Hochschule Bremen
City University of Applied Sciences

Assignment III

Image Processing and Pattern Recognition. Winter Semester 2021-2022 MSc E.E.

Lecturer: Yannik Steiniger

Date of Submission: 1/09/2022

Jose Luis Quinones Gonzalez - 5172885

Contents

1. Introduction.

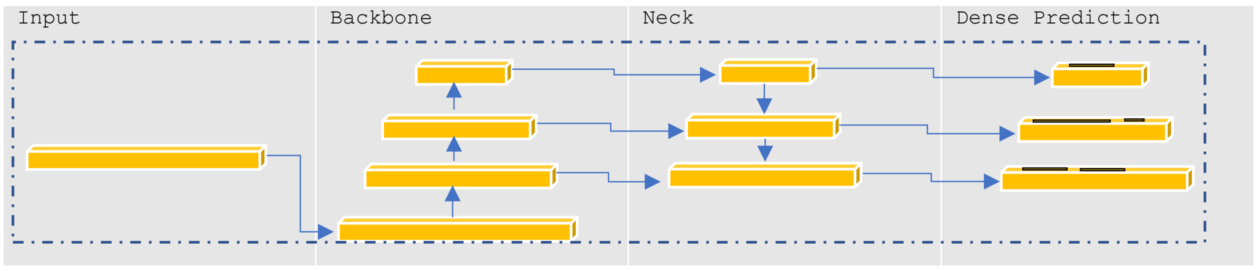
In this assignment the resolution for object detection in a set of images, throughout the use of deep learning methods, is put into practice. At the beginning simple programming functionalities, from Python and OpenCV, as "matchTemplate()" are utilized.

Later, as further demands for detecting a wider range of objects within an image, the method "matchTemplate()" is found to be inadequate and inaccurate. As a result, more complex and robust programming techniques are deployed. The prefer method used in this assignment is a subset of artificial intelligence (AI) and machine learning. That is, deep learning, (model: YOLOv4).

1.1 YOLOv4, deep learning model.

YOLOv4 stands for "You only look once" version 4. It is mainly constructed as a one stage detector, which consists of different levels, sometimes refer as hierarchical trinity search framework, of "computation". The first part is the image itself (Input), then comes convolutional neural networks, "CNN"(Backbone), approximation of different receptive fields or merging features(Neck), and dense prediction/ classification and regression(Head). [1]

Table 1. One stage detector.



The latter table represents a typical structure from a one stage detector. Some of the architectures commonly used at every part of the process can be:

Backbone: VGG16, ResNet-50, ResNetXt-101, Darknet53 (used in YOLOv4), and others.

Neck: FPN, PANet, Bi-FPN, others.

Head: Dense prediction: RPN, YOLO, SSD, RetinaNet, FCOS, others.

For the YOLOv4 the following architectures are selected [1]:

- Backbone: CSPDarknet53
- Additional module: SPP
- Neck: PANet path-aggregation
- Head: YOLOv3 (anchor based)

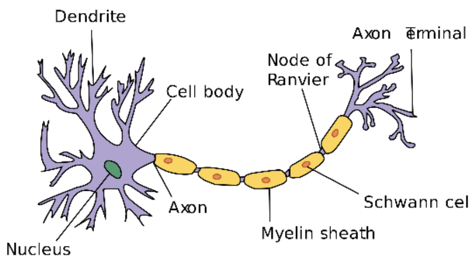
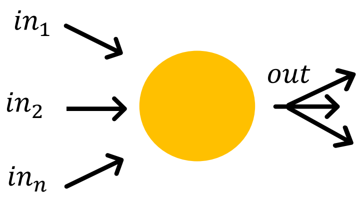
It is also important to mention the following parameters from the YOLOv4 backbone. [1]

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPDarknet53	512X512	725X725	27.6 M	950 K	52 (26.0 FMA)	66

1.2 Traditional, machine & deep learning.

Deep learning uses a multi-layered artificial neural network(s) which provides, perhaps one of the most accurate results in terms of object detection and speech recognition applications, just to mention a few. This technique is not a traditional one, since it does not involve or at least, drastically reduces the common human manipulation, intervention, or discrimination of results/ final data. Since it is a technique which learns and when trained properly, it can offer a great flexibility when the model is tested with new and similar trained data. [2]

Table 2. Natural and Artificial Neuron.

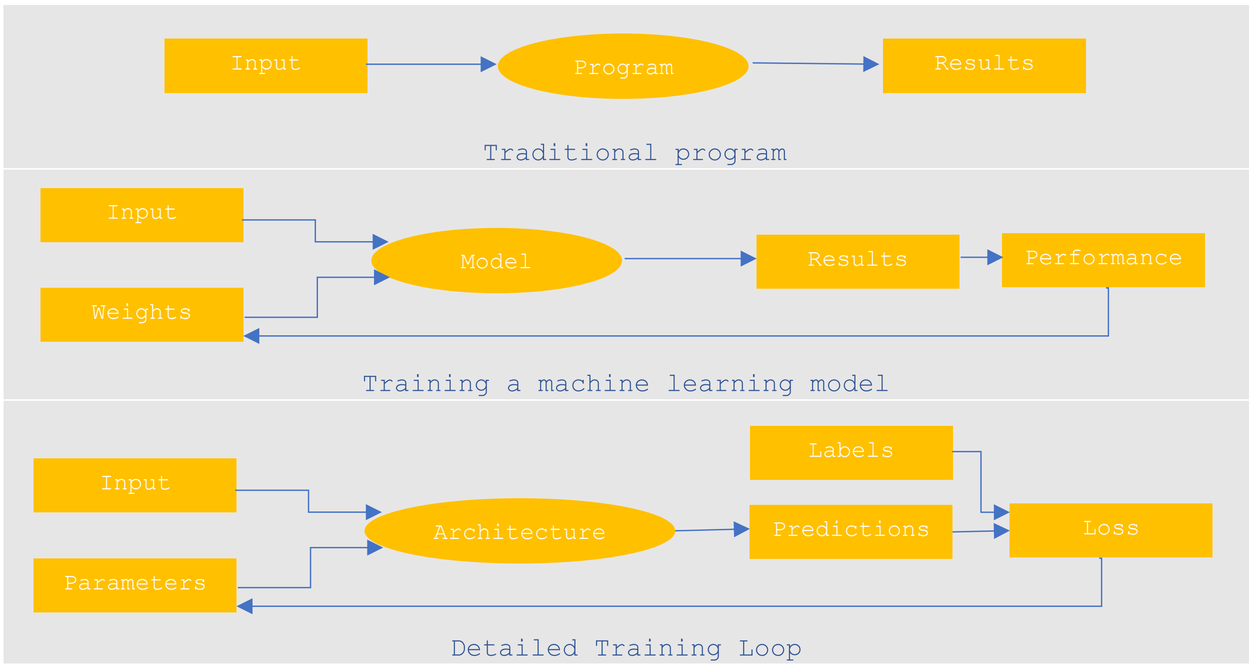
Natural Neuron	 Dendrite Axon Terminal Node of Ranvier Cell body Axon Schwann cel Myelin sheath Nucleus
Artificial Neuron	

The table from above illustrates a simple way of how an artificial neuron is constructed. As any other function in programming, it depends upon inputs as to yield an output. The process which all inputs go through initially, is typically a code which transforms the data into a more convenient value; afterwards, this same process involves a series of computations which finally yield a result. [3]

To have a better comprehension, the difference between machine learning and deep learning must be expounded.

As mentioned before, both areas, machine learning and deep learning pertain to the same field of artificial intelligence. Nonetheless, the manner a machine learning model is trained differs from that of a deep learning model. The following tables explain both training processes respectively:

Table 3. Traditional, machine & deep learning training.



Using the table from above, each learning process can be differentiated in the way each of these make proper modifications as it learns from its experience. From machine and deep learning a new variable named weights and parameters respectively, is added at the initial part of the computational process. In the middle part, a simple program is now a model, and from model to an architecture, one more complex than the former one.

One stage before reaching the end of the process cycle, results are obtained, and predictions based upon labels are inferred in machine and deep learning respectively. Deep learning being a more "brain-like structure" and as a result more complex and accurate the result is.

Then feedback in the process is as well included as to perform modifications, in accordance with the performance/ losses. [3]

2. Preparations/ Resolution

2.1 MatchTemplate() Function.

OpenCV a common visual library in python, offers a simple convolutional process using a template and a threshold value in an image. It basically slides the template through all the image and in accordance with a certain threshold value, it matches the template in a specific region of the image. As variations are performed in such threshold, a different number of matches can be obtained. Nonetheless, precision is undermined, and results are not accurate.

For performing such program, an image is selected. Afterwards, a template can be extracted using a simple image editor. At the end both images are read using a simple python code and results are obtained.

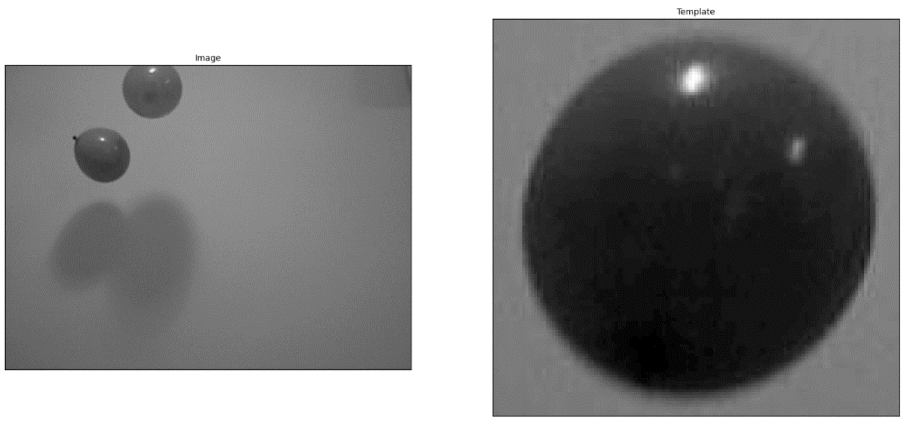


Figure 1. Image and Template.

2.2 YOLOv4 Deep learning model.

The preparations for designing and training this model based upon a particular dataset is somehow extensive. Nonetheless, here is a summary of the most essential parts when it comes to train a customized data.

The first steps involved the download of the darknet library (which contains a pretrained model). This library has already a pretrained set of data and a total of 80 different object categories. Since the purpose of this assignment is that of identifying balloons, and balloons' shadows the model shall be trained using new data.

A total selection of 50 images is considered in this experiment. The configuration file within the darknet library (architecture) is then modified to:

Parameter	Value
batch	64 (it must be a multiple of 8)
subdivisions	16 (Not greater than the batch No.)
width	416 (size which images are transformed)
height	416 (size which images are transformed)
learning_rate	0.001

max_batches	4000 (No. of classes*2000)
steps	3200,3600 (max_batches*0.8, *0.9)
[Convolutional] filters	21 (No. of classes+5)*(3)
[yolo] classes	2 (balloon, shadow)

Configuration values are recommended by the creator of YOLOv4 accordingly to the specific requirements of the training data. (Further information can be found within the github site from the same YOLOv4 creator) [4]

The next step is to label the corresponding data, here a common useful tool is utilized. LabelImg.

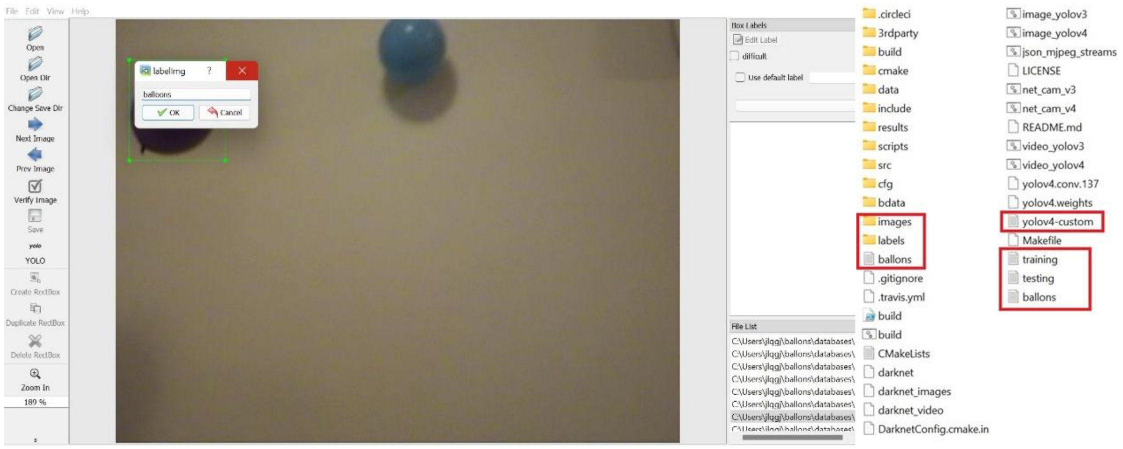


Figure 2. LabelImg, Darknet files.

Afterwards a total of seven files are added to the darknet folder. Each file contents a specific data regarding this training. From labels, classes, images, and coordinates from labeled data (text file) to YOLOv4 configuration (.cfg) and data files (.data).

The .data file being the one which indicates the program which file contains a specific information as to run the training. From here some of the text files, in addition, specify the path of training/ testing images and labels. *(Data file: classes = 2, train = training.txt, valid = testing.txt, names = ballons.names, backup = backup).*

Finally, the complete folder is compressed into a zip file and uploaded to a google drive account as to utilize google colab (co) for free cloud server, TPU & GPU. Since the training process demands a certain graphic card for faster and accurate results, co turns out to be the best choice. Otherwise, the user may be forced to face poor results of training, when using only the CPU.

3. Results

The results regarding the matchTemplate() function from CV were very limited in terms of accuracy. When the selected template was included in the test image, that is, object was 100% same resemblance, the program turned out to be precise. Yet when the threshold value was altered the precision dropped. Not to mention, the results when the template was tested upon a different image, which contain a similar object but not same object. The template sometimes identified the template as a balloon and vice versa.

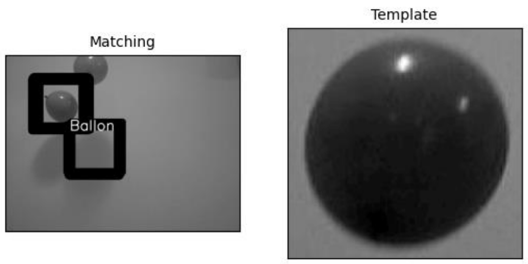


Figure 3. matchTemplate(), OpenCV function.

As for the training of the YOLOv4 model, the results, weight, and configuration file were successfully obtained. It took around 3.5 hours to reach the 1000 best weights for the program.

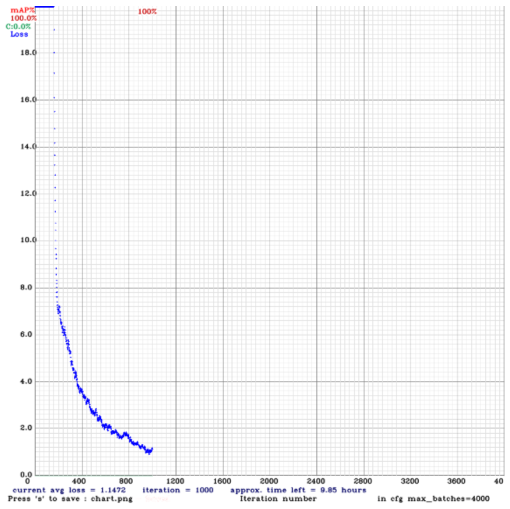


Figure 4. 1000 best weights graph

The graph demonstrates an accuracy of 100%, and as the number of batches progressed, the loss was reaching a lower value. It must be clarified, that, even though the total number of batches was of 4000, the training was stopped at 1000. This decision was inferred as sometimes the accuracy may also decreased as the training reaches the final batches value. For general purposes, a 1000 batches run is more than enough when the data set is not considerably large (above ~100 images).

When the files, yolov4-custom_1000.cfg, yolov4-custom_1000.weights were loaded to the python's code, the code could have a successful and accurate prediction to all images within the dataset.

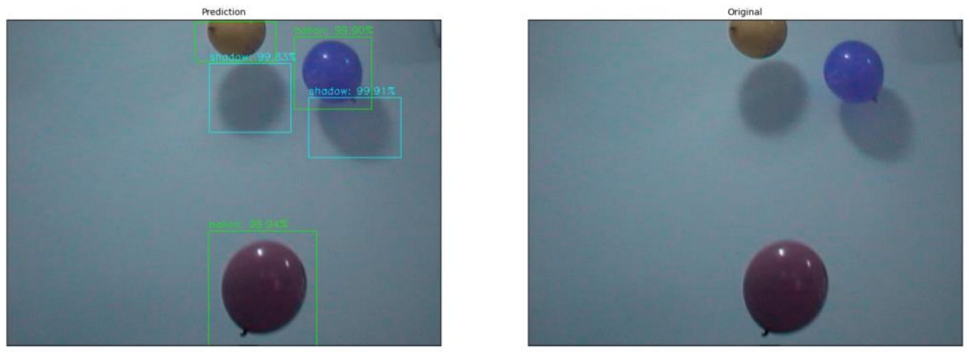


Figure 5. Customized YOLOv4 results.

In addition, the confidence percentage prediction was very close to 100%. In the last figure, three balloons were detected, couple with two shadows, the confidence values were: Balloon 99.94%, Balloon 99.90%, Balloon 99.4%, shadow 99.91%, shadow 99.83%. Now, when the same model was tested with different images, which included balloons, but were not included within the same process of training, the result turned out to be not that accurate.

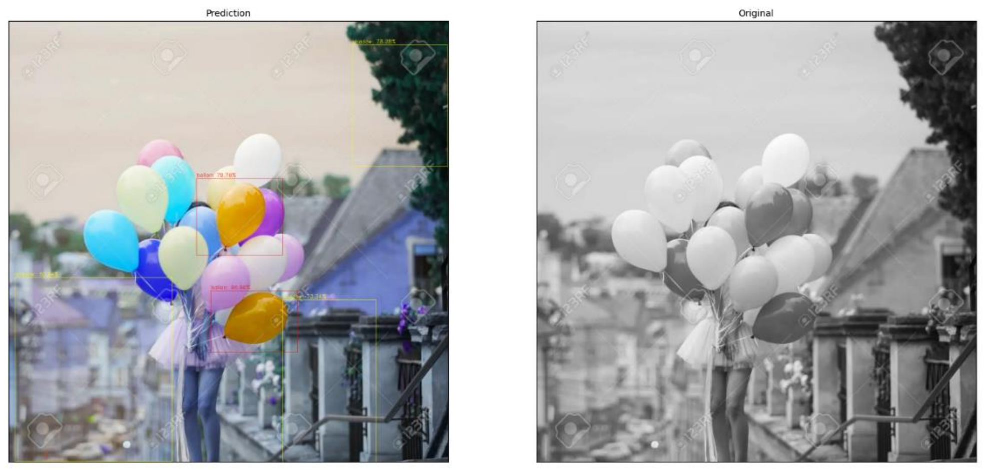


Figure 6. Customized YOLOv4 applied in random image.

The prediction was poor. *Predicted object ballon: 96.96%, predicted object ballon: 79.76%, predicted object shadow: 78.28%, predicted object shadow: 73.34%, predicted object shadow: 60.84%.*

Only two balloons were detected from approximately 15. While the class shadow was attributed to “empty” space, curiously to similar regions where typically a shadow from within any image from the dataset is located.

4. Conclusion

It can be inferred that if both methods are compared, matchTemplate() and the customized YOLOv4 model, one is much more superior to the other. When the matchTemplate() lacks certain flexibility, the deep learning model is by far a better solution. Yet, when such a robust architecture is trained/ tested with data which somehow has a certain patter, as similar background, and lacks the versatility of different “scenarios”, either the results might be astonishing when detecting desired objects from images which were included in the training dataset, or the results may tend to err a correct prediction when detection is performed in completely new images.

It can be also confirmed that this YOLOv4 model suffered from an overfitting problem.

Overfitting remains a great challenge and perhaps is the most important one, when it comes to machine or deep learning algorithms. Nonetheless, it might be impossible for a machine to posses such an intuitive logical intelligence as to generate some sort of intuitive improvised prediction when new data, without previous training, is tested. One of the best solutions is to keep feeding the training with new data. As though the model could be updated and as a result a new constant training is always performed to the model.

However, it may happen that the model itself may be saturated with lots of data, when trying to feed an infinite number of images.

Even for complex and state of the art algorithms limitations exist. Yet, these models can be of great use when optimized for a specific application.

References

[1] C.-Y. W. H.-Y. M. L. Alexey Bochkovskiy, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020.

[2] A. Adams, Python Programming, Deep Learning, 2020.

[3] J. H. &. S. Gugger, Deep Learning for Coders with fastai & PyTorch, Sebastopol, CA: O'Reilly, July, 2020.

[4] A. Bochkovskiy, "https://github.com," 2020. [Online]. Available: https://github.com/AlexeyAB/darknet.

Appendix (Python's Code)

MatchTemplate()

```
Created on Sat Dec 25 21:31:38 2021

@author: jlqgj
"""

import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
from skimage.io import imread, imsave, imshow
from skimage.morphology import binary_opening, binary_closing, square, erosion


# gray_templ = cv.cvtColor(templ, cv.COLOR_RGB2GRAY)

# edges_orig = cv.Canny(gray_orig,25,60)
# edges_templ = cv.Canny(gray_templ,190,110)


orig = cv.imread('C:/Users/jlqgj/ballons/p38.JPG',0)
# gray_orig = cv.cvtColor(orig, cv.COLOR_RGB2GRAY)
templ = cv.imread('C:/Users/jlqgj/ballons/template_2.png',0)
w, h = templ.shape[::-1]

img = cv.medianBlur(orig,1)
ret,th1 = cv.threshold(img,120,255,cv.THRESH_BINARY)
# im_o = binary_opening(th1, square(13))
# im_c = binary_closing(im_o,square(12))
# im_e = erosion(im_c, square(2))
# plt.imshow(im_e)
# plt.show()


res = cv.matchTemplate(th1,templ,cv.TM_CCOEFF_NORMED)
upperth = 0.5
lowerth = 0.4

loc = np.where(  res >= upperth)
for pt in zip(*loc[::-1]):
    cv.rectangle(orig, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

cv.imwrite('res.png',orig)
print(loc)


p = cv.imread('C:/Users/jlqgj/programs/res.png',0)
cv.putText(p, 'Ballon',pt, cv.FONT_HERSHEY_SIMPLEX,1.3,(255, 0, 0),2)
titles = ['Matching', 'Template']
images = [p,templ]
for i in range(2):
    plt.subplot(1,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i], fontsize=10)
    plt.xticks([]),plt.yticks([])
plt.show()
```

Deep Learning. YOLOV4

```
Created on Mon Jan  3 21:29:57 2022

@author: jlqgj
"""

import numpy as  np
import cv2
from matplotlib import pyplot as plt

# load the image to detect, get width, height

img_to_detect = cv2.imread('C:/Users/jlqgj/programs/images/ballons3.jpg')
img_height = img_to_detect.shape[0]
img_width = img_to_detect.shape[1]

# convert to blob to pass into model
img_blob = cv2.dnn.blobFromImage(img_to_detect, 0.003922, (416, 416), swapRB=True,
crop=False)

# set of 2 class labels
class_labels = ["ballon","shadow"]

#Declare List of colors as an array
#Red, yellow
class_colors = ["238,12,12","255,255,0"]
class_colors = [np.array(every_color.split(",")).astype("int") for every_color in
class_colors]
class_colors = np.array(class_colors)
class_colors = np.tile(class_colors,(2,1))

# Loading pretrained model
yolo_model = cv2.dnn.readNetFromDarknet('model/yolov4-custom_1000.cfg','model/yolov4-
custom_1000.weights')

# Get all layers from the yolo network
yolo_layers = yolo_model.getLayerNames()
yolo_output_layer = [yolo_layers[yolo_layer - 1] for yolo_layer in
yolo_model.getUnconnectedOutLayers()]

# input preprocessed blob into model and pass through the model
yolo_model.setInput(img_blob)
# obtain the detection layers by forwarding through till the output layer
obj_detection_layers = yolo_model.forward(yolo_output_layer)


# initialization for non-max suppression (NMS)
class_ids_list = []
boxes_list = []
confidences_list = []



# loop over each of the layer outputs
for object_detection_layer in obj_detection_layers:
        # loop over the detections
    for object_detection in object_detection_layer:

        # obj_detections[1 to 4] => will have the two center points, box width and box
height
        # obj_detections[5] => will have scores for all objects within bounding box
        all_scores = object_detection[5:]
        predicted_class_id = np.argmax(all_scores)
        prediction_confidence = all_scores[predicted_class_id]

        # take only predictions with confidence more than 20%
        if prediction_confidence > 0.20:
            #get the predicted label
            predicted_class_label = class_labels[predicted_class_id]
            #obtain the bounding box co-oridnates for actual image from resized image
size
            bounding_box = object_detection[0:4] * np.array([img_width, img_height,
img_width, img_height])
            (box_center_x_pt, box_center_y_pt, box_width, box_height) =
bounding_box.astype("int")
            start_x_pt = int(box_center_x_pt - (box_width / 2))
            start_y_pt = int(box_center_y_pt - (box_height / 2))



            class_ids_list.append(predicted_class_id)
            confidences_list.append(float(prediction_confidence))
            boxes_list.append([start_x_pt, start_y_pt, int(box_width),
int(box_height)])


max_value_ids = cv2.dnn.NMSBoxes(boxes_list, confidences_list, 0.5, 0.4)

# loop through the final set of detections remaining after NMS and draw bounding box
and write text
for max_valueid in max_value_ids:
    max_class_id = max_valueid
    box = boxes_list[max_class_id]
    start_x_pt = box[0]
    start_y_pt = box[1]
    box_width = box[2]
    box_height = box[3]

    #get the predicted class id and label
    predicted_class_id = class_ids_list[max_class_id]
    predicted_class_label = class_labels[predicted_class_id]
    prediction_confidence = confidences_list[max_class_id]

    end_x_pt = start_x_pt + box_width
    end_y_pt = start_y_pt + box_height


    box_color = class_colors[predicted_class_id]


    box_color = [int(c) for c in box_color]

    # print the prediction in console
    predicted_class_label = "{}: {:.2f}%".format(predicted_class_label,
prediction_confidence * 100)
    print("predicted object {}".format(predicted_class_label))

    # draw rectangle and text in the image
    cv2.rectangle(img_to_detect, (start_x_pt, start_y_pt), (end_x_pt, end_y_pt),
box_color, 1)
    cv2.putText(img_to_detect, predicted_class_label, (start_x_pt, start_y_pt-5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 1)

cv2.imshow("Detection Output", img_to_detect)

#Plot prediction and original
img_orig = cv2.imread('C:/Users/jlqgj/programs/images/ballons3.jpg',0)
titles = ['Prediction', 'Original']
images = [img_to_detect,img_orig]
for i in range(2):
    plt.subplot(1,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i], fontsize=10)
    plt.xticks([]),plt.yticks([])
plt.show()
```