****

A PROJECT REPORT ON

**“Helmet Detection and Traffic Signal Control using Computer Vision”**

UNDER THE GUIDANCE OF

**DR. R. SRILAKSHMI**

SUBMITTED BY

**K. JAHNAVI – 245321733095**

**G. ASHOK – 245321733305**

**T. SAI SANOJ - 245321733306**

**V. MARUTHI SRIMANNARAYANA – 245321733307**

**D. RISHITHA – 245321733310**

**K. SURESH NAIK – 24532173312**

**ABSTRACT**

In our country, the motorcycle is the most used vehicle. The continual motorization of traffic has led to an increase in road accidents across the globe. As a recommendation, a system for automatically detecting helmet wear is mandatory for road safety. This project's main aim is to automatically detect bike-riders without helmets using computer vision.

As a result, a machine learning-based algorithm is used to generate a unique object detection model that can specify motorcycle riders. Then it detects whether the rider is wearing a helmet or not. If the person who is riding the motor-cycle is detected without helmet, then the picture of that rider is displayed on the screen, making him to feel very embarrassing.

Additionally, design a traffic signal control mechanism that prevents the signal from transitioning to green if any person without a helmet is detected in the frame. Once all individuals are wearing helmets, the signal can transition to green.

**CONTENTS**

CHAPTER 1: INTRODUCTION

CHAPTER 2: COMPUTER VISION

CHAPTER 3: OBJECT DETECTION

3.1: ONE SHOT OBJECT DETECTION

3.2: TWO SHOT OBJECT DETECTION

CHAPTER 4: CONVOLUTIONAL NEURAL NETWORKS

CHAPTER 5: ACTIVATION FUNCTIONS

5.1: STEP ACTIVATION FUNCTION

5.2: SIGMOID ACTIVATION FUNCTION

5.3: ReLU ACTIVATION FUNCTION

5.4; LEAKY ReLU ACTIVATION FUNCTION

5.5: SOFTMAX ACTIVATION FUNCTION

5.6: TANH ACTIVATION FUNCTION

CHAPTER 6: ROLE OF CNN IN YOLO

CHAPTER 7: ADVANCED TECHNIQUES IN COMPUTER

VISION OBJECT DETECTION

7.1: R-CNN

7.2: YOLO

7.3: SSD

CHAPTER 8: WHY YOLO

CHAPTER 9: FLOW DIAGRAM OF THE PROJECT

CHAPTER 10: YOLO

10.1: DEFINITION

10.2: ROLE OF YOLO

10.3: ARCHITECTURE

10.4: TRAINING

10.5: DETECTION

10.6: BENEFITS OF YOLO

10.7: DISADVANTAGES OF YOLO

CHAPTER 11: SOURCE CODE

CHAPTER 12: OUTPUT

CHAPTER 13: FUTURE WORK

1. **Introduction**

This document explores the implementation of computer vision techniques for two crucial aspects of road safety: helmet detection and traffic signal control.

By leveraging computer vision algorithms and machine learning models, we aim to develop a system capable of automatically detecting helmet usage among motorcyclists and controlling traffic signals intelligently to enhance safety on roads.

In summary, this document presents an overview of our research and development efforts in utilizing computer vision for improving road safety through helmet detection and traffic signal control.

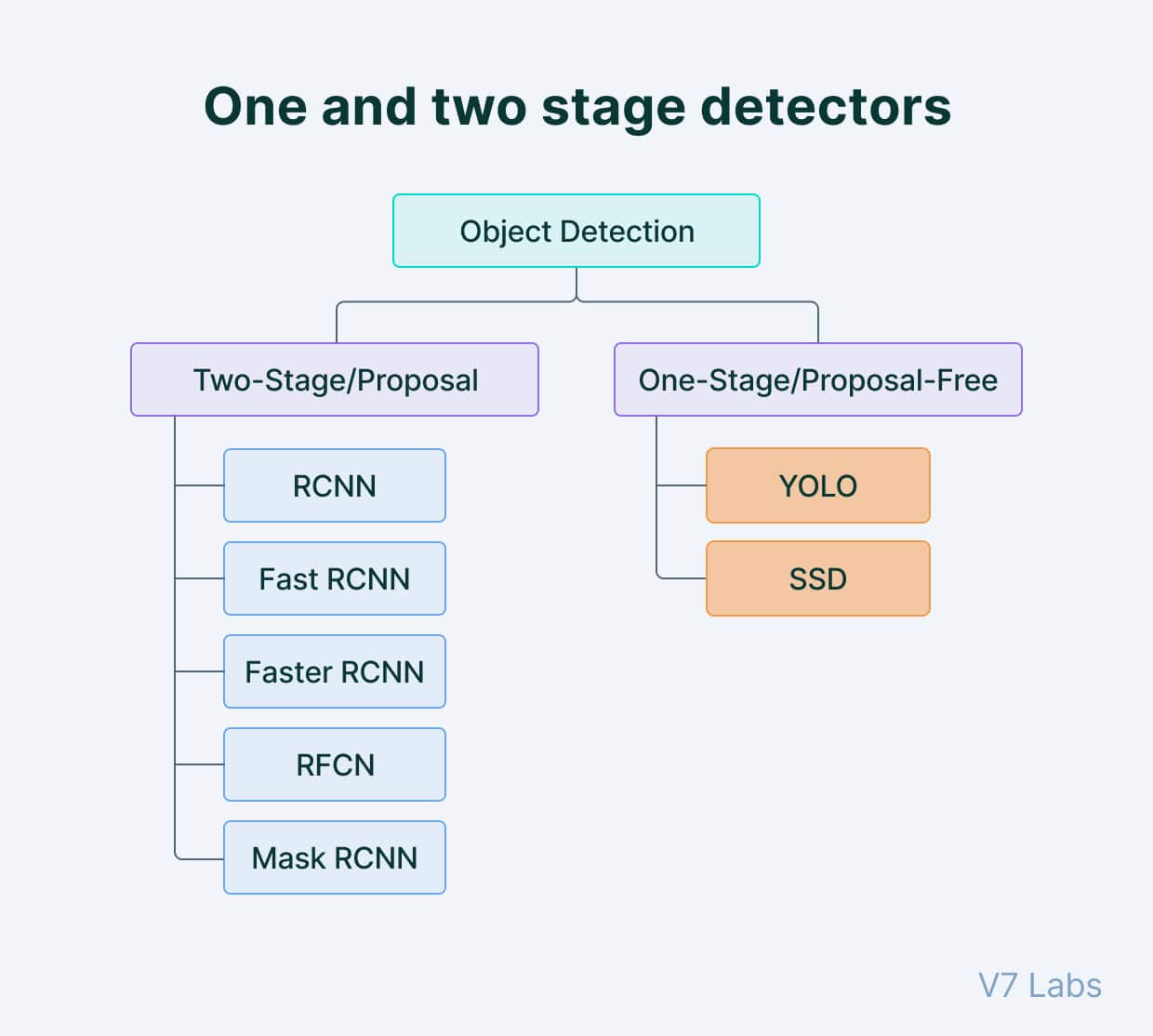
1. **Computer Vision**

Computer vision is a field of artificial intelligence and computer science that enables computers to interpret and understand visual information from the real world, similar to how humans perceive and analyse visual data. It involves developing algorithms and systems that can extract, analyse, and interpret visual data from images or videos.

Computer vision algorithms enable machines to perform tasks such as object recognition, detection, tracking, segmentation, and scene understanding. These algorithms can recognize objects, people, text, and other visual elements within images or videos, enabling applications such as facial recognition, autonomous vehicles, medical image analysis, surveillance systems, and augmented reality.

1. **Object Detection**

Object detection is a computer vision task that involves identifying and locating objects in images or videos. It is an important part of many applications, such as surveillance, self-driving cars, or robotics. Object detection algorithms can be divided into two main categories: single-shot detectors and two-stage detector.



* 1. **Single-shot object detection**

Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient.

However, single-shot object detection is generally less accurate than other methods, and it’s less effective in detecting small objects. Such algorithms can be used to detect objects in real time in resource-constrained environments.

YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. We will dive deeper into the YOLO model in the next section.

* 1. **Two-shot object detection**

Two-shot object detection uses two passes of the input image to make predictions about the presence and location of objects. The first pass is used to generate a set of proposals or potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than single-shot object detection but is also more computationally expensive.

Overall, the choice between single-shot and two-shot object detection depends on the specific requirements and constraints of the application.

Generally, single-shot object detection is better suited for real-time applications, while two-shot object detection is better for applications where accuracy is more important.

1. **CONVOLUTION NEURAL NETWORKS**

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order. It is the sequential design that give permission to CNN to learn hierarchical attributes. In CNN, some of them followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers.

1. **ACTIVATION FUNCTIONS**
   1. **Step Function**

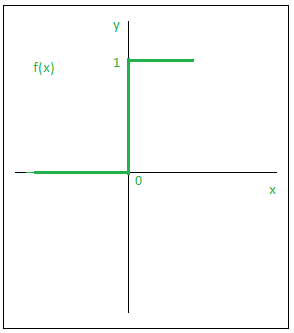
Step Function is one of the simplest kinds of activation functions. In this, we consider a threshold value and if the value of net input say **y** is greater than the threshold then the neuron is activated.

Mathematically,

**f(x)=1, if x>=0**

**f(x)=0, if x<0**

Graphical representation of step function



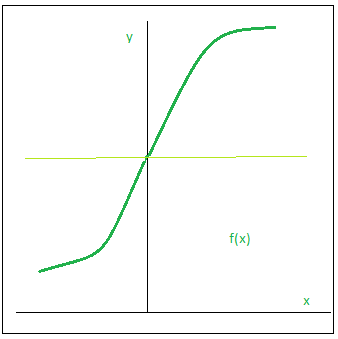
* 1. **Sigmoid Function**

Sigmoid function is a widely used activation function. This is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear. This essentially means that when I have multiple neurons having sigmoid function as their activation function the output is non-linear as well. The function ranges from 0-1 having an S shape.

Mathematically,

**F(x)=1/(1+e^-x)**

Graphical representation of sigmoid function.



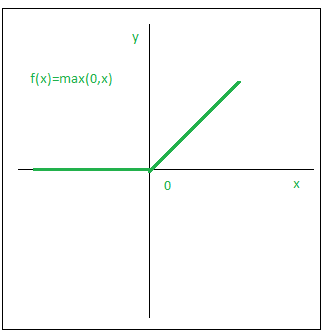
* 1. **ReLU**

The ReLU function is the Rectified linear unit. It is the most widely used activation function. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. What does this mean? If you look at the ReLU function if the input is negative, it will convert it to zero and the neuron does not get activated.

Mathematically,

**f(x)=max (0, x)**

Graphical representation of ReLU function



* 1. **Leaky ReLU**

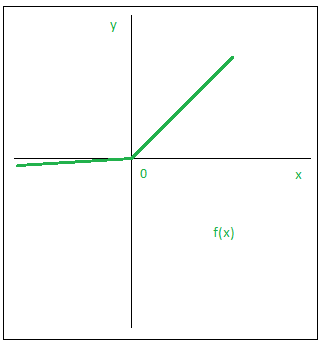
Leaky ReLU function is nothing but an improved version of ReLU function. Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x.

Mathematically,

**f(x)=ax, x<0**

**f(x)=x, otherwise**

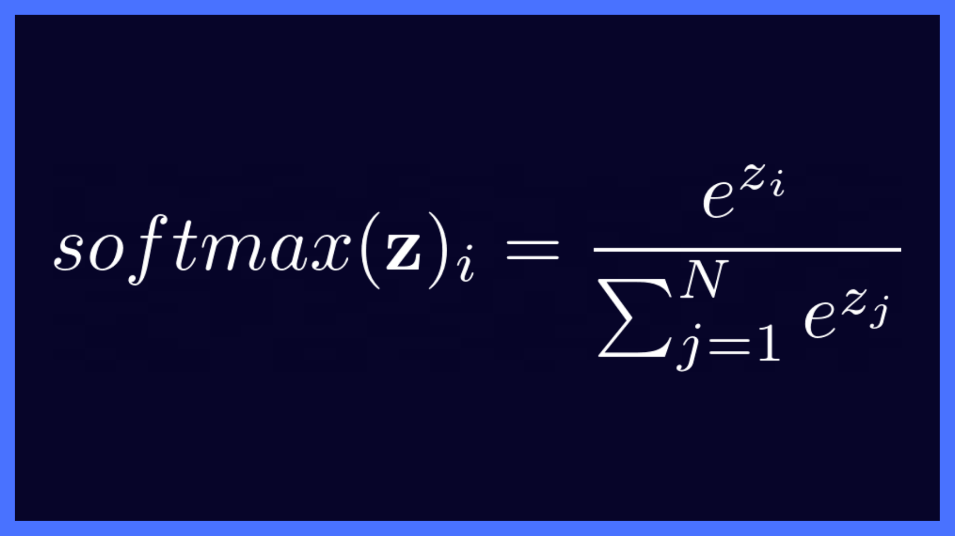
Graphical representation of Leaky ReLU function.



* 1. **SOFTMAX ACTIVATION FUNCTION**

The SoftMax activation function takes in a vector of **raw outputs** of the neural network and returns a vector of **probability scores**.

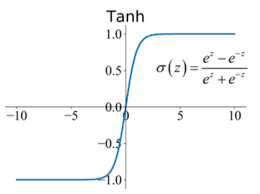
Mathematically,



* 1. **TANH ACTIVATION FUNCTION**

Tanh activation function used for neural networks. Tanh function became preferred over sigmoid function as it gave better performance for multi-layer neural networks.

Mathematically,



1. **ROLE OF CNN IN YOLO**

**Feature Extraction**: YOLO employs a CNN as a feature extractor. The input image is passed through several convolutional layers to extract features. These layers are usually pre-trained on a large dataset (like ImageNet) and then fine-tuned on the target dataset.

**Convolutional Layers**: The convolutional layers help in learning hierarchical features from the input image. These layers apply various filters to detect patterns such as edges, textures, and shapes.

**Pooling Layers:** After convolution, pooling layers are applied to reduce the spatial dimensions of the feature maps while retaining important information. This helps in reducing computation and extracting more abstract features.

**Fully Connected Layers**: The output of the convolutional layers is flattened and fed into fully connected layers. These layers learn complex relationships between features extracted by earlier layers.

**Bounding Box Prediction**: At the end of the network, YOLO uses convolutional layers to predict bounding boxes. For each grid cell, YOLO predicts multiple bounding boxes with associated confidence scores. These bounding boxes are adjusted based on offsets and anchor boxes (prior boxes representing predefined shapes and sizes).

**Class Prediction**: Alongside bounding box predictions, YOLO also predicts the probability distribution over all classes for each bounding box. This is usually done using softmax activation function in the output layer, assigning a probability to each class.

**Loss Computation and Backpropagation**: YOLO uses a specific loss function (often a combination of localization loss and classification loss, such as the sum of squared error for bounding box coordinates and cross-entropy for class probabilities) to compare the predicted outputs with the ground truth annotations. The network parameters are then updated using backpropagation to minimize this loss.

1. **Advanced Techniques in**

**Computer Vision Object Detection**

* 1. **R-CNN**

**(Region-based Convolutional Neural Networks)**

R-CNN combines region proposal algorithms (like selective search) with convolutional neural networks (CNNs) to detect objects within an image. It first generates a set of region proposals, typically using selective search, which proposes regions likely to contain objects. Each proposed region is then fed into a CNN, which extracts features from the region.

Finally, these features are used by a classifier to determine the presence of objects and refine their bounding boxes.Although effective, R-CNN tends to be slower due to its multi-stage process.

* 1. **YOLO (You Only Look Once)**

YOLO is a single-stage object detection algorithm that processes the entire image in a single pass through a neural network. It divides the image into a grid and predicts bounding boxes and class probabilities directly from grid cells.YOLO is known for its speed and efficiency, as it doesn't require multiple passes or region proposals.

* 1. **SSD (Single Shot Detector)**

SSD is another single-stage object detection method that combines feature extraction and classification into one network architecture. Similar to YOLO, SSD divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell. Additionally, SSD uses a set of default bounding boxes with different aspect ratios and scales to improve accuracy for detecting objects of various sizes. SSD is fast and accurate for detecting multiple objects within images or video frames, making it suitable for real-time applications.

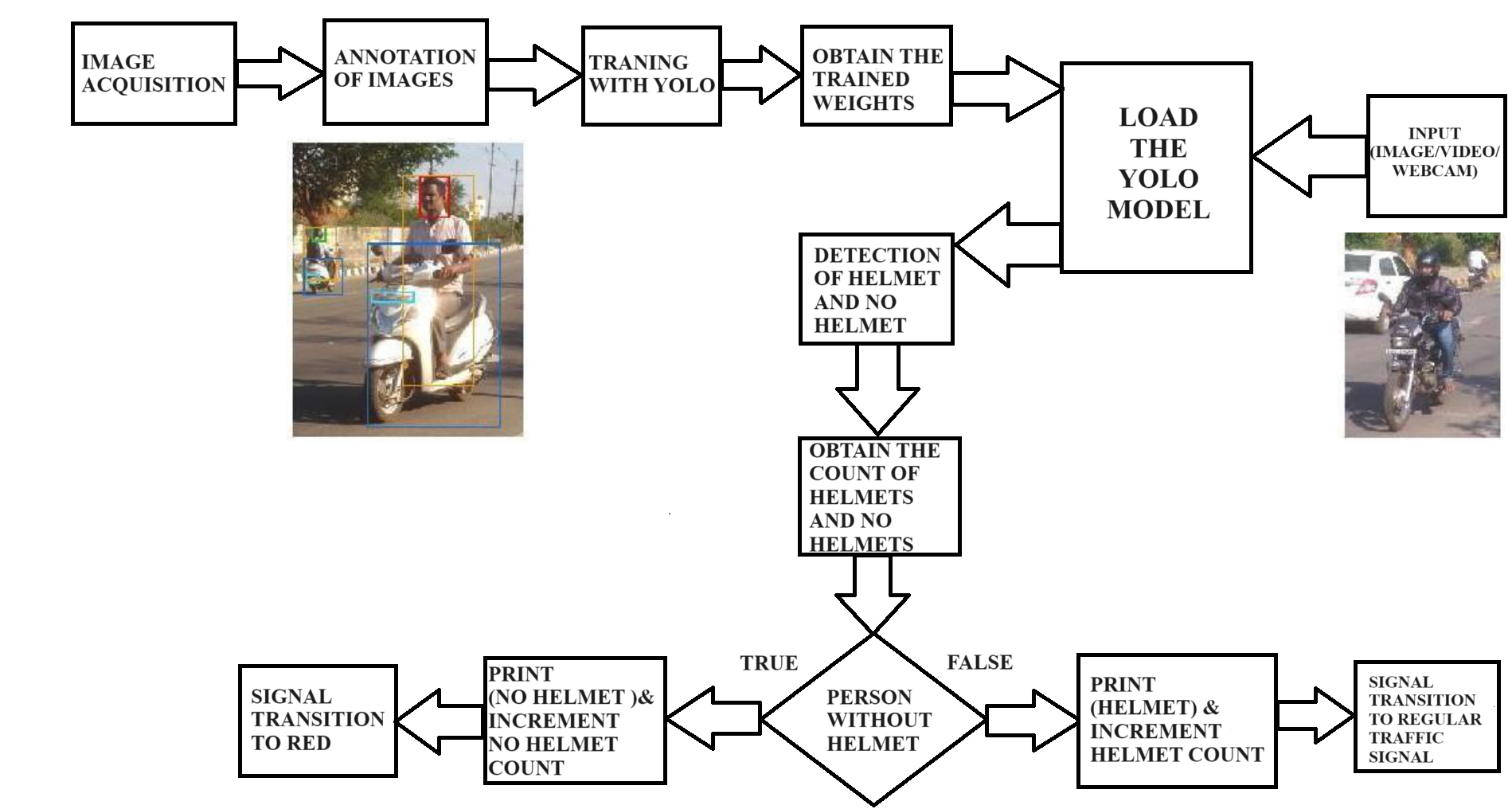
1. **WHY YOLO (You Only Look Once)**

Among the techniques mentioned YOLO (You Only Look Once) is generally considered the best for real-time object detection applications.

**Single-pass Processing**: YOLO processes the entire image in a single pass through the neural network, making it very fast compared to multi-stage approaches like R-CNN.

**Consistent Speed**: YOLO maintains a consistent processing speed regardless of the number of objects in the image, making it suitable for applications where real-time performance is crucial.

1. **FLOW DIAGRAM OF THE PROJECT**



1. **YOLO (You Only Look Once)**
   1. **Definition**

YOLO (You Only Look Once) is a popular object detection model known for its speed and accuracy.

YOLO stands for "You Only Look Once," and it refers to a family of real-time object detection algorithms in computer vision. The YOLO algorithm is designed to detect and classify objects within images or video frames, providing bounding box coordinates and class probabilities for each detected object.

The key innovation of YOLO is its ability to perform object detection in a single forward pass through the neural network, making it very fast and suitable for real-time applications.

* 1. **ROLE OF YOLO**

**Single Pass Detection:** YOLO processes the entire image or frame in a single forward pass through the neural network. This is in contrast to traditional object detection methods that involve multiple passes or region-based approaches.

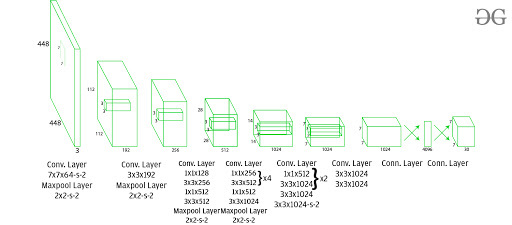
**Grid System:** The image is divided into a grid, and each grid cell is responsible for predicting bounding boxes and class probabilities. This grid-based approach helps YOLO efficiently cover the entire image.

**Bounding Box Predictions:** YOLO predicts bounding boxes (coordinates) for each detected object along with confidence scores. The confidence score indicates how likely the predicted bounding box contains an object.

**Class Predictions:** YOLO predicts class probabilities for each bounding box, indicating the likelihood of the object belonging to a specific class (e.g., person, car, dog).

**Multiple Scales:** YOLO typically predicts objects at multiple scales to handle objects of different sizes. This is achieved by predicting bounding boxes at different resolutions or scales within the grid.

**Non-Maximum Suppression:** After predictions are made, a post-processing step called non-maximum suppression is applied to filter out redundant and overlapping bounding boxes, keeping only the most confident ones.

* 1. **Architecture**

Convolutional layers play a crucial role in Convolutional Neural Networks (CNNs) and are responsible for extracting hierarchical features from input data, particularly in the context of image processing. These layers use convolutional operations to detect patterns, edges, and higher-level features in the input data.

The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pre-trained model is converted to perform detection since previous research showcased that adding convolution and connected layers to a pre-trained network improves performance. YOLO’s final fully connected layer predicts both class probabilities and bounding box coordinates.

YOLO divides an input image into an S × S grid. If the centre of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted box is.

YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be “responsible” for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at forecasting certain sizes, aspect ratios, or classes of objects, improving the overall recall score.

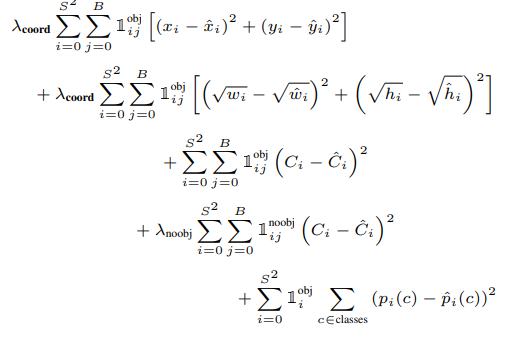
One key technique used in the YOLO models is non-maximum suppression (NMS). NMS is a post-processing step that is used to improve the accuracy and efficiency of object detection. In object detection, it is common for multiple bounding boxes to be generated for a single object in an image. These bounding boxes may overlap or be located at different positions, but they all represent the same object. NMS is used to identify and remove redundant or incorrect bounding boxes and to output a single bounding box for each object in the image.

This architecture uses Leaky ReLU as its activation function in whole architecture except the last layer where it uses linear activation function. Batch normalization also helps to regularize the model. Dropout technique is also used to prevent overfitting.

* 1. **Training**

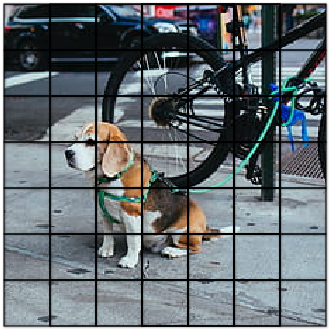
This model is trained on the *ImageNet-1000* dataset. The model is trained over a week and achieve top-5 accuracy of *88% on ImageNet 2012* validation which is comparable to GoogLeNet (2014 ILSVRC winner), the state of the art model at that time. Fast YOLO uses fewer layers *(9 instead of 24)* and fewer filters. Except this, the fast YOLO have all parameters similar to YOLO.YOLO uses sum-squared error loss function which is easy to optimize. However, this function gives equal weight to the classification and localization task. The loss function defined in YOLO as follows:

The first two parts of the above loss equation represent localization mean-squared error, but the other three parts represent classification error. In the localization error, the first term calculates the deviation from the ground truth bounding box. The second term calculates the square root of the difference between height and width of the bounding box. In the second term, we take the square root of width and height because our loss function should be able to consider the deviation in terms of the size of the bounding box. For small bounding boxes, the little deviation should be more important as compared to large bounding boxes.

There are three terms in classification loss, the first term calculates the sum-squared error between the predicted confidence score that whether the object present or not and the ground truth for each bounding box in each cell. Similarly, the second term calculates the mean-squared sum of cells that do not contain any bounding box, and a regularization parameter is used to make this loss small. The third term calculates the sum-squared error of the classes belongs to these grid cells.

**10.5 Detection**

This architecture divides the image into a grid of *S\*S* size. If the centre of the bounding box of the object is in that grid, then this grid is responsible for detecting that object. Each grid predicts bounding boxes with their confidence score. Each confidence score shows how accurate it is that the bounding box predicted contains an object and how precise it predicts the bounding box coordinates with respect to ground truth prediction.



**10.6 Benefits of YOLO:**

* Process frames at the rate of *45 frames for second* (larger network) to *150 frames per second* (smaller network) which is better than real-time.
* The network is able to generalize the image better.

**10.7 Disadvantages of YOLO:**

* Comparatively low recall and more localization error compared to Faster R\_CNN.
* Struggles to detect close objects because each grid can propose only 2 bounding boxes.
* Struggles to detect small objects.

1. **Source code**

import cv2

import numpy as np

import os

import imutils

from tensorflow.keras.models import load\_model

import time

os.environ['TF\_FORCE\_GPU\_ALLOW\_GROWTH'] = 'true'

net = cv2.dnn.readNet("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/yolov3.weights", "C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/yolov3.cfg")

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)

model = load\_model("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/yolo\_model.h5")

print('model loaded!!!')

cap = cv2.VideoCapture(0) # Use webcam, change to the appropriate index if you have multiple cameras

COLORS = [(0, 0, 255)] # Red color only

fourcc = cv2.VideoWriter\_fourcc(\*"XVID") # open-source MPEG-4 video codec

writer = cv2.VideoWriter('output.avi', fourcc, 5, (888, 500))

# Folder to save cropped images

output\_folder = 'output\_images'

os.makedirs(output\_folder, exist\_ok=True)

def helmet\_or\_nohelmet(helmet\_roi):

try:

helmet\_roi = cv2.resize(helmet\_roi, (224, 224))

helmet\_roi = np.array(helmet\_roi, dtype='float32')

helmet\_roi = helmet\_roi.reshape(1, 224, 224, 3)

helmet\_roi = helmet\_roi / 255.0

return int(model.predict(helmet\_roi)[0][0])

except:

pass

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

previous = 0

def control\_traffic\_signal(total\_no\_helmet\_count, all\_wearing\_helmet, elapsed\_time, red\_signal\_duration):

global previous

cycle\_time = 46

if all\_wearing\_helmet:

red\_signal\_duration = 0

green\_time = 20

yellow\_time = 6

remaining\_time = elapsed\_time % cycle\_time

if remaining\_time < green\_time:

signal\_color = (0, 255, 0) # Green

signal\_text = f"Green Signal: {green\_time - remaining\_time} seconds"

signal\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/GREEN SIGNAL.jpg")

elif remaining\_time < green\_time + yellow\_time:

signal\_color = (0, 255, 255) # Yellow

signal\_text = f"Yellow Signal: {green\_time + yellow\_time - remaining\_time} seconds"

signal\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/YELLOW SIGNAL.jpg")

else:

signal\_color = (0, 0, 255) # Red

signal\_text = f"Red Signal: {cycle\_time - remaining\_time} seconds"

signal\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/RED SIGNAL.jpg")

cv2.putText(signal\_image, signal\_text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, signal\_color, 2, cv2.LINE\_AA)

cv2.imshow('Traffic Signal', signal\_image)

cv2.waitKey(1) # Allow the GUI to process events

return elapsed\_time + 1, max(0, red\_signal\_duration - 1) # Increment elapsed time,

else:

signal\_color = (0, 0, 255) # Red

signal\_text = f" PLEASE WEAR HELMET Red Signal: {max(0,red\_signal\_duration)} seconds"

signal\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/RED SIGNAL.jpg")

if total\_no\_helmet\_count > 0:

if red\_signal\_duration == 0:

red\_signal\_duration = 5\*total\_no\_helmet\_count

previous = 5 \* total\_no\_helmet\_count

else:

# Increment red signal duration by 5 seconds for every person without a helmet

red\_signal\_duration = ((5 \* total\_no\_helmet\_count) - previous) + red\_signal\_duration

red\_signal\_duration = max(0, red\_signal\_duration)

previous = 5 \* total\_no\_helmet\_count

if elapsed\_time >= 35:

green\_signal\_duration = 16

remaining\_green\_time = (elapsed\_time - 35) % (green\_signal\_duration + cycle\_time)

if remaining\_green\_time < green\_signal\_duration:

signal\_color = (0, 255, 0) # Green

signal\_text = f"Passing Green Signal: {green\_signal\_duration - remaining\_green\_time} seconds"

signal\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/GREEN SIGNAL.jpg")

red\_signal\_duration = 0 # Reset red signal duration when green signal is released

previous = 0

else:

signal\_text = f"PLEASE WEAR HELMET Red Signal: {cycle\_time - remaining\_green\_time} seconds"

print("2 minutes have passed. Resuming normal operation")

else:

signal\_text = f"PLEASE WEAR HELMET Red Signal: {red\_signal\_duration} seconds"

print("Not all individuals have helmets. Traffic signal: Red")

cv2.putText(signal\_image, signal\_text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, signal\_color, 2, cv2.LINE\_AA)

cv2.imshow('Traffic Signal', signal\_image)

cv2.waitKey(1) # Allow the GUI to process events

return elapsed\_time + 1, max(0, red\_signal\_duration - 1) # Increment elapsed time, update red signal duration

red\_signal\_duration = 0

elapsed\_time = 0

while True:

# Clear the list for each frame

no\_helmet\_images = []

ret, img = cap.read()

if not ret:

break

img = imutils.resize(img, height=500)

height, width = img.shape[:2]

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output\_layers)

confidences = []

boxes = []

classIds = []

total\_helmet\_count = 0

total\_no\_helmet\_count = 0

all\_wearing\_helmet = True

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

classIds.append(class\_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

for i in range(len(boxes)):

if i in indexes:

x, y, w, h = boxes[i]

helmet\_roi = img[max(0, y):max(0, y) + max(0, h) // 4, max(0, x):max(0, x) + max(0, w)]

if classIds[i] == 0:

if helmet\_or\_nohelmet(helmet\_roi) == 1:

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

total\_helmet\_count += 1

else:

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

all\_wearing\_helmet = False

if 0 <= y < img.shape[0] and 0 <= x < img.shape[1]:

image\_path = os.path.join(output\_folder, f'no\_helmet\_{total\_no\_helmet\_count}.jpg')

cropped\_image = img[y:y+h, x:x+w]

no\_helmet\_images.append(cropped\_image)

total\_no\_helmet\_count += 1

elapsed\_time, red\_signal\_duration = control\_traffic\_signal(total\_no\_helmet\_count, all\_wearing\_helmet, elapsed\_time, red\_signal\_duration)

# Display the original frame with the count of people without helmets

cv2.putText(img, f'Total no-helmet count: {total\_no\_helmet\_count}', (10, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

cv2.imshow('Original Frame with Count', img)

cv2.waitKey(1) # Allow the GUI to process events

# Display cropped images of riders without helmets in a single pop-up window

if no\_helmet\_images:

print("Displaying all cropped images without helmets in one pop-up:")

max\_height = max(image.shape[0] for image in no\_helmet\_images)

resized\_images = [cv2.resize(image, (int(image.shape[1] \* max\_height / image.shape[0]), max\_height)) for image in no\_helmet\_images]

collage = np.concatenate(resized\_images, axis=1) # Concatenate resized images horizontally

cv2.imshow('Cropped Images Collage', collage)

else:

# Display "OFF SCREEN.jpeg" when there are no individuals without helmets

off\_screen\_image = cv2.imread("C:/Helmet Detection2/Helmet and Number Plate Detection and Recognition/Off Screen.jpeg")

cv2.imshow('Cropped Images Collage', off\_screen\_image)

time.sleep(2) # Wait for 1 second

# Release resources

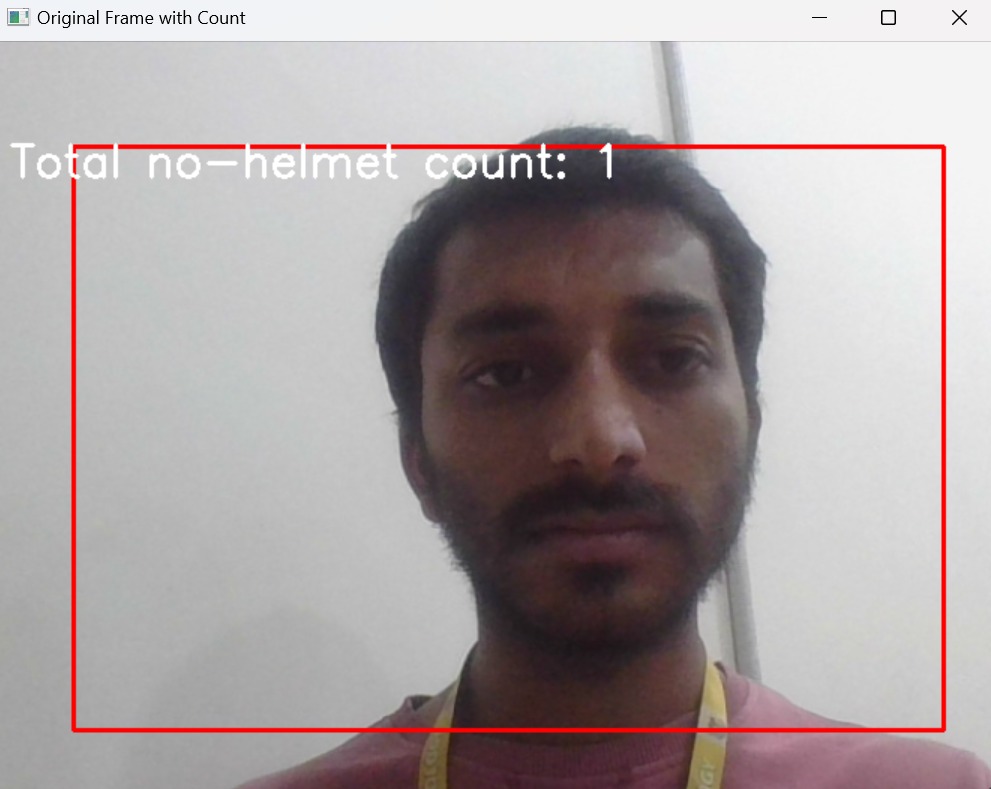
writer.release()

cap.release()

cv2.destroyAllWindows()

1. **OUTPUT**

When there is a detection of not wearing helmet, the signal transition changes to red signal.





1. **Future Work**

The future work of the project mainly focuses on implementing it in daily traffic. Where we can find a large number of bike riders. For those who don’t wear the helmet when riding. It will be a big advantage to the road traffic authority to identify and punish them.

A black line with black text

Description automatically generated