PEDESTRIAN DETECTOR

Using machine learning to detect pedestrians

Project Engineering

Year 4

Jack Sheridan

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Galway-Mayo Institute of Technology

2020/ 2021

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

TABLE OF CONTENTS

1. Project summary

For my final year project, I decided to build an image classifier/ object detector which would be able to classify an image and/ or detect a specified object within an image. Initially I was not aware of the differences between an image classifier and an object detector so the first thing which I needed to do was research the topic to be able to make a better-informed decision as to what I would like to do.

Once I had done some research on the topic, I decided that the best course of action would be to first try and build an image classifier. I built a model using Google colab to host my runtime where I coded a model in Python using TensorFlow and Keras , which would classify images of handwritten digits 0-9. This first model was the basis for my January project presentation. After the January presentation I worked on modifying the model to instead detect pedestrians. I found datasets online [1]. which I used to train the new model to an accuracy of ≈98% as well as moving the code off Google Colab to host it on my own machine allowing the model to train faster.

Once I was happy with my image classifier, I wanted to move on to try making an object detector. To do this I looked at several different techniques but eventually settled on using the YOLOv3 algorithm. YOLO is based on the Darknet framework which is a 53-layer Convolutional neural network. I won't go into too much detail about this yet as this is all talked about further down in the report.

Going into this project I had no prior knowledge of machine learning models other than the very basics. By the end of my study, I was able to not only understand how machine learning works but also build and test my own image classification/ object detection models.

2. Poster

Pedestrian Detctor

By Jack Sheridan

GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

What is a pedestrian detector?

A pedestrian detector is an object detector designed specifically to look for Pedestrians on roadways.

Why a pedestrian detector?

The goal with this project was to gain a greater knowledge of how machine learning models work and how to develop my own. I chose the subject of pedestrians as there is an abundance of images on the internet which I could use in my dataset e.g. PennFudan and PennPed datasets, image from which can be seen below

Image from the PennFudan dataset

Image from the PennPed dataset

Description

The model is built using the Yolov3/ Darknet framework. This means the model is 106 layers in total (53 from base Darkenet plus additional 53 from Yolov3).

The model makes its predictions at 3 layers (82, 94, 106). These predictions happen at different scales (13x13, 26x26, 52x52). This is so the model can accuratly etect ddifferent sized objects.

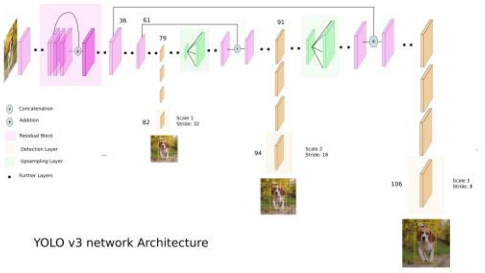
Small objects are detected at layer 106 at a scale of 52x52.
medium objects are detected at layer 94 at a scale of 26x26.
Small objects are detected at layer 82 at a scale of 13x13.

YOLO (You Only Look Once)

YOLO is an object detection algorithm/ framework which uses classes and creates bounding boxes around individual objects for each object in the image, in one run of the algorithm. Hence the name.

Darknet

Darknet is an open-source neural network framework.
The main difference with Darknet and other models is that other models are applied at multiple locations and scales. Darknet applies the model once over the whole image.
This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities



YOLO v3 network Architecture

Here is what a model prediction looks like!

Technologies used

Google colab
It allows you to write and execute python code as well as text cells through the browser.

LableImg, used to label the images in our dataset.

YOLOv3, our object detecion algorith

Darknet, the framework YOLO is based on

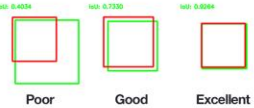
Model Output

When training the model we are looking for a few things which would indicate that the model is performing well.

The first is "avg loss". We want this value to be as low as possible. Generally anything less than 5 is considered good.

The other thing to look for is IoU or Intersection over Union, which tells us how accurately the bounding boxes are being placed relative to the "ground truth" bounding box (the bounding box we placed ourselves in LabelIng). The higher the value the better.

Here is an example:



Poor Good Excellent

3. What is an object detector?

As opposed to an image classifier, which only tells you if an image contains a certain object, an object detector also tells you where in an image the object is. This process is known as object localization.

4. Darknet

Darknet is an open-source neural network framework [2]. It is a 53-layer Convolutional neural network used mainly for object detection.

The main difference with Darknet and other models is that other models are applied at multiple locations and scales. Darknet applies the model once over the whole image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

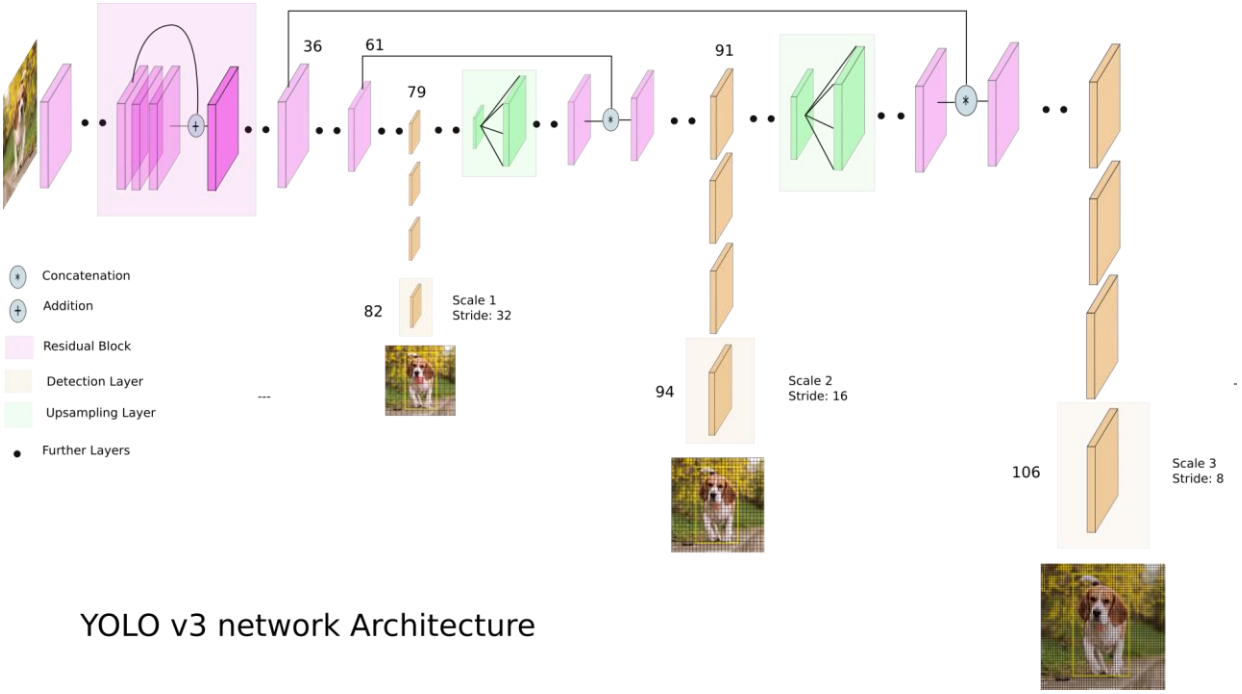
Here is an example of an output vector where P_c is our class id e.g. 0/ 1/ 2 etc. b_x is our x coordinate for the top left of the bounding box, b_y is our x coordinate for the top left of the bounding box, b_h is the height of the bounding box and b_w is the width of the bounding box. C_1 , C_2 , and C_3 are the classes.

$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

5. You Only Look Once

YOLO is an object detection algorithm/ framework which uses classes and creates bounding boxes around individual objects for each object in the image, in one run of the algorithm. Hence the name.

Here is the base YOLOv3 architecture:



YOLO v3 network Architecture

Diagram by Ayoosh Kathuria [3].

6. YOLO network architecture

Yolo is based on the Darknet-53 framework which is a Convolutional neural network (CNN) consisting of 53 layers. The yolo framework adds an additional 53 layers for a total of 106 layers. Predictions are made at 3 layers. Layer 82, layer 94 and layer 106. Each of the 3 layers is responsible for making predictions at different scales. There are feature maps created at each of these layers of size: 13x13, 26x26, 52x52 respectively each with a depth of 255.

- The 82nd layer (13x13 feature map) is responsible for detecting large images.
- The 94th layer (26x26 feature map) is responsible for detecting medium sized images.
- The 106th layer (52x52 feature map) is responsible for detecting small images.

Each prediction layers that is, 82, 94 ,106 are followed by an up-sampling layer so that the input image to the next layers will be of greater resolution. These up-sampled images are also concatenated with previous layers which helps to preserve smaller details making detection of smaller objects easier for the model.

7. YOLO configuration parameters

YOLOv3 needs a configuration file “yolov3.cfg”. This config file contains all the important training parameters. I will now go over what these values all mean. [4].

Here are the parameters in the .cfg file:

```
[net]
# Testing
batch=1
subdivisions=1
# Training
# batch=64
# subdivisions=8
height=416
width=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

```

7.1 batch size

Our dataset contains around 100 images, but this is low for a typical dataset, which would usually contain several thousand images. The whole point of the training process is to update the weights of the neural network with each iteration, based on how many mistakes it is making on the training dataset.

A subset of images is used in one iteration, and this subset is called the batch size. In the code snippet above the batch size is set to 1 however in mt own example I have the batch=64 meaning that the model updates the weights after looking at 64 images.

7.2 subdivisions

The subdivisions parameter is used in case you do not have a powerful enough Graphis Processing Unit (GPU) to load all 64 images into memory at once. The GPU will process batch/subdivision number of images at any time, but the weight will only update after all the 64 images are processed.

7.3 Image dimensions

The image input into the model will be resized to whatever value is given for “width” and “height”. The channel parameter (in this case is equal to 3) means the model will process 3 channels of the image I.e., Red, Green, Blue (RGB).

7.4 Momentum and decay

Momentum and decay are both parameters which control how the weights are updated. Because the weights are updated after each iteration through a small batch of the data, the weight updates can be volatile. The momentum parameter is used to mitigate this problem. Momentum penalizes large weight changes between iterations.

Decay is used to mitigate overfitting by penalizing large weight values. Overfitting is an issue whereby the model becomes too familiar with the training dataset. It is said to “memorize” the training data as opposed to learning the features in the image.

7.5 Data augmentation

The values for angle, saturation, exposure and hue can be used to augment the dataset by modifying the images to be slightly different. For example, you can use the angle parameter to have the images rotated which would force the model to learn based on these “new” images. This also applies to saturation, exposure and hue.

7.6 Learning rate

- learning_rate=0.001
- burn_in=1000
- policy=steps
- steps=400000, 450000
- scales=.1, .1

At the beginning of the training the learning rate needs to be high but as the model sees more and more images the learning rate should decrease. A higher learning rate means the weight changes will be more volatile. In the example above the model starts with a learning rate of 0.001 and stays constant through to 400,000 iterations where it will then multiply by "scales". To reiterate the learning rate starts at 0.001, the policy states it will remain constant until a "steps" number of iterations (in this case it's 400,000) where the learning rate then is multiplied by "scales" (in this case 0.1).

7.7 Burn in

As I mentioned earlier the learning rate should be high towards the start, however it has been found that it is beneficial to have a short period of time at the start where the learning rate is slow. To accomplish this, we use "burn_in". In this case the burn in period is set to 1000.

7.8 Max batches

The "max_batches" parameter is used to set how many iterations the model will run for while training. In the example above it is set to 500,200 iterations. It is recommended that the max batches parameter is set to 2000*n batches, where n is equal to the number of classes you are training for.

8. Why YOLO?

Well firstly Yolo is open source meaning that as well as it being free to use there are countless examples online making it easier to learn.

The biggest advantage of using YOLO is that it is fast which in the context of pedestrian detection is a must. Having a quick way of detecting people (or objects) in self driving vehicles means the car has more time to respond and plan accordingly making it safer. This however is not as much of a concern in our case as we are only looking at still images.

Another advantage of YOLO over other object detection frameworks is its ability to generalize. Compared to other methods such as SSD (Single Shot MultiBox Detector) which divides the image using a grid where each grid cell is responsible for detecting objects in that region of the image. YOLO looks at the whole image at once, so its predictions are informed by global context in the image.

9. LabelImg

To build our YOLO object detector, we first need to provide it with some images along with their corresponding labels. YOLO has its own specific format for image labels, that is a .txt file with the same name as the image file in the same directory. This .txt file contains the annotations for the image, which are object class, object coordinates, height, and width.

```
<object-class> <x> <y> <width> <height>
```

LabelImg is the best way to label our images in the YOLO format. Once you have downloaded the software [5].

10. Setup for the model

Before we begin to budling our model we first need to do some set up. The first step is to label our images. We do this using the LabelImg application mentioned above. A link to it can be found below.

Once we have our images labeled, we take our image files along with the .txt files which labelImg created for us and zip them together. We then upload the .zip file to Google Drive in a folder. Give this folder a memorable name such as “yolo”. We do this so we have a place for our model to save the weights files to which we can easily find again later.

Next, we connect Google Colab to our Google Drive. Here are some code snippets to show how this is done.

```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')
     !ln -s /content/gdrive/My\ Drive/ /mydrive
```

Now that we have access to our Google Drive, we need to clone the Darknet repository by AlexeyAB.

```
[ ]  !git clone https://github.com/AlexeyAB/darknet
```

11. Building our model

Once we have the repo cloned successfully, we need to change into the directory where is saved and make a few changes to the Makefile. These are to enable GPU acceleration. This is done using the stream editor (!sed) command and the !make command to build the file once the changes are made.

```
[ ]  # change makefile to have GPU and OPENCV enabled
     # change directory into cloned repo
     %cd darknet
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile

     !make
```

Next, we copy the contents of Darknet yolov3.cfg file to a new file called “yolov3_training.cfg”

```
[ ]  !cp cfg/yolov3.cfg cfg/yolov3_training.cfg # copies the contents of the first file into the second file.
```

Once this is done, we need to modify the new config file to suit our needs. Descriptions for these parameters can be found in section seven of the report.

```
[6]  !sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg # use stream editeor to
     !sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3_training.cfg
     !sed -i 's/max_batches = 500200/max_batches = 4000/' cfg/yolov3_training.cfg
     !sed -i '610 s@classes=80@classes=1@' cfg/yolov3_training.cfg
     !sed -i '696 s@classes=80@classes=1@' cfg/yolov3_training.cfg
     !sed -i '783 s@classes=80@classes=1@' cfg/yolov3_training.cfg
     !sed -i '603 s@filters=255@filters=18@' cfg/yolov3_training.cfg
     !sed -i '689 s@filters=255@filters=18@' cfg/yolov3_training.cfg
     !sed -i '776 s@filters=255@filters=18@' cfg/yolov3_training.cfg
```

N.B. the equation for calculating the depth of detection filters/ kernels is as follows.

`(b * (5 + c))` where b = number of bounding boxes and c = number of classes.

We then create our obj.names file which contains our class names. In our case we only have the one class “Person”. Next, we create our obj.data file which contains the number of classes, out train and test sets as well as our backup where the weights are saved to.

```
!echo "Person" > data/obj.names # creates file “data/obj.names” with “Person” in it.
!echo -e 'classes= 1\ntrain  = data/train.txt\nvalid  = data/test.txt\nnames = data/obj.names\nbackup = /mydrive/yolov3' > data/obj.data
!mkdir data/obj fil
```

All that's left to do now is download the weights file and unzip our images.

```
[9]  # Download weights darknet model 53
     !wget https://pjreddie.com/media/files/darknet53.conv.74

     !unzip /mydrive/yolov3/images.zip -d data/obj # Unzip our images
```

12. Extra bits

One extra step before we start our model training is to change the class in each of our .txt files from 1 to 0. This is because we are only dealing with 1 class "Person". Alternatively, you could also go into the original classes folder to begin with and remove all other classes except for "Person". E.g., dog, cat etc.

```
import glob # global
import os # operating system
import re # regex

txt_file_paths = glob.glob(r"data/obj/*.txt") # returns all file paths that match "data/obj/" and end in ".txt"
for i, file_path in enumerate(txt_file_paths): # returns a tuple with the counter and value i = count e.g. 88
    # get image size
    with open(file_path) as fo: # used for exception handling similar to try catch, opens file_path for reading
        lines = fo.readlines() # readline()reads until EOF and returns a list containing the lines.

        converted = [] # list
        for line in lines: # for each string in the list
            print(line) # print the string
            numbers = re.findall("[0-9.]+", line) # list where regex finds all instances of "0-9" in string line
            print(numbers) # print new list
            if numbers:

                # Define coordinates
                text = "{} {} {} {} {}".format(0, numbers[1], numbers[2], numbers[3], numbers[4]) # takes entries
                converted.append(text) # appends text to converted list
                print(i, file_path) # prints count and txt file path
                print(text) # prints newly converted string
        # Write file
        with open(file_path, 'w') as fp: # opens file_path for writing
            for item in converted:
                fp.writelines("%s\n" % item) # replaces contents of file_path with newly converted values
```

Once this is done, we simply need to put a list of our images into a .txt file so that the model knows which images to train on.

13. Training the model

The last step is to start the model training. To do this type the following.

```
[ ]  # Start the training
     !./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show
```

The model will run continuously, and a weights file will be saved to the folder on Google Drive which we created earlier every 100 and 1000 iterations.

After leaving the model to train for some time you should see the weights file appear in the folder on Drive once it has reached 100+ iterations. This may take anywhere from 20 minutes to 2 hours depending on the GPU (graphical processing units) which the runtime is using. You can check this by typing `!nvidia-smi`

Here is what the output looks like:

```
[ ]  !nvidia-smi

     Sun May 16 12:45:33 2021
     +-----------------------------------------------------------------------------+
     | NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
     |-------------------------------+----------------------+----------------------+
     | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
     | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
     |                               |                      |               MIG M. |
     |===============================+======================+======================|
     |   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
     | N/A   38C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
     |                               |                      |                  N/A |
     +-------------------------------+----------------------+----------------------+

     +-----------------------------------------------------------------------------+
     | Processes:                                                                  |
     |  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
     |        ID   ID                                                   Usage      |
     |=============================================================================|
     |  No running processes found                                                 |
     +-----------------------------------------------------------------------------+
```

In this case we are using an NVIDIA Tesla P4 with 15GB of memory.

In order to get good results, you want to allow the model to train for as long as possible. After ≈3 hours the model had gone through 2065 iterations and reached an avg loss of 0.217953. The IoU was also consistently in the high 80's low 90's which is good.

Keep in mind that Google Colab has a maximum runtime of 12 hours after which you will be automatically disconnected.

Here is the result of ≈3 hours of training with an NVIDIA Tesla P4

```
 2065: 0.232548, 0.217953 avg loss, 0.001000 rate, 6.548656 seconds, 132160 images, 3.138192
Loaded: 0.000068 seconds
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.865983), c
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.875667), c
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000),
 total_bbox = 427157, rewritten_bbox = 0.158958 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.885296), c
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.815772), c
Can't open label file. (This can be normal only if you use MSCOCO): data/obj/Copy of pic1 (1
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.795941),
 total_bbox = 427170, rewritten_bbox = 0.158953 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.906046), c
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), c
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000),
```

In Red you see the number of iterations or epochs, which is the number of times the model has ran through the training dataset.

In Green you can see the average loss

In yellow is the “Intersection over Union” (IoU) which is defined as Area of Intersection/ Area of Union, a higher value indicates a more accurate prediction. Here is an example:

14. Testing our model

We know our model is accurate base on the avg loss and IoU values it was giving us during the training phase, to test this we can see our model in action by running a python script.

This python script does a few things. It loads the model feeds it the images you want to have it detect on and then displays the output. Firstly, we import our dependencies and load the YOLO network using the OpenCV command **cv2.dnn.readNet**. readNet requires two parameters, the weights file and .cfg config file. Keep in mind these files must be in the same directory as the python script file.

```
import cv2
import numpy as np
import glob
import random

# Load Yolo using cv2
net = cv2.dnn.readNet("yolov3_training_last (12).weights", "yolov3_testing.cfg")
```

Next, we declare our class “Person” and give it the path to the images you want it to detect on.

```
# Name custom object
classes = ["Person"]

# Images path
images_path = glob.glob(r"D:\yoloTest\*.jpg")
```

We Then use the **net.getLayerNames()** command to get our layers. For refence here is what “layers” looks like when printed to the terminal.

```
254 ['conv_0', 'bn_0', 'leaky_1', 'conv_1', 'bn_1', 'leaky_2', 'conv_2', 'bn_2', 'leaky_3', 'conv_3', 'bn_3', 'leaky_4', 'shortcut_4', 'conv_5', 'bn_5', 'leaky_6', 'conv_6', 'bn_6', 'leaky_7', 'conv_7', 'bn_7', 'leaky_8', 'shortcut_8', 'conv_9', 'bn_9', 'leaky_10', 'conv_10', 'bn_10', 'leaky_11', 'shortcut_11', 'conv_12', 'bn_12', 'leaky_13', 'conv_13', 'bn_13', 'leaky_14', 'conv_14', 'bn_14', 'leaky_15', 'shortcut_15', 'conv_16', 'bn_16', 'leaky_17', 'conv_17', 'bn_17', 'leaky_18', 'shortcut_18', 'conv_19', 'bn_19', 'leaky_20', 'conv_20', 'bn_20', 'leaky_21', 'shortcut_21', 'conv_22', 'bn_22', 'leaky_23', 'conv_23', 'bn_23', 'leaky_24', 'shortcut_24', 'conv_25', 'bn_25', 'leaky_26', 'conv_26', 'bn_26', 'leaky_27', 'shortcut_27', 'conv_28', 'bn_28', 'leaky_29', 'conv_29', 'bn_29', 'leaky_30', 'shortcut_30', 'conv_31', 'bn_31', 'leaky_32', 'conv_32', 'bn_32', 'leaky_33', 'shortcut_33', 'conv_34', 'bn_34', 'leaky_35', 'conv_35', 'bn_35', 'leaky_36', 'shortcut_36', 'conv_37', 'bn_37', 'leaky_38', 'conv_38', 'bn_38', 'leaky_39', 'conv_39', 'bn_39', 'leaky_40', 'shortcut_40', 'conv_41', 'bn_41', 'leaky_42', 'conv_42', 'bn_42', 'leaky_43', 'shortcut_43', 'conv_44', 'bn_44', 'leaky_45', 'conv_45', 'bn_45', 'leaky_46', 'shortcut_46', 'conv_47', 'bn_47', 'leaky_48', 'conv_48', 'bn_48', 'leaky_49', 'shortcut_49', 'conv_50', 'bn_50', 'leaky_51', 'conv_51', 'bn_51', 'leaky_52', 'shortcut_52', 'conv_53', 'bn_53', 'leaky_54', 'conv_54', 'bn_54', 'leaky_55', 'shortcut_55', 'conv_56', 'bn_56', 'leaky_57', 'conv_57', 'bn_57', 'leaky_58', 'shortcut_58', 'conv_59', 'bn_59', 'leaky_60', 'conv_60', 'bn_60', 'leaky_61', 'shortcut_61', 'conv_62', 'bn_62', 'leaky_63', 'conv_63', 'bn_63', 'leaky_64', 'conv_64', 'bn_64', 'leaky_65', 'shortcut_65', 'conv_66', 'bn_66', 'leaky_67', 'conv_67', 'bn_67', 'leaky_68', 'shortcut_68', 'conv_69', 'bn_69', 'leaky_70', 'conv_70', 'bn_70', 'leaky_71', 'shortcut_71', 'conv_72', 'bn_72', 'leaky_73', 'conv_73', 'bn_73', 'leaky_74', 'shortcut_74', 'conv_75', 'bn_75', 'leaky_76', 'conv_76', 'bn_76', 'leaky_77', 'conv_77', 'bn_77', 'leaky_78', 'conv_78', 'bn_78', 'leaky_79', 'conv_79', 'bn_79', 'leaky_80', 'conv_80', 'bn_80', 'leaky_81', 'conv_81', 'permute_82', 'yolo_82', 'identity_83', 'conv_84', 'bn_84', 'leaky_85', 'upsample_85', 'concat_86', 'conv_87', 'bn_87', 'leaky_88', 'conv_88', 'bn_88', 'leaky_89', 'conv_89', 'bn_89', 'leaky_90', 'conv_90', 'bn_90', 'leaky_91', 'conv_91', 'bn_91', 'leaky_92', 'conv_92', 'bn_92', 'leaky_93', 'conv_93', 'permute_94', 'yolo_94', 'identity_95', 'conv_96', 'bn_96', 'leaky_97', 'upsample_97', 'concat_98', 'conv_99', 'bn_99', 'leaky_100', 'conv_100', 'bn_100', 'leaky_101', 'conv_101', 'bn_101', 'leaky_102', 'conv_102', 'bn_102', 'leaky_103', 'conv_103', 'bn_103', 'leaky_104', 'conv_104', 'bn_104', 'leaky_105', 'conv_105', 'permute_106', 'yolo_106']
```

And the code snippet...

```
17  layers = net.getLayerNames()
18  # len=254
19  print(len(layers), layers)
```

Use the net.getUnconnectedOutLayers() command to get the indexes of layers with unconnected outputs. I.e. our output layers. We also set the colour of our bounding boxes. In this case it is random.

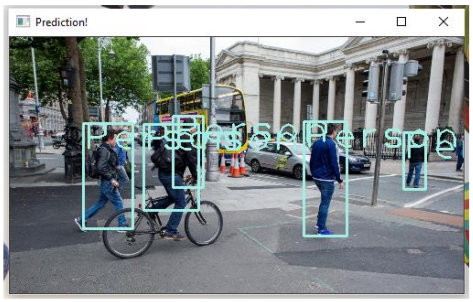
```
21  # Returns names of layers with unconnected outputs.
22  output_layers = [layers[i[0] - 1] for i in net.getUnconnectedOutLayers()]
23  print(output_layers)
24  # The colour of our bboxes
25  colors = np.random.uniform(0, 255, size=(len(classes), 3))
```

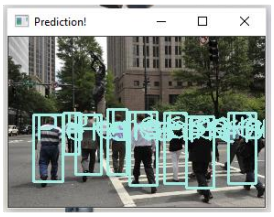
When you print "output_layers" you get: ['yolo_82', 'yolo_94', 'yolo_106']

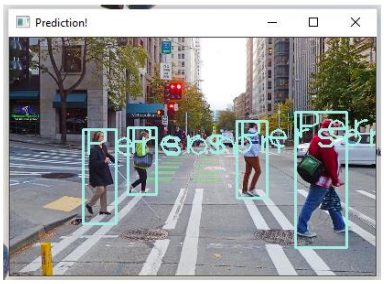
Next it loops through all the images and creates a 4-dimensional blob for each image. Blob in python stands for Binary Large Object. It is a data type that can store binary data. Then net.setInput(blob) sets the blob to the new input for the model. Use net.forward to create a blob for the first output of the layer.

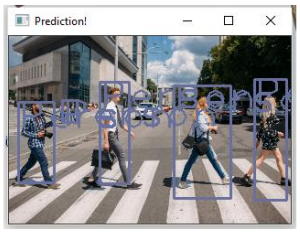
```
38      blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), (0, 0, 0), True, crop=False)
39      print(blob)
40      net.setInput(blob)
41      outs = net.forward(output_layers)
```

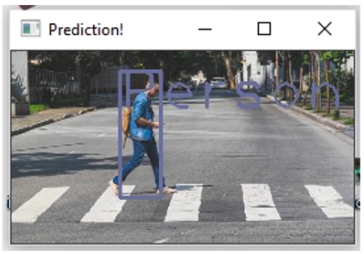
Then all that is left to do is to display the output on the screen. Here are some examples:

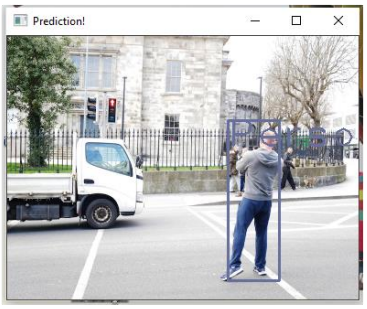

Prediction!


Prediction!


Prediction!


Prediction!


Prediction!


Prediction!

15. Ethics

Machine learning and artificial intelligence are powerful tools to humanity and as a result there is a lot of talk about ethics in the industry, Whether it's people's right to privacy in relation to facial recognition software being used to track your every move in public or what a car should do when it cannot avoid an accident, should it hit a car and potentially injure the owner and the occupants of the other vehicle or swerve and hit a pedestrian practically guaranteeing the drivers safety but potentially killing or seriously injuring the person.

There is no doubt ethics have/ will play a huge role in the machine learning and artificial intelligence discipline and I believe that it is important that no matter what the technology should be used to benefit the most amount of people at any given time.

16. Conclusion

There are countless real-world applications for machine learning models and the computer vision subgroup is no different with its possibilities. Whether its spotting pedestrians or detecting abnormalities in cells computer vision is proving to be a great solution to a lot of today's problems. Clearly the computer vision filed has great growth potential with the global market expected to be worth 17.4 billion by 2024. [6].

I have a lot of learning still to do but I feel as though getting the chance to work on this project has cemented my interest in the topic and I will be looking at opportunities to further my career in this field.

References

[1]. Liming W. PennPed dataset. Accessed May 2021.

[2]. Joseph Chet Redmon. Darknet. pjreddie.com/darknet. Accessed May 2021.

[3]. Ayoosh Kathuria. Yolov3 architecture diagram. Accessed May 2021.

[4]. Sunita Nayak. YOLO parameters. Accessed May 2021.

[5]. Tzutalin. LabelImg. tzutalin/labelImg. Accessed May 2021.

[6]. Markets&Markets. CODE SE 6082. computer vision market to be worth 17.4 billion by 2024. Accessed May 2021.

Other sources of information

Blogs:

pyimagesearch.com/object-detection-with-keras-tensorflow-and-deep-learning/

kdnuggets.com/object-detection-image-classification-yolo.html

towardsdatascience.com/classification-regression-and-prediction-whats-the-difference

towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained

towardsdatascience.com/creating-your-own-object-detector

YouTube:

Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras

How to Create a Custom Object Detector with TensorFlow in 2020

Yolov3 tutorial

GitHub:

Bengemon825/TF_Object_Detection2020

datitran/raccoon_dataset

Other:

coursera.org/learn/neural-networks-deep-learning#syllabus

pysource.com/train-yolo-to-detect-a-custom-object

tensorflow.org/tutorials/load_data/tfrecord