YOLOv7 Model Training and NAVQ+ Deployment

Colby Gallaher, Collin Lloyd, Rhia Bipin Roy

**Training**

To begin utilizing the object detection capabilities of YOLOv7, a model must first be trained using predefined weights and biases. To do this, we need to utilize a Google Colab notebook with commands that include pulling down Yolov7 code for object detection, then training the provided dataset (which consists of a file of images) with the ‘yolov7\_training.pt’, which comes included with the repo. We define a range of values for weights and choose a random number so that every training is distinct and leads to consistent output. Afterwards we iterate through the yaml files associated with our images and finally we run detection on the extracted yaml. Once the code iterates through the folder, detecting objects and drawing bounding boxes, once finished, another folder is created.

**Directory** containing results of running model script on NAVQ+: Downloads/colab\_and\_images/yolov7/runs/detect/exp# (Each run will get stored in its own exp folder within this specified directory.)

**Code**

**Link to GitHub:** [**https://github.com/WongKinYiu/yolov7**](https://github.com/WongKinYiu/yolov7)

**Script to allow trained YOLO model to make inferences on specified dataset:**

#!/bin/bash

# Pulls the yolov7 repo from Github  
git clone <https://github.com/WongKinYiu/yolov7>;

# Goes into the ‘yolov7’ directory  
#cd yolov7;

# Installs ‘requirements.txt’ which has a bunch of other packages that need to be install  
#pip install -r requirements.txt;  
  
# Can specify whichever ‘database’ you want, in this case we used images  
dataset="/home/user/Downloads/colab\_and\_images/images"

# In order to iterate through, we used this data.yaml, $ lets you use the dataset variable  
data\_yaml="$dataset/data.yaml"  
  
  
# download COCO starting checkpoint  
#cd /content/yolov7;  
# Allows user to get the training.pt file

wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7\_training.pt;  
# run this cell to begin training  
cd yolov7;

#begin training (You can also specify your own waits by generating another pt file), we tried making a giant array of numbers to serve as weights, but further research needs done into the format (yolov7\_training.pt is an unrecognized format)  
python train.py --batch 16 --epochs 25 --data "$dataset" --weights 'yolov7\_training.pt' ;  
  
# iterate through the data\_yaml variable we mentioned  
for data\_yaml in "$dataset"/\*.yaml; do  
  
 # Use $data\_yaml in your script for testing  
 python test.py --task 'test' --batch 16 --data "$data\_yaml" --weights /home/user/Downloads/colab\_and\_images/yolov7/runs/train/exp/weights/best.pt;

# Runs the detect function from yolov7 repo to draw detection boxes around images and make ids

python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source /home/user/Downloads/traffic\_lights/traffic\_lights  
done

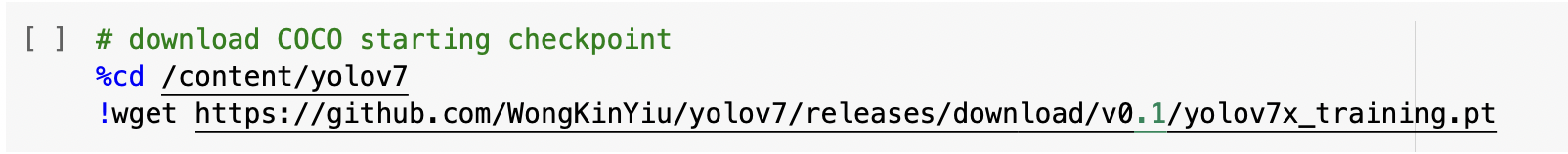
**Note: The line that contains detect.py comes from the Google Colab**

**Note: If encountering a permission denied error, type in: chmod +x ./filename and then run the executable**

**Note: If there are any errors with the image's identification, additional runs of the test.py will make future detection more accurate**

**To train with different weights you can follow the instructions specified in:** [**https://colab.research.google.com/drive/1bIhpSpYLeTV3RFexQmQoDmA17OdKkm7M**](https://colab.research.google.com/drive/1bIhpSpYLeTV3RFexQmQoDmA17OdKkm7M)

**Specify the training file on this line:**

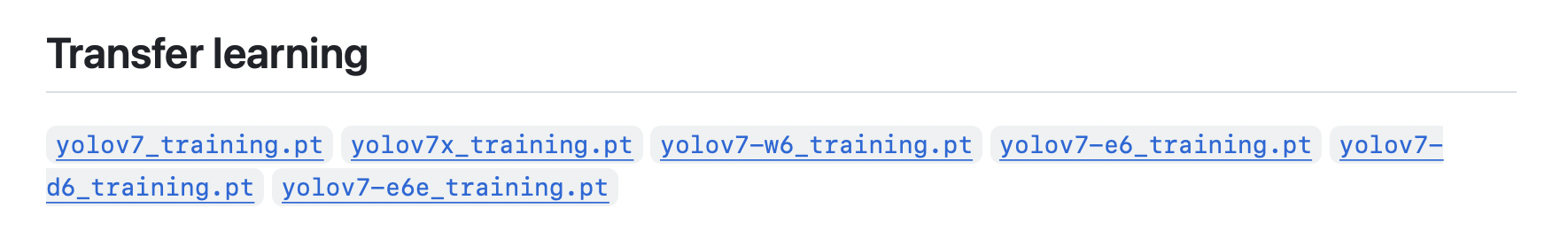


**As shown below, replace yolov7\_training.pt with the new file with new weights**



The following files from the GitHub can be used for this purpose:

For training:



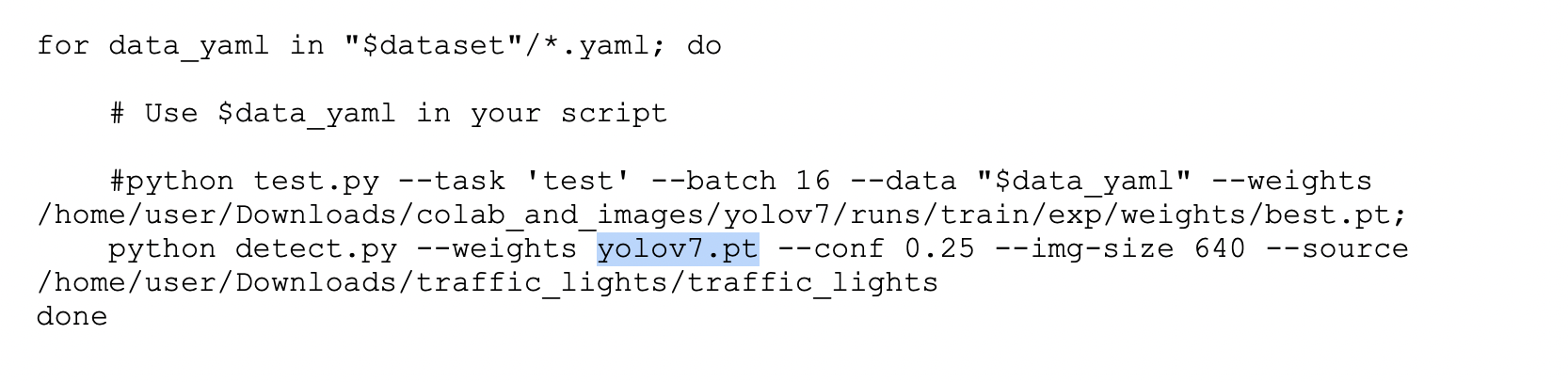
For testing:



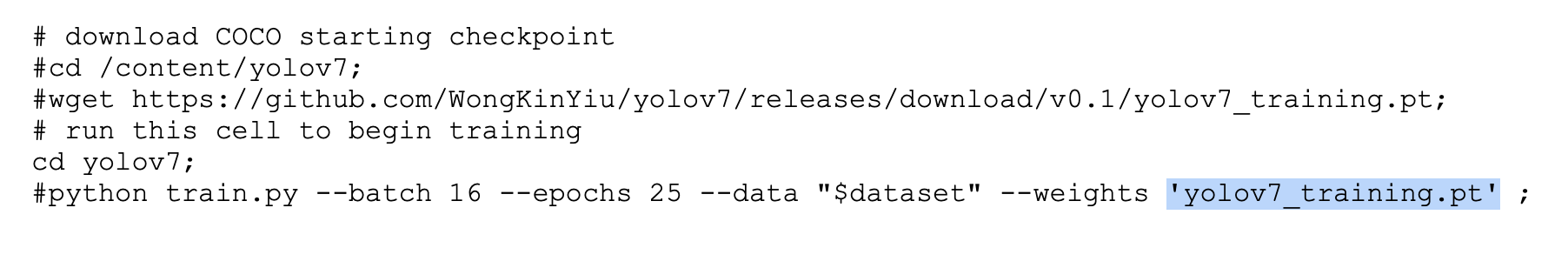
\*The training and testing file versions must match

**If directly editing the provided script above:**

You may specify the name of the new weights file by replacing the highlighted portion shown below:



Important: Make sure that the training file also matches!



**Script to capture screenshots at fixed time intervals from livestream of Google Coral:**

import cv2 #include OpenCV library functions in python  
import os  
import datetime  
  
  
#Create an object to hold reference to camera video capturing  
vidcap = cv2.VideoCapture(3)  
  
  
  
directory='/home/user/Downloads'  
  
os.chdir(directory)  
  
n=0  
  
  
  
#check if connection with camera is successfully  
if vidcap.isOpened():  
   
   
  
 #check whether frame is successfully captured  
   
 # continue to display window until 'q' is pressed  
 while(True):  
 #check whether frame is successfully captured  
   
 ret, frame = vidcap.read() #capture a frame from live video   
   
 if ret:  
   
 cv2.imshow("Frame",frame) #show captured frame  
   
 #press 'q' to break out of the loop  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 if n==300:  
 n=0  
 cv2.imwrite('{}\_{}.{}'.format("test", datetime.datetime.now().strftime('%Y\_%m\_%d\_%H\_%M\_%S%f'),'jpg'), frame)  
 n+=10  
 #print error if frame capturing was unsuccessful  
   
# print error if the connection with camera is unsuccessful  
else:  
 print("Cannot open camera")

**Script to capture screenshots with key press from livestream of Google Coral:**

import time   
import sys   
  
#time.sleep(10)   
  
import cv2 as cv   
import numpy as np   
import os  
  
  
  
capture = cv.VideoCapture('v4l2src device=/dev/video3 ! video/x-raw,framerate=30/1,width=640,height=480 ! appsink', cv.CAP\_GSTREAMER)  
  
#Object to write output. 'XVID' extension is best for linux (allegedly)   
#writer = cv.VideoWriter\_fourcc(\*'XVID')   
writer = cv.VideoWriter\_fourcc('M','J','P','G')  
  
# sets width and height to size that fits webcam (The frame width and height of the capture)   
# Can use capture.set() to set the properties of the video capture   
width = int(capture.get(cv.CAP\_PROP\_FRAME\_WIDTH) )   
height = int(capture.get(cv.CAP\_PROP\_FRAME\_HEIGHT) )   
  
#Object to write output to selected file. Can change extension on file as needed   
out = cv.VideoWriter('/home/user/Downloads/test.avi',writer, 20.0, (width,height))   
  
def store\_capture():   
  
 if not capture.isOpened():   
 print("Camera cannot be opened")   
 exit()   
   
 while capture.isOpened():   
 # ret is a bool indicating the status of frame capture   
 # frame is the actual frame that will be sent to a destination   
 ret, frame = capture.read()   
   
 # case that ret is not true indicating a failed capture of the frame   
 if not ret:   
 print("Frame not captured properly")   
 break   
   
 out.write(frame)   
   
 cv.imshow('frame',frame)   
   
 if cv.waitKey(1) == ord('q'):   
 break   
   
 capture.release()   
 out.release()   
 cv.destroyAllWindows()   
   
store\_capture()   
  
def save\_frame\_camera\_key(device\_num, dir\_path, basename, ext='jpg', delay=1, window\_name='frame'):  
 #cap = cv.VideoCapture(device\_num)  
  
 if not capture.isOpened():  
 return  
  
 os.makedirs(dir\_path, exist\_ok=True)  
 base\_path = os.path.join(dir\_path, basename)  
   
 n = 0  
 while True:  
 ret, frame = capture.read()  
 cv.imshow(window\_name, frame)  
 key = cv.waitKey(delay) & 0xFF  
 if key == ord('c'):  
 cv.imwrite('{}\_{}.{}'.format(base\_path, n, ext), frame)  
 n += 1  
 elif key == ord('m'):  
   
 break  
  
 cv.destroyWindow(window\_name)  
  
  
save\_frame\_camera\_key(3, '/home/user/Downloads/temp', 'camera\_capture')

**UDP (Server):**

import socket  
  
localIP = "192.168.137.233"  
localPort = 1234  
bufferSize = 1024  
  
UDPServerSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
UDPServerSocket.bind((localIP, localPort))  
  
print("UDP Server Listening at:", UDPServerSocket)  
  
# Receive the file data from the client and write it to a file  
with open("received\_file.png", "wb") as file:  
 while True:  
 chunk, address = UDPServerSocket.recvfrom(bufferSize)  
 if not chunk:  
 break  
 print(f"Received chunk of size {len(chunk)} bytes")  
 file.write(chunk)  
  
print("File received from the client and saved as 'received\_file.png'")  
  
# Close the server socket when done  
UDPServerSocket.close(

**UDP (Client):**

import socket  
import os  
  
serverIP = "192.168.137.233" # IP address of UDP Server  
serverPort = 1234  
  
clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
image\_to\_be\_sent = "/home/user/Downloads/walter.jpg"  
  
if not os.path.exists(image\_to\_be\_sent):  
 print(f"Image file '{image\_to\_be\_sent}' does not exist")  
 exit()  
  
# Read the binary content of the file  
with open(image\_to\_be\_sent, "rb") as file:  
 file\_data = file.read()  
  
# Send the file data to the server in chunks  
chunk\_size = 1024 # Needs to be same as   
for i in range(0, len(file\_data), chunk\_size):  
 chunk = file\_data[i:i + chunk\_size]  
 print(f"Sending chunk of size {len(chunk)} bytes")  
 clientSocket.sendto(chunk, (serverIP, serverPort))  
  
# Signal the end of file transfer  
clientSocket.sendto(b"", (serverIP, serverPort))  
  
print("File sent to the server")  
  
# Close the client socket when done  
clientSocket.close()

**Revision**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Date** |
| Rhia Bipin Roy | First Draft of Document | 12/08/23 |
| Colby Gallaher | First Draft of Document | 12/08/23 |
| Colin Lloyd | First Draft of Document | 12/08/23 |