Deep Learning Project Report

Rohil Gupta
University Jean Monnet
rohil.gupta@etu.univ-st-etienne.fr

Abdul Rahman Zakarya
University Jean Monnet
abdul.rahman.zakarya@etu.univ-st-etienne.fr

Karthik Bhaskar
University Jean Monnet
karthik.bhaskar@etu.univ-st-etienne.fr

Abstract

In this work, we focus on Deep Learning machine learning method, to implement the state of art algorithm for object detection in the images. For the purpose of this project, we were provided with dataset of images taken from 9 different cameras. These images have a 24 colors checkerboard. The objective was to detect these 24 color patches in the test images by learning the deep learning model. In this project, we implemented the state of art algorithm of YOLO version 3 or YOLOv3 by Joseph Redmon. The literature review, data pre-processing and results are discussed in this report.

1. Introduction

Object detection is the technique of computer vision and image processing which is able to detect particular objects in an image and video with human level accuracy in today's time. The basic idea followed by the object detection algorithms is to capture the implicit or latent features of the object that is to be detected. In this report, we will describe the YOLO V3 in Literature Review section 2, then in section 3 the data pre-processing is explained for the provided dataset. In the next section 4, modelling and procedure is being discussed with the results in section 5.2. The report discusses the final conclusions in the section 6.

2. Literature Review

In our project we have used YOLOv3[6] with Darknet[3]. In the following subsections we will discuss about this.

2.1. Darknet

Darknet[3][1] is an opensource neural network framework written in C and CUDA programming languages and this framework supports CPU and GPU computations and this framework is used as a backbone in training and testing YOLOv3 as shown in the figure 2.[1][2]

It uses convolutional layers without large fully connected layers at the end.[3] The comparison of backbone shown in the figure1 as explained in the YOLOv3 paper[6] explains that it is trained with identical strings and tested on 256x256, shows Darknet-53 outperforms the state of the art classiffier and 2x faster compared to ResNet-152 and 1.5x faster compared to Resnet-101.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figure 1. Backbones Comparison
Source: YOLOv3: An Incremental Improvement [6]

2.2. YOLOv3

You Only Look Once (YOLO) is a state of the art object detection system[4]. YOLOv3 which is an incremental improvement over YOLOv2[5]. As the authors explain in their paper, YOLO segments the image into NxN grid, where each grid cell predicts a fixed number of boundary box using dimention clusters.

YOLOv3 uses bounding box prediction where the network predicts 4 coordinates in each of the bounding box. The coordinates are t_x,t_y,t_w,t_h where t_x and t_y are the offset positions of the box with width and height, p_w and p_h are the prior width and height.

The prediction of the bounding box corresponding to the cell c(x,y) from top left corner of the image is given by,

$$b_x = \sigma(t_x) + c_x$$

[1] https://pjreddie.com/darknet/
[2] https://github.com/pjreddie/darknet
[3] https://pjreddie.com/darknet/imagenet/darknet53

$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

The sum of squared error loss is used during the training and the confidence score is calculated using logistic regression for each of the bounding box. The gradient is based on the ground truth minus the prediction.

The confidence score is 1 if the bounding box overlaps the ground truth object more than any other bounding box. YOLOv3 uses 0.5 as the threshold to keep the predictions. Even if the bounding box overlaps the ground truth object which is not the best, based on the threshold the prediction is removed or kept.

In class predictions each of the box predicts classes using the multilable classification and independent logistic classifiers are used to obtain good performance.Binary cross-entropy loss is used for the prediction of classes.[6]

The authors of the paper[6] explains YOLOv3 predicts the bounding boxes in 3 different scales, At first more convolutional layers are added to the base feature extractor in which the last of these predicts a 3-D tensor bounding box, confidence score of object and the class predictions.

In the second scale, the feature map of previous 2 layers are upsampled by 2x and the feature map from earlier network is taken and concatenated with the upsampled features and later more convolutional layers are added to process the newly added feature map to predict the tensor similar to the previous scale with twice the size as before.

At the final scale similar design is followed for box prediction which benefits from previous computation with the finely grained features from the early network, k-means clustering is used to predict the prior bounding boxes.

The network for performing feature extraction uses a successive 3x3 and 1x1 convolutional layers, the authors explains it is significantly larger with 53 convolutional layers and call it Darknet-53 and it is shown in the figure 2.

The authors claims that the training was performed on full images with no negative mining by using multi-scale training by using Darknet[3] for training and testing.

The figure 3 shows the plot accuracy vs speed on AP_{50} metric and it can be seen that YOLOv3 is a strong detector compared to other detecting systems and authors prove that its faster and better and has at par accuracy compared two best detecting systems on 0.5 IOU[4][5].

3. Data pre-processing

In this report we are going to use the "Illuminant Estimation for Color Constancy" data-set which consists of a set of 1853 PNG images taken from nine different cameras,

[4]https://mc.ai/yolo3-a-huge-improvement/
[5]https://github.com/AlexeyAB/darknet

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2. Darknet-53
Source: YOLOv3: An Incremental Improvement[6]

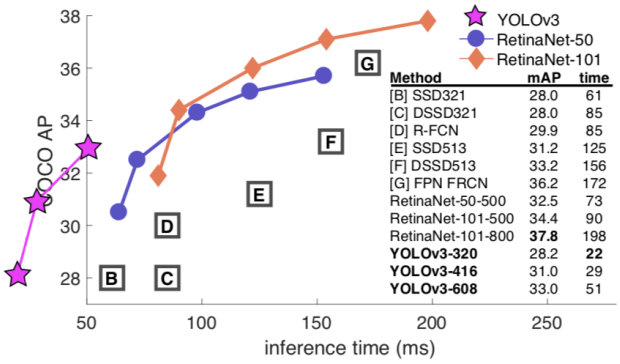

Figure 3. YOLOv3 Performance
Source: YOLOv3: An Incremental Improvement [6] [2]

each image contains a color checker palette with 24 patches of colors.

However, the provided images have to be pre-processed, that's because the color components in the images are linearly related to the energy quantities captured by the sensors, so in order to obtain a visible color components expressed in a classical display color space, such as sRGB. we have to as apply an inverse gamma correction.

Along with each image in the dataset, there is checker files that contains the color checker information including the mask positions and colors of all the patches.

3.1. Image gamma correction

The first problem we faced here is that the color channel ranges are not fixed for all the images, so we need to do a min-max normalization for each color channel per image, apply the gamma correction to the normalized image with $\gamma = 2.2$ and then save the image as 8-bit/color RGB. the following pseudocode illustrates the process:

$R, G, B \leftarrow readImage(image)$

$// \; min - max \; normalization$
$R \leftarrow (R - min(R))/(max(R) - min(R))$
$G \leftarrow (G - min(G))/(max(G) - min(G))$
$B \leftarrow (B - min(B))/(max(B) - min(B))$

$// \; gamma \; correction$
$R, G, B \leftarrow R^{1/\gamma}, G^{1/\gamma}, B^{1/\gamma}$

$// \; save \; image$
$saveImage([R, G, B], depth = 8bits)$

3.2. Checker mask pre-processing

Each image has a corresponding mask text file that contains the annotations, it includes the ROI of the color-checker as well as all 24 color patches. The first line is the ROI of the color-checker while the next 48 lines are respectively x-coordinates and y-coordinates of four warping rectangle corners of 24 patches. For example, the first 3 lines maybe:

$92.318, 1304, 591.65, 425.28$
$35.232, 35.452, 91.167, 90.676$
$317.83, 369.93, 367.37, 315.28$

the first line represents the coordinates of the ROI Rectangle which takes the form x, y, $width$, $height$. the x and y are the coordinates of the top left corner of the color checker. The next two lines represents the coordinates of the 4 corners of the polygon that covers the color patch which takes the form of two lines the first $x1$, $x2$, $x3$, $x4$ and the next line $y1$, $y2$, $y3$, $y4$. we can extract the color patch coordinates as the following:

$ROI_X = 92.318$
$ROI_Y = 1304$
$ROI_{Width} = 591.65$
$ROI_{Height} = 425.28$
$X_m = [35.232, 35.452, 91.167, 90.676]$
$Y_m = [317.83, 369.93, 367.37, 315.28]$

We note that the patches coordinates Xm and Ym are relative to ROI so in order to get the real coordinates of the color patch polygon with respect to the image coordinates we have to add the x and y coordinates of the ROI as follows:

$X_m = X_m + ROI_X$
$Y_m = Y_m + ROI_Y$

as shown in Figure 4, we plot the ROI rectangle as well as the 24 ploygons of the color patches.

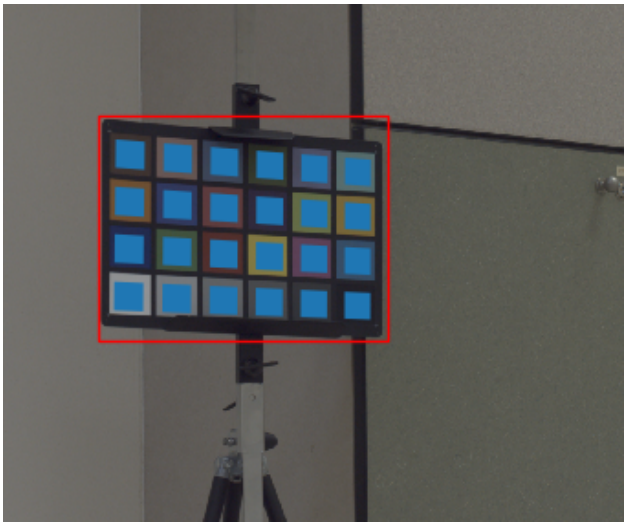

Figure 4. Color checker attributes

The next step is to approximate the polygon coordinates into a simple point of interest rectangle that contains that polygon. To get the approximate coordinates for the rectangle given a polygon of 4 points $p1$, $p2$, $p3$, $p4$ each has x and y coordinates we apply the following:

$// \; top \; left \; corner$
$X1_{rect} = min(x_{p1}, x_{p4})$
$Y1_{rect} = min(y_{p1}, y_{p2})$

$// \; buttom \; right \; corner$
$X2_{rect} = max(x_{p3}, x_{p2})$
$Y2_{rect} = max(y_{p3}, y_{p4})$

$// \; rectangle \; center, \; width \; and \; height$
$X_{center} = (X1_{rect} + X2_{rect})/2$
$Y_{center} = (Y1_{rect} + Y2_{rect})/2$
$width = X2_{rect} - X1_{rect}$
$height = Y2_{rect} - Y1_{rect}$

in the following figure 5 we illustrate the rectangle coordinates with respect to to the polygon coordinates.

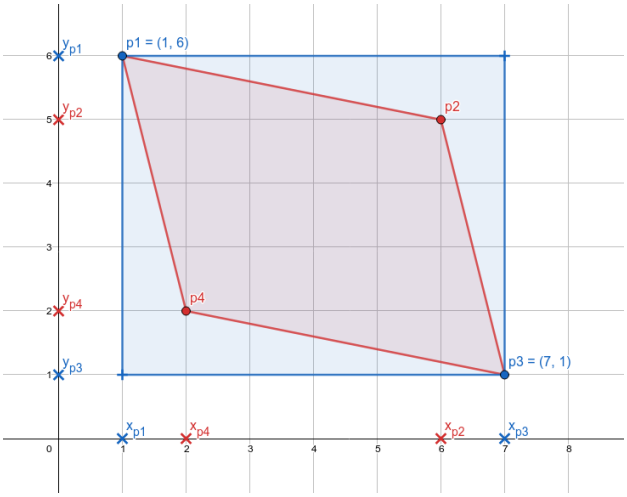

Figure 5. polygon to rectangle approximation

3.3. Validation data-set

In addition to "Illuminant" data-set, and in order to test the detection network learned on the previous data-set, we are going to use the "Colour Constancy as a different data-set to validate the learnt model, the data-set consists of 592 tif downsampled images, each image is provided with a text files that holds the color patches coordinates, we note that the annotations are different than the Illuminant data-set used for learning the model, each annotation file holds the coordinates of the color checker patches in the following form. For example the first lines of the file read:

4368 2912
3802.209323 1965.799878
4353.823178 2153.363335
3571.739386 2350.683607
4057.026568 2615.989413
3810.309933 2007.675337
3857.134494 2024.267649
3823.522691 2079.804637
3777.473875 2062.255419
...

The first line gives the image dimensions (4368 x-axis, 2912 y-axis). The next four lines the corner points of the Color Checker Chart, in the example: $(3802, 1965), (4354, 2153), (3572, 2351), (4057, 2616)$. The following lines give the corner points of the colored patches within the color checker chart row-by-row starting at the upper left. With a similar approach to the steps used by the previous checker mask pre-processing we generate the approximate coordinates for the point of view rectangle.

To sum up, the the input for our neural network will be the images with a generated annotation file per image that contains the Class Identifier and the coordinates of 24 point of interest rectangles, the generated annotation file is a csv file that takes the following form: $ClassId, X_{center}, Y_{center}, Width, Height$. Each line represents a color patch to be used for classification in the next steps.

4. Modelling and procedure followed

The next part to be followed was to learn a model for the images that can detect these color patches. The algorithm implemented was YOLO version 3 described in Literature Review section.

For the purpose of this project the modified version of YOLO v3 implemented in Darknet by AlexyAB github [6] repository was followed. This repository comes with additional features made on base version of the YOLO v3 by Joseph Redmon [7]. The procedure involved using the university server. Due to the limitations involved uploading data to the server, ssh connection was used in manifold steps, to transfer the data to the university server. Else, the documentation provided on the AlexyAB github respository was very clear and the training of the model was conducted smoothly.

The model was initialized with a pre-trained weights named *darknet53.conv.74*. Also, the batch size of 64 and sub-divisions 8 was making the GPU run out of memory, thus it was reduced to a batch size of 8 and sub divisions 4.

The model was trained for 30,000 iterations but, it was later found out that at 30,000 iterations it started overfitting, which can be seen in the average *mAP* for 30,000 iterations. The average *mAP* for 20,000 iterations gave better results, thus was used to produce the final results. These figures are discussed more in 5th section.

5. Evaluation parameters and Results

This section describes the various evaluation parameters used to test our model, the test datasets used and results in separate section.

5.1. Evaluation parameters

There are three evaluation parameters used, which are as follows:

1. **Precision:**
   The Precision is the ratio of true positives over total

[6] https://github.com/AlexeyAB/darknet
[7] https://github.com/pjreddie/darknet

number of positives predicted. In the context of the object detection, it is defined as the ratio given in figure 6 .

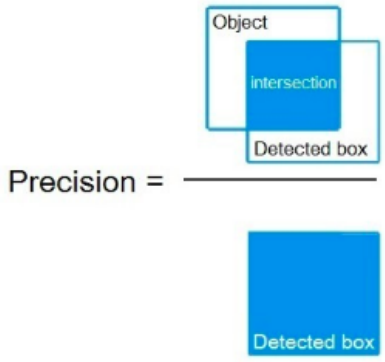

Figure 6. Precision
Source: https://github.com/AlexeyAB/darknet

2. **Recall:**
   The recall is the ratio of true positives over total actual positives. In the context of the object detection, it is defined as the ratio given in figure 7.

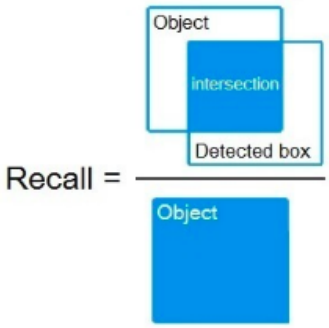

Figure 7. Recall
Source: https://github.com/AlexeyAB/darknet

3. **IOU or mean average dinner(mAP):**
   Intersection over union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. It is described in figure 8.

5.2. Results

This section is sub divided into two parts, since originally at the start of the project a camera was chosen out of 9 cameras to be used as a test set but, it was realized that the same scene is taken from the different cameras. So, this leads to the biased results, as the model has already seen that scene of the image during the training.

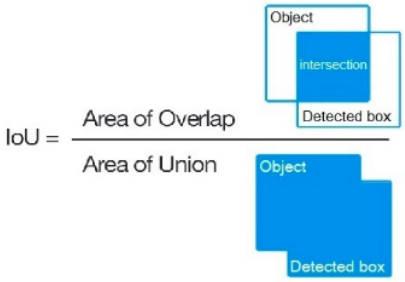

Figure 8. IOU
Source: https://github.com/AlexeyAB/darknet

There was a new test-set [8] used to evaluate our model. The results are described in following sections.

5.2.1 Old Test-Set

The old test-set performed with 98.06% mAP for a model trained for 20,000 iterations. The different mAPs for 10,000, 20,000 and 30,000 can be seen the figure in 9, 10 and 11.

It can be seen clearly that the mAP with 30,000 iterations is overfitting with worse values of IOU, Precision, Recall and F1 score compared to the 20,000 iterations model. It has precision of 1.0, recall of 1.0 and F1 score of 1.0 and also an average IOU of 79.41% and mAP of 98.06%.

```
Loading weights from ./yolo-obj_10000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
120
 detections_count = 3937, unique_truth_count = 2808
class_id = 0, name = 0, 3937      ap = 90.75 %
class_id = 1, name = 1,           ap = 90.29 %
class_id = 2, name = 2,           ap = 89.98 %
class_id = 3, name = 3,           ap = 90.75 %
class_id = 4, name = 4,           ap = 90.91 %
class_id = 5, name = 5,           ap = 90.51 %
class_id = 6, name = 6,           ap = 90.83 %
class_id = 7, name = 7,           ap = 90.74 %
class_id = 8, name = 8,           ap = 90.75 %
class_id = 9, name = 9,           ap = 90.17 %
class_id = 10, name = 10,         ap = 100.00 %
class_id = 11, name = 11,         ap = 90.91 %
class_id = 12, name = 12,         ap = 99.42 %
class_id = 13, name = 13,         ap = 90.18 %
class_id = 14, name = 14,         ap = 90.58 %
class_id = 15, name = 15,         ap = 90.91 %
class_id = 16, name = 16,         ap = 90.66 %
class_id = 17, name = 17,         ap = 90.75 %
class_id = 18, name = 18,         ap = 90.91 %
class_id = 19, name = 19,         ap = 90.83 %
class_id = 20, name = 20,         ap = 90.28 %
class_id = 21, name = 21,         ap = 90.91 %
class_id = 22, name = 22,         ap = 90.59 %
class_id = 23, name = 23,         ap = 90.75 %
 for thresh = 0.25, precision = 0.98, recall = 0.98, F1-score = 0.98
 for thresh = 0.25, TP = 2746, FP = 61, FN = 62, average IoU = 73.22 %

 mean average precision (mAP) = 0.913897, or 91.39 %
Total Detection Time: 7.000000 Seconds
```

Figure 9. mAP for 10,000 iterations in training for old test-set

[8] http://files.is.tue.mpg.de/pgehler/projects/color/index.html

```
Loading weights from ./yolo-obj_20000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
120
 detections_count = 3809, unique_truth_count = 2808
class_id = 0, name = 0, 3809      ap = 90.91 %
class_id = 1, name = 1,           ap = 90.91 %
class_id = 2, name = 2,           ap = 100.00 %
class_id = 3, name = 3,           ap = 100.00 %
class_id = 4, name = 4,           ap = 90.52 %
class_id = 5, name = 5,           ap = 90.43 %
class_id = 6, name = 6,           ap = 100.00 %
class_id = 7, name = 7,           ap = 100.00 %
class_id = 8, name = 8,           ap = 100.00 %
class_id = 9, name = 9,           ap = 100.00 %
class_id = 10, name = 10,         ap = 100.00 %
class_id = 11, name = 11,         ap = 100.00 %
class_id = 12, name = 12,         ap = 100.00 %
class_id = 13, name = 13,         ap = 100.00 %
class_id = 14, name = 14,         ap = 100.00 %
class_id = 15, name = 15,         ap = 100.00 %
class_id = 16, name = 16,         ap = 100.00 %
class_id = 17, name = 17,         ap = 100.00 %
class_id = 18, name = 18,         ap = 100.00 %
class_id = 19, name = 19,         ap = 100.00 %
class_id = 20, name = 20,         ap = 100.00 %
class_id = 21, name = 21,         ap = 100.00 %
class_id = 22, name = 22,         ap = 99.92 %
class_id = 23, name = 23,         ap = 90.75 %
 for thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
 for thresh = 0.25, TP = 2799, FP = 12, FN = 9, average IoU = 79.41 %

 mean average precision (mAP) = 0.980603, or 98.06 %
Total Detection Time: 7.000000 Seconds
```

Figure 10. mAP for 20,000 iterations in training for old test-set

```
Loading weights from ./yolo-obj_30000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
120
 detections_count = 3375, unique_truth_count = 2808
class_id = 0, name = 0, 3375      ap = 100.00 %
class_id = 1, name = 1,           ap = 100.00 %
class_id = 2, name = 2,           ap = 90.10 %
class_id = 3, name = 3,           ap = 90.91 %
class_id = 4, name = 4,           ap = 90.75 %
class_id = 5, name = 5,           ap = 90.13 %
class_id = 6, name = 6,           ap = 100.00 %
class_id = 7, name = 7,           ap = 90.91 %
class_id = 8, name = 8,           ap = 100.00 %
class_id = 9, name = 9,           ap = 90.18 %
class_id = 10, name = 10,         ap = 90.13 %
class_id = 11, name = 11,         ap = 90.20 %
class_id = 12, name = 12,         ap = 100.00 %
class_id = 13, name = 13,         ap = 100.00 %
class_id = 14, name = 14,         ap = 98.89 %
class_id = 15, name = 15,         ap = 90.29 %
class_id = 16, name = 16,         ap = 100.00 %
class_id = 17, name = 17,         ap = 99.70 %
class_id = 18, name = 18,         ap = 90.75 %
class_id = 19, name = 19,         ap = 90.91 %
class_id = 20, name = 20,         ap = 90.91 %
class_id = 21, name = 21,         ap = 100.00 %
class_id = 22, name = 22,         ap = 99.42 %
class_id = 23, name = 23,         ap = 90.91 %
 for thresh = 0.25, precision = 0.99, recall = 0.99, F1-score = 0.99
 for thresh = 0.25, TP = 2776, FP = 22, FN = 32, average IoU = 74.66 %

 mean average precision (mAP) = 0.947952, or 94.80 %
Total Detection Time: 6.000000 Seconds
```

Figure 11. mAP for 30,000 iterations in training for old test-set

The model with mAP with the best result was selected to produce the results on the test images. These results are shown in figure 12 and 13. It can be seen through these test images that the model is robust enough to recognize the color patches in small areas also.

5.2.2 New Test-set

The new dataset performed with best mAP of 51.67% for a model trained on 20,000 iterations. The different mAPs obtained for 10,000, 20,000 and 30,000 can be seen the figure

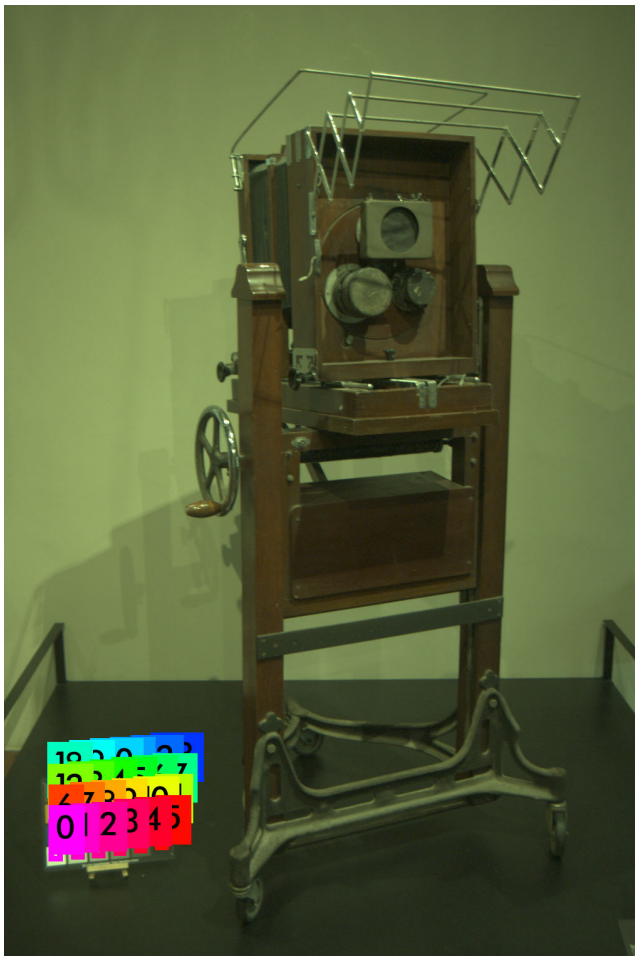

Figure 12. Detection of color patches with best model obtained

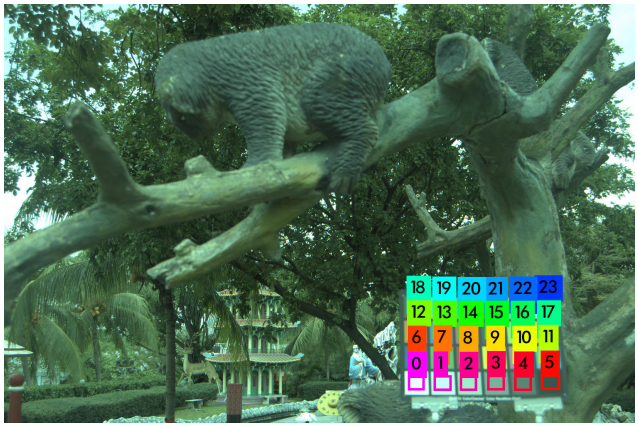

Figure 13. Detection of color patches with best model obtained

in 14, 15 and 16. It can be seen that mAp and every other evaluation parameter performs worse on new test-set. The possible reasons are discussed later for this drop in performance. For the new dataset, same weights of the models were used to predict and the results were thus obtained in

```
Loading weights from ./yolo-obj_10000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
568
 detections_count = 52566, unique_truth_count = 13632
class_id = 0, name = 0,  52566   ap = 36.73 %
class_id = 1, name = 1,          ap = 37.46 %
class_id = 2, name = 2,          ap = 38.17 %
class_id = 3, name = 3,          ap = 41.80 %
class_id = 4, name = 4,          ap = 38.80 %
class_id = 5, name = 5,          ap = 41.31 %
class_id = 6, name = 6,          ap = 41.71 %
class_id = 7, name = 7,          ap = 45.14 %
class_id = 8, name = 8,          ap = 43.71 %
class_id = 9, name = 9,          ap = 50.17 %
class_id = 10, name = 10,        ap = 49.34 %
class_id = 11, name = 11,        ap = 53.87 %
class_id = 12, name = 12,        ap = 51.98 %
class_id = 13, name = 13,        ap = 46.71 %
class_id = 14, name = 14,        ap = 47.49 %
class_id = 15, name = 15,        ap = 48.87 %
class_id = 16, name = 16,        ap = 48.25 %
class_id = 17, name = 17,        ap = 53.49 %
class_id = 18, name = 18,        ap = 44.73 %
class_id = 19, name = 19,        ap = 46.73 %
class_id = 20, name = 20,        ap = 44.62 %
class_id = 21, name = 21,        ap = 38.19 %
class_id = 22, name = 22,        ap = 47.55 %
class_id = 23, name = 23,        ap = 50.82 %
 for thresh = 0.25, precision = 0.60, recall = 0.54, F1-score = 0.57
 for thresh = 0.25, TP = 7420, FP = 4941, FN = 6212, average IoU = 38.59 %

 mean average precision (mAP) = 0.453182, or 45.32 %
Total Detection Time: 29.000000 Seconds
```

Figure 14. mAP for 10,000 iterations in training for new test-set

```
Loading weights from ./yolo-obj_20000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
568
 detections_count = 44362, unique_truth_count = 13632
class_id = 0, name = 0,  44362   ap = 42.61 %
class_id = 1, name = 1,          ap = 49.93 %
class_id = 2, name = 2,          ap = 46.32 %
class_id = 3, name = 3,          ap = 42.94 %
class_id = 4, name = 4,          ap = 41.71 %
class_id = 5, name = 5,          ap = 41.57 %
class_id = 6, name = 6,          ap = 52.61 %
class_id = 7, name = 7,          ap = 54.50 %
class_id = 8, name = 8,          ap = 54.54 %
class_id = 9, name = 9,          ap = 53.99 %
class_id = 10, name = 10,        ap = 52.80 %
class_id = 11, name = 11,        ap = 55.80 %
class_id = 12, name = 12,        ap = 52.30 %
class_id = 13, name = 13,        ap = 54.03 %
class_id = 14, name = 14,        ap = 53.58 %
class_id = 15, name = 15,        ap = 51.32 %
class_id = 16, name = 16,        ap = 55.10 %
class_id = 17, name = 17,        ap = 63.69 %
class_id = 18, name = 18,        ap = 53.37 %
class_id = 19, name = 19,        ap = 56.58 %
class_id = 20, name = 20,        ap = 51.93 %
class_id = 21, name = 21,        ap = 49.51 %
class_id = 22, name = 22,        ap = 53.23 %
class_id = 23, name = 23,        ap = 56.16 %
 for thresh = 0.25, precision = 0.64, recall = 0.60, F1-score = 0.62
 for thresh = 0.25, TP = 8232, FP = 4658, FN = 5400, average IoU = 41.35 %

 mean average precision (mAP) = 0.516711, or 51.67 %
Total Detection Time: 29.000000 Seconds
```

Figure 15. mAP for 20,000 iterations in training for new test-set

the end. As shown in the figure 17, 18 and 19. It can be particularly seen in figure 17 that the patches are not at all correctly detected for many patches. These types of results were seen in many other images also. This is the reason for the overall drop in the performance of the model trained on these new test images.

There are few reasons that were seen as the reason for this drop in the performance:

- The first reason was the very fact that the model was not trained on the images incorporating large rotation of the checkerboard in the images, as the original dataset had images directly in the plane of the image. Thus, the model never was introduced to capture these rotation so was not able to detect it properly.

```
Loading weights from ./yolo-obj_30000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
568
 detections_count = 35457, unique_truth_count = 13632
class_id = 0, name = 0,  35457   ap = 42.47 %
class_id = 1, name = 1,          ap = 41.72 %
class_id = 2, name = 2,          ap = 38.91 %
class_id = 3, name = 3,          ap = 37.25 %
class_id = 4, name = 4,          ap = 31.75 %
class_id = 5, name = 5,          ap = 34.10 %
class_id = 6, name = 6,          ap = 41.46 %
class_id = 7, name = 7,          ap = 43.39 %
class_id = 8, name = 8,          ap = 41.05 %
class_id = 9, name = 9,          ap = 41.05 %
class_id = 10, name = 10,        ap = 40.75 %
class_id = 11, name = 11,        ap = 42.34 %
class_id = 12, name = 12,        ap = 43.62 %
class_id = 13, name = 13,        ap = 43.53 %
class_id = 14, name = 14,        ap = 41.66 %
class_id = 15, name = 15,        ap = 43.05 %
class_id = 16, name = 16,        ap = 42.87 %
class_id = 17, name = 17,        ap = 49.72 %
class_id = 18, name = 18,        ap = 43.04 %
class_id = 19, name = 19,        ap = 49.19 %
class_id = 20, name = 20,        ap = 48.04 %
class_id = 21, name = 21,        ap = 40.56 %
class_id = 22, name = 22,        ap = 41.95 %
class_id = 23, name = 23,        ap = 44.46 %
 for thresh = 0.25, precision = 0.58, recall = 0.53, F1-score = 0.55
 for thresh = 0.25, TP = 7196, FP = 5149, FN = 6436, average IoU = 37.98 %

 mean average precision (mAP) = 0.419969, or 42.00 %
Total Detection Time: 28.000000 Seconds
```

Figure 16. mAP for 30,000 iterations in training for new test-set

Figure 17. Detection of color patches with best model obtained

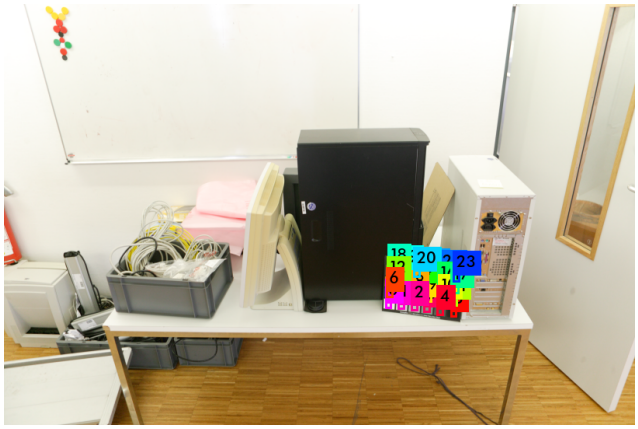

Figure 18. Detection of color patches with best model obtained

- The second reason can also be the fact that the images on which the model was tested on originally, has al-

Figure 19. Detection of color patches with best model obtained

ready seen those particular scenes while training, so there was a leakage of the information and model was over-fitted.

The results obtained were robust on the clear images of the new test set also, which can be seen in figure 18 and 19, thus confirming the claims made before.

6. Conclusions

This project can be concluded on following points:

- The YOLOv3 algorithm performed at par, with very satisfactory results.
- It was realized that the data pre-processing of the images was a critical task for effective learning. The assumptions taken which was discussed before proved to be every effective and produced very robust results.
- It was learned during this project that how the leakage of information can though produce good results for some test images but will fail on other set of similar test images.
- The mean average precision is a very robust performance evaluation metric which produced consistent results.
- The project overall produced satisfactory results and this work can be extended to detect other types of the images. The YOLOv3 is easy to implement and has very robust performance both on images and video data.

References

[1] Yolo-v3 and yolo-v2 for windows and linux. https://github.com/AlexeyAB/darknet. 1

[2] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. 2

[3] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. 1, 2

[4] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1

[5] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. 1

[6] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 1, 2