enhancements, YOLOv3 additionally makes use of residual connections and up sampling layers to boost prediction accuracy over earlier iterations. Additionally, YOLOv3 makes use of anchor boxes with various aspect ratios to manage objects of various sizes and forms.

One of the fastest and most reliable object detection algorithms currently in use is YOLOv3. It can operate in real-time on a GPU and can recognise a variety of things, such as people, cars, animals, and more. Furthermore, it is very scalable since the network can be trained on a big dataset to increase accuracy or fine-tuned on a smaller dataset to recognise certain items.

YOLOv3 is a strong and effective object recognition technique that has gained widespread acceptance in several computer vision applications. It is an appealing option for many use cases due to its quick, real-time performance, excellent precision, and capacity to handle objects of various shapes and sizes.

3.2 YOLOv3 Architecture

3.2.1 Description of Architecture

3.2.1.1 YOLOv3 object detection procedures

- The input is a collection of form images ($m, 416, 416, 3$).
- This image is sent to a convolutional neural network by YOLO v3. (CNN).
- The output volume obtained by flattening the final two dimensions of the output is $(19, 19, 425)$:
 - Here, each cell of a 19×19 grid returns 425 numbers.
 - $425 = 5 * 85$, where 5 is the number of anchor boxes per grid.
 - $85 = 5 + 80$, where 5 is (pc, bx, by, bh, and bw) and 80 is the total number of classes we're looking to find.
- The output provides a list of bounding boxes and any classes that were found. Six numerals are used as symbols for each bounding box (pc, bx, by, bh, dw, and c). Each bounding box in an 80-dimensional vector of c is represented by 85 values.
- To prevent choosing overlapping boxes, we next do the IoU (Intersection over Union) and Non-Max Suppression.

3.2.1.2 Regarding the Architecture

- A Darknet derivative called YOLOv3 makes use of an Imagenet-trained 53-layer network.
- For the purpose of detection, 53 more layers are added, giving YOLOv3 a 106-layer, fully convolutional underlying architecture.
- Applying 1×1 detection kernels to feature maps of three different sizes at three different locations across the network is how YOLO v3's detection procedure is carried out.
- Dimensions of the detecting kernel are $1 \times 1 \times (B \times (5 + C))$. Thus, "5" denotes the four bounding box characteristics and one object confidence, and "B" denotes the maximum

number of bounding boxes that a feature map cell is capable of predicting. There are "C" number of classes.

- YOLOv3 employs logistic regression to estimate object confidence and class predictions and binary cross-entropy to calculate the classification loss for each label.

3.2.1.3 Hyper-parameters utilised

- class threshold - Determines the threshold for the projected item's likelihood.
- Non-Max suppression Threshold: This threshold prevents the same object from being recognised more than once in a picture. To do this, the highest probability boxes are chosen, while the neighbouring, lower probability boxes are suppressed. (Below the established threshold).
- Input image size, input height, and input shape.

3.2.2 Architecture Diagram of YOLOv3

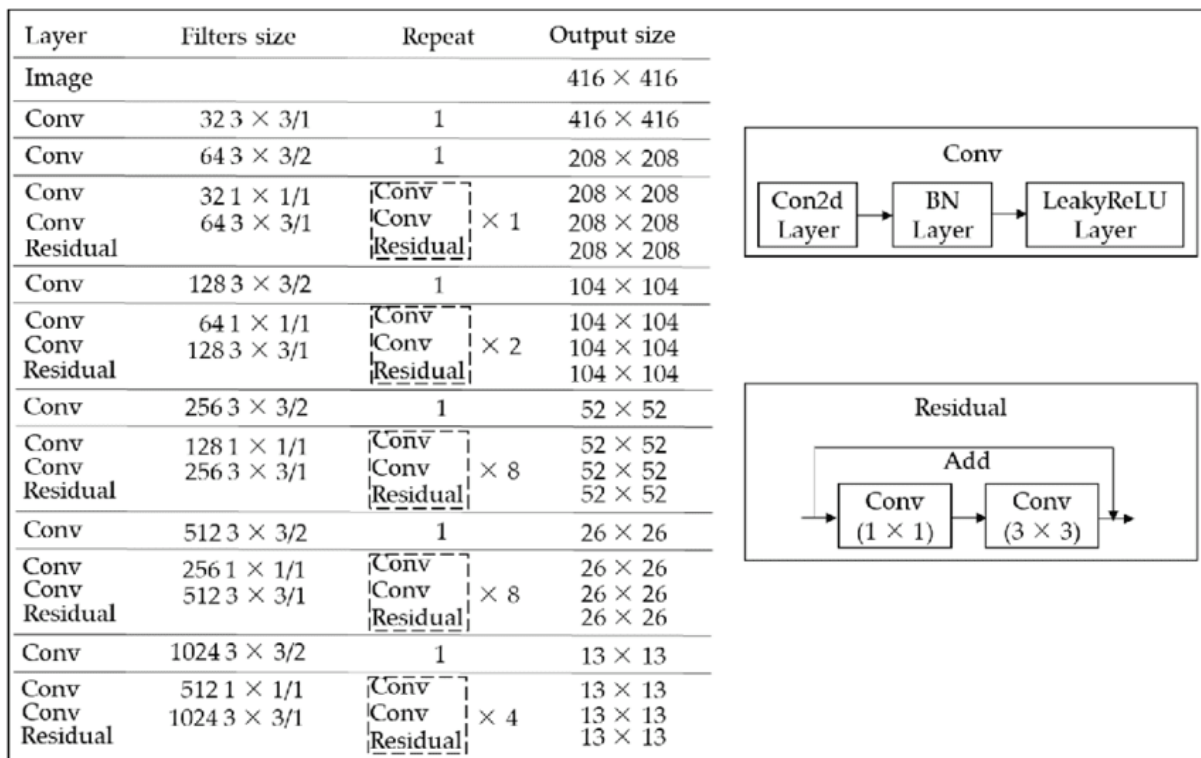


Figure 3.2.1: Architecture Diagram of YOLOv3

3.2.3 Details of Layers

When only convolutional layers are used, YOLO is a fully convolutional network (FCN). A more sophisticated feature extractor by the name of Darknet-53 is used in YOLOv3.

3.2.3.1 Convolution Layers in YOLOv3

Like other convolutional neural networks, YOLOv3's architecture is mostly composed of convolution layers. In order to build representations that are helpful for the job of object detection, convolutional layers are employed to extract features from the input picture.

The input layer, the backbone network, and the detection layers are the three primary components of the YOLOv3 deep neural network architecture in which the convolutional layers are placed. The backbone network oversees extracting features from the pre-processed picture after the input layer pre-processes the raw image that it gets as input.

The YOLOv3 backbone network is made up of several convolutional layers with various filter sizes and strides, followed by residual connections that enhance gradient flow and aid in preventing disappearing gradients during training. Darknet-53, a deep neural network with 53 layers that serves as the feature extractor, is a component of the backbone network.

The detection layers in YOLOv3 forecast the bounding boxes and class probabilities for each item in the picture. These layers use the feature map generated by the backbone network as input and use a series of convolutional layers and up sampling layers to build the final output.

Overall, the convolution layers of YOLOv3 are crucial to the network's capacity to recognise objects quickly and accurately.

3.2.3.2 Darknet – 53 in YOLOv3

The YOLOv3 object identification technique is built on the convolutional neural network architecture known as Darknet-53. A deep neural network with 53 layers called Darknet-53 makes advantage of residual connections to enhance gradient flow and stop gradients from disappearing during training.

Darknet-53 was developed by the creator of YOLO, Joseph Redmon, as an alternative to the widely used ResNet architecture. It was designed specifically for object detection tasks and has shown superior performance on image classification benchmarks such as ImageNet.

In YOLOv3, the input picture is processed by Darknet-53 to create a feature map, which is then applied to the input image for object detection. Additional convolutional layers are used to the feature map during processing in order to forecast bounding boxes and class probabilities for each item in the picture.

Overall, Darknet-53 is an important component of YOLOv3 and contributes to its speed and accuracy in real-time object detection.

3.2.3.3 *Different Layers inside YOLOv3*

Code for Layer 1 to 53 in Tensorflow: Consider `res_block()` method for below code

```
def res_block(inputs, filters):
    shortcut = inputs
    net = conv2d(inputs, filters * 1, 1)
    net = conv2d(net, filters * 2, 3)
    net = net + shortcut
    return net

#First two conv2d layers with 32 and 64 filters
net = conv2d(inputs, 32, 3, strides = 1)
net = conv2d(net, 64, 3, strides = 2)
res_block * 1
net = res_block(net, 32)

# Convolutional block with 128 filters
net = conv2d(net, 128, 3, strides = 2)
res_block * 2
for i in range(2):
    net = res_block(net, 64)

# Convolutional layer with 256 filters
net = conv2d(net, 256, 3, strides = 2)
res_block * 8
for i in range(8):
    net = res_block(net, 128)

# Convolutional layer with 512 filters
route_1 = net
```

```

net = conv2d(net, 512, 3, strides = 2)
res_block * 8
for i in range(8):
    net = res_block(net, 256)
# Convolutional layer with 1024 filters
route_2 = net
net = conv2d(net, 1024, 3, strides = 2)
res_block * 4
for i in range(4):
    net = res_block(net, 512)
route_3 = net

```

3.2.4 Input Details for Model Inference

- i. Input Pre-processing: Before being input into our algorithm, the images must be scaled to 416×416 pixels. Providing the dimensions while the Python code is running is one option.
- ii. Input Dimensions: While the user may choose to select a different size, the model anticipates inputs to be coloured pictures with square shapes measuring 416×416 pixels.

3.2.5 Details on Model Inference program and output

3.2.5.1 The output of the model

- The outcome gives a list of bounding boxes and any discovered classes.
- Six integers are used to symbolise each bounding box (pc, bx, by, bh, dw, and c). The number of classes we want to forecast is represented by the 80-dimensional vector c , and each bounding box is represented by 85 values.

3.2.5.2 Post-processing Results

- The output of the inference software is a collection of numpy arrays that are provided with their shapes. Although being encoded, these arrays already know what their bounding boxes and class labels will be.

- The candidate bounding boxes and class predictions are then decoded for each of the numpy arrays. Bounding boxes with class probabilities below the threshold value of 0.3 and that do not clearly define an object will be disregarded.
- In this case, a picture can take a maximum of 200 bounding boxes into account.
- The bounding box coordinates have been translated using the `correct_yolo_boxes()` function so that the original picture may be plotted and rendered.
- The overlap will once again be assessed using the non-max suppression threshold = 0.45 in order to exclude candidate bounding boxes that could be referring to the same item.
- The bounding boxes' coordinates also need to be scaled back to match the original picture, and each one must have the title and scores displayed on top of it.
- Before the bounding boxes and recognised classes in our output image are obtained, all these post-processing procedures must be completed.

3.2.5.3 Model output from the inference program

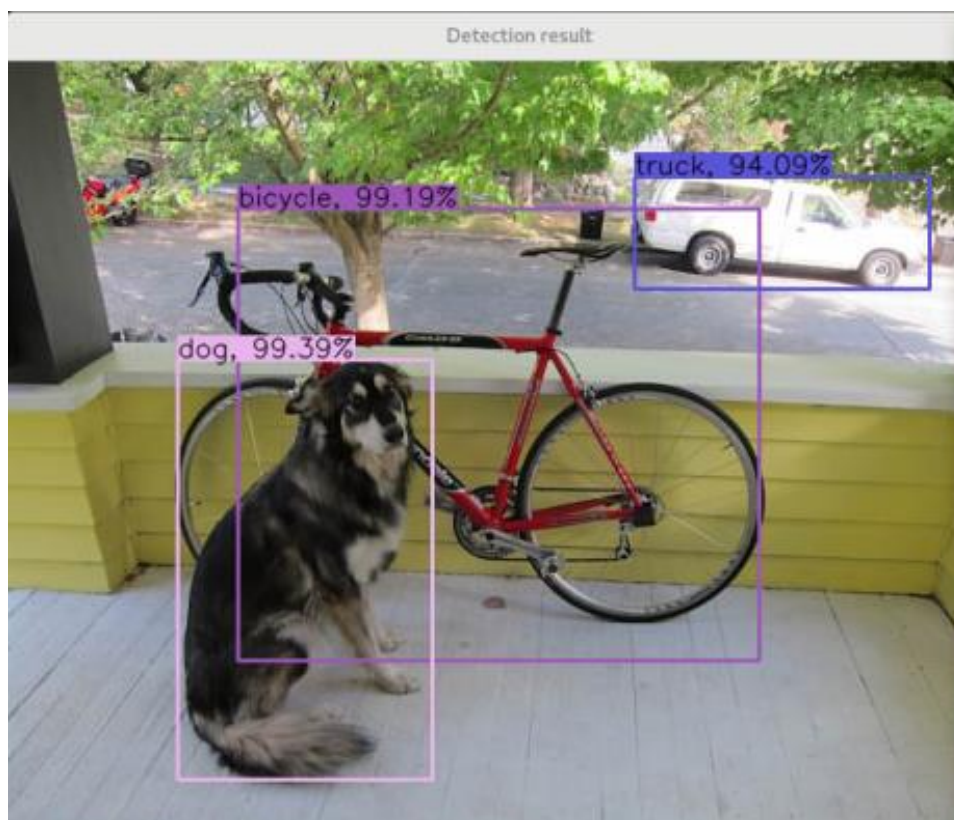


Figure 3.2.2: Object Detection Result using YOLOv3 Example 1

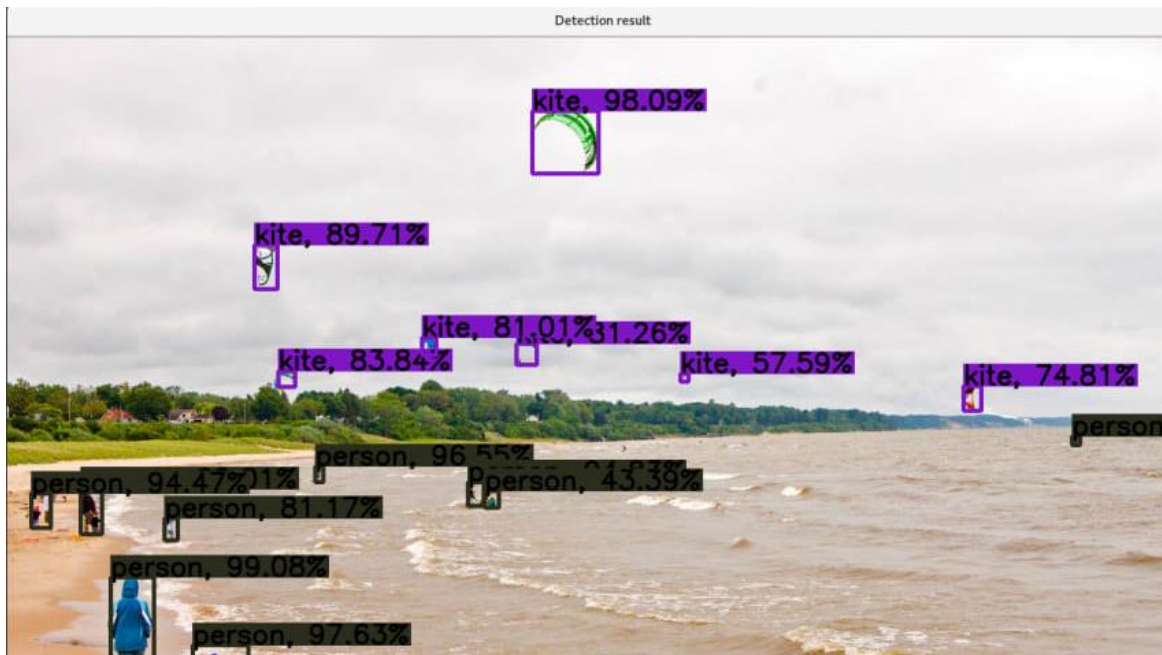


Figure 3.2.3: Object Detection Result using YOLOv3 Example 2



Figure 3.2.4: Object Detection Result using YOLOv3 Example 3

3.2.6 YOLOv3's Speed and Accuracy

- Model Speed (in FPS) - It processes pictures at 30 FPS on a Pascal Titan X.
- Model Accuracy (On Testing dataset) - The model's accuracy (based on testing dataset) is 87.54% (Mean Average Precision).

CHAPTER 4: RESULTS

4.1 Summary of Code

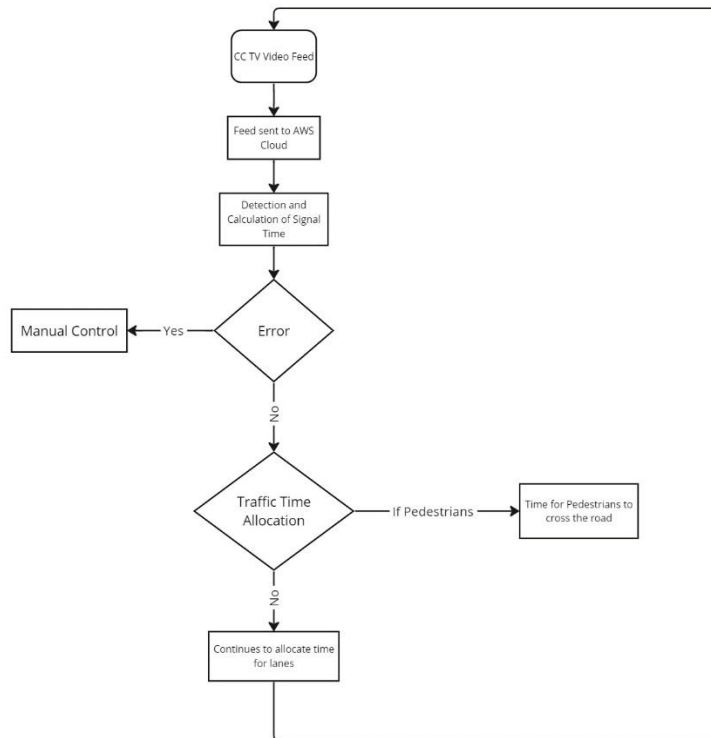
This is a YOLO (You Only Look Once) deep neural network implementation of object detection. When a video file is supplied, the program analyses it frame by frame, using the YOLO pre-trained model to identify items in each frame. The program makes use of the argparse module to parse command-line parameters and the OpenCV library for image processing and manipulation. During the processing of the video frames, a progress bar is provided by the tqdm module.

The COCO (Common items in Context) dataset, which contains 80 classes of items, is used to train the YOLO model that is employed in this program. To take use of the computing capability of NVIDIA GPUs, the program loads the YOLO model configuration and weights from disc and specifies CUDA as the execution backend and target.

After reading each frame of the video, the program runs each frame through the YOLO model to produce bounding boxes and class probabilities for any objects that were recognised. The remaining bounding boxes are then drawn on the frame along with the class name and probability after the program uses non-maxima suppression to eliminate duplicate bounding boxes for the same item. Additionally, the program stores the edited video on disc.

Additionally, the program creates two log files: one for things that enter the frame and another for those that exit the frame. The logs are produced depending on the number of objects in each frame and the direction in which they moved relative to the previous frame. To guarantee that the logs are correct, the program records the ID of each object.

4.2 Model creation and Training



4.3 Execution of Code

4.3.1 Procedure to run the software

1. Create a folder with all the required libraries, tools present in that folder and define the paths in the program wherever necessary.
2. Save the file with .py extension and Open Command prompt or Windows Terminal
3. The Command Prompt or Windows terminal should be in the folder where all the files have been stored.
4. Next, type the following command at the command prompt or terminal:
`Python file_name.py -i input_file_location -o output_file_location`
5. In this case, file_name denotes the name of the Python code file, input_file_location denotes the location of the video file to be processed, and output_file_location denotes the location of the processed video file.
6. The tqdm module will be used in the terminal to display the timer once the video processing is complete and to indicate the progress of the video processing. The pedestrian time and the green signal time on this timer both vary automatically.

4.4 Obtained Results

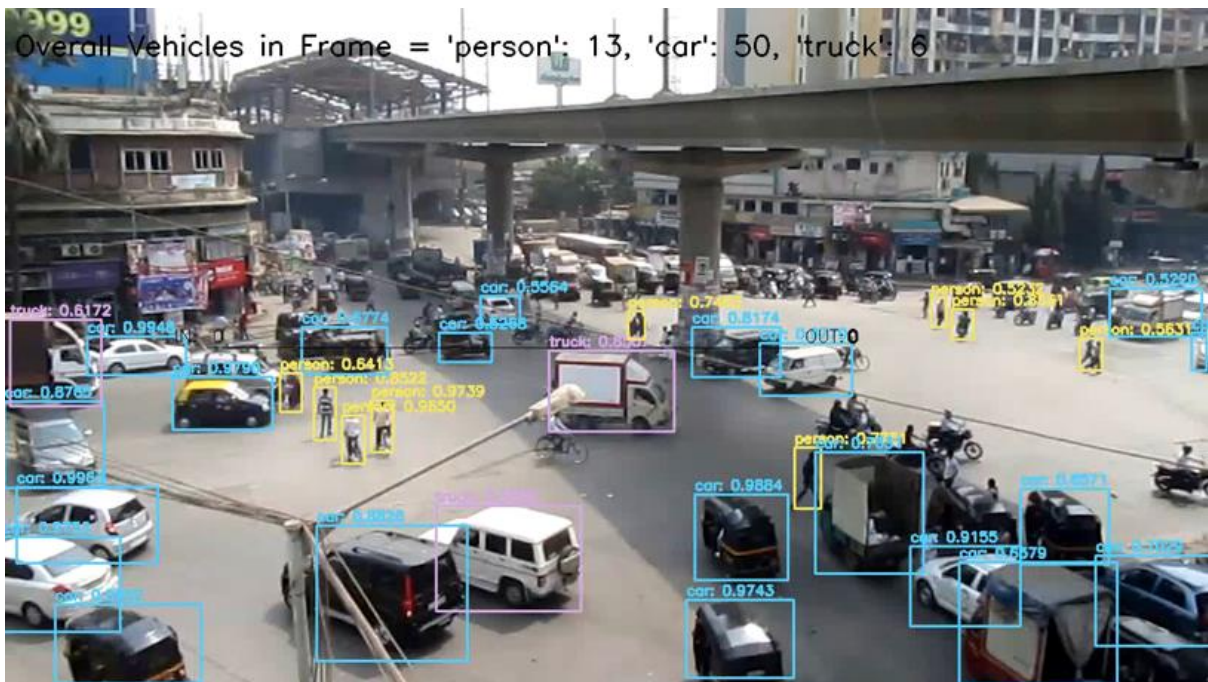


Figure 4.4.1: Output of Video 1



Figure 4.4.2: Output of Video 2

```
Windows PowerShell
PS D:\B.Tech\Final Year Project\Codes\Pedestrian & Vehicle Detection> python .\VoLo_Detector.py -i videos/overpass.mp4 -o output/op.avi

-----
[INFO] Loading Network Weights & Configuration from disk...
-----

[INFO] Total frames in the video: 155

0% | 0/155 [00:00<?, ?it/s][
WARN:000.390] global D:\opencv-python\opencv-python\opencv\modules\dnn\src\net_impl.cpp (179) cv::dnn::dnn4_v20220524::Net::Impl::setUpNet DNN module was
not built with CUDA backend; switching to CPU

[INFO] Estimated time taken to process single frame: 0.6509 seconds
[INFO] Estimated total time to finish object detection: 1.6815 minutes

-----

Object Detection Progress:
90% | 140/155 [01:15<00:08, 1.75it/s]
```

Figure 4.4.3: Output in Windows Terminal at the time of Video Processing

```
Windows PowerShell
PS D:\B.Tech\Final Year Project\Codes\Pedestrian & Vehicle Detection> python .\VoLo_Detector.py -i videos/overpass.mp4 -o output/op.avi

-----
[INFO] Loading Network Weights & Configuration from disk...
-----

[INFO] Total frames in the video: 155

0% | 0/155 [00:00<?, ?it/s][
WARN:000.390] global D:\opencv-python\opencv-python\opencv\modules\dnn\src\net_impl.cpp (179) cv::dnn::dnn4_v20220524::Net::Impl::setUpNet DNN module was
not built with CUDA backend; switching to CPU

[INFO] Estimated time taken to process single frame: 0.6509 seconds
[INFO] Estimated total time to finish object detection: 1.6815 minutes

-----

Object Detection Progress:
100% | 155/155 [01:23<00:00, 1.87it/s]
{'person': 1, 'traffic light': 2, 'car': 24, 'bus': 18}
51

-----Green Signal-----

14
```

Figure 4.4.4: Output in Windows Terminal after Video Processing

CHAPTER 5: TOOLS USED

5.1 Python

5.1.1 Introduction

Guido van Rossum created Python, an interpreted, object-oriented, high-level, dynamically semantic programming language. The product was launched in 1991. The word "Python" is designed to be both straightforward and amusing, and it is a reference to the British comedy group Monty Python. As it handles the bulk of the complexity for the user, Python has a reputation for being a beginner-friendly language. It has replaced Java as the most popular beginning language and allows newcomers to concentrate on really understanding programming concepts rather than minute details.

Python is renowned for its dynamic typing, dynamic binding, and high-level built-in data structures. Moreover, it is often used as a scripting or glue language to link already existing components as well as for quick application creation. Python is also used for system programming, software development, arithmetic, and server-side web development. Python's easy-to-learn syntax and focus on readability reduce the expenses associated with maintaining programs. Moreover, Python's support for modules and packages streamlines the process of building modular applications and reusing code. Due to Python's status as an open-source community language, a sizable number of independent programmers are continually developing libraries and features for it. A sizable community of independent programmers are continually developing libraries and features for Python as a result of its open-source nature.

5.1.2 Requirements

The requirements for Python vary depending on the specific use case and environment, but some of the general requirements for using Python are:

1. Python interpreter: Having the Python interpreter installed on the system is a must for using Python. The application that runs Python code is known as the Python interpreter. From the official Python website, Python may be downloaded, and installation is usually simple.
2. Text editor or IDE: To create and modify Python code, you need a text editor or integrated development environment (IDE). There are several options available,

including as paid and free software like Visual Studio Code, PyCharm, Sublime Text, Notepad++, and Atom.

3. Libraries and modules: Python offer a sizable standard library that contains several modules and routines for carrying out frequent operations. To increase Python's capability, nevertheless, certain applications might need extra libraries or modules. NumPy, Pandas, TensorFlow, and Pygame are a few well-known Python libraries.
4. Operating system: Python is compatible with Windows, Linux, and macOS, among other operating systems. Depending on the version of Python and the individual libraries or modules being used, the operating system requirements for Python may change.
5. Hardware requirements: The hardware requirements for Python are relatively low, and Python can run on most modern computers. However, some libraries or modules may have specific hardware requirements, such as a certain amount of RAM or processing power.

In summary, the basic requirements for using Python include a Python interpreter, a text editor or IDE, and any necessary libraries or modules. Additionally, the specific operating system and hardware requirements may vary depending on the project and the specific tools being used.

5.1.3 Features of Python

Python's features include:

- Easy-to-learn – Given that it has a clear syntax, few keywords, and a simple organizational structure, Python is simple to learn. The pupil may swiftly pick up the language thanks to this.
- Simple to read: Since Python code is more clearly defined and visible, it is simpler to comprehend.
- Simple to maintain – Python's source code is relatively easy to keep up with.
- A big standard library – The bulk of the Python library runs on UNIX, Windows, and Macintosh and is cross-platform compatible.
- Interactive mode – Python has an interactive mode that makes it possible to test and debug code snippets while they are being performed.
- Portability – Python offers the same user interface on all hardware platforms and is very portable.

- Extendable – The Python interpreter can take low-level modules, making it extensible. With the help of these modules, programmers may improve or customize their tools to make them more helpful.
- Databases – Python provides interfaces to all important commercial databases.
- GUI Programming – It is simpler to develop and adapt GUI programs to various platforms thanks to Python's support for system calls, libraries, and Windows-based operating systems like Windows MFC, Macintosh, and the Unix X Window system.
- Scalable – Python is more structured and supportive of larger projects than shell programming, making it more scalable.

5.1.4 Setting Up Python

Setting up Python involves a series of steps that will enable a user to run Python code and applications. Here are the general steps involved in setting up Python:

1. Download Python: Downloading the Python installation package from the official Python website is the first step. Depending on their operating system and architecture, the user should select the suitable version of Python.
2. Install Python: The user should launch the installer after downloading the Python installation package, then follow the on-screen instructions to finish the installation. Python will by default be set up in the "Program Files" directory on Windows, or in the "/usr/local/bin" directory on macOS or Linux.
3. Verify the installation: The user should start the Python interpreter at a command prompt or terminal window after installing Python to make sure it was installed correctly. The user should see the Python version number and a ">>>" prompt if Python has been installed properly.
4. Install a code editor: Any text editor may be used to create Python code, although it's advised to choose one that has debugging tools, syntax highlighting, and code completion. Some well-liked choices are Atom, PyCharm, Visual Studio Code, and PyCharm.
5. Install Python packages: With its extensive library of packages and modules, Python may be used for a wide range of applications. The pip package manager, which is pre-installed with Python 2.7.9+ and Python 3.4+, allows the user to install Python packages. The user should launch a command window or terminal window and run "pip install package-name>" to install a package.

By completing these steps, the user should be able to start writing and running Python code on their computer. They can test their setup by creating a simple "Hello, World!" program and running it in their code editor or Python interpreter.

5.1.5 Conditional Statements in Python

Python's conditional statements let programmers run several sets of code based on whether a given condition is true or false. The "if" statement and the "if-else" statement are the two primary categories of conditional statements in Python.

5.1.5.1 If statement

A piece of code is only run using the "if" statement if a certain condition is met. The "if" statement's fundamental grammar is seen here:

```
if condition:  
    statement1  
    statement2  
    ...
```

A Boolean expression that is evaluated to true or false in this syntax is referred to as the "condition." The indented chunk of code below "condition" is run if "condition" is true. The code block is skipped if "condition" returns false.

5.1.5.2 If - else statement

When using an "if-else" statement, a block of code is executed once if a condition is true and once if it is false. Here is the "if-else" statement's fundamental syntax:

```
if condition:  
    statement1  
    statement2  
    ...  
  
else:  
    statement3  
    statement4  
    ...
```

In this syntax, the block of code following the "if" statement is run and the block of code following the "else" statement is skipped if the "condition" is true. The block of code beneath the "if" statement is skipped if the "condition" is false, and the block of code under the "else" statement is carried out instead.

5.1.5.3 Nested "if" statement

Multiple conditions can be tested using nested "if" expressions. An "if" statement is nestled inside another "if" statement in this instance. A sample of nested "if" statements is shown below:

```
if condition1:
    statement1
    statement2
    ...
    if condition2:
        statement3
        statement4
        ...
    else:
        statement5
        statement6
        ...

else:
    statement7
    statement8
    ...
```

In this format, the code block after the "if" statement is run if "condition1" is true. The code block after the nested "if" statement is performed if "condition2" is also true. The code block after the nested "else" expression is run if "condition2" is false. The code block after the "else" expression is performed if "condition1" is false.

5.1.5.4 If - elif statement

Another conditional statement in Python that enables the programmer to check many conditions in a single block of code is "elif" (short for "else if"). In order to build a branching

structure in the code, the "elif" statement is combined with the "if" statement and the "else" statement.

The basic syntax of the "elif" statement is as follows:

```
if condition1:
    statement1

elif condition2:
    statement2

elif condition3:
    statement3

else:
    statementN
```

If "condition1" is true, the code block following the "if" statement in this syntax is run. The "elif" statement is assessed if "condition1" is false. The code block in the "elif" expression is run if "condition2" is true. The subsequent "elif" sentence is evaluated if "condition2" is false, and so on. The "else" statement's code block is run if none of the criteria are true.

5.1.6 Loops in Python

Loops are used in Python to continually run a section of code. In Python, there are two different forms of loops: "for" loops and "while" loops.

5.1.6.1 For Loops

For loops are used to repeatedly iterate over any iterable object or sequence (such as a list, tuple, or string). The following is the fundamental syntax for a for loop in Python:

for variable in sequence:

```
    statement(s)
```

The iteration variable that receives the values of each element in the sequence or iterable object is referred to in this syntax as "variable," while the sequence or iterable object being iterated over is referred to as "sequence." For each loop iteration, the code block included within the for loop is run.

5.1.6.2 While Loops

If a specific condition is met, a block of code is continuously executed using a while loop. The following is the fundamental syntax for a while loop in Python:

while condition:

 statement(s)

In this syntax, "statement(s)" refers to the block of code that is run for each iteration of the loop if "condition" is true. "Condition" refers to the Boolean expression that is verified before each iteration of the loop.

Both for loops and while loops can contain nested loops, conditional statements, and other control flow instructions, which is a crucial point to remember.

5.1.7 Python Dictionary

The items are separated by commas and a colon (:) separates each key from its value, and curly brackets enclose the whole structure. Just two curly braces are used to denote an empty dictionary with no entries: {}.

Values may not be unique within a dictionary, but keys always are. A dictionary's keys must be immutable data types like texts, integers, or tuples, but its values can be of any kind.

5.1.7.1 Accessing Values in Dictionary

You may make use of the well-known square brackets and the key to access dictionary items.

Here is a straightforward illustration:

```
dict={'Name': 'Karan', 'Age': 20, 'Class': 'Third'}
```

```
print(dict['Name'])
```

```
print(dict['Age'])
```

When the above code is executed, the result will be:

```
Karan
```

```
20
```

5.1.8 Functions in Python

The function in Python is a name-define code block. When we need to repeat a job again without writing the same code again, we utilize functions. It can accept parameters and give

back the value. Python has a DRY principle like other programming languages. DRY stands for Don't Repeat Yourself. Consider a scenario where we need to do some actions/tasks many times. We can define that action only once using a function and call that function whenever required to do the same activity. Function improves efficiency and reduces errors because of the reusability of a code. Once we create a function, we can call it anywhere and anytime. The benefit of using a function is reusability and modularity.

5.1.8.1 Creating a Function

- Use the def keyword with the function name to define a function.
- Next, pass the number of parameters as per your requirement. (Optional).
- Next, define the function body with a block of code. This block of code is nothing but the action you wanted to perform.

In Python, there is no need to specify curly braces for the function body. The only indentation is essential to separate code blocks. Otherwise, you will make an error.

Syntax of creating a function:

```
def function_name(parameter1, parameter2):  
    #block of code  
return value
```

Here,

- `function_name`: Function name is the name of the function. We can give any name to function.
- `parameter`: Parameter is the value passed to the function. We can pass any number of parameters. Function body uses the parameter's value to perform an action.
- `function_body`: The function body is a block of code that performs some tasks. This block of code is nothing but the action you wanted to accomplish.
- `return value`: Return value is the output of the function.

5.1.8.2 Calling a Function

Once we define a function or finalized structure, we can call that function by using its name. We can also call that function from another function or program by importing it. To call a function, use the name of the function with the parenthesis, and if the function accepts parameters, then pass those parameters in the parenthesis.

```
def function_name(parameter1, parameter2):  
    #block of code  
return value  
function_name(parameter1, parameter2)
```

5.2 Computer Vision

The technique of understanding pictures and videos, how they are stored, and how to change and extract data from them is known as computer vision. The foundation or primary tool utilized in artificial intelligence is computer vision. Self-driving vehicles, robots, and photo-editing applications all heavily rely on computer vision.

5.2.1 Open CV

People use the expansive open-source package known as OpenCV for image processing, machine learning, and computer vision. One area in which it currently plays a crucial role is real-time operation, which is critical in modern systems. It may be used to find individuals, items, and even handwriting in pictures and movies. The OpenCV array structure may be handled by Python for analysis when combined with a number of libraries, like NumPy. We use vector space and mathematical operations on these features to detect visual patterns and their numerous characteristics.

OpenCV's first version was 1.0. Since it is offered under a BSD license, OpenCV is free for both academic and commercial use. It has interfaces for C++, C, Python, and Java and is interoperable with Windows, Linux, Mac OS, iOS, and Android. Maximizing processing effectiveness for real-time applications was a design aim of OpenCV. Each piece of code is written in C/C++ and has been enhanced to take advantage of multi-core processing.

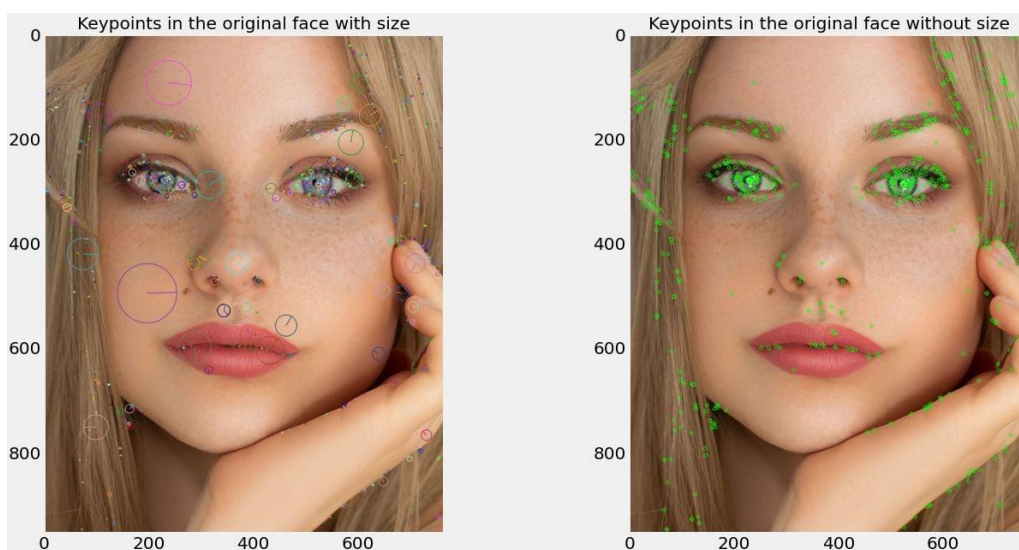


Figure 5.2.1 Feature Extraction and Image Classification using Open CV

Numerous details that are available in the original image may be accessed from the original image above. Like in the image up top, there are two faces to choose from in addition to the person (I) in the picture. Therefore, using OpenCV, it is possible to extract every aspect of a face from a picture.

5.2.2 Installation of Open CV on Windows

1. Install Python: OpenCV is a Python library, Python needs to be pre-installed on the computer before installing Open CV. Python can be downloaded from the official Python website, and the installation process is usually straightforward.
2. Install pip: Pip is a package manager for Python that allows the user to easily install and manage Python packages. To install pip, open a command prompt or terminal and run the following command:

```
python -m pip --default -- pip
```

3. Install OpenCV: Once Python and pip have been installed, OpenCV can be installed using pip. Open a command prompt or terminal and run the following command:

```
pip install opencv -- python
```

This command installs the latest version of OpenCV and all its dependencies.

4. Test the installation: After installing OpenCV, the installation can be tested to ensure that it was successful. Open a Python shell or create a new Python file and enter the following code:

```
import cv2  
print(cv2.__version__)
```

5. If the installation was successful, the version of OpenCV that was installed will be printed to the console.

The OpenCV library supports the following file types:

- *.bmp and *.dib Windows bitmaps
- JPEG files, such as *.jpg and *.jpeg; PNG files, such as *.png; and WebP files, such as *.webp
- TIFF files: *.tiff, *.tif; Sun rasters: *.sr, *.ras
- GDAL supports both raster and vector geographic data.

5.2.3 Read, Display, and write an Image using Open CV

5.2.3.1 Steps to read and display an Image

In OpenCV, use these steps to read and show an image:

1. Use the `imread()` method to read an image.
2. Create a GUI window and use the `imshow()` method to display an image.
3. To retain an image window on the screen for the specified number of seconds—that is, until the user shuts it—use the method `waitkey(0)`.
4. After displaying, use the `destroyAllWindows()` method to remove the image window from memory.

5.2.3.2 Reading an Image

To view the pictures, It uses the `cv2.imread()` technique. With this procedure, a picture is loaded from the given file. This function produces an empty matrix if the picture cannot be read (due to a missing file, poor permissions, an unsupported or invalid format, etc.).

Syntax: `cv2.imread(path, flag)`

1. Open the openCV library in python:

```
import cv2
```

2. Load the image you want to read using the `cv2.imread()` function. The function takes one argument, which is the file path to the image:

```
image = cv2.imread('path/to/image.png')
```

Replace 'path/to/image.png' with the actual file path to your image. The function returns a NumPy array that represents the image.

3. If you wish to see the image, then display it. The picture may be shown using the `cv2.imshow()` method. The window name and the image data are the first and second parameters, respectively, for the function:

```
cv2.imshow('Image', image)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

The `cv2.destroyAllWindows()` method eliminates all windows, whereas the `cv2.waitKey()` function watches for a key event. Keep in mind that, depending on your environment, you might need to add these lines of code to see the picture.

5.2.3.3 Displaying an Image

A collection of Python bindings called OpenCV-Python was created to address issues with computer vision. A window containing an image is displayed using the `cv2.imshow()` technique. The picture size is automatically adapted to the window.

Syntax: `cv2.imshow(window_name, image)`

To display an image in OpenCV using Python, the `cv2.imshow()` function can be used.

1. Import the OpenCV library in Python:

```
import cv2
```

2. Utilize the `cv2.imread()` method to load the required picture. The file path to the picture is the only parameter the function accepts:

```
image = cv2.imread('path/to/image.png')
```

Replace 'path/to/image.png' with the actual file path to image to be displayed.

3. Use the `cv2.imshow()` method to show the image. The window name and the image data are the first and second parameters, respectively, for the function:

```
cv2.imshow('Image', image) cv2.waitKey(0)
```

The picture is shown by the `cv2.imshow()` method in a new window with the specified window name. The `cv2.waitKey()` method observes for the occurrence of a key event. If the argument is set to 0, it will continuously wait until a key is pushed.

4. Close the window. After displaying the image, you can close the window using the `cv2.destroyAllWindows()` function:

```
cv2.destroyAllWindows()
```

This will destroy all windows.

5.2.3.4 Writing an Image

An image may be saved to any storage medium using the `cv2.imwrite()` API. This will store the image in the current working directory using the selected format.

Syntax: `cv2.imwrite(filename, image)`

To save an image in OpenCV using Python, the `cv2.imwrite()` function can be used.

1. Import the OpenCV library in Python:

```
import cv2
```

2. Load the image to be saved using the `cv2.imread()` function:

```
image = cv2.imread('path/to/image.png')
```

3. Use the `cv2.imwrite()` method to save the picture. The function accepts two arguments: first, the file name you wish to save the image as, and second, the image data:

```
cv2.imwrite('path/to/save/image.png', image)
```

The image will be saved in the specified location.

Note that the format of the stored picture will depend on the file extension chosen in the file name. The picture will be stored in PNG format, for instance, if it is saved as a PNG file. The format of the stored image will alter if a different file extension is chosen.

5.3 COCO Dataset

5.3.1 Introduction to COCO

Commonly utilised in computer vision research, the Common Objects in Context (COCO) dataset is a sizable object recognition, segmentation, and captioning dataset. The COCO Consortium, which is made up of Microsoft, Carnegie Mellon University, and a number of other colleges and businesses, hosted it after Microsoft built it.

The COCO dataset contains more than 330,000 images with more than 2.5 million object instances that have been tagged and annotated with bounding boxes, segmentation masks, and descriptions. The images are diverse and show a range of scenes, including indoor and outdoor locations, human activities, and other object categories, such as animals, cars, and furniture.

Each instance of an item in an image is meticulously labelled with a bounding box or segmentation mask by a team of experienced annotators, who also offer a short summary of the image's content for the annotations in the COCO dataset. The annotations are then examined and corrected by a second team of annotators to ensure high quality and accuracy.

A common benchmark for assessing object detection, segmentation, and captioning algorithms is the COCO dataset. The dataset may be used by researchers to develop, test, and compare their models' performance to that of cutting-edge techniques. The dataset also has a leader board where researchers may post their findings and see how they compare to those of their peers.

The COCO dataset additionally includes subtitles for each image in addition to the object recognition and segmentation annotations. Short, descriptive sentences that summarise the substance of the images are used as captions. The captions may be utilised for additional applications including image search and retrieval in addition to training and assessing image captioning models.

The COCO dataset may be accessible via the COCO API, a Python-based interface for accessing and altering the dataset, which can be downloaded from the COCO website. The API offers a collection of tools for testing the effectiveness of object identification, segmentation,

and captioning algorithms in addition to methods for importing and modifying the photos and annotations.

In many different areas of computer vision research, such as object identification, segmentation, tracking, and captioning, the COCO dataset has been employed. The dataset has also been used to research a variety of other subjects, including picture retrieval, human-object interaction, and visual question answering. The dataset is a useful resource for academics in the field due to its scale and diversity, and it has helped advance the state of the art in several computer vision-related fields.

In conclusion, a large dataset for object detection, segmentation, and captioning known as the COCO dataset is often used in computer vision research. It contains around 330,000 images and more than 2.5 million instances of labelled and annotated objects with bounding boxes, segmentation masks, and descriptions. The dataset has improved the state of the art in various areas of computer vision, making it a valuable tool for researchers in the field.

5.3.2 Inside COCO Dataset

A sizable image recognition dataset for tasks including object detection, segmentation, and captioning is called COCO (Common Objects in Context). More than 330,000 photos with 80 item categories and 5 scene-specific subtitles are included. The COCO dataset has been utilised to train and assess several cutting-edge object identification and segmentation algorithms in computer vision research.

The photos and their annotations make up the dataset's two primary components.

- The photos are arranged in a hierarchy of folders, with the train, validation, and test sets' subdirectories located under the top-level directory.
- JSON format is used for the annotations, and each file corresponds to a single image.

The following details are included in each annotation in the dataset:

- Image file name
- Image size (width and height)
- A list of items that includes the following details: Bounding box coordinates (x, y, width, and height); segmentation mask (polygon or RLE format); key points and their locations; object class (e.g., "person," "car"); (if available)

- A scenario description in five captions.

Additional data is also provided by the COCO dataset, including picture super categories, licenses, and coco-stuff. (Pixel-wise annotations for stuff classes in addition to 80 object classes).

Multiple sorts of annotations are offered by MS COCO,

- Object recognition for 80 distinct objects with complete segmentation masks and bounding box coordinates.
- Include pixel maps showing 91 amorphous background regions in picture segmentation.
- Panoptic segmentation uses 91 categories of "stuff" and 80 categories of "things" to identify objects in pictures.
- Dense posture with over 39,000 photographs and over 56,000 tagged people, including pixel mapping, a template 3D model, and explanations in natural language for each picture.
- Annotations on over 250,000 people's important spots, including the left hip, right eye, and nose.



Figure 5.3.1: Image Classification using COCO Dataset

5.3.3 MS COCO Dataset Classes

The two primary categories of the COCO (Common Objects in Context) dataset classes are "things" and "stuff."

Things classes contain things that can be picked up or handled readily, such as home goods, automobiles, and animals. In COCO, examples of "things" classes include:

- Person
- Bicycle
- Car

- Motorcycle

Classes of "stuff" include backdrop or ambient elements like the sky, the sea, and the road. In COCO, examples of "stuff" classes include:

- Sky
- Tree
- Road

A comprehensive list of COCO's 80 classes is shown in the graphic below.

person	fire hydrant	elephant	skis	wine glass	broccoli	dining table	toaster
bicycle	stop sign	bear	snowboard	cup	carrot	toilet	sink
car	parking meter	zebra	sports ball	fork	hot dog	tv	refrigerator
motorcycle	bench	giraffe	kite	knife	pizza	laptop	book
airplane	bird	backpack	baseball bat	spoon	donut	mouse	clock
bus	cat	umbrella	baseball glove	bowl	cake	remote	vase
train	dog	handbag	skateboard	banana	chair	keyboard	scissors
truck	horse	tie	surfboard	apple	couch	cell phone	teddy bear
boat	sheep	suitcase	tennis racket	sandwich	potted plant	microwave	hair drier
traffic light	cow	frisbee	bottle	orange	bed	oven	toothbrush

Figure 5.3.2: List of 80 classes in COCO Dataset

It's crucial to remember that the COCO dataset has a built-in bias caused by class imbalance. When there is a considerable disparity between the quantity of samples in one class and other classes, there is a class imbalance. Some object classes contain much more picture occurrences than others in the COCO dataset context.

Bias in machine learning model training and assessment might result from the class imbalance. This is due to the model being exposed to more samples of the common classes, which helps it become more adept at identifying them. As a result, the model could require assistance in identifying the less common classes and doing badly in them.

Additionally, if the dataset is biased, the model may be overfit on the majority class and perform better in this class but worse in others. There are several methods for addressing the class imbalance problem, including oversampling, under sampling, and the fabrication of synthetic data.

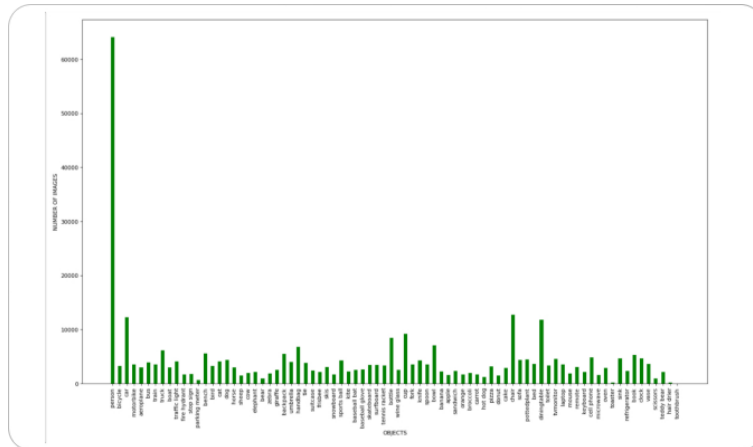


Figure 5.3.3: Graphical Representation of Classes v/s Number of images present in it.

5.3.4 Using COCO Dataset

The COCO dataset provides a starting point for computer vision to develop, test, improve, and scale up models for the annotation process more quickly.

5.3.4.1 Object detection

The most common use of computer vision is object detection. In order to enable their categorization and localisation in a picture, it recognises objects with bounding boxes.

Models for object detection may be trained using the COCO dataset. The dataset offers bounding box coordinates for 80 distinct types of items, allowing models to be trained to recognise bounding boxes and identify objects in the photos.

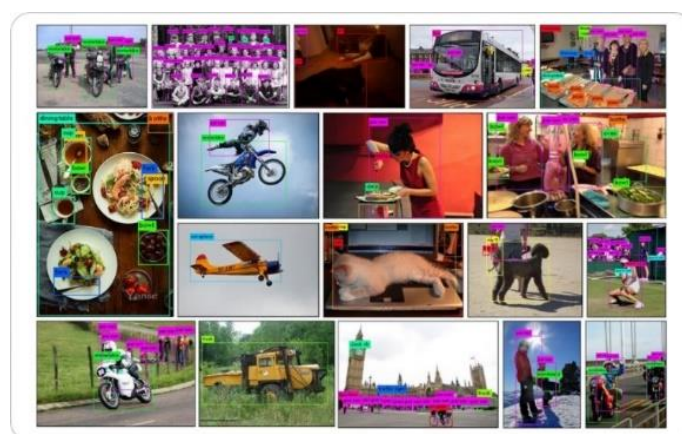


Figure 5.3.4: Object Detection in Different Images

5.3.4.2 Instance segmentation

Instance segmentation is a computer vision problem that entails locating and classifying individual items inside an image while also giving each instance of an object a distinct label.

Instance segmentation models frequently start by locating the objects in the picture using object recognition methods like bounding box regression and non-maximum suppression. Then, the models separate the items inside the bounding boxes using semantic segmentation approaches, including Convolutional Neural Networks (CNNs), and give distinct labels to each instance. Instance segmentation annotations from the COCO dataset can be utilised to train models for this job.

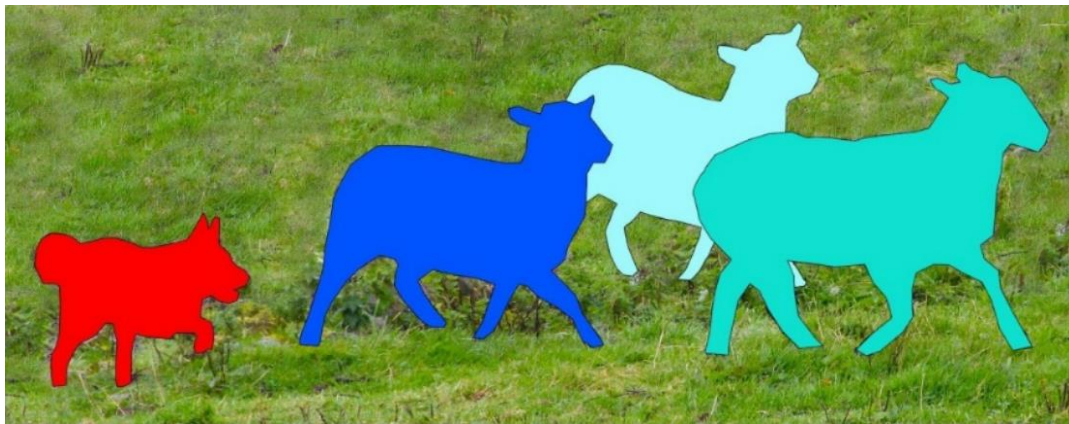


Figure 5.3.5: Representation of Instance Segmentation

CONCLUSION

An extremely efficient method for managing and monitoring traffic is the Real Time Traffic Management method utilising the YOLOv3 Algorithm. The COCO dataset and OpenCV library were used in the research to recognise and track automobiles and people in real-time video streams using the YOLOv3 technique.

The findings from the planned CNN model were extremely accurate, and they were made even more precise by using non-max suppression and overlapping bounding box removal approaches. The system proved to be effective in properly detecting and tracking objects in real-time by being put through a variety of realistic scenarios. This project may be used to monitor and control traffic flow on real-time videos, which will increase safety, lessen congestion, and optimise traffic flow. A very effective and efficient answer to the issues of contemporary traffic management is provided by the Real Time Traffic Management System employing the YOLOv3 Algorithm.

FUTURE SCOPE

The Real Time Traffic Management System employing the YOLOv3 Algorithm can be improved in several ways in the future. Among them are:

1. Integration of more sophisticated features: To give more thorough insights into traffic management, the present system may be enhanced by incorporating more sophisticated features such as vehicle categorization, licence plate identification, and driver behaviour analysis.
2. Support for many cameras: The system may be improved to accommodate several cameras, offering more thorough coverage and improved traffic flow monitoring.
3. Integration with traffic control systems: To enhance general traffic flow and ease congestion, the system may be combined with traffic control systems including traffic lights, road signs, and other traffic control mechanisms. Real-time traffic flow prediction: Future work could focus on developing algorithms to predict traffic flow patterns based on real-time data from the system, which could aid in decision-making for traffic management authorities.
4. Cloud-based implementation: The system might be set up on a cloud-based platform, allowing for more effective data processing and real-time data access from anywhere.

The YOLOv3 Algorithm-based Real Time Traffic Management System has a lot of room to grow and improve, and it can offer traffic management authorities useful information and solutions.

REFERENCES

- [1] S. S. P. Moka, S. M. Pilla, and S. Radhika, "Real time density-based traffic surveillance system integrated with acoustic based emergency vehicle detection," in 2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP). IEEE, 2020, pp. 1–7.
- [2] R. Chauhan, K. K. Ghanshala, and R. Joshi, "Convolutional neural network (CNN) for image detection and recognition," in 2018 first international conference on secure cyber computing and communication (ICSCCC). IEEE, 2018, pp. 278–282.
- [3] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in 2017 international conference on engineering and technology (ICET). Ieee, 2017, pp. 1–6.
- [4] B. F. Momin and T. M. Mujawar, "Vehicle detection and attribute-based search of vehicles in video surveillance system," in 2015 international conference on circuits, power, and computing technologies [ICCPCT-2015]. IEEE, 2015, pp. 1–4.
- [5] N. Zhang and J. Fan, "A lightweight object detection algorithm based on yolov3 for vehicle and pedestrian detection," in 2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC). IEEE, 2021, pp. 742–745.
- [6] R. B. Diwate, A. Zagade, M. Khodaskar, and V. R. Dange, "Optimization in object detection model using yolo. v3," in 2022 International Conference on Emerging Smart Computing and Informatics (ESCI). IEEE, 2022, pp. 1–4.
- [7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. Springer, 2014, pp. 740–755.
- [8] J. Redmon, "YOLO: Real-Time Object Detection." [Online]. Available: <https://pjreddie.com/darknet/yolo/>

PUBLISHED PAPER STATUS

My Submissions

Check Feedback, Status & Deadlines

Abstracts Presentation Material **Papers**

REAL TIME TRAFFIC MANAGEMENT SYSTEM USING YOLOv3 ALGORITHM
1st International Conference on Recent Innovations in Science, Engineering and Technology

Pending

View Paper
Submitted on
Mar 20, 2023

My paper

New submission

Paper number	File number	Status	Time	Operation
Paper title: REAL TIME TRAFFIC MANAGEMENT SYSTEM USING YOLOv3 ALGORITHM				Download
(IJIGSP) PAPER008656	PAPER008656-1	Under review	2023-03-29 12:12:56	Details