REAL TIME OBJECT TRACKING USING YOLOv5

MINI PROJECT REPORT

Submitted by

AKSSHAYA B (202009003)

AMIRTHAVARSHINI B (202009004)

BHUVANA S (202009008)

MANJU BALA S (202009026)

in

19AD651 – DEEP LEARNING LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2023

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **"AKSSHAYA B (202009003), AMIRTHAVARSHINI B (202009004), BHUVANA S (202009008), MANJU BALA S (202009026)"** for the mini project titled **"REAL TIME OBJECT TRACKING USING YOLOv5"** in 19AD651 – Deep Learning Laboratory during the sixth semester Jan 2023 – May 2023 under my supervision.

SIGNATURE

Mrs.L.Prasika,
Assistant Professor,
Dept. of Artificial Intelligence and Data Science,
Mepco Schlenk Engg.College,Sivakasi.

SIGNATURE

Dr.J.Angela Jennifa Sujana,
Asso. Professor (Senior Grade) & Head,
Dept. of Artificial Intelligence and Data Science,
Mepco Schlenk Engg.College,Sivakasi.

ABSTRACT

Real time object detection is a vast, vibrant, and complex area of computer vision. If there is a single object to be detected in an image, it is known as Image Localization and if there are multiple objects in an image, then it is Object Detection. This detects the semantic objects of a class in digital images and videos. The applications of real time object detection include tracking objects, video surveillance, pedestrian detection, people counting, self-driving cars, face detection, ball tracking in sports and many more. Convolution Neural Networks is a representative tool of Deep learning to detect objects using OpenCV (Opensource Computer Vision), which is a library of programming functions mainly aimed at real time computer vision. Object tracking refers to the ability to estimate or predict the position of a target object in each consecutive frame in a video once the initial position of the target object is defined. Object tracking is a deep learning process where the algorithm tracks the movement of an object. In other words, it is the task of estimating or predicting the positions and other relevant information of moving objects in a video.

The project "Real Time Object Tracking Using YOLOv5" aims to develop a robust and efficient object tracking system using the state-of-the-art You Only Look Once version 5 (YOLOv5) deep learning algorithm. The system is designed to detect and track multiple objects in real-time video streams, with applications in various fields such as surveillance, traffic monitoring, and robotics. The project involves training the YOLOv5 model on a large dataset, optimizing its parameters, and implementing it in a real-time environment using Python programming language and OpenCV library. The accuracy and performance of the system are evaluated using various metrics, including detection rate, tracking precision, and frame rate. The results demonstrate the effectiveness of the proposed system in detecting and tracking objects in real-time with high accuracy and efficiency, making it a promising tool for a wide range of applications.

INTRODUCTION

Project Objective:

The motive of object tracking is to recognize and locate all known objects in a video. Preferably in 3D space, recovering pose of objects in 3D is very important for robotic control systems. Imparting intelligence to machines and making robots more and more autonomous and independent has been a sustaining technological dream for the mankind. It is our dream to let the robots take on tedious, boring, or dangerous work so that we can commit our time to more creative tasks. Unfortunately, the intelligent part seems to be still lagging. In real life, to achieve this goal, besides hardware development, we need a software that can enable robot the intelligence to do the work and act independently. A robot cannot be too intelligent if it cannot see and adapt to a dynamic environment. The searching or recognition process in real time scenario is very difficult. So far, no effective solution has been found for this problem. Object detection is relatively simpler if the machine is looking to detect one particular object. However, recognizing all the objects inherently requires the skill to differentiate one object from the other, though they may be of same type. Such a problem is very difficult for machines if they do not know about the various possibilities of objects.

Motivation:

The motivation behind the project "Real Time Object Tracking Using YOLOv5" is the growing demand for efficient and accurate object tracking systems in various fields such as surveillance, traffic monitoring, and robotics. Traditional object tracking methods often rely on hand-crafted features and have limited accuracy and efficiency in complex real-world scenarios. On the other hand, deep learning-based object detection and tracking algorithms such as YOLOv5 have shown promising results in recent years, but their real-time performance remains a challenge. Thus, this project aims to develop a real-time object tracking system that can accurately and efficiently detect and track multiple objects in a video stream using the latest advancements in deep learning and computer vision. The successful implementation of such a system can have numerous practical applications, such as improving public safety, optimizing traffic flow, and enhancing the performance of autonomous robots.

BACKGROUND

- **DATA PREPERATION STAGE**

In the data preparation stage, we collect and annotate images for training the model. The annotation involves marking the objects of interest in the images with bounding boxes and assigning labels to the objects. For this purpose, we use the open-source tool called LabelImg for annotation. After annotation, the images are split into training and validation sets.

We select and download high quality images from the web. Then, we annotate it with the above said tool, LabelImg. From these annotated images, we split up the testing images.

- **MODEL TRAINING STAGE**

The first step in the model training process is to collect and prepare a dataset of images that includes the objects of interest. The video author uses the COCO dataset, which contains over 330,000 images and 80 object categories. Once the dataset is prepared, it is split into training and validation sets to evaluate the performance of the model during training.

The next step is to configure the model architecture and hyperparameters. In this project, YOLOv4 is used as the base model, and the author modifies its architecture and parameters to improve the model's accuracy and speed. The author also uses transfer learning, which involves pre-training the model on a large dataset and fine-tuning it on the specific dataset of interest. The pre-training is performed on the ImageNet dataset, which contains over 14 million labeled images.

Once the model architecture and parameters are set up, the training process can begin. The author trains the model on a GPU using the Darknet framework, which is optimized for deep learning tasks such as object detection. During training, the model is fed with batches of images and their corresponding labels, and the weights of the model are updated using the backpropagation algorithm to minimize the loss between the predicted and ground-truth labels.
The author monitors the training progress by evaluating the loss function and the accuracy on the validation set. If the model overfits the training set, regularization techniques such as dropout or weight decay can be used to prevent the model from memorizing the training data. The author

also uses data augmentation techniques such as flipping and rotating the images to increase the diversity of the training data and improve the model's generalization ability.

The training process can take several hours or even days depending on the size of the dataset and the complexity of the model architecture. Once the training is complete, the author evaluates the performance of the model on a test set, which contains images that were not used during training or validation. The author also visualizes the model's predictions on sample images to assess its accuracy and identify areas for improvement.

- **OBJECT TRACKING STAGE**

The aim of object tracking is to detect all instances of objects from a known class, such as people, cars or faces in an video. Generally, only a small number of instances of the object are present in the video, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored. Each detection of the frame in the video is reported with some form of pose information. This is as simple as the location of the object, a location and scale, or the extent of the object defined in terms of a bounding box. In some other situations, the pose information is more detailed and contains the parameters of a linear or non-linear transformation. For example, for face detection in a face detector may compute the locations of the eyes, nose and mouth, in addition to the bounding box of the face. An example of bicycle detection in an image that specifies the locations of certain parts is shown in Figure. The pose can also be defined by a three-dimensional transformation specifying the location of the object relative to the camera. Object detection systems always construct a model for an object class from a set of training examples. In the case of a fixed rigid object in an image, only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability.

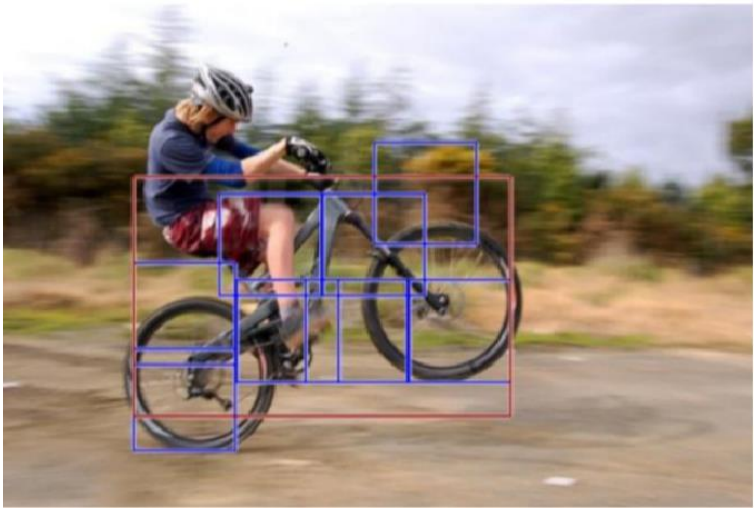

Figure 1: Image Detection of Cycle

Convolutional implementation of the sliding windows Before we discuss the implementation of the sliding window using convents, let us analyze how we can convert the fully connected layers of the network into convolutional layers. Fig. 2 shows a simple convolutional network with two fully connected.

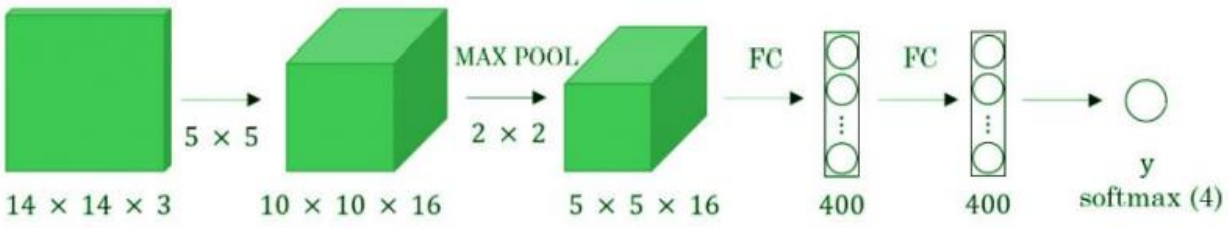

Figure 2: Simple convolution network

A fully connected layer can be converted to a convolutional layer with the help of a 1D convolutional layer. The width and height of this layer is equal to one and the number of filters are equal to the shape of the fully connected layer. An example of this is fig 3.

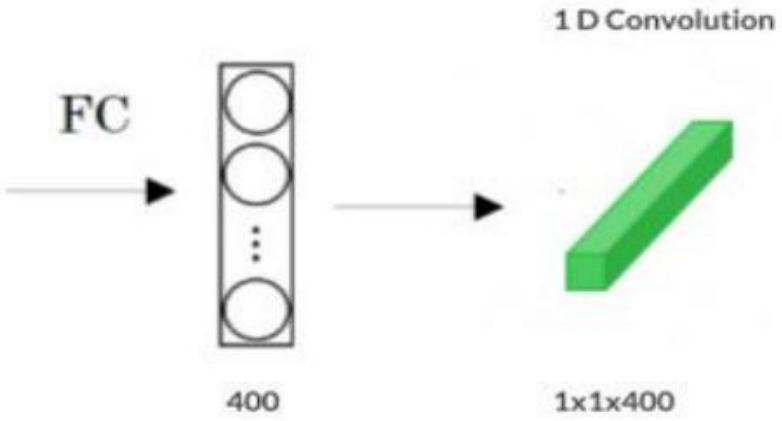

Figure 3: Convolution Layer

We can apply the concept of conversion of a fully connected layer into a convolutional layer to the model by replacing the fully connected layer with a 1-D convolutional layer. The number of filters of the 1D convolutional layer is equal to the shape of the fully connected layer. This representation is shown in Fig 4. Also, the output softmax layer is also a convolutional layer of shape (1, 1, 4), where 4 is the number of classes to predict.

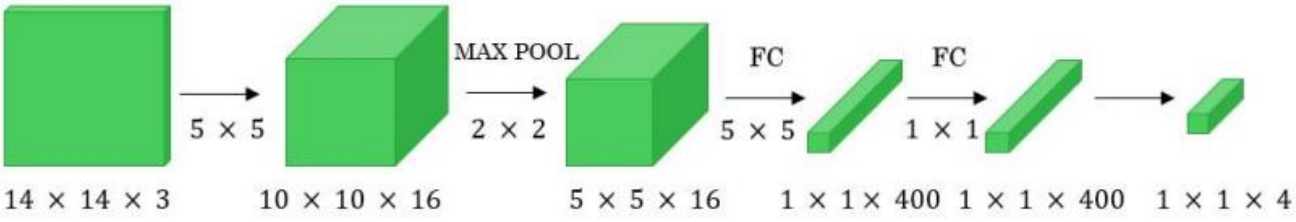

Figure 4: Resizing grid size

Now, let's extend the above approach to implement a convolutional version of the sliding window. First, let us consider the ConvNet that we have trained to be in the following representation (no fully connected layers).

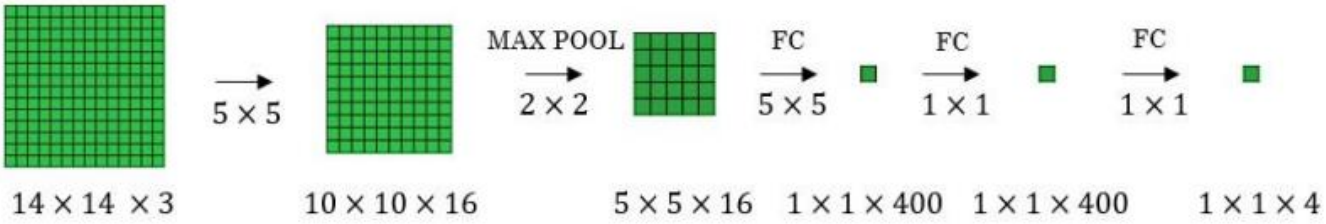

Figure 5: Griding

Let's assume the size of the input image to be 16 × 16 × 3. If we are using the sliding window approach, then we would have passed this image to the above ConvNet four times, where each time the sliding window crops the part of the input image matrix of size 14 × 14 × 3 and pass it through the ConvNet. But instead of this, we feed the full image (with shape 16 × 16 × 3) directly into the trained ConvNet (see Fig. 6). This results will give an output matrix of shape 2 × 2 × 4. Each cell in the output matrix represents the result of the possible crop and the classified value of the cropped image. For example, the left cell of the output matrix(the green one) in Fig. 6 represents the result of the first sliding window. The other cells in the matrix represent the results of the remaining sliding window operations.

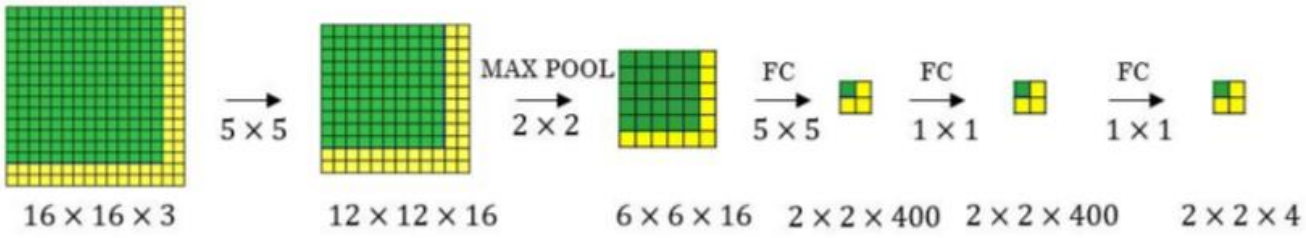

Figure 6: ConvNet

The stride of the sliding window is decided by the number of filters used in the Max Pool layer. In the example above, the Max Pool layer has two filters, and for the result, the sliding window moves with a stride of two resulting in four possible outputs to the given input. The main advantage of using this technique is that the sliding window runs and computes all values simultaneously. Consequently, this technique is really fast. The weakness of this technique is that

the position of the bounding boxes is not very accurate. A better algorithm that tackles the issue of predicting accurate bounding boxes while using the convolutional sliding window technique is the YOLO algorithm. YOLO stands for you only look 6 once which was developed in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. It is popular because it achieves high accuracy while running in real-time. This algorithm requires only one forward propagation pass through the network to make the predictions. This algorithm divides the image into grids and then runs the image classification and localization algorithm (discussed under object localization) on each of the grid cells. For example, we can give an input image of size 256 × 256. We place a 3 × 3 grid on the image.

USED LIBRARY

OpenCV

OpenCV stands for Open supply pc Vision Library is associate open supply pc vision and machine learning software system library. The purpose of creation of OpenCV was to produce a standard infrastructure for computer vision applications and to accelerate the utilization of machine perception within the business product. It becomes very easy for businesses to utilize and modify the code with OpenCV as it is a BSD-licensed product. It is a rich wholesome library as it contains 2500 optimized algorithms, which also includes a comprehensive set of both classic and progressive computer vision and machine learning algorithms. These algorithms is used for various functions such as discover and acknowledging faces. Identify objects classify human actions. In videos, track camera movements, track moving objects. Extract 3D models of objects, manufacture 3D purpose clouds from stereo cameras, sew pictures along to provide a high-resolution image of a complete scene, find similar pictures from a picture information, remove red eyes from images that are clicked with the flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality. Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls The main contributors to the project included a number of optimization experts in Intel Russia, as

well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.

- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.

- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

Numpy

NumPy is library of Python programming language, adding support for large, multi-dimensional array and matrice, along with large collection of high-level mathematical function to operate over these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several developers. In 2005 Travis Olphant created NumPy by incorporating features of computing Num array into Numeric, with extension modifications. NumPy is open-source software and has many contributors.

Matplotlib

Matplotlib is a Python programming language plotting library and its NumPy numerical math extension. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib is an amazing visualization library in Python for 2D plots of arrays. It is built on NumPy arrays and designed to work with the broader SciPy stack. It is easy to use and consists of several plots like line, bar, scatter, histogram, etc.

YOLO- YOU ONLY LOOK ONCE

All the previous object detection algorithms have used regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which has high probabilities of containing the object. YOLO or You Only Look Once is an object detection algorithm much is different from the region based algorithms which seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

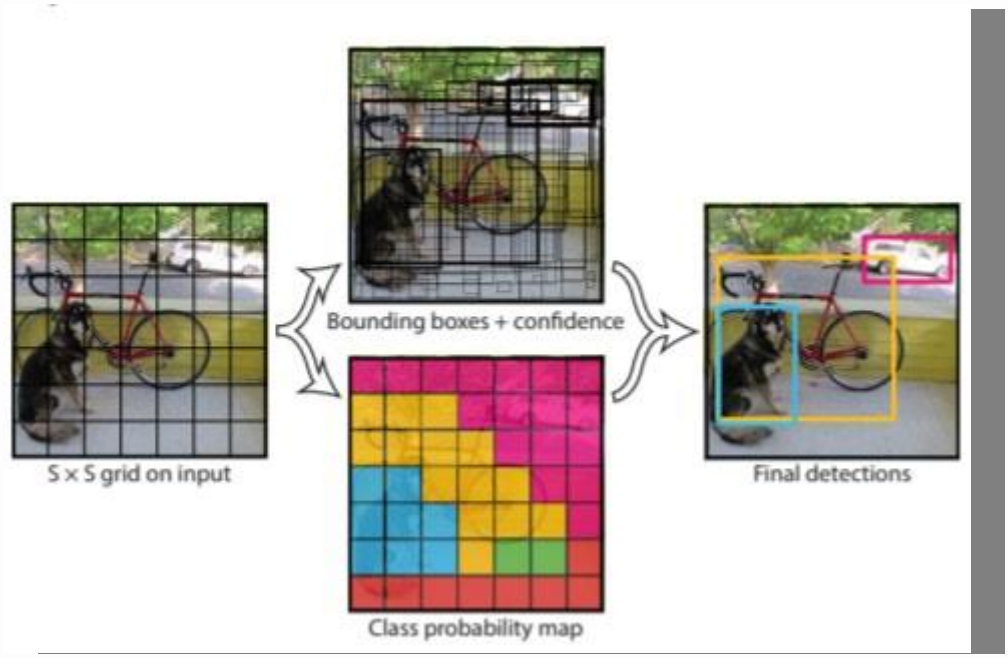

Figure 7: Image Detection

YOLO works by taking an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network gives an output a class probability and offset values for the bounding box. The bounding boxes have the class probability above a threshold value is selected and used to locate the object within the image. YOLO is orders of magnitude faster(45 frames per second) than any other object detection algorithms. The limitation of YOLO algorithm is that it struggles with the small objects within the image, for example, it might have difficulties in identifying a flock of birds. This is due to the spatial constraints of the algorithm.

YOLOv5:

YOLOv5 is an object detection algorithm that is used for identifying and localizing objects within an image or video. It is the latest version of the YOLO (You Only Look Once) family of models, and is known for its accuracy, speed, and ease of use. YOLOv5 was developed by Ultralytics, an AI research company, and was released in June 2020. It is built on the PyTorch framework and uses a deep neural network architecture consisting of convolutional layers, max pooling layers, and fully connected layers. The YOLOv5 model is designed to work with a wide range of image sizes, from 320x320 to 640x640, and can detect objects of varying sizes and aspect ratios. It is also able to detect objects in real-time, with speeds of up to 140 frames per second on a single GPU. One of the key features of YOLOv5 is its use of a novel architecture called the CSP (Cross-Stage Partial) Darknet backbone. This architecture is designed to improve both the accuracy and speed of the model by reducing the number of parameters while increasing the depth and width of the network. The CSP Darknet backbone consists of a series of convolutional layers that are split into two branches, with one branch performing a partial convolution and the other performing a cross-stage connection. This architecture helps to reduce overfitting and improve the generalization of the model. Another important feature of YOLOv5 is its use of a hybrid approach to training the model. This approach combines traditional supervised learning with a technique called self-supervised learning. In traditional supervised learning, the model is trained on labeled data, where each image is labeled with the location and class of each object in the image. In self-supervised learning, the model is trained on unlabeled data, where the model is tasked with learning to predict the next frame in a video sequence. This approach helps to improve the model's ability to generalize to new data and to adapt to changes in the environment. Its novel architecture and hybrid approach to training make it a state-of-the-art model in the field of computer vision, with applications in a wide range of industries, including autonomous driving, robotics, and surveillance.

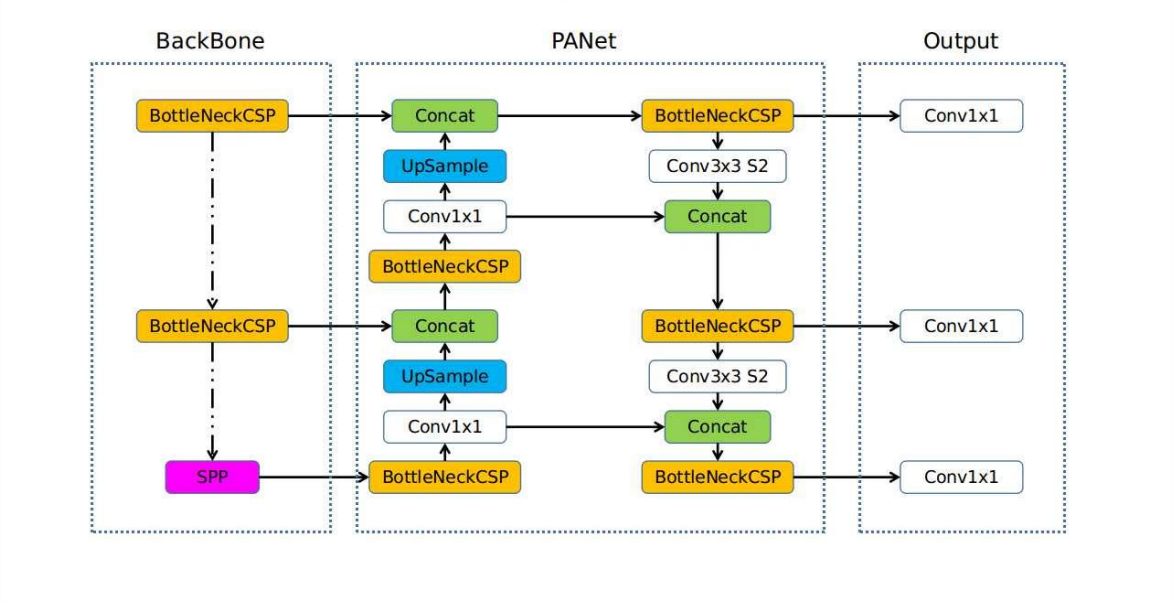

Figure 8: YOLOv5 Model

SOURCE CODE AND OUTPUT

Extract class name from XML

```
import os
from glob import glob # extract path of each file
import pandas as pd # data preprocessing
from xml.etree import ElementTree as et # parse information from XML
from functools import reduce
import warnings
warnings.filterwarnings(&#39;ignore&#39;)
# step-1: get path of each xml file
xmlfiles = glob(&#39;./data_images/*.xml&#39;)
# replace \\ with /
replace_text = lambda x: x.replace(&#39;\\&#39;,&#39;/&#39;)
xmlfiles = list(map(replace_text,xmlfiles))
xmlfiles
# step-2: read xml files
# from each xml file we need to extract
# filename, size(width, height), object(name, xmin, xmax, ymin, ymax)
def extract_text(filename):
tree = et.parse(filename)
root = tree.getroot()
```

```
# extract filename
image_name = root.find(&#39;filename&#39;).text
# width and height of the image
width = root.find(&#39;size&#39;).find(&#39;width&#39;).text
height = root.find(&#39;size&#39;).find(&#39;height&#39;).text
objs = root.findall(&#39;object&#39;)
parser = []
for obj in objs:
name = obj.find(&#39;name&#39;).text
bndbox = obj.find(&#39;bndbox&#39;)

xmin = bndbox.find(&#39;xmin&#39;).text
xmax = bndbox.find(&#39;xmax&#39;).text
ymin = bndbox.find(&#39;ymin&#39;).text
ymax = bndbox.find(&#39;ymax&#39;).text
parser.append([image_name, width, height, name,xmin,xmax,ymin,ymax])

return parser
parser_all = list(map(extract_text,xmlfiles))
data = reduce(lambda x, y : x+y,parser_all)
df                      =                      pd.DataFrame(data,columns                      =
[&#39;filename&#39;,&#39;width&#39;,&#39;height&#39;,&#39;name&#39;,&#39;xmin&#39;,&#39;xmax&#
39;,&#39;ymin&#39;,&#39;ymax&#39;])
df.head()
df.shape
df[&#39;name&#39;].value_counts()
df.info()
# type conversion
cols                                                                                    =
[&#39;width&#39;,&#39;height&#39;,&#39;xmin&#39;,&#39;xmax&#39;,&#39;ymin&#39;,&#39;ymax&#39;]
df[cols] = df[cols].astype(int)
df.info()
# center x, center y
df[&#39;center_x&#39;] = ((df[&#39;xmax&#39;]+df[&#39;xmin&#39;])/2)/df[&#39;width&#39;]
df[&#39;center_y&#39;] = ((df[&#39;ymax&#39;]+df[&#39;ymin&#39;])/2)/df[&#39;height&#39;]
# w
df[&#39;w&#39;] = (df[&#39;xmax&#39;]-df[&#39;xmin&#39;])/df[&#39;width&#39;]
# h
df[&#39;h&#39;] = (df[&#39;ymax&#39;]-df[&#39;ymin&#39;])/df[&#39;height&#39;]
df.head()
### split data into train and test
```

```
images = df[&#39;filename&#39;].unique()
len(images)
# 80% train and 20% test
img_df = pd.DataFrame(images,columns=[&#39;filename&#39;])

img_train = tuple(img_df.sample(frac=0.8)[&#39;filename&#39;]) # shuffle and pick 80% of images
img_test = tuple(img_df.query(f&#39;filename not in {img_train}&#39;)[&#39;filename&#39;]) # take rest 20% images
len(img_train), len(img_test)
train_df = df.query(f&#39;filename in {img_train}&#39;)
test_df = df.query(f&#39;filename in {img_test}&#39;)
train_df.head()
test_df.head()
### Assign id number to object names
# label encoding
def label_encoding(x):
labels = {&#39;person&#39;:0, &#39;car&#39;:1, &#39;chair&#39;:2, &#39;bottle&#39;:3, &#39;potted plant&#39;:4, &#39;bird&#39;:5, &#39;dog&#39;:6,
&#39;sofa&#39;:7, &#39;bicycle&#39;:8, &#39;boat&#39;:9, &#39; motor bike&#39;:10, &#39;TV monitor&#39;:11,
&#39;cow&#39;:12, &#39;sheep&#39;:13, &#39;aeroplane&#39;:14, &#39;train&#39;:15, &#39;table&#39;:18, &#39;bus&#39;:19,&#39;horse&#39;:20}
return labels[x]
train_df[&#39;id&#39;] = train_df[&#39;name&#39;].apply(label_encoding)
test_df[&#39;id&#39;] = test_df[&#39;name&#39;].apply(label_encoding)
train_df.head(10)
### Save Image and Labels in text
import os
from shutil import move
train_folder = &#39;data_images/train&#39;
test_folder = &#39;data_images/test&#39;

os.mkdir(train_folder)
os.mkdir(test_folder)
cols = [&#39;filename&#39;,&#39;id&#39;,&#39;center_x&#39;,&#39;center_y&#39;, &#39;w&#39;, &#39;h&#39;]
groupby_obj_train = train_df[cols].groupby(&#39;filename&#39;)
groupby_obj_test = test_df[cols].groupby(&#39;filename&#39;)
#groupby_obj_train.get_group(&#39;000009.jpg&#39;).set_index(&#39;filename&#39;).to_csv(&#39;sample.txt&#39;,index=False,h
```

```
eader=False)

# save each image in train/test folder and repective labels in .txt
def save_data(filename, folder_path, group_obj):
# move image
src = os.path.join(&#39;data_images&#39;,filename)
dst = os.path.join(folder_path,filename)
move(src,dst) # move image to the destination folder

# save the labels
text_filename = os.path.join(folder_path,
os.path.splitext(filename)[0]+&#39;.txt&#39;)
group_obj.get_group(filename).set_index(&#39;filename&#39;).to_csv(text_filename,sep=&#39;
&#39;,index=False,header=False)

filename_series = pd.Series(groupby_obj_train.groups.keys())
filename_series.apply(save_data,args=(train_folder,groupby_obj_train))
filename_series_test = pd.Series(groupby_obj_test.groups.keys())
filename_series_test.apply(save_data,args=(test_folder,groupby_obj_test))
```

YOLO Training

```
import os

os.chdir('/content/drive/MyDrive/Deep Learning Project/yolo_training')

from google.colab import drive

drive.mount('/content/drive')

!git clone https://github.com/ultralytics/yolov5.git

os.chdir('yolov5')

!pip install -r requirements.txt

"""Training YOLO v5 Model"""

!python train.py --data data.yaml --cfg yolov5s.yaml --batch-size 8 --name Model --epochs 50

!python export.py --weights yolov5s.pt --include torchscript onnx
```

YOLO Predicition Module

```
import cv2
import numpy as np
import os
import yaml
from yaml.loader import SafeLoader

class YOLO_Pred():
    def __init__(self,onnx_model,data_yaml):
        # load YAML
        with open(data_yaml,mode=&#39;r&#39;) as f:
            data_yaml = yaml.load(f,Loader=SafeLoader)
        self.labels = data_yaml[&#39;names&#39;]
        self.nc = data_yaml[&#39;nc&#39;]

        # load YOLO model
        self.yolo = cv2.dnn.readNetFromONNX(onnx_model)
        self.yolo.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        self.yolo.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

    def predictions(self,image):
        row, col, d = image.shape
        # get the YOLO prediction from the the image
        # step-1 convert image into square image (array)
        max_rc = max(row,col)
        input_image = np.zeros((max_rc,max_rc,3),dtype=np.uint8)
        input_image[0:row,0:col] = image
        # step-2: get prediction from square array
        INPUT_WH_YOLO = 640
        blob =
cv2.dnn.blobFromImage(input_image,1/255,(INPUT_WH_YOLO,INPUT_WH_YOLO),swapRB=T
rue,crop=False)
        self.yolo.setInput(blob)
        preds = self.yolo.forward() # detection or prediction from YOLO

        # Non Maximum Supression
        # step-1: filter detection based on confidence (0.4) and probability
score (0.25)
        detections = preds[0]
        boxes = []
```

```
        confidences = []
        classes = []

        # width and height of the image (input_image)
        image_w, image_h = input_image.shape[:2]
        x_factor = image_w/INPUT_WH_YOLO
        y_factor = image_h/INPUT_WH_YOLO

        for i in range(len(detections)):

            row = detections[i]
            confidence = row[4] # confidence of detection an object
            if confidence &gt; 0.4:
                class_score = row[5:].max() # maximum probability from 20
objects
                class_id = row[5:].argmax() # get the index position at which
max probabilty occur
                if class_score &gt; 0.25:

                    cx, cy, w, h = row[0:4]
                    # construct bounding from four values
                    # left, top, width and height
                    left = int((cx - 0.5*w)*x_factor)
                    top = int((cy - 0.5*h)*y_factor)
                    width = int(w*x_factor)
                    height = int(h*y_factor)
                    box = np.array([left,top,width,height])
                    # append values into the list
                    confidences.append(confidence)
                    boxes.append(box)
                    classes.append(class_id)
        # clean
        boxes_np = np.array(boxes).tolist()
        confidences_np = np.array(confidences).tolist()
        # NMS
        index = cv2.dnn.NMSBoxes(boxes_np,confidences_np,0.25,0.45)

        # Draw the Bounding
        if len(index) &gt; 0:
         for ind in index.flatten():
            # extract bounding box
```

```
            x,y,w,h = boxes_np[ind]
            bb_conf = int(confidences_np[ind]*100)
            classes_id = classes[ind]
            class_name = self.labels[classes_id]
            colors = self.generate_colors(classes_id)
            text = f&#39;{class_name}: {bb_conf}%&#39;

            cv2.rectangle(image,(x,y),(x+w,y+h),colors,2)
            cv2.rectangle(image,(x,y-30),(x+w,y),colors,-1)
            cv2.putText(image,text,(x,y-
10),cv2.FONT_HERSHEY_PLAIN,0.7,(0,0,0),1)
            return image

    def generate_colors(self,ID):
        np.random.seed(10)
        colors = np.random.randint(100,255,size=(self.nc,3)).tolist()
        return tuple(colors[ID])
```

Final Prediction

```
import cv2
from yolo_predictions import YOLO_Pred
yolo = YOLO_Pred(&#39;./Model/weights/best.onnx&#39;,&#39;data.yaml&#39;)
img      =      cv2.imread(&quot;C:\\Users\\amirt\\Pictures\\Wallpaper\\photo_2023-04-23_15-33-
46.jpg&quot;)

# predictions
img_pred = yolo.predictions(img)
cv2.imshow(&#39;prediction image&#39;,img_pred)
cv2.waitKey(0)
cv2.destroyAllWindows()



## Real Time Object Tracking
cap = cv2.VideoCapture(&quot;./People.mp4.crdownload&quot;)

while True:
ret, frame = cap.read()
if ret == False:
print(&#39;unable to read video&#39;)
break
```

```
pred_image = yolo.predictions(frame)

cv2.imshow(&#39;YOLO&#39;,pred_image)
if cv2.waitKey(1) == 27:
break

cv2.destroyAllWindows()
cap.release()

cap = cv2.VideoCapture(&quot;./Bird.mp4.crdownload&quot;)

while True:
ret, frame = cap.read()
if ret == False:
print(&#39;unable to read video&#39;)
break

pred_image = yolo.predictions(frame)

cv2.imshow(&#39;YOLO&#39;,pred_image)
if cv2.waitKey(1) == 27:
break

cv2.destroyAllWindows()
cap.release()
cap = cv2.VideoCapture(0)

while True:
ret, frame = cap.read()
if ret == False:
print(&#39;unable to read video&#39;)
break

pred_image = yolo.predictions(frame)

cv2.imshow(&#39;YOLO&#39;,pred_image)
if cv2.waitKey(1) == 27:
break
cv2.destroyAllWindows()
cap.release()
```

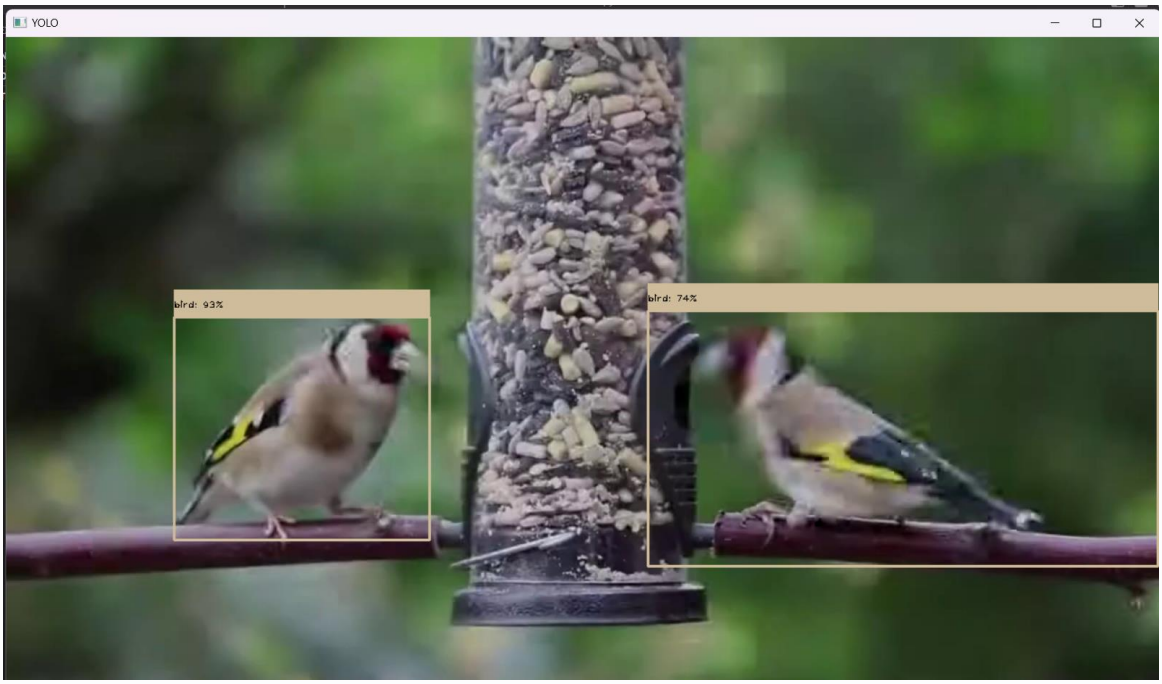

Figure 9: Object Detection in Image

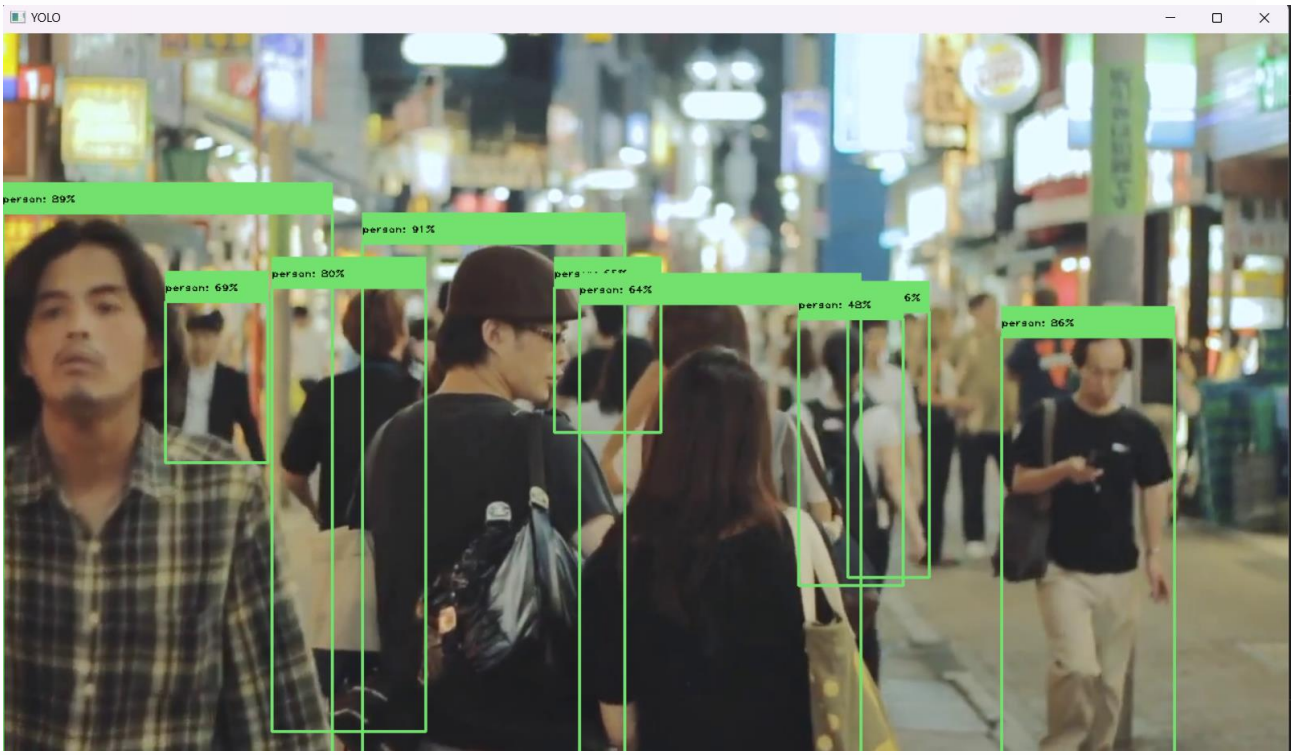

Figure 10: Object Detection in Video

Figure 11: Object Detection in Real-Time Webcam

CONCLUSION

YOLO is a futuristic recognizer that has faster FPS and is more accurate than available detectors. The detector can be trained and used on a conventional GPU which enables widespread adoption. New features in YOLOv5 improve accuracy of the classifier and detector. The workflow of the YOLO is cloning the Darknet then enabling GPU, Loading YOLOV5 weights and names, Building Darknet Frame work, Testing image, Setting bounding boxes, labels and Confidence, Open web cam, Detect the object. We have used Darknet and python libraries to do object detection using YOLOV5 algorithm and have also detected object precisely using YOLO weights and YOLO names.