SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SCHOOL OF COMPUTING

DEPARTMENT OF DATASCIENCE AND BUSINESS SYSTEMS

18CSC305J ARTIFICIAL INTELLIGENCE

MINI PROJECT REPORT

Object Detection

Name: Adarsh Shailendra

Register Number: RA1911027010004

Mail ID: as8969@srmist.edu.in

Department: DSBS

Specialization: CSE-Big Data

Semester: 6

Team Members

Name – Chirag Bengani	**Registration Number – RA1911027010002**
Name – Adarsh Shailendra	**Registration Number – RA1911027010004**
Name – Nishok Ilangovan	**Registration Number – RA1911027010009**
Name – Pranav Natarajan	**Registration Number – RA1911027010016**

Content Page

Abstract

Object detection is a computer vision technique that works to identify and locate objects within an image or video. Specifically, object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene. Our model utilizes YOLOv5 for better accuracy and less computation

YOLO is an acronym that stands for You Only Look Once. We are employing Version 5, which was launched by Ultralytics in June 2020 and is now the most advanced object identification algorithm available. It is a novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each component. These bounding boxes are weighted by the expected probability. The method “just looks once” at the image in the sense that it makes predictions after only one forward propagation run through the neural network. It then delivers detected items after non-max suppression (which ensures that the object detection algorithm only identifies each object once).

We have used Roboflow to make boundary boxes for the labels of the images in our training dataset. Roboflow Annotate is a self-serve annotation tool included with all Roboflow accounts that greatly streamlines the process of going from raw images to a trained and deployed computer vision model.

Finally we host it as a webapp using streamlit. Streamlit is an open-source python framework for building web apps for Machine Learning and Data Science.

Chapter 1: Introduction and motivation

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.

Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

Object detection is breaking into a wide range of industries, with use cases ranging from personal security to productivity in the workplace. Object detection and recognition is applied in many areas of computer vision, including image retrieval, security, surveillance, automated vehicle systems and machine inspection. Significant challenges stay on the field of object recognition. The possibilities are endless when it comes to future use cases for object detection.

The motive of object detection is to recognize and locate (localize) all known objects in a scene. Preferably in 3D space, recovering pose of objects in 3D is very important for robotic control systems. The information from the object detector can be used for obstacle avoidance and other interactions with the environment.

Our motivation for this project was to evolve it into something useful for a product project. An application which hasn't been researched on is Object Detection in Online Shopping.

Chapter 2: Review of Existing methods and their Limitations

Existing object detection methods involve copying and cloning heavy libraries with a large number of dependencies and unalterable methods.

Some limitations of object detection are:

1) Illumination conditions - Lighting has a very large influence on the definition of objects. The same objects will look different depending on the lighting conditions.
2) Cluttered or textured background - Objects that need to be identified may blend into the background, making it difficult to identify them.
3) Variety - The same object can have completely different shapes and sizes. Computer vision needs to do a lot of research to read an object and understand what it means.
4) Speed - When it comes to video, detectors need to be trained to perform analysis in an ever-changing environment. It means that object detection algorithms must not only accurately classify important objects but also be incredibly fast during prediction to be able to identify objects that are in motion.

Chapter 3: Proposed Method with System Architecture / Flow Diagram

We propose a webapp for the sole purpose of object detection. Since Object detection modules are very process heavy and slow in their entirety, we plan to stop and bridge the gap between these. Anyone who plans to identify objects in a video, photo or real-time for any purpose be it research, identifying Region of Interests or just identifying unknown objects can use the webapp

This webapp will use a custom pre-trained model which will be a variation of the model published by popular Object Detection Module YOLOv5

We plan on leveraging the model provided, enhance it a bit, and deploy it in the webapp. The webapp's entire framework will be based on markdown with the help of a popular Web application library called Streamlit and can be hosted on hosting clouds (i.e. Heroku, Netlify)

Chapter 4: Modules Description

Module 1 – Environment Setup and library Installation

We initiate our work by setting up a virtual environment inside our desired directory. A virtual environment helps us to easily set up our environment and enables us to install all the required libraries in one go. After we're done setting up the virtual environment, we clone YOLOv5's Github Repository in order to perform our own operations and get the list of libraries required.

After downloading the repository, we use their 'Requirements.txt' file to install all the required libraries using the command line and pip in our virtual environment.

Module 2 – Custom Training of Pre-made model

YOLOv5 model takes pictures with bounding boxes drawn around objects and annotation files for them in text format defining the specifications for the drawn bounding box for training. In order to get that, we proceed to Roboflow which is an online tool used for making bounding boxes in images and getting their annotation files.

We make the bounding boxes in Roboflow and save the dataset. Later, we import that dataset into our notebook with the help of Roboflow's API. Succeeding that process, we initiate training on top of the pre-made model with the help of the command line and a training file provided by YOLO itself. We give the necessary arguments like training epochs, weights, batch size etc and let the model train.

Module 3 – Testing of the model

In this part, we use completely different inputs to test our model's accuracy and confidence. The model takes three different kinds of inputs i.e. Still Picture, Video and Realtime Video. We test for all three kinds of inputs using the detection file provided by the Model after training.

Module 4 – Construction of Webapp and Final Modifications

In this module, we work on the Webapp which is made with the help of a Python Library called Streamlit. Streamlit is a library which is based on markdown and helps us to create webapps with the help of its inbuilt functions. We integrate our detection functions with the webapp and modify it's frontend according to our needs.

Chapter 5: Implementation requirements

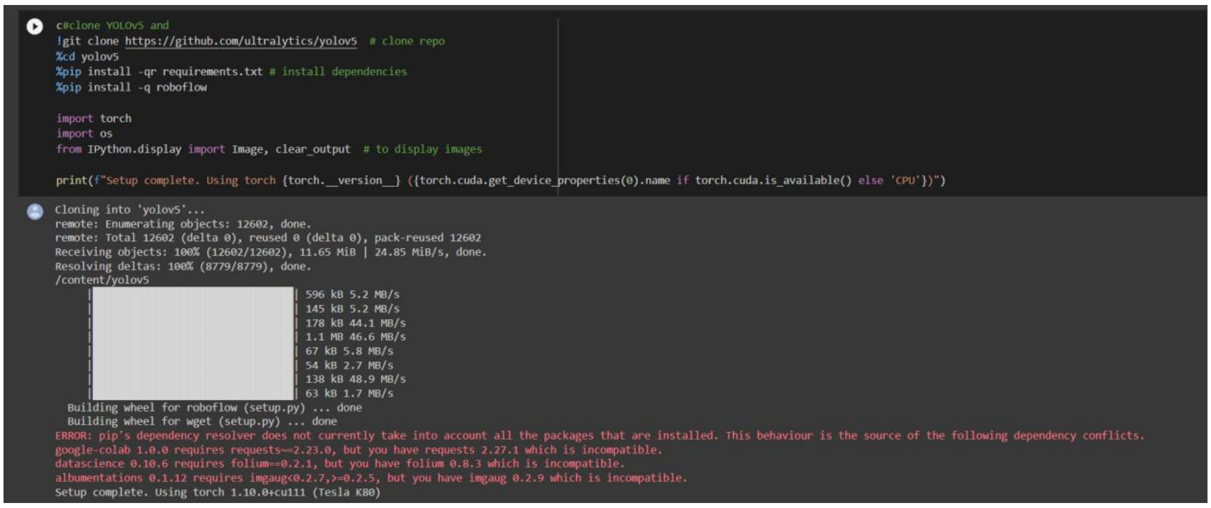
Essential Software Requirements and Libraries

- Python
- Markdown 3.3.6
- Tensorboard
- YOLOv5
- PyTorch – vision
- Streamlit

External Requirements

- Fast internet connection
- CPU – i3 5th Gen +
- RAM – 4GB +
- Git bash (Standalone / Integrated)

Chapter 6: Output Screenshots



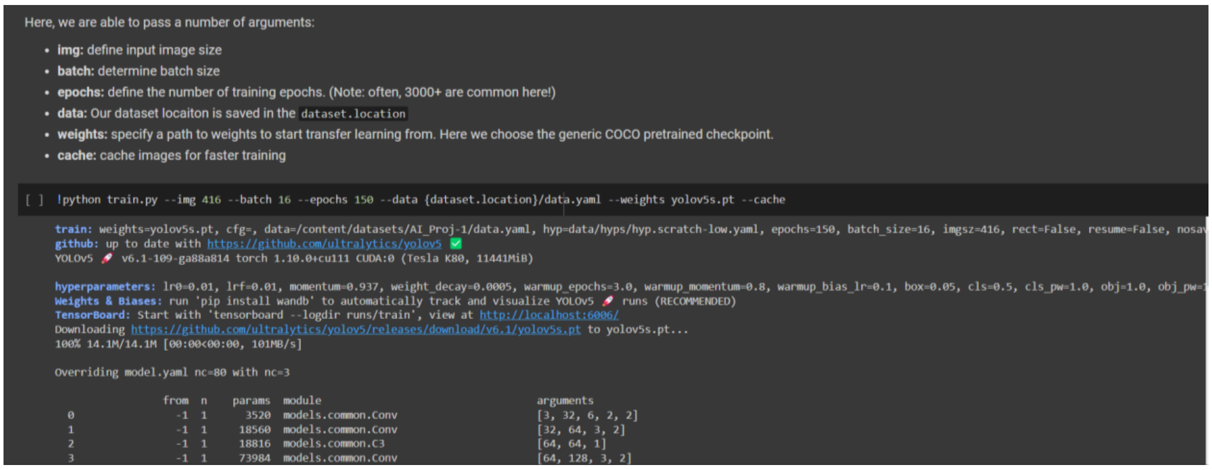
```
c#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 12602, done.
remote: Total 12602 (delta 0), reused 0 (delta 0), pack-reused 12602
Receiving objects: 100% (12602/12602), 11.65 MiB | 24.85 MiB/s, done.
Resolving deltas: 100% (8779/8779), done.
/content/yolov5
     |                                | 596 kB 5.2 MB/s
     |                                | 145 kB 5.2 MB/s
     |                                | 178 kB 44.1 MB/s
     |                                | 1.1 MB 46.6 MB/s
     |                                | 67 kB 5.8 MB/s
     |                                | 54 kB 2.7 MB/s
     |                                | 138 kB 48.9 MB/s
     |                                | 63 kB 1.7 MB/s
  Building wheel for roboflow (setup.py) ... done
  Building wheel for wget (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.27.1 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
Setup complete. Using torch 1.10.0+cu111 (Tesla K80)
```

Fig 1: Importing some necessary libraries



Here, we are able to pass a number of arguments:

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
- **data:** Our dataset locaiton is saved in the `dataset.location`
- **weights:** specify a path to weights to start transfer learning from. Here we choose the generic COCO pretrained checkpoint.
- **cache:** cache images for faster training

```
[ ] !python train.py --img 416 --batch 16 --epochs 150 --data {dataset.location}/data.yaml --weights yolov5s.pt --cache
```

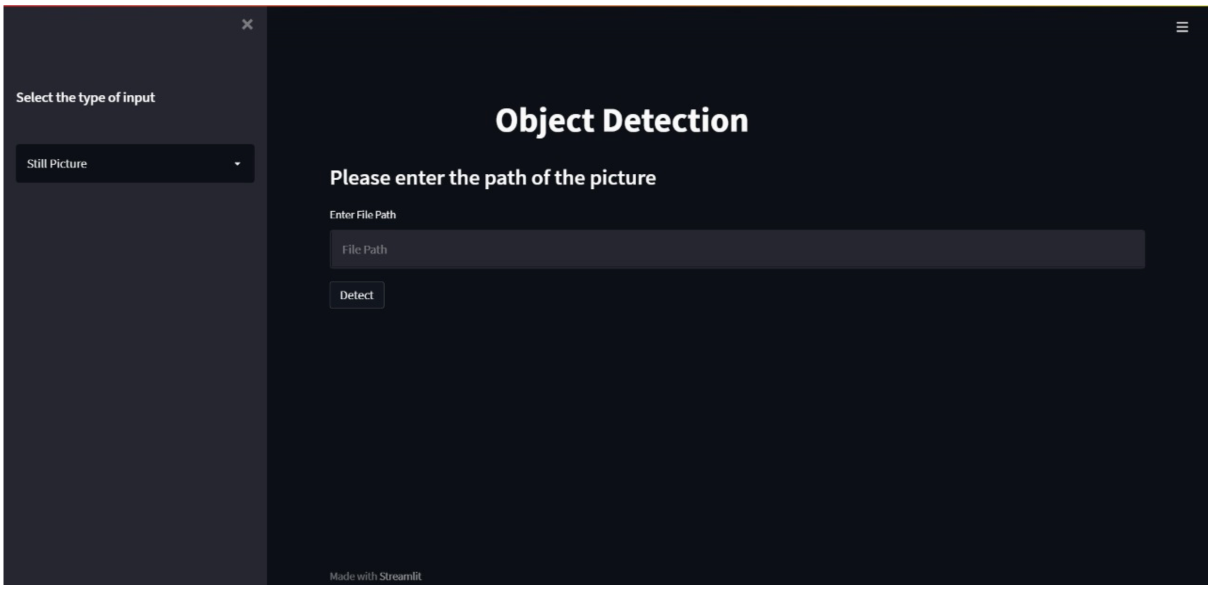
```
train: weights=yolov5s.pt, cfg=, data=/content/datasets/AI_Proj-1/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=150, batch_size=16, imgsz=416, rect=False, resume=False, nosav
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v6.1-109-ga88a814 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://github.com/ultralytics/yolov5/releases/download/v6.1/yolov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 101MB/s]

Overriding model.yaml nc=80 with nc=3

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
```

Fig 2: Training on custom self-annotated dataset



Select the type of input

Still Picture

Object Detection

Please enter the path of the picture

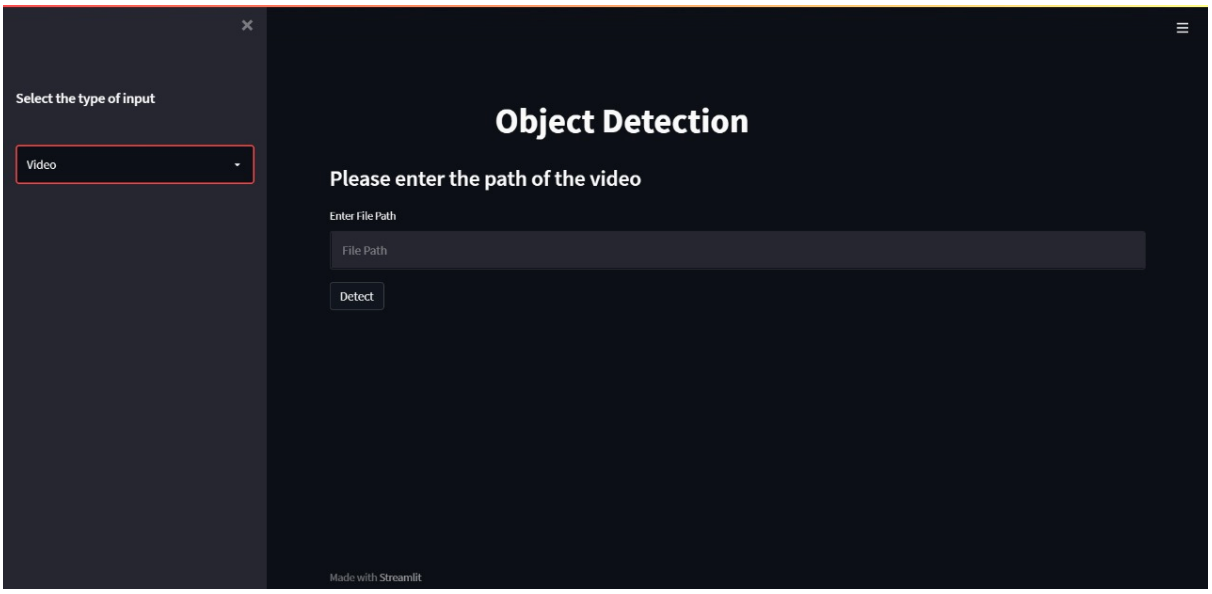
Enter File Path

File Path

Detect

Made with Streamlit

Fig 3: Webapp Still picture input interface



Select the type of input

Video

Object Detection

Please enter the path of the video

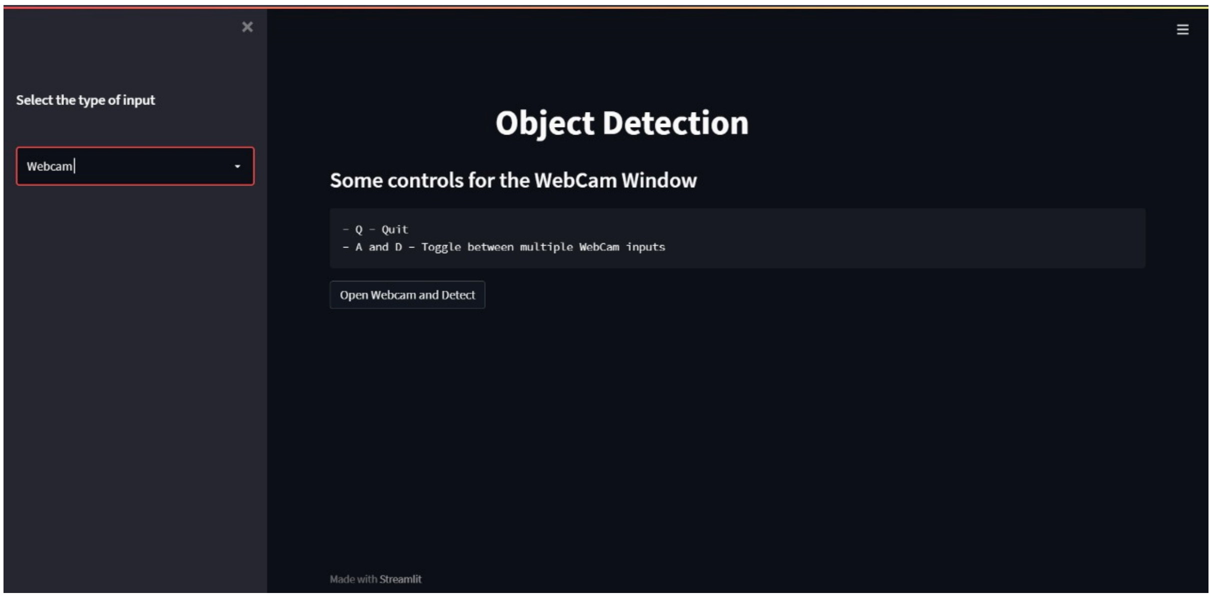
Enter File Path

File Path

Detect

Made with Streamlit

Fig 4: Webapp Video input interface



Some controls for the WebCam Window

```
- Q - Quit
- A and D - Toggle between multiple WebCam inputs
```

Fig 5: Webapp Webcam input Interface

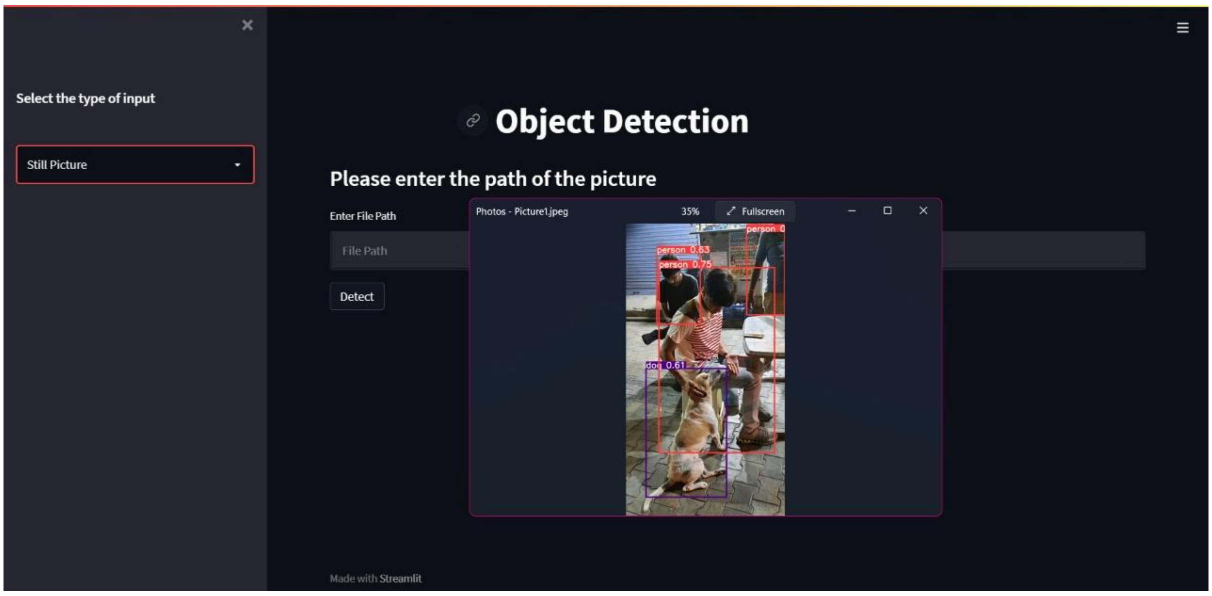


Fig6: Detected Objects in a still picture

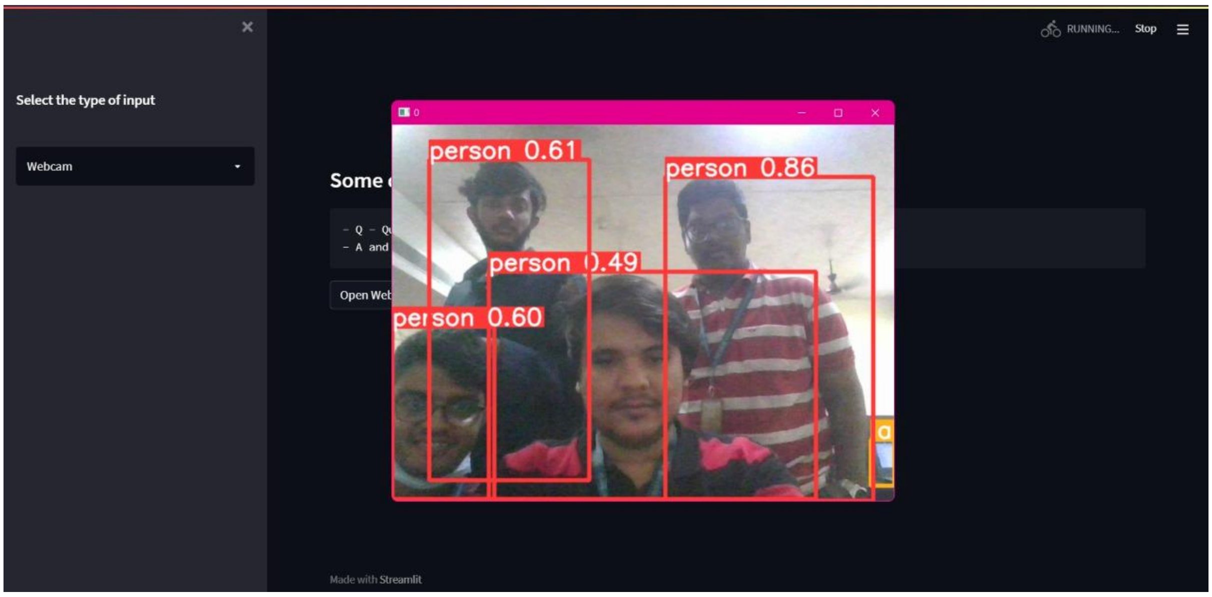


Fig 7: Detected Objects in real-time using Webcam

Conclusion

The object detection feature which we wanted to display successfully works and is able to detect objects in videos, static images and real time videos with high accuracy and low computation. This object detection model can be implemented in many real time applications and projects and one such application with major potential is shopping sites where we can browse for products by giving images and videos as the input instead of searching by their product names.

References

https://machinelearningmastery.com/object-recognition-with-deep-learning/

https://ieeexplore.ieee.org/abstract/document/8627998

https://www.sciencedirect.com/science/article/pii/S1877050918308767

https://arxiv.org/pdf/1807.05511.pdf

Appendix A:

Source Code – Webapp

```
import os
import streamlit as st

st.set_page_config(layout='wide',initial_sidebar_state='expanded')

titles0, title, titles1 = st.columns((1.6,6,0.5))
title.title('Object Detection')

st.sidebar.subheader('Select the type of input')
selectbox = st.sidebar.selectbox('',('Still Picture','Video','Webcam'))

if selectbox=='Still Picture' :
    st.markdown("""### Please enter the path of the picture""")
    filepath1 = st.text_input('Enter File Path',placeholder='File Path')
    raw_filepath = r'{}'.format(filepath1)
    if st.button('Detect'):
        stream = os.popen(f'python detect.py --source ./{raw_filepath}')
        output = stream.read()
        if output is not None:
            st.write(output)

if selectbox=='Video':
    st.markdown("""### Please enter the path of the video""")
    filepath1 = st.text_input('Enter File Path',placeholder='File Path')
    raw_filepath = r'{}'.format(filepath1)
    if st.button('Detect'):
        stream = os.popen(f'python detect.py --source ./{raw_filepath}')
        output = stream.read()
        if output is not None:
            st.write(output)


if selectbox=='Webcam':
    st.markdown("""### Some controls for the WebCam Window
    - Q - Quit
    - A and D - Toggle between multiple WebCam inputs""")
    if st.button('Open Webcam and Detect'):
        stream = os.popen(f'python detect.py --source 0')
        output = stream.read()
        if output is not None:
            st.write(output)
```

Appendix B:

Github Profile Link - https://github.com/Adarsh-gif-crypt

Project Repository Link - https://github.com/Adarsh-gif-crypt/Object-Detection-Webapp