Q2. Train a deep learning classification model for chess piece dataset.

Steps Involved :

→ Find dataset

→ Select a model to train the images on

→ Train the model with training data

→ Test with model with testing data

→ Evaluation of the model

I tried a few datasets available online, but they were not suitable for real-time detection or classification tasks. Either they had a high dimensions or random images that are irrelevant to chess pieces in real life.

I even tried to train YOLOv5 on the above datasets for detection task, but they did not perform well.

So, I decided to collect my own dataset from the chess pieces I had.

Also thought it would be fun to do it.

Why YOLOV5?

YOLOv5 is **fast, easy to use, and capable of achieving state-of-the-art results for object detection tasks**. It is also more accurate and easier to train than its predecessors. It requires less data and give better results with it.

Collecting dataset :

I took 25 images of each chess piece and then resized them to 640*640 .

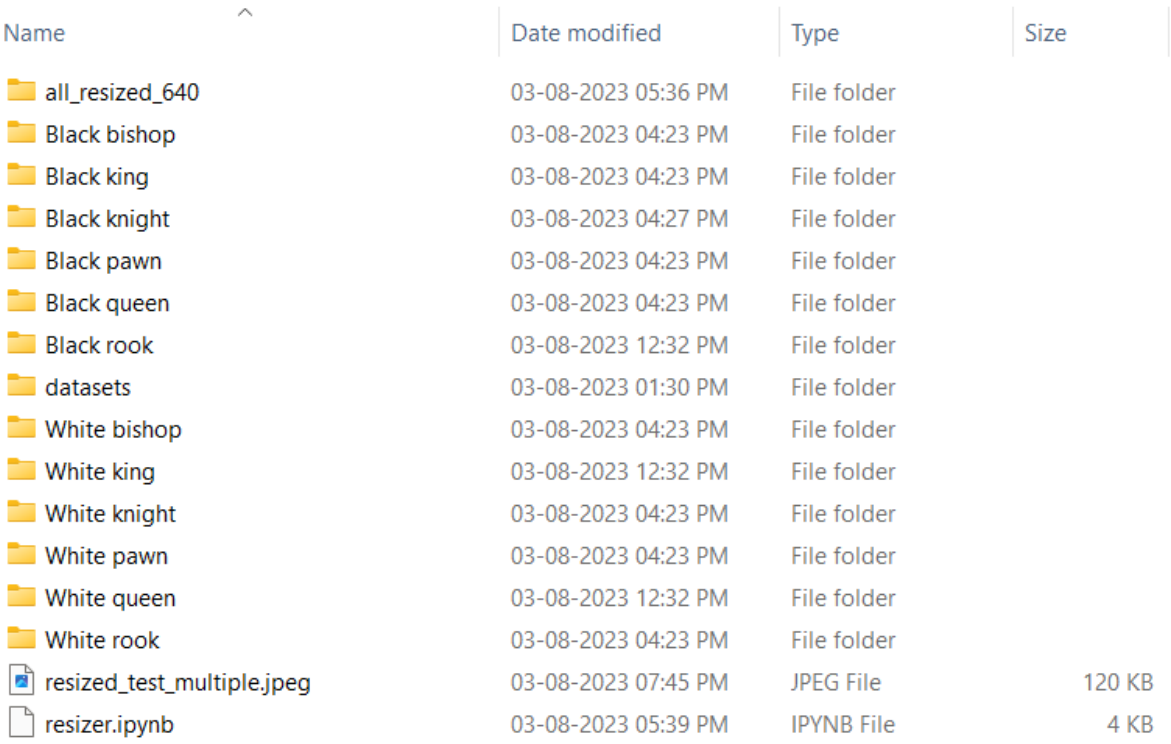
Then annotated them and stored the labels.

Split them into train and test datasets.

The classes include - White king, White queen, White rook, White bishop, White pawn, White knight, Black king, Black queen, Black rook, Black bishop, Black pawn, Black knight. Total of 12 classes.

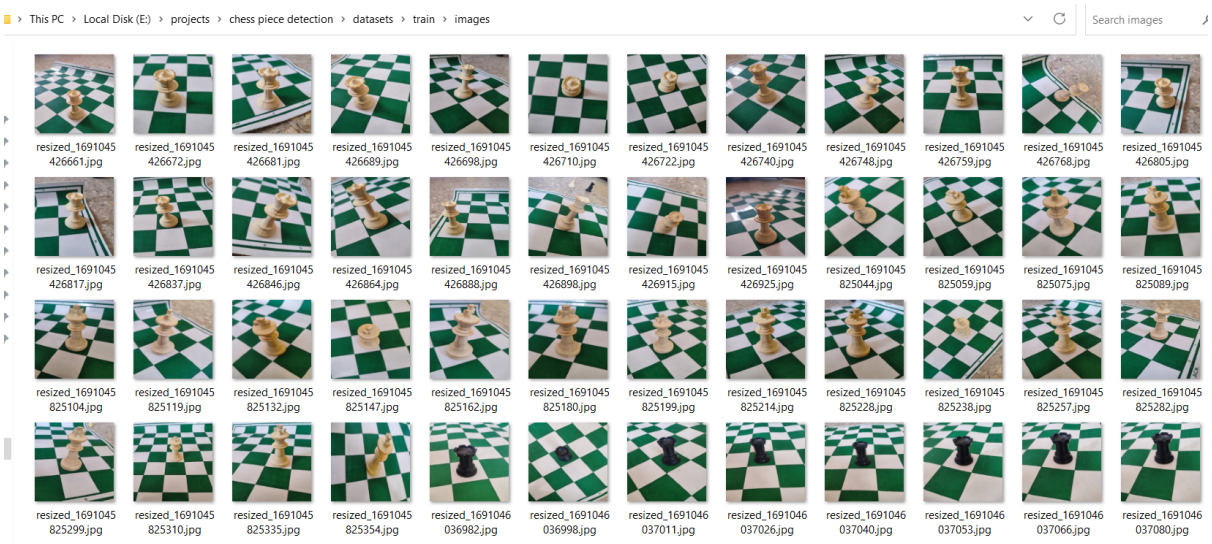
So, 12*25 = 300

Did a train-test split as : train - 20 images of each class , test - 5 images of each class. Then annotated each image using labelimg



Name	Date modified	Type	Size
all_resized_640	03-08-2023 05:36 PM	File folder	
Black bishop	03-08-2023 04:23 PM	File folder	
Black king	03-08-2023 04:23 PM	File folder	
Black knight	03-08-2023 04:27 PM	File folder	
Black pawn	03-08-2023 04:23 PM	File folder	
Black queen	03-08-2023 04:23 PM	File folder	
Black rook	03-08-2023 12:32 PM	File folder	
datasets	03-08-2023 01:30 PM	File folder	
White bishop	03-08-2023 04:23 PM	File folder	
White king	03-08-2023 12:32 PM	File folder	
White knight	03-08-2023 04:23 PM	File folder	
White pawn	03-08-2023 04:23 PM	File folder	
White queen	03-08-2023 12:32 PM	File folder	
White rook	03-08-2023 04:23 PM	File folder	
resized_test_multiple.jpeg	03-08-2023 07:45 PM	JPEG File	120 KB
resizer.ipynb	03-08-2023 05:39 PM	IPYNB File	4 KB

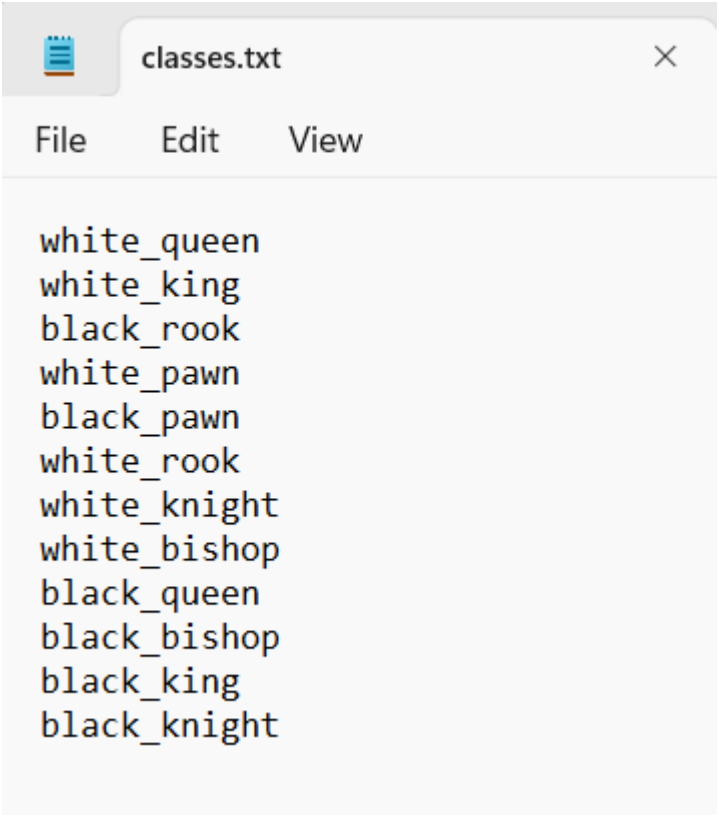
Then dumped them it into a single folder as YOLOv5 takes dataset in such format. Now, we have both images and labels for each image into corresponding folders.



This PC › Local Disk (E:) › projects › chess piece detection › datasets › train › labels

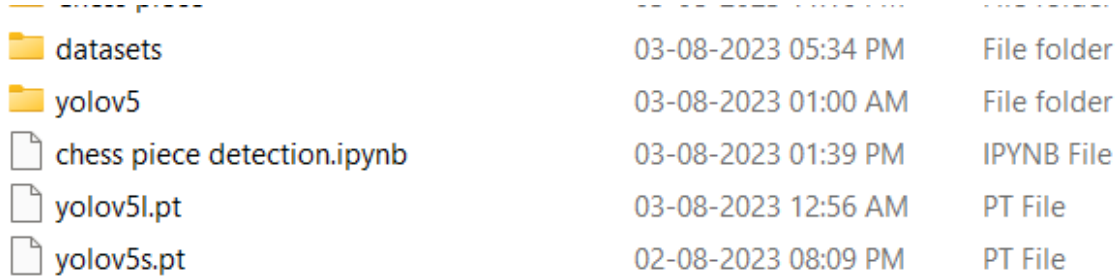
Name	Date modified	Type	Size
resized_1691045426672.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426681.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426689.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426698.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426710.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426722.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426740.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426748.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426759.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426768.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426805.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426817.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426837.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426846.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426864.txt	03-08-2023 01:30 PM	Text Document	1 KB
resized_1691045426888.txt	03-08-2023 01:30 PM	Text Document	1 KB

classes.txt file - defines the classes and labels for the images depends on their index of the class

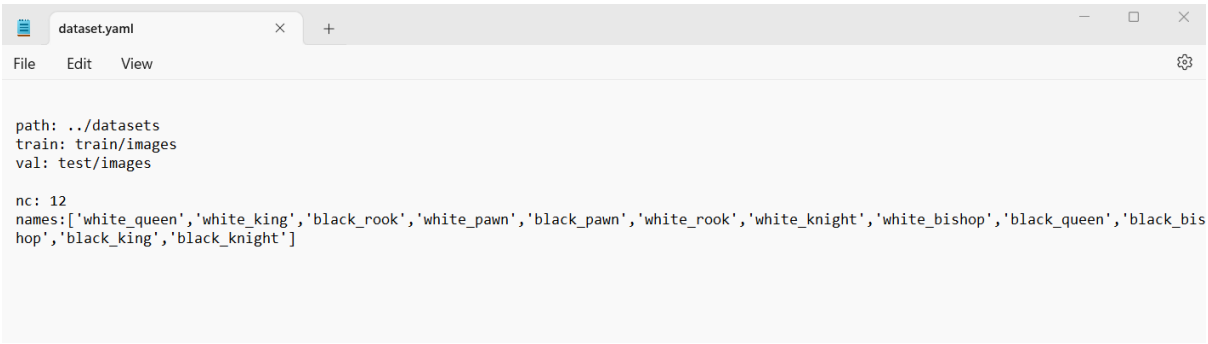


I cloned the YOLOv5 repository from ultralytics and continue to train it using the dataset we just created. Used the YOLOV5 Large model for better results.

Directory setup



Before training the model, We need the define the classes and dataset paths in dataset.yaml file within yolov5 folder



Install and import the requirements for yolov5.

```
!cd yolov5 & pip install -r requirements.txt

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

model = torch.hub.load('ultralytics/yolov5', 'yolov5l')
```

We use the command prompt to train the model using the following command. cd into the project folder and run :

```
!cd yolov5 && python train.py --img 640 --batch 8 --epochs 300 --data dataset.yaml
  --weights yolov5l.pt
  --name yolov5l_results
  --cache
```

We run the train.py file with following parameters.

‘ img ’ parameter denotes the smaller side of the image for kernel size setting as YOLO uses variable kernel size for first kernel. It is 640 in our case, as the image is of dimensions 640*640.

‘batch’ size is set 8 for minimal RAM usage.

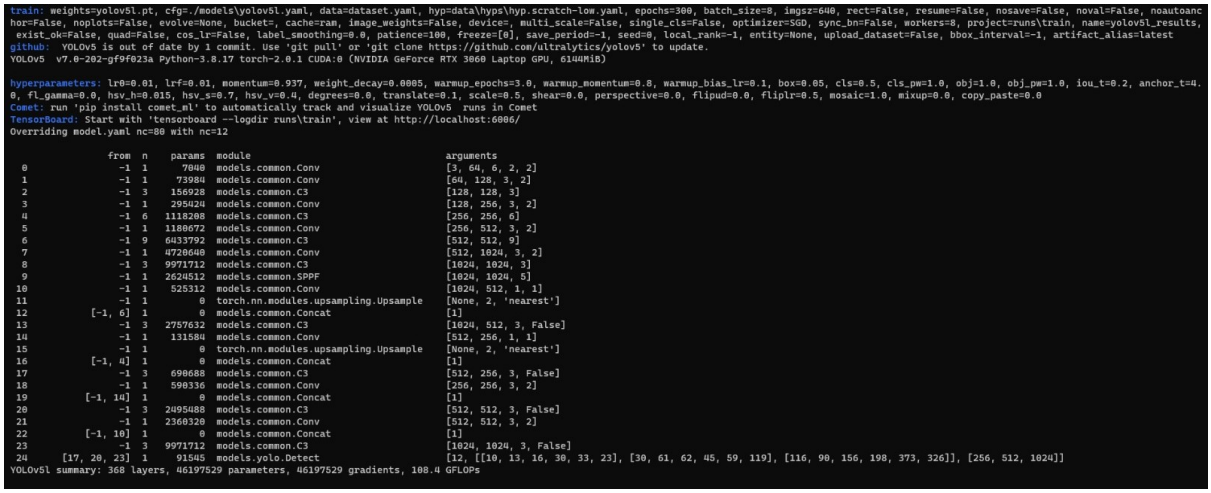
‘epoch’ is set 300

‘data’ parameter denotes the location where the dataset is located and the number of classes involved in training.

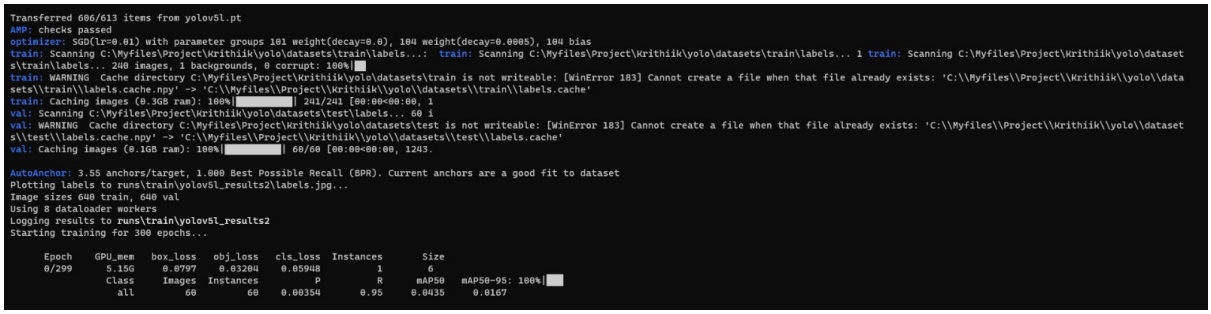
‘weights’ denotes the initial weights, we use the pretrained weights of yolov5l model.

‘cache’ is used to cache the results, so we can continue training later.

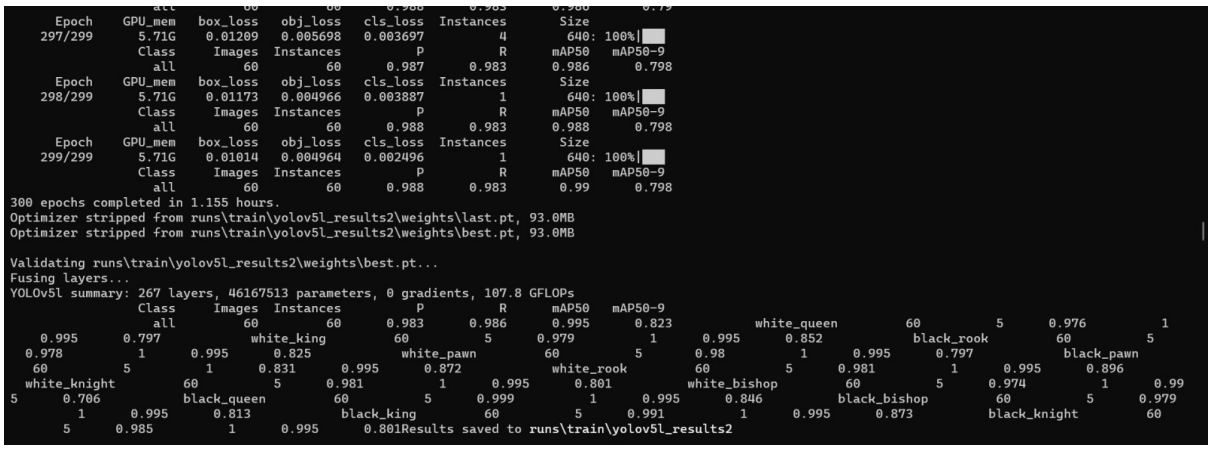
Fusing layers



Epoch start

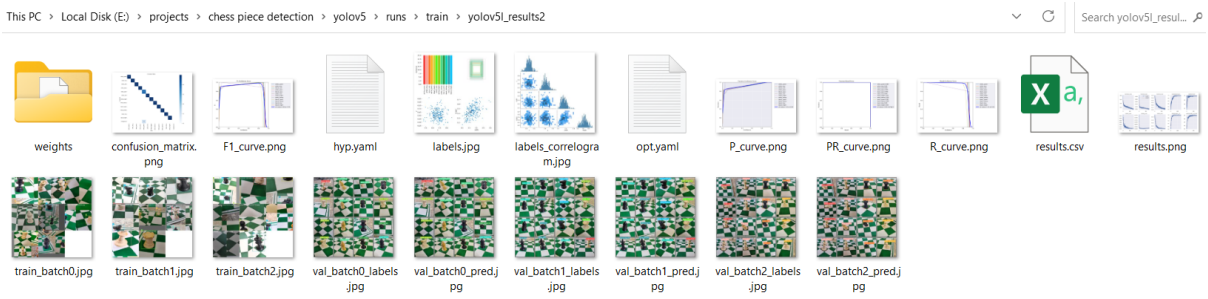


Epoch end



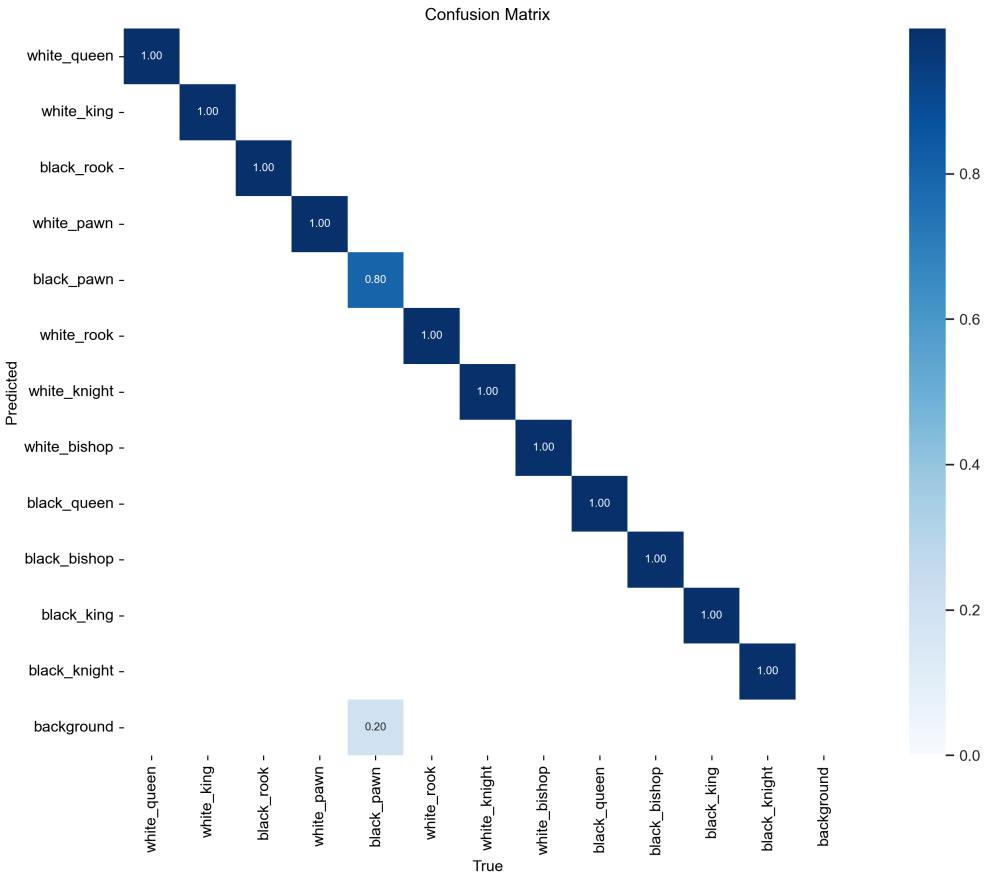
After the model is trained, we will have the results in the runs folder.

It has several evaluation metrices executed.



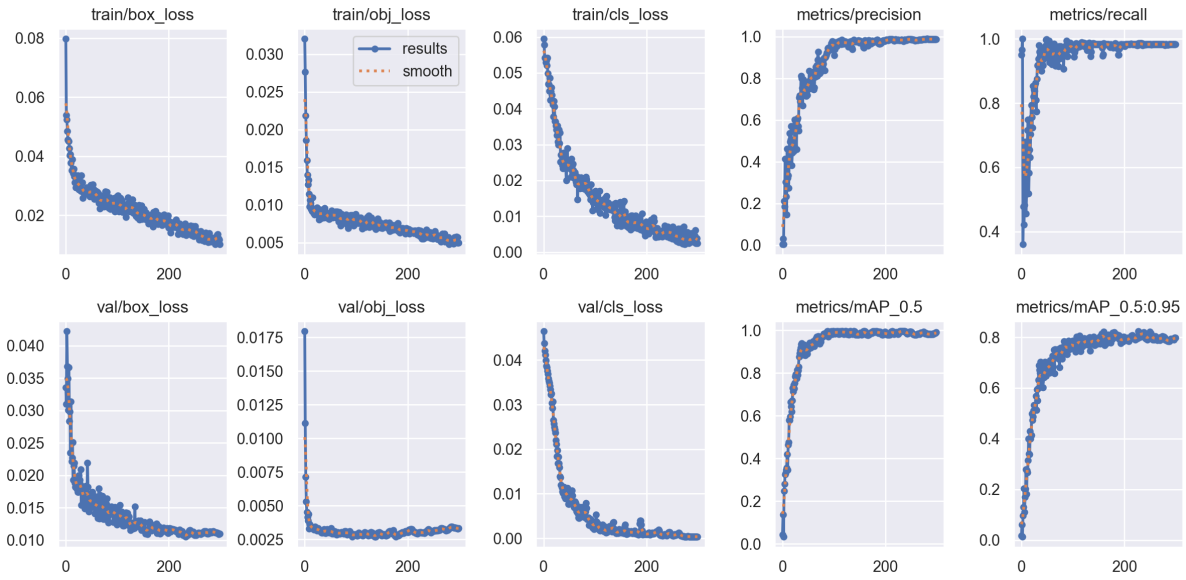
results.csv has the progress of the model on over 300 epochs.

Confusion matrix



We can notice that the model performs really good on the given task.

We can also have a look at the results.png



We can notice that the model achieved better precision and recall on training.

To be precise the *precision is - 0.988 and recall is 0.983*, which can be seen in results.csv or at the end of training session.

Detection:


```
!python detect.py --weights runs/train/yolov5l_results/weights/best.pt
 --img 640 --conf 0.5 --source ../test/images
```

we run the detect.py to test our model on the test images.

‘weights’ parameter is set to our best weights ie. best.pt achieved during training.

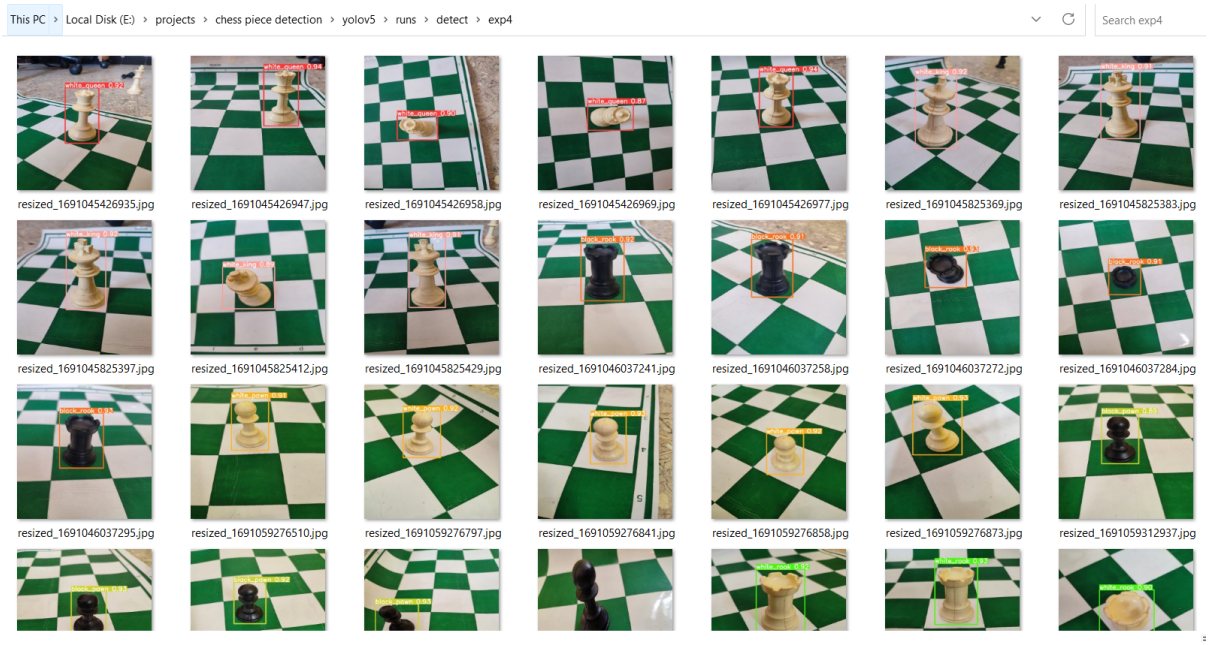
‘conf’ is the threshold on confidence metric for detection.

‘source’ denotes the file path for test images or ‘0’ for live camera detection.

```
detect: weights=['runs/train/yolov5l_results2/weights/best.pt'], source=../datasets/test/images, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.6, iou_thres=0.45, max_det=1000, device=, view_img=False
, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=
3, hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
YOLOv5  v7.0-202-gf9f823a Python-3.8.17 torch-2.0.1 CUDA:0 (NVIDIA GeForce RTX 3060 Laptop GPU, 6144MiB)

Fusing layers...
YOLOv5l summary: 267 layers, 46167513 parameters, 0 gradients, 107.8 GFLOPs
image 1/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045426935.jpg: 640x640 1 white_queen, 22.9ms
image 2/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045426947.jpg: 640x640 1 white_queen, 23.3ms
image 3/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045426958.jpg: 640x640 1 white_queen, 23.0ms
image 4/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045426969.jpg: 640x640 1 white_queen, 22.2ms
image 5/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045426977.jpg: 640x640 1 white_queen, 22.4ms
image 6/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045825369.jpg: 640x640 1 white_king, 22.1ms
image 7/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045825383.jpg: 640x640 1 white_king, 23.0ms
image 8/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045825397.jpg: 640x640 1 white_king, 22.6ms
image 9/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045825412.jpg: 640x640 1 white_king, 22.1ms
image 10/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691045825429.jpg: 640x640 1 white_king, 22.4ms
image 11/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691046037241.jpg: 640x640 1 black_rook, 22.1ms
image 12/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691046037258.jpg: 640x640 1 black_rook, 22.0ms
image 13/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691046037272.jpg: 640x640 1 black_rook, 22.3ms
image 14/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691046037284.jpg: 640x640 1 black_rook, 22.5ms
image 15/60 C:\Myfiles\Project\Krithiik\yolo\datasets\test\images\resized_1691046037295.jpg: 640x640 1 black_rook, 21.9ms
```

It can be seem that the detection on images occur within milli-seconds. That is the advantage of using YOLO models. They are relatively small and provide better results , faster.



Utility functions

resizes the image into a 640*640 dimensions by crawling through defined sub folders of images.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import glob
import pickle
import xml.etree.ElementTree as ET
from os import listdir, getcwd
from os.path import join

def getImagesInDir(dir_path):
    image_list = []
    for filename in glob.glob(dir_path + '/*.jpg'):
        image_list.append(filename)

    return image_list


dirs = [ 'Black rook', 'White king', 'White queen','White rook', 'White pawn',
'White bishop', 'White knight', 'Black king', 'Black queen','Black pawn',
 'Black bishop', 'Black knight'  ]


cwd = getcwd()


for dir_path in dirs:
    full_dir_path = cwd + '/' + dir_path
    output_path = cwd +'/resized/' + dir_path + '/'
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    image_paths = getImagesInDir(full_dir_path)
    for image_path in image_paths:
       image = cv2.imread(r''+image_path)
       new_image = cv2.resize(image, (640, 640))

       filename = 'resized_' + os.path.basename(image_path).split('/')[-1]
       cv2.imwrite(filename, new_image)
```

Q1. Chess piece classification - inception

An inception network is a deep neural network with an architectural design that consists of repeating components referred to as Inception modules.

It is widely used for image classification tasks.

It has following advantages over other network such as VGG:

→ It can provide improved accuracy

→ Deep architecture with reduced parameters

→ Multi-scale feature extraction

→ Dimensionality reduction

→ Transfer learning capabilities

Model summary

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3  0           []
                             )]

 conv2d_94 (Conv2D)          (None, 111, 111, 32  864         ['input_2[0][0]']
                             )

 batch_normalization_94 (BatchN  (None, 111, 111, 32  96      ['conv2d_94[0][0]']
 ormalization)               )

 activation_94 (Activation)  (None, 111, 111, 32  0           ['batch_normalization_94[0][0]']
                             )

 conv2d_95 (Conv2D)          (None, 109, 109, 32  9216        ['activation_94[0][0]']
                             )

 batch_normalization_95 (BatchN  (None, 109, 109, 32  96      ['conv2d_95[0][0]']
 ormalization)               )

 activation_95 (Activation)  (None, 109, 109, 32  0           ['batch_normalization_95[0][0]']
                             )

...
Total params: 22,417,196
Trainable params: 614,412
Non-trainable params: 21,802,784
```

We use transfer learning method by initializing the weights for ‘ imagenet ’.

We modify the input and output layers according to the dataset. Here we need to add the input layer according to the image size. We resized our images of chess pieces to 224 * 224 for this purpose.

Then, we add the output layer depending on the number of classes that we want our model to predict. In our case it is 12 (number of distinct chess pieces).

Preprocessing :

We preprocess the image data for training the network.

→ rescale - scale down the values with a certain range.

1./255 is the scaling value we choose. It reduces the model complexity.

→ we use sheer range and zoom range parameters as well.

Fit:

While training, we define the training set and validation set. The number of epochs is 10. It takes sometime to train as it is a complex model. So, we run the model for less epochs only.

The train and validation loss get reduced and the accuracy increases as the training steps increase. This can be seen in the graph in the ipynb file.

Predict:

Load the model with trained weights stored in h5 format. Inorder to predict an image, the image should be in numpy array format. We scale down the image in the same ratio as we used while training.

Format - (1,224,224,3)

Then, we call the .predict() method and call argmax function that returns the prediction category of the given image.

```
    model.predict(img_data)
```

```
1/1 [==============================] - 1s 1s/step

array([[3.9231088e-02, 4.2647470e-02, 5.4844004e-01, 8.4745284e-04,
        1.2884967e-04, 7.1436830e-06, 3.0903568e-02, 4.0902891e-05,
        2.9366493e-01, 3.9114268e-04, 1.3219624e-02, 3.0477740e-02]],
      dtype=float32)
```

```
    a=np.argmax(model.predict(img_data), axis=1)
```

```
1/1 [==============================] - 0s 29ms/step
```

```
    a
```

```
array([2], dtype=int64)
```