

Contents

[Abstract 1](#_heading=h.gjdgxs)

[First Step: 1](#_heading=h.30j0zll)

[Phase 1:Data collection 1](#_heading=h.1fob9te)

[Second step 3](#_heading=h.3znysh7)

[Phase 1:Extracting the number of fish on each frame 3](#_heading=h.2et92p0)

[The Flow of the Phase 3](#_heading=h.tyjcwt)

[Some Real time Images 5](#_heading=h.3dy6vkm)

[Our Code: 5](#_heading=h.1t3h5sf)

[Phase 2:Plotting the extracted data 6](#_heading=h.4d34og8)

# Abstract

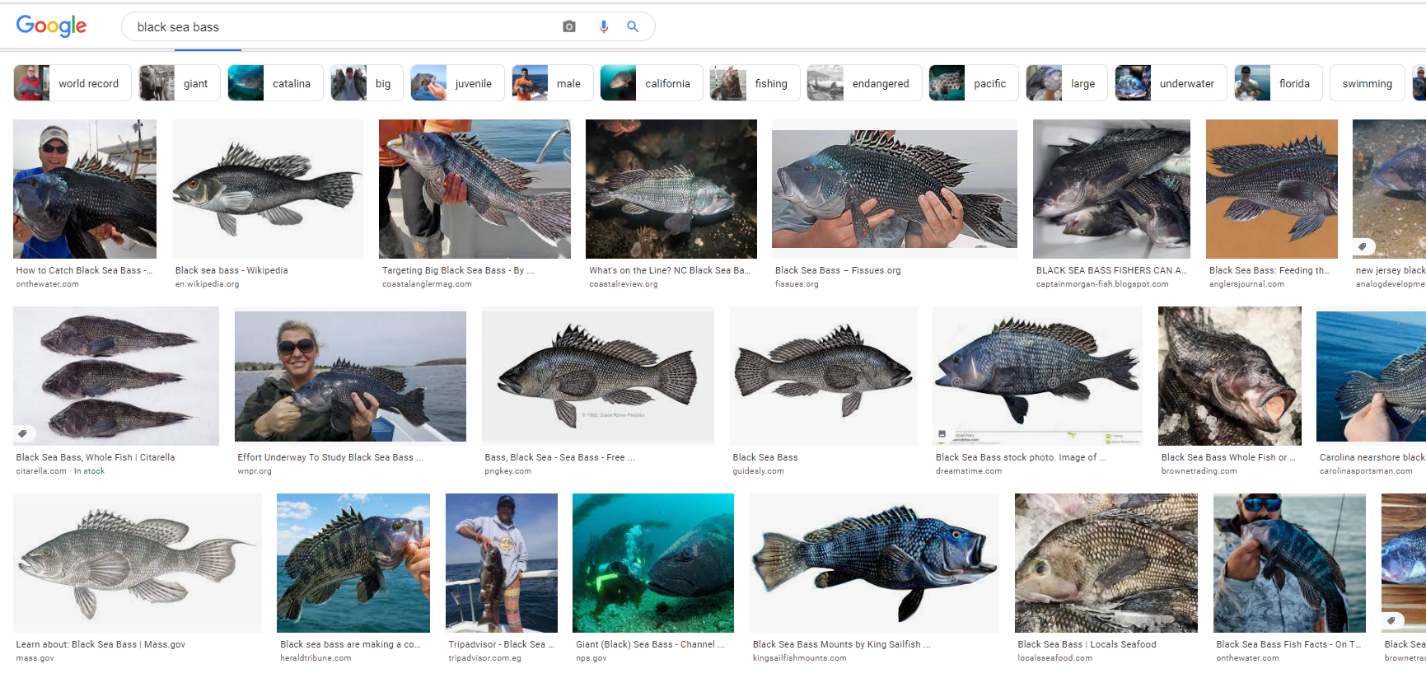
As the last challenge we distributed the work on the same manner but with some different steps (1.Data collection and preparation, 2.Extracting number of Black Sea Bass fish on every frame and plotting the result).However we faced many challenges on getting the job done, that we will discuss in the following.

# First Step:

## Phase 1:Data collection

The hardest and consumed most time as we trained our model several times with new data on each time.

We collected our data from:

* Google

We annotated the images using LabelImg Yolo and then we uploaded it to Google drive.

## Phase 2:Preparing the data

Diagram

Description automatically generated

Fig.1 The flow of data from the drive to custom vision service

The same as the previous challenge we pulled the data using google drive API and transferred it to the custom vision service

Text

Description automatically generated

Fig.1 assigning the regions to the objects

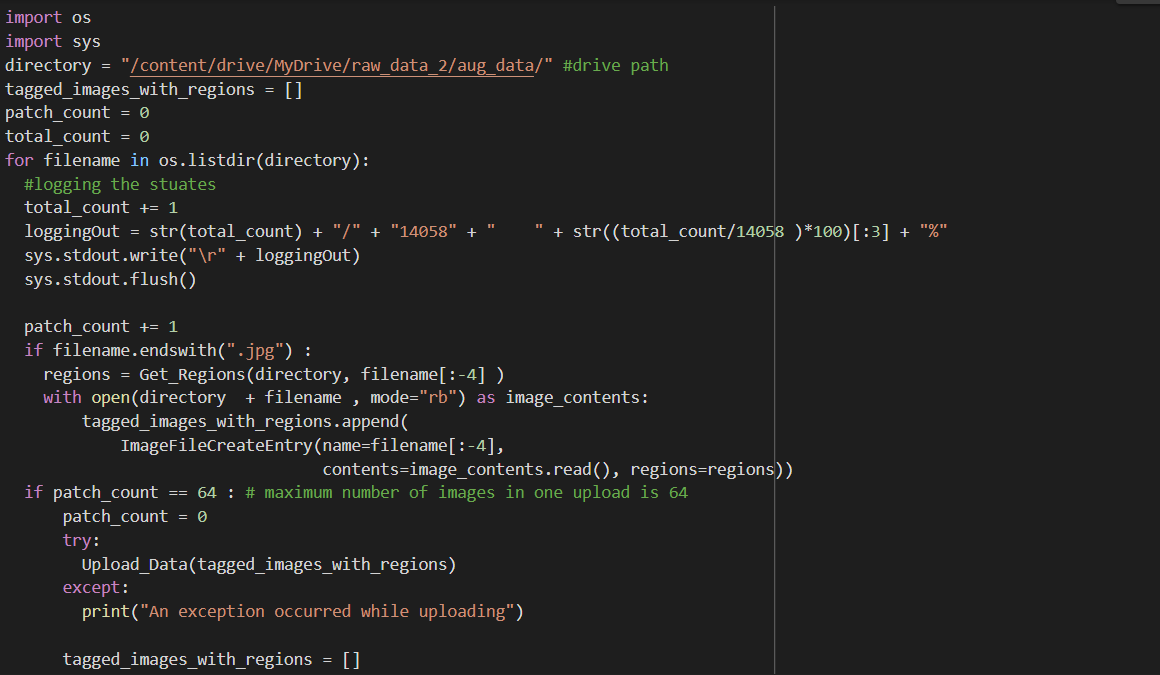


Fig.2 Code used to upload to custom vision service

# Second step

## Phase 1:Extracting the number of fish on each frame

The Flow of the Phase

As shown in Fig.3 we started by getting the video URL to fetch it ,then cutting it to 30 frames per second where we pass every frame to our custom vision service. As shown in Fig.4 The output will be the number of the predicted fish ,we will loop through every prediction to check its probability if it is high, so the count of fish will increase by one and a bounding box is drawn around it, then we extract an image and store it and store the count in an array. After collecting all the output images we store our count data in a CSV file and stack the images again to create our final video as shown in Fig.5.

:Graphical user interface, application

Description automatically generated

Fig.3

Diagram

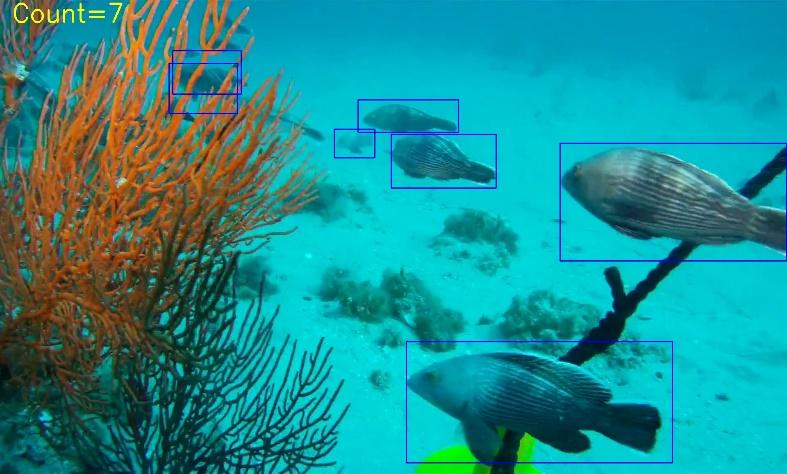
Description automatically generated

Fig.4

Diagram

Description automatically generated

Fig.5

Some Real time Images:A screenshot of a video game

Description automatically generated

Fig.6 Fig.7

### Our Code:

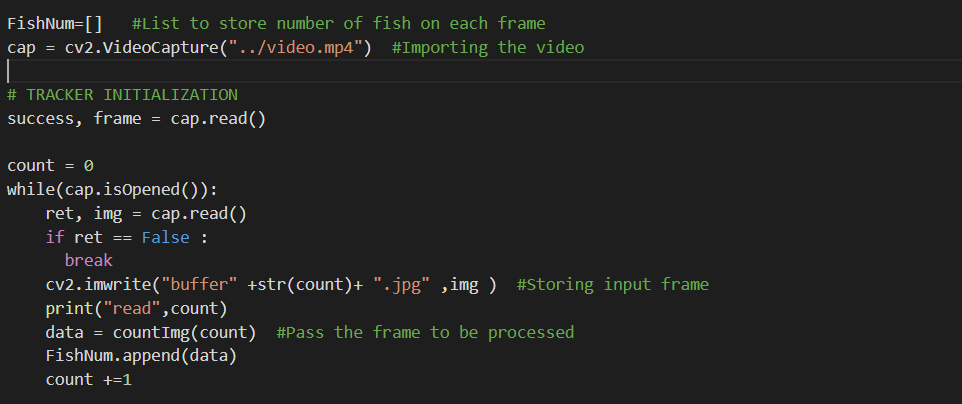


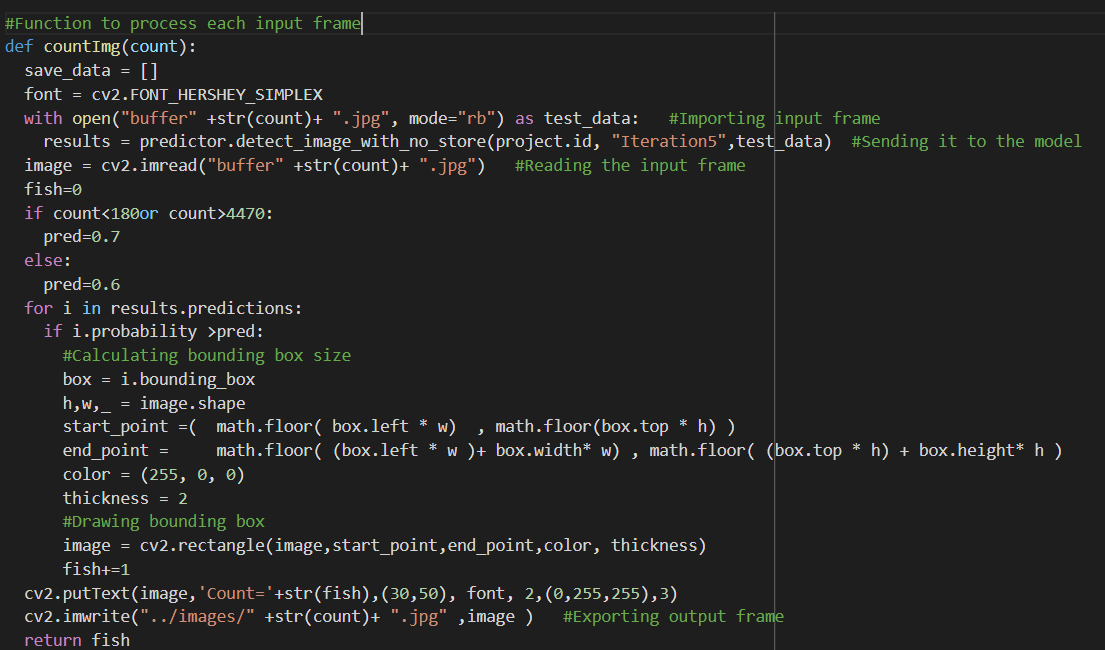
Fig.8

Fig.9

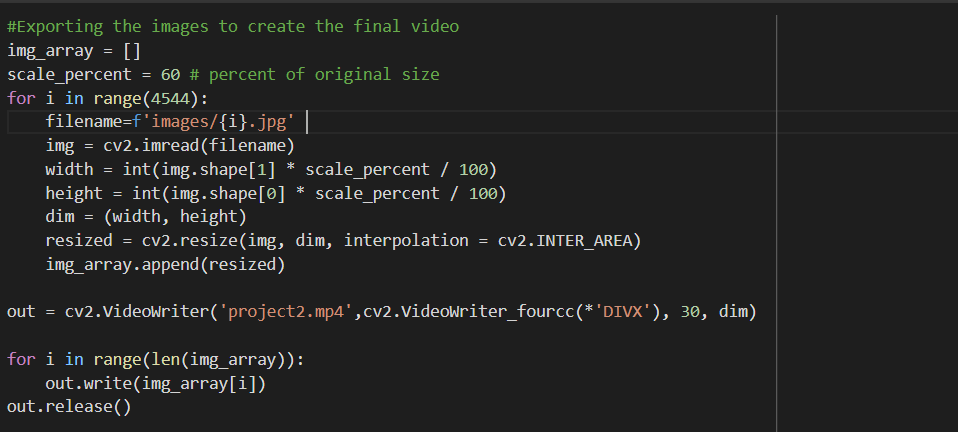


Fig.10

## Phase 2:Plotting the extracted data

We receive a csv file from the model that contains the number of fish in every frame, then we plot the data using matplotlib.

At first we plotted the data in matplotlib without any processing but it had a lot of spikes and it wasn’t smooth at all. So we decided to  do the 3-frames method which is getting every 3 frames in the list to do calculations on them **ex:[1,2,3],[4,5,6]**. At first we did an average to the 3-frame method by getting the total of the fish in 3 frames and dividing it by 3 instead of displaying fish in every frame but it was way far from being accurate so we searched for a different solution then we made the moving method which is getting the 3 frames to do calculations then getting another 3 frames but in a single step **ex: [1,2,3],[2,3,4].** We averaged the 3 frames from the moving method like the 3-frames and it was better than the 3-frames method but wasn’t good enough to satisfy us then we decided to try plotting the maximum of the points instead of the average. We tried it first on the 3-frames method and the result was good, then we made it on the moving method and the result was pretty much what we needed. and the following code is what we used :

importing libraries

import random

import collections

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

Functions:

Averaging function to average a given list

def Average(lst):

Average = int(sum(lst) / len(lst))

return Average

Simplifying function to simplify the plot

def simpler(Fish\_Number\_List):

simp = [ ]

x = 16

simp.append(Average(Fish\_Number\_List[:x]))

while 1:

try:

simp.append(Average(Fish\_Number\_List[x:x+30]))

x += 30

except:

if len(Fish\_Number\_List[x:]) > 0:

simp.append(Average(Fish\_Number\_List[x:]))

print(x)

break

return simp

plotting function to plot the data

def plot(x, y, color, label, order, max\_x, max\_y, linewidth=1, markersize=12):

markerfacecolor = color

plt.subplot(3, 1, order)

plt.plot(x, y, color=color, linewidth=linewidth, markerfacecolor=markerfacecolor, markersize=markersize,

label=label)

plt.grid()

plt.legend([label])

plt.ylim(0, max\_y + 10)

plt.xlim(0, (max\_x + 2) / 30)

plt.yticks(np.arange(0, max\_y + 10, 5))

plt.xticks(np.arange(0, max\_x + 2, 200) / 30)

plt.xlabel('Seconds')

plt.ylabel('Fish')

# now we plot the data

df = pd.read\_csv('Data1.csv')

Fish\_Number\_List = df['num'].tolist()

# Ordinary Average every 3 frames

maximum = []

time = []

frame = []

t = 0.0

MovAvgList = []

MovAvgFrameLst = []

for p in range(len(Fish\_Number\_List)):

if (p+2) == len(Fish\_Number\_List):

break

MovAvgFrameLst.append(Fish\_Number\_List[p])

MovAvgFrameLst.append(Fish\_Number\_List[p+1])

MovAvgFrameLst.append(Fish\_Number\_List[p+2])

MovAvgList.append(Average(MovAvgFrameLst))

temp\_lst = [Fish\_Number\_List[p], Fish\_Number\_List[p+1], Fish\_Number\_List[p+2]]

maximum.append(max(temp\_lst))

frame.append(t)

t += 1

MovAvgFrameLst.clear()

temp\_lst.clear()

frame.append(t + 0.1)

MovAvgList.append(Fish\_Number\_List[-2])

frame.append(t + 0.2)

MovAvgList.append(Fish\_Number\_List[-1])

for t in frame:

time.append(t/30)

simp = simpler(Fish\_Number\_List)

simp2 = simpler(MovAvgList)

simp3 = simpler(maximum)

max\_x = max([t])

max\_y = max([max(MovAvgList), max(Fish\_Number\_List), max(maximum)])

plot(np.arange(0, len(simp), 1), simp, 'blue', 'Simplified Fish List', 1, max\_x, max\_y)

plot(np.arange(0, len(simp2), 1), simp2, 'red', 'Simplified Moving Average', 2, max\_x, max\_y)

plot(np.arange(0, len(simp3), 1), simp3, 'orange', 'Simplified Moving Maximum', 3, max\_x, max\_y)

plt.show()

# Lessons learnt

One of the major troubles we faced is that we used one model on one computer to train our data.However we noticed lately that everyone should have made his own model to choose the best one and avoid any future troubles.