DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

FINAL PROJECT REPORT for

Fabric fault detection using AI and machine learning

by

Aaron Blanco

Student ID: 1911261

Supervisor: Dr Qingping Yang

Submitted in part fulfilment of the requirements for

the degree of Bachelors of Engineering

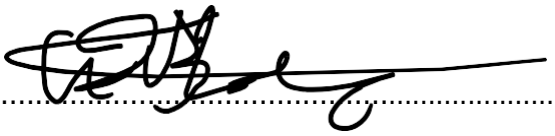
in

Mechanical Engineering

APRIL 2022

DECLARATION

I declare that the work in this project was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation report that is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material, which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

SIGNED:  DATE: 25/04/2022

ABSTRACT

This project is about developing and testing a prototype system to apply image processing and advanced machine learning to detect possible fabric faults, so the fabric defects will not be used for the manufacturing. To develop the prototype, deep learning was studied in MATLAB and research into fabric fault detection using deep learning was conducted. After this the programming for the prototype begun. The program was based on a pretrained network which utilised transfer learning to create a network specifically for fabric defect detection. The data used is provided by a company called SoundSorba which specialises in sound-proofing products. The company sent over a sample of different fabrics with defects and without. The fabrics were analysed by using a YOLOv4 object detection network to determine if a fabric has a fault or not. The pretrained object detector had an accuracy of 88.2% for detecting defects however it was unable to detect no defects. The ideas for improvement were stated that images be relabelled and to make the detector more accurate and to have a larger dataset too.

ACKNOWLEDGEMENTS

Thank you for my superviose Dr Qingping Yang and the owner and staff of SoundSorba for allowing me to create this project and providing the necessary data to complete it.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

CNN	**Convolutional Neural Network**
DCNN	**Deep convolutional Neural Network**
YOLO	**You Only Look Once**
IoU	**intersection-over-union**
AP	**Average Precision**

1 Introduction

1.1 Context for the project

The prototype developed will help in the manufacturing of acoustic panels by detecting faults in the fabrics used for the company Soundsorba. Soundsorba are acoustic specialists providing sound control solutions for building interior and provide acoustic products in the following: sound absorbing panels, acoustic wall and ceiling tiles, acoustic doors, acoustic diffusers, acoustic fabric, sound absorbers and sound absorbing foam. The problem with the fabrics is a client-side issue that focuses on the design aspect. For example, if there was an error in the pattern or some of the fabric is coming off, etc. This will be solved by creating a program using techniques of machine and deep learning. As the program will need to be able to identify multiple fabrics by accessing a database of images. Through these images it will have learnt what fabrics in good or bad condition look like. In return, the program will apply this knowledge and decide whether the fabric has a defect or not – at the same time the program will need to be maintained so you'd need to be able to add new images to the database whenever a new type of fabric is being used.

In the manufacturing of acoustic panels for Soundsorba, panels are wrapped in fabric. Fabric that is formed by yarning, weaving, knitting, tatting, felting, crocheting, or braiding natural or synthetic yarns. However, these fabrics can contain some of these defects such as a hole, knot, thick bar, thin bar, missing picks, broken end, stain weft yarn, oil spot, double pick, double end, loose weft, broken end, missing picks, etc [1] [2]. These defects usually occur due to the failure in design and machine production equipment, which reduces the quality of the fabric and wastes it as well. Hence, detecting defects is a core competency that enterprises should possess to improve the quality of manufacturing products without affecting production [3].

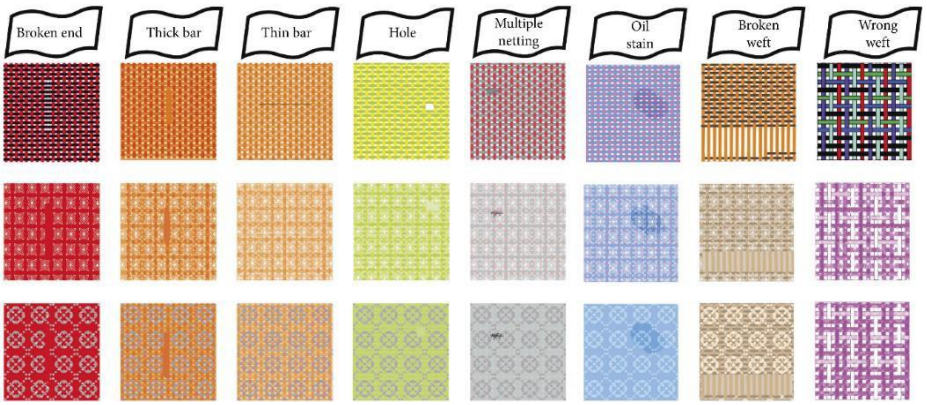


Figure 1) Examples of textile fabric defects [4]

Traditionally, inspection process is completed manually through human efforts to ensure the quality of the fabric. The main drawbacks with manual inspections are as follows: training individuals to make them a fabric inspector, major defects can be detected while small defects can be ignored due to human carelessness, a lot of effort is required to locate fabric defects; and it is difficult for fabric inspectors to keep focus on the production process for a time that is more than 10 minutes [4]. Also, according to research, 60-75% human accuracy is reported to detect fabric defects and the wastage due to fabric defects leads to the high price of product in market [5]. Due to this looking into automating the process of defect detection would speed up the process and accuracy if done correctly. This project will be using a deep learning algorithm, specifically a convolutional neural network (CNN). A Convolutional neural network are multilayer neural networks which are skilled in dealing with related machine learning problems, especially large-scale image classification tasks.

1.2 Aim and objectives for the project

The aim of this project is to develop and test a prototype system to apply image processing and advanced machine learning to detect possible fabric faults, so the fabric defects will not be used for the manufacturing.

The objectives to be met to reach the aim of this project are:

- Create a database of images of the fabrics without and with faults
- Train a network using the database of images
- Validate results of the program by testing the network

2 Literature Review

2.1 Review of Neural Networks

There are various methods to detect defects in fabrics using neural networks, all with different accuracies and network structures. Since the aim of the project is to use advanced machine learning, the articles that are being investigated will associate with deep learning-based approaches to solve the problem.

2.1.1 DefectNet Network

In an article titled Real-time Fabric Defect Detection based on Lightweight Convolutional Neural Network [6], the authors design a lightweight CNN model called DefectNet. It is based on a streamlined architecture that uses depthwise separable convolutions instead of standard convolutions, and significantly reduces the computational complexity of the model. The model uses a multi-scaled feature extraction method to improve detection ability of the model for fabric defects of various sizes. The authors plan to extract features of a deep convolutional neural network (DCNN) as it has strong feature extraction abilities and combine them with a lightweight model, to improves detection times as a DCNN can takes a lot of computational power and time. When testing the network, the authors created their own database of 3000-images of fabrics with and without defects.

The authors recognised that a fabric defect image is mainly composed of low-order features such as contour and texture. Therefore, when detecting fabric defects, the network should mainly extract the low-semantic features of the image. Therefore, a lightweight network architecture is designed with fewer convolutional layers, which reduces the depth of the network, but achieves feature extraction in low-order feature maps.

L	Layer type	Filter Size	Stride	Output Size w ×h ×d
0	Input	-	-	512×512×3
1	Conv	3×3，16	1	512×512×16
2	Dw conv Conv	3×3，16 1×1×16，32	2 1	256×256×16 256×256×32
3	Dw conv Conv	3×3，32 1×1×32，64	2 1	128×128×32 128×128×64
4	Dw conv Conv	3×3，64 1×1×64，128	2 1	64×64×64 64×64×128
5	Dw conv Conv	3×3，128 1×1×128，256	2 1	32×32×128 32×32×256
6	Conv Conv	3×3，256 1×1×256，128	1 1	32×32×256 32×32×128
7	Dw conv Conv	3×3，128 1×1×128，256	2 1	16×16×128 16×16×256
8	Dw conv Conv	3×3，256 1×1×256，256	1 1	16×16×256 16×16×256

Figure 2) Network architecture of DefectNet [6]

The structure of DefectNet is built on depthwise separable convolutions. It also uses continuous depthwise convolution and pointwise convolutional layers for dimensionality reduction and feature fusion. Figure 3 shows the factorized layer with depthwise convolution, batch-norm, Leaky ReLU layer and 1 × 1 pointwise convolution after each convolutional layer. In the forward propagation process, the dimension transformation of tensor is shown by adjusting the step size of the convolution core instead of the max pooling layer, which reduces

the loss of feature information, ensures the high resolution of convolution output, and improves the detection accuracy.

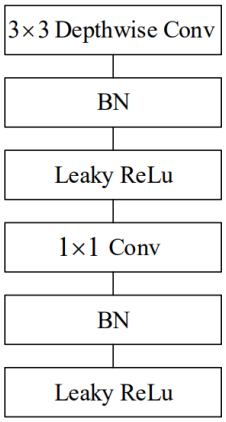


Figure 3) The factorised layer with depthwise convolution [6]

In the detection network, the authors use a 1x1 convolutional layers instead of the output of the full connection layers to realise the full convolutional network model and reduce the network parameters and computation. To detect the defects in a fabric, multi-scaled feature extraction methods are used to extract features from different scales.

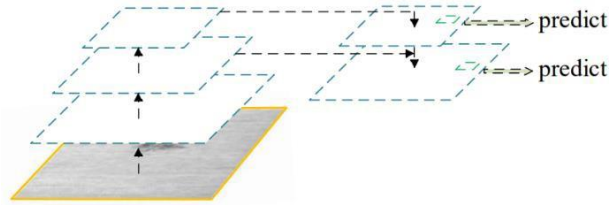


Figure 4) Multi-scaled feature extraction [6]

This is done by extracting features from two feature maps of different scales. When extracting these features, high-resolution information of low-order features and high-semantic information of high-order features are utilised. Features are extracted from feature maps after 16× and 32× downsampled, and fuse features between the two layers of feature maps by using concatenation. This method obtained more meaningful semantic information from the upsampled features and can improve the feature extraction ability of small targets.

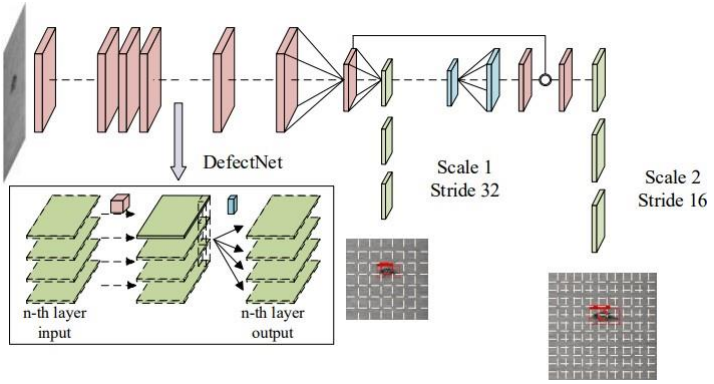


Figure 5) Network Structure diagram of DefectNet model [6]

The DefectNet model was inspired by the YOLO network. The model functioned by using regression methods to detect targets directly, a full convolutional network structure is adopted, an anchor box mechanism is added to predict boundary frames and its coordinate values. In the process of detection, the input image is divided into SxS cells of the same size. Each cell is responsible for predicting the target located in the centre of the cell. One cell predicts B boundary boxes. The input image is processed through several deep convolutional layers to obtain the feature maps after 32× downsampled. This is so more meaningful semantic information can be obtained by feature extraction from feature maps of this layer. Then it is upsampled by 2× and fused with the feature maps of 16× downsampled of the previous layer, which can extract the feature information with better accurate information. From the basic feature extractor, several convolutional layers are utilized to predict bounding box, objectness, and class predictions. The model predicts three different scale boxes in each element layer to detect defects of various sizes.

	ACC (%)	Speed (ms)	Model size (MB)
SSD	75.3	420.2	163
YOLO-v3	98.0	132.2	241
Tiny-Yolo-v3	86.1	56.5	34
DefectNet	97.7	32.7	9

Figure 6) Comparison of the accuracy, speed and parameter [6]

After testing the DefectNet network, collecting data and compared it to three other networks called SSD, YOLO-v3 and Tiny-YOLO-v3. The authors managed to create a high accuracy model with fast detection times. The model reaches an accuracy of 97.7% which is close to YOLO-v3 but the model size is 26.8 times smaller than that of YOLO-v3 and the detection speed is increased by 4 times. Figure 6 clearly represents the aims and objectives set by the authors when deciding to create this network model.

2.1.2 YOLO-LFD Network

In an article called Fabric Defect Detection Based on Lightweight Neural Network [7], the authors also propose the idea that a deep convolution neural network (DCNN) is good due its detection accuracy however it comes with the drawbacks of large computational costs and storage services. Therefore, they suggest a lightweight CNN as it can be computation efficient. This network is named YOLO-LFD, which adopts Darknet-53 and uses a multi-scale feature extraction method to improve the detection ability of the model for different size targets. Lastly, it adopts a K-means clustering method on the fabric defect image dataset to find the optimal size of anchor boxes. In the YOLO-LFD network, it uses a continuous 3×3 and 1×1 convolution layers for dimensionality reduction and feature fusion. In the forward propagation process, the dimension transformation of tensor is realized by adjusting the step size of convolution core instead of max pooling layer, which ensures the high resolution of convolution output and improves the detection accuracy.

The network also increases its depth and improves detection accuracy by using feature maps on different channels that are fused by 1x1 convolutional layers, which can highlight the features of the detection target. Another reason to use feature maps is to reduce the computational complexity as a dimension reduction module. The method allows the convolution results of the previous layer to be fused by 1x1 convolutional layers, which improve the ability of feature fusion and reduce the computational complexity.

L	Layer type	Filter size (w × h, n)	Stride	Output size w × h × d
0	Input	–	–	512 × 512 × 3
1	Convolution	3 × 3, 16	2	256 × 256×16
	Convolution	1 × 1, 8	–	256 × 256×8
2	Convolution	3 × 3, 32	2	128 × 128×32
	Convolution	1 × 1, 16	–	128 × 128×16
3	Convolution	3 × 3, 64	2	64 × 64×64
	Convolution	1 × 1, 32	–	64 × 64×32
4	Convolution	3 × 3, 128	2	32 × 32×128
	Convolution	1 × 1, 64	–	32 × 32×64
5	Convolution	3 × 3, 256	1	32 × 32 × 256
6	Convolution	1 × 1, 128	–	32 × 32 × 128
	Convolution	3 × 3, 256	2	16 × 16 × 256
7	Convolution	1 × 1, 128	–	16 × 16 × 128
	Convolution	3 × 3, 256	1	16 × 16 × 256
8	Convolution	1 × 1, 128	–	16 × 16 × 128
	Convolution	3 × 3, 256	1	16 × 16 × 256

Figure 7) Framework of YOLO-LFD

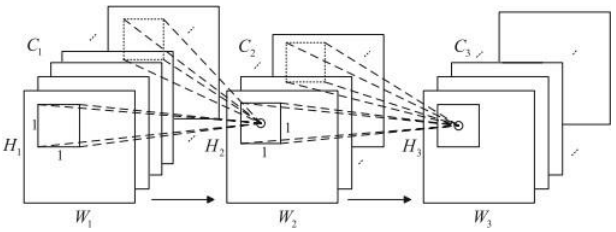


Figure 8) Feature fusion of 1x1 convolutional layers

The authors understood that fabric defect images mainly consist of direction, boundary, and curvature information. Also, most fabrics defect sizes tend to be very small, so they adopt multi-scale feature extraction in the low-level features and medium level features. The mulit-scale detection module of YOLO-v3 is revised by using two different scaled feature maps to extract features, which can make use of both high-resolution information of low-features and high-semantic information of high-level features. Features are extracted from the feature maps after 16x and 32x downsampled and fuse the features between the two layers of feature maps. This method improves the feature extraction ability of small targets and gets more meaningful semantic information from upsampled features.

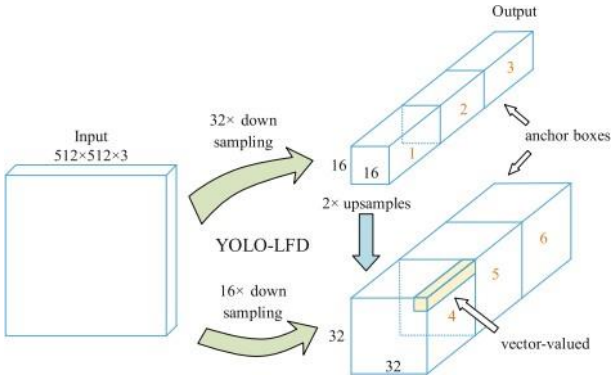


Figure 9) Multi-scale detection module

	ACC (%)	Speed (ms)	Model size (MB)
SSD	75.3	420.2	163
YOLO-v3	98.0	132.2	241
Tiny-Yolo-v3	86.1	56.5	34
YOLO-LFD	97.2	51.0	8

Figure 10) Table comparing accuracy, speed and parameter size of each model

Lastly, the algorithm uses K-means clustering method is used to get compact and independent clusters, and the data can be classified into different categories according to the distance relationship. It is also used to cluster the size of ground truth in fabric image database and get anchor boxes which are closest to the size of the fabrics defects bounding box. The network was tested using a dataset created by the authors of 3000 fabric defect images that contain five classes. Those five classes being: holes, surface debris, oil stains, longitude and weft breakage. Among them, 2000 pictures were randomly selected as the training set and 1000 pictures as the test set. The experimental results were compared with YOLO-v3, Tiny-YOLO-v3 and SSD. The results are shown in Figure 10. The results show that the accuracy is close to YOLO-v3, however the model is 30x smaller than that of YOLO-v3 and the detection speed is increased by 2.6x. The authors therefore have been able to create the lightweight model as they proposed. Now is the case of increasing the accuracy and finding ways to make the model computationally more efficient.

3 Experimental Method

This section details the steps in creating a prototype for fabric defect detection, where the program was written in MATLAB R2022a. In addition, additional apps were installed on MATLAB that supported the progression of the project. These apps were the Deep Learning Toolbox, Computer Vision Toolbox and Computer Vision Toolbox Model for YOLO v4 Object Detection.

3.1 Creating the Dataset

The dataset was created with the aid of SoundSorba providing a total of 7 unique coloured fabric. Majority of the fabrics that were provided had no defects, so this meant that defects had to be made manually. The number of defects that could be made were limited, which resulted in only oil stain and hole defects to be made. Other datasets on fabric defects that are found on the internet can be used to make up for the other defects, however the datasets were either unreliable or wasn't available to the public. Despite most of the fabrics provided by SoundSorba had no defects, the few that did, contain multi-netting or dirt.

The images of the fabrics were taken using the back-facing camera of an iPhone 11 Pro. The images were captured under the following conditions: fabric must be placed on a flat surface, photos taken in a well-lit room, camera flash on, photos must be taken from an approximate height of 14cm above the fabric and photos are taken in square crop. A total of 903 images were taken and 479 of them don't contain defects. These images were separated into folders according to their colour and additional subfolders categorised the images into "Defects" and "No defects". In the defect folder, the subfolders "Oil Stain", "Hole", "Multiple Netting" and "Dirt" were created. However, the categorie "Multiple Netting" was only used for certain fabrics, as not all the fabrics contained these defects. After the images were filed correctly, they were renamed according to their colour and type of defect or not at all. In MATLAB, an image datastore was created to store the images of the fabrics

```
 fabricds =
imageDatastore("Fabric_Images","IncludeSubfolders",true,"LabelSource","folder
names");
```

3.2 Creating the Neural Network

3.2.1 Network Architecture

After reviewing neural networks that were designed to detect fabric in defects, it was decided to create neural network that uses the YOLOv4 object detector with resnet50 as its base network. In the literature review, the articles mentioned the use of YOLOv3. YOLOv4 was used as it is an improved version of YOLOv3 for the following reasons: it uses CSPDarknet53 as its backbone that enhances the learning capability of CNN, the spatial pyramid pooling block is added over CSPDarknet53 to increase the receptive field and separate out the most significant context features and uses PANet as a method for parameter aggregation for different detector levels. These changes make it twice as fast as EfficientDet (competitive recognition model). In addition, AP (Average Precision) and FPS (Frames Per Second) increased by 10% and 12% compared to YOLOv3 [7].

As already stated in the paragraph above, for extracting features from the input photos, the YOLO v4 network employs CSPDarkNet-53 as the backbone. Five residual block modules make up the backbone, and the feature map outputs from the residual block modules are fused at the YOLO v4 network's neck. To extract the most representative features, the SPP module in the neck concatenates the low-resolution feature map's max-pooling outputs. For the max-pooling process, the SPP module employs kernels of size 1-by-1, 5-by-5, 9-by-9, and

13-by-13. The stride is set at one. Concatenating the feature maps expands the backbone features' receptive area and improves the network's accuracy in recognising small objects. Using a PAN, the SPP module's concatenated feature maps are fused with the high-resolution feature maps. The PAN sets bottom-up and top-down routes for merging low-level and high-level features using upsampling and downsampling processes.

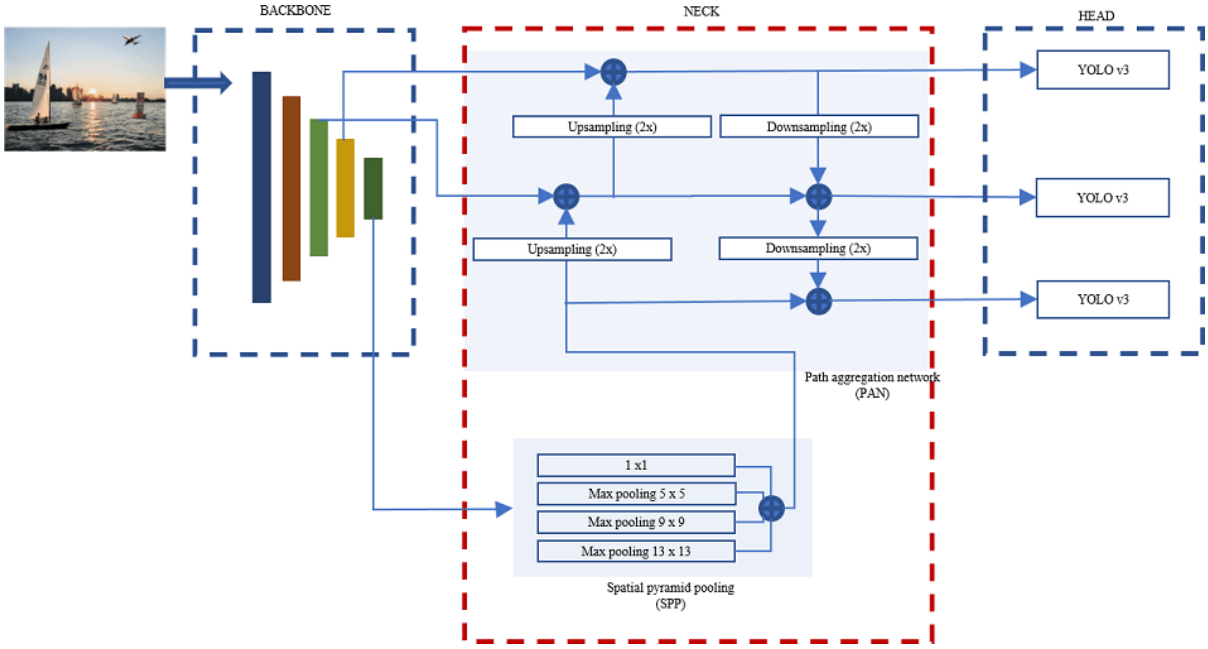


Figure 11) Image of YOLO v4 architecture [8]

The PAN module generates a collection of aggregated feature maps that can be used to make predictions. Three networks make up the YOLO v4 network. The final predictions are computed by each detection head, which is a YOLO v3 network. To anticipate the bounding boxes, classification scores, and objectness scores, the YOLO v4 network outputs use maps of sizes 19-by-19, 38-by-38, and 76-by-76 [8].

3.2.2 Ground Truth Table

To train a YOLOv4 object detector, you must have labelled images with their bounding boxes defined. Bounding boxes represent the labelled area of an image, where its position on the image is recorded. This information is stored in a ground truth table. A ground truth object contains information about the data source, label definitions, and marked label annotations for a set of ground truth labels. To create a ground truth table, the Image Labeler app was used. The Image Labeler app allows you to label ground truth data in a collection of images. Using the app, you can: define rectangular regions of interest (ROI) labels, polyline ROI labels, pixel ROI labels, polygon ROI labels, and scene labels. These labels are used to interactively label your ground truth data. Secondly, you can export the labelled ground truth as a groundTruth object. This will be used in training an object detector.

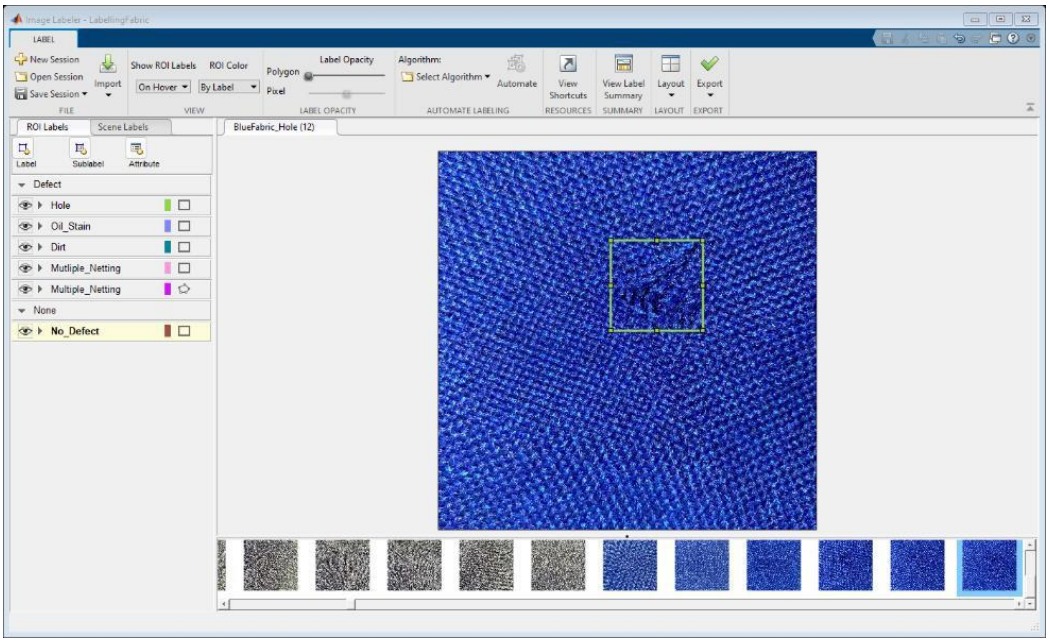


Figure 12) Screenshot of Image Labeler application

	1 imageFilename	2 Hole	3 Oil_Stain	4 Mutliple_Netting	5 No_Defect
1	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (1).JPEG'	[689,455,62,...	[]	[]	[]
2	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (10).JPEG'	[492,630,42...	[]	[]	[]
3	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (11).JPEG'	[520,605,35...	[]	[]	[]
4	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (12).JPEG'	[561,464,37...	[]	[]	[]
5	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (13).JPEG'	[495,302,29...	[]	[]	[]
6	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (14).JPEG'	[589,427,22...	[]	[]	[]
7	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (15).JPEG'	[655,499,33...	[]	[]	[]
8	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (16).JPEG'	[748,580,41...	[]	[]	[]
9	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (17).JPEG'	[742,439,46...	[]	[]	[]
10	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (2).JPEG'	[430,851,31...	[]	[]	[]
11	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (22).JPEG'	[583,492,21...	[]	[]	[]
12	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (23).JPEG'	[492,527,25...	[]	[]	[]
13	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (3).JPEG'	[467,848,34...	[]	[]	[]
14	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (4).JPEG'	[467,839,41...	[]	[]	[]
15	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (5).JPEG'	[533,842,45...	[]	[]	[]
16	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (6).JPEG'	[536,876,49...	[]	[]	[]
17	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (7).JPEG'	[467,904,56...	[]	[]	[]
18	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (8).JPEG'	[436,773,54...	[]	[]	[]
19	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Hole\BlackFabric_Hole (9).JPEG'	[439,736,52...	[]	[]	[]
20	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (1).JPEG'	[]	[327,290,74...	[]	[]
21	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (10).JPEG'	[]	[589,196,59...	[]	[]
22	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (11).JPEG'	[]	[555,127,68...	[]	[]
23	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (12).JPEG'	[]	[676,187,60...	[]	[]
24	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (13).JPEG'	[]	[761,258,54...	[]	[]
25	'D:\Aaron\Majot Individual Project\Fabric_Images\Black Fabric\Defect\Oil Stain\BlackFabric_Oil (14).JPEG'	[]	[701,233,55...	[]	[]

Figure 13) Screenshot of Ground Truth table that was created after exporting it from Image Labeler application

3.2.3 Loading the Dataset

Before creating the object-detector and training it, the data was split up into three categories: training, validation, and testing. However, when you train an object detector you need to create an image and box label datastores which are used to store the file locations and bounding boxes of the fabric images respectively. After the datastores were created, the program validated them to find any invalid images, bounding boxes, or labels.

Create image datastore

```
fabricds =
imageDatastore("Fabric_Images","IncludeSubfolders",true,"LabelSource","folder
names");
```

Splitting data

Split the dataset into training, validation, and test sets. Select 55% of the data for training, 15% for validation, and the rest for testing the trained detector.

```
load fabricGTv2.mat

source = gTruth.DataSource.Source;
labelData = gTruth.LabelData;
fabricDataset = [source labelData];

rng("default");
shuffledIndices = randperm(height(fabricDataset));
idx = floor(0.55 * length(shuffledIndices));

trainingIdx = 1:idx;
trainingDataTbl = fabricDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.15 * length(shuffledIndices) );
validationDataTbl = fabricDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
testDataTbl = fabricDataset(shuffledIndices(testIdx),:);
```

Use `imageDatastore` and `boxLabelDatastore` to create datastores for loading the image and label data during training and evaluation.

```
imdsTrain =
imageDatastore(trainingDataTbl{:,"Var1"},"LabelSource","foldernames");
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));

imdsValidation =
imageDatastore(validationDataTbl{:,"Var1"},"LabelSource","foldernames");
bldsValidation = boxLabelDatastore(validationDataTbl(:,2:end));

imdsTest =
imageDatastore(testDataTbl{:,"Var1"},"LabelSource","foldernames");
bldsTest = boxLabelDatastore(testDataTbl(:,2:end));
```

Combine image and box label datastores.

```
trainingData = combine(imdsTrain,bldsTrain);
validationData = combine(imdsValidation,bldsValidation);
testData = combine(imdsTest,bldsTest);
```

Validate data

```
validateInputData(trainingData);
validateInputData(validationData);
validateInputData(testData);
```

Functions

```
function validateInputData(ds)
% Validates the input images, bounding boxes and labels and displays the
% paths of invalid samples.

% Copyright 2021 The MathWorks, Inc.

% Path to images
info = ds.UnderlyingDatastores{1}.Files;

ds = transform(ds, @isValidDetectorData);
data = readall(ds);

validImgs = [data.validImgs];
validBoxes = [data.validBoxes];
validLabels = [data.validLabels];

msg = "";

if(any(~validImgs))
    imPaths = info(~validImgs);
    str = strjoin(imPaths, '\n');
    imErrMsg = sprintf("Input images must be non-empty and have 2 or 3
dimensions. The following images are invalid:\n") + str;
    msg = (imErrMsg + newline + newline);
end

if(any(~validBoxes))
    imPaths = info(~validBoxes);
    str = strjoin(imPaths, '\n');
    boxErrMsg = sprintf("Bounding box data must be M-by-4 matrices of
positive integer values. The following images have invalid bounding box
data:\n") ...
        + str;

    msg = (msg + boxErrMsg + newline + newline);
end

if(any(~validLabels))
    imPaths = info(~validLabels);
    str = strjoin(imPaths, '\n');
```

```
        labelErrMsg = sprintf("Labels must be non-empty and categorical. The
following images have invalid labels:\n") + str;

        msg = (msg + labelErrMsg + newline);
    end

    if(~isempty(msg))
        error(msg);
    end

    end

    function out = isValidDetectorData(data)
    % Checks validity of images, bounding boxes and labels
    for i = 1:size(data,1)
        I = data{i,1};
        boxes = data{i,2};
        labels = data{i,3};

        imageSize = size(I);
        mSize = size(boxes, 1);

        out.validImgs(i) = iCheckImages(I);
        out.validBoxes(i) = iCheckBoxes(boxes, imageSize);
        out.validLabels(i) = iCheckLabels(labels, mSize);
    end

    end

    function valid = iCheckImages(I)
    % Validates the input images.

    valid = true;
    if ndims(I) == 2
        nDims = 2;
    else
        nDims = 3;
    end
    % Define image validation parameters.
    classes        = {'numeric'};
    attrs          = {'nonempty', 'nonsparse', 'nonnan', 'finite', 'ndims',
nDims};
    try
        validateattributes(I, classes, attrs);
    catch
        valid = false;
    end
    end
```

```
function valid = iCheckBoxes(boxes, imageSize)
% Validates the ground-truth bounding boxes to be non-empty and finite.

valid = true;
% Define bounding box validation parameters.
classes = {'numeric'};
attrs   = {'nonempty', 'integer', 'nonnan', 'finite', 'positive', 'nonzero',
'nonsparse', '2d', 'ncols', 4};
try
    validateattributes(boxes, classes, attrs);
    % Validate if bounding box in within image boundary.
    validateattributes(boxes(:,1)+boxes(:,3)-1, classes, {'<=',
imageSize(2)});
    validateattributes(boxes(:,2)+boxes(:,4)-1, classes, {'<=',
imageSize(1)});
catch
    valid = false;
end
end

function valid = iCheckLabels(labels, mSize)
% Validates the labels.

valid = true;
% Define label validation parameters.
classes = {'categorical'};
attrs   = {'nonempty', 'nonsparse', '2d', 'ncols', 1, 'nrows', mSize};
try
    validateattributes(labels, classes, attrs);
catch
    valid = false;
end
end
```

3.2.4 Anchor Boxes

To predict objects using YOLO v4, you use anchor boxes to detect classes of objects in an image, but also improves the speed and efficiency for the detection portion of a deep learning neural network frame. Anchor boxes are a series of predetermined bounding boxes of a specific height and width. These boxes are often chosen depending on item sizes in your training datasets to capture the scale and aspect ratio of various object classes you want to detect. To use anchor boxes, you must estimate them using the training data obtained in section 3.2.3 and pre-process them before using it in training the network.

Preprocessing and Estimating Anchor Boxes

```
% data = read(TrainingData);
inputSize = [224 224 3];

rng("default")
```

```
 trainingDataForEstimation =
transform(trainingData,@(fabricDataset)preprocessData(fabricDataset,inputSize
));
 numAnchors = 18;
 [anchors,meanIoU] =
estimateAnchorBoxes(trainingDataForEstimation,numAnchors);

 area = anchors(:,1).*anchors(:,2);
 [~,idx] = sort(area,"descend");
 sortedAnchors = anchors(idx,:)
```

```
sortedAnchors = 18×2

   224   224
   197   195
   131   224
   206   117
   224    90
   146   134
    99   116
   224    28
    64    67
    59    53
     ⋮
```

```
 anchorBoxes = {sortedAnchors(1:6,:) sortedAnchors(7:12,:)
sortedAnchors(13:18,:)};

 meanIoU
```

```
meanIoU = 0.8288
```

Functions

```
 function data = preprocessData(data,targetSize)
 % Resize the images and scale the pixels to between 0 and 1. Also scale the
 % corresponding bounding boxes.

 for ii = 1:size(data,1)
     I = data{ii,1};
     imgSize = size(I);

     bboxes = data{ii,2};

     I = im2single(imresize(I,targetSize(1:2)));
     scale = targetSize(1:2)./imgSize(1:2);
     bboxes = bboxresize(bboxes,scale);

     data(ii,1:2) = {I,bboxes};
 end
 end
```

When estimating anchor boxes, the number of anchors passed through to the estimateAnchorBoxes function matters. This is because the greater number of anchors can improve the mean IoU (IoU stands for intersection-over-union distance metric). However, using more anchor boxes in an object detector can also increase the computation cost and lead to overfitting, which results in poor detector performance. The IoU distance metric leads to boxes of similar aspect ratios and sizes being clustered together, which results in anchor box estimates that fit the data. In the code above, the mean IoU is calculated to determine how well the anchor boxes overlap with the bounding boxes in the training data. Any value above 0.5 ensures the boxes overlap well.

Determining the ideal number of Anchor boxes

```
maxNumAnchors = 30;
meanIoU = zeros([maxNumAnchors,1]);
anchorBoxes = cell(maxNumAnchors, 1);
for k = 1:maxNumAnchors
    % Estimate anchors and mean IoU.
    [anchorBoxes{k},meanIoU(k)] = estimateAnchorBoxes(trainingData,k);
end

figure
plot(1:maxNumAnchors,meanIoU,'-o')
ylabel("Mean IoU")
xlabel("Number of Anchors")
title("Number of Anchors vs. Mean IoU")
```

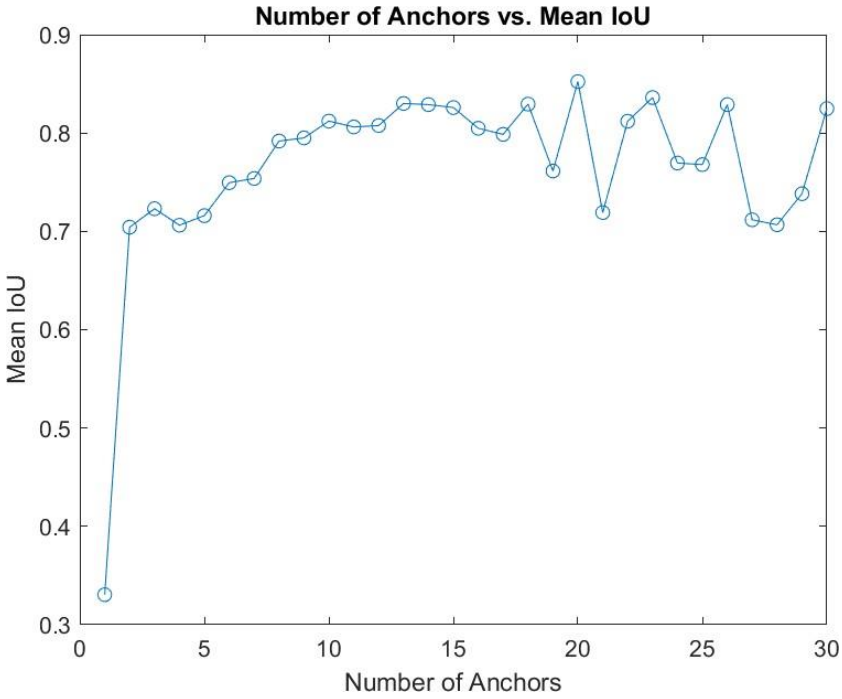


Figure 14) Graph of number of anchors vs mean IoU

The graph was made to visualise a range of values and plot the mean IoU versus the number of anchor boxes to measure the trade-off between number of anchors and mean IoU. Using a two anchor boxes results in a mean IoU value of 0.7 and using more than ten anchor boxes displays a slow gradual increase in mean IoU. Given these results, the next step is to train

and evaluate an object detector using a value of 18. 18 is the final decision as any higher will increase the computational cost, and a smaller value will produce a smaller mean IoU value, which will decrease the accuracy of the anchor boxes.

3.2.5 Creating Custom YOLO v4 Object Detector

Transfer learning was considered the best option to go about creating the network. The YOLO v4 object detector was created using the pretrained network resnet50 as its base network. Through this the specific layers of the base network were replaced and removed to prepare itself to be merged with YOLO v4 architecture. The function yolov4ObjectDetector() is used to create the detector and the following parameters: network, class names, anchor boxes and detection network source are needed to create a full functioning network.

Analysing pretrained network

```
basenet = resnet50;
analyzeNetwork(basenet);
basenet.Layers(1)
```

```
ans =

  ImageInputLayer with properties:


                     Name: 'input_1'

                InputSize: [224 224 3]


   Hyperparameters

          DataAugmentation: 'none'

             Normalization: 'zerocenter'

    NormalizationDimension: 'auto'

                      Mean: [224×224×3 single]
```

Setting the Normalisation property value as 'none' and removing classification layer of the input layer

```
imageSize = basenet.Layers(1).InputSize;
layerName = basenet.Layers(1).Name;
newinputLayer =
imageInputLayer(imageSize,"Normalization","none","Name",layerName);

lgraph = layerGraph(basenet);
lgraph = removeLayers(lgraph,"ClassificationLayer_fc1000");
lgraph = replaceLayer(lgraph,layerName,newinputLayer);
```

```
dlnet = dlnetwork(lgraph);
```

Specify names of the feature extraction layers in the base network to use as detection heads

Three layers used as there are three detection heads in the YOLO v4 network.

```
featureExtractionLayers =
["activation_10_relu","activation_22_relu","activation_40_relu"];
```

Specify the class names and anchor boxes to use for training

```
classes = trainingDataTbl.Properties.VariableNames(2:end);
anchorBoxes = transpose(anchorBoxes);
```

Create a YOLO v4 object detector

```
detector =
yolov4ObjectDetector(dlnet,classes,anchorBoxes,DetectionNetworkSource=feature
ExtractionLayers);

disp(detector)
```

```
  yolov4ObjectDetector with properties:


       Network: [1×1 dlnetwork]
   AnchorBoxes: {3×1 cell}
    ClassNames: {4×1 cell}
     InputSize: [224 224 3]
     ModelName: ''
```

```
analyzeNetwork(detector.Network)
```

3.2.6 Data Augmentation

To improve training accuracy, data augmentation is used. Custom data augmentations are applied to the training data using the transform function. The augmentData function adds the following enhancements to the data it receives: Color jitter enhancement in HSV space, random horizontal flip, and 10% random scaling Additionally, the test and validation data are not subjected to data augmentation. For unbiased review, test and validation data should ideally be reflective of the original data and left unmodified.

```
augmentedTrainingData = transform(trainingData,@augmentData);
```

Functions

```
function data = augmentData(A)
% Apply random horizontal flipping, and random X/Y scaling. Boxes that get
```

```
% scaled outside the bounds are clipped if the overlap is above 0.25. Also,
% jitter image color.

data = cell(size(A));
for ii = 1:size(A,1)
    I = A{ii,1};
    bboxes = A{ii,2};
    labels = A{ii,3};
    sz = size(I);

    if numel(sz) == 3 && sz(3) == 3
        I = jitterColorHSV(I,...
            contrast=0.0,...
            Hue=0.1,...
            Saturation=0.2,...
            Brightness=0.2);
    end

    % Randomly flip image.
    tform = randomAffine2d(XReflection=true,Scale=[1 1.1]);
    rout = affineOutputView(sz,tform,BoundsStyle="centerOutput");
    I = imwarp(I,tform,OutputView=rout);

    % Apply same transform to boxes.
    [bboxes,indices] = bboxwarp(bboxes,tform,rout,OverlapThreshold=0.25);
    labels = labels(indices);

    % Return original data only when all boxes are removed by warping.
    if isempty(indices)
        data(ii,:) = A(ii,:);
    else
        data(ii,:) = {I,bboxes,labels};
    end
end
```

3.2.7 Train Object Detector

The options variable was established to store the training choices and their specified values to train the network. To improve the network's accuracy, the training options can be changed. Then, to train the network, the function trainYOLOv4ObjectDetector was called and filled with the appropriate parameters. The training progress was outputted into the variable "info" while training the network to track if the training was going properly or not.

Training the Network

```
options = trainingOptions("adam",...
    GradientDecayFactor=0.9,...
    SquaredGradientDecayFactor=0.999,...
    InitialLearnRate=0.001,...
    LearnRateSchedule="none",...
```

```
    MiniBatchSize=4,...
    L2Regularization=0.0005,...
    MaxEpochs=70,...
    BatchNormalizationStatistics="moving",...
    DispatchInBackground=true,...
    ResetInputNormalization=false,...
    Shuffle="every-epoch",...
    VerboseFrequency=20,...
    CheckpointPath=tempdir,...
    ValidationData=validationData);

 [trainingDetector,info] =
trainYOLOv4ObjectDetector(augmentedTrainingData,detector,options);
```

4 Analysis and Results

4.1 Training progress

```
    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss    ValidationLoss
    _____    _________    ___________    _________    ____________    ______________
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
      1          20        00:02:35        0.001         2796.2
      1          40        00:03:00        0.001         965.34
      1          60        00:03:43        0.001         577.41
      1          80        00:04:10        0.001         309.34
      1         100        00:04:39        0.001         223.74          7.2114
      1         120        00:05:21        0.001         185.48
      2         140        00:07:16        0.001         149.67
      2         160        00:07:54        0.001         111.98
      2         180        00:08:20        0.001         93.467
      2         200        00:08:48        0.001         92.981          2.6487
      2         220        00:09:28        0.001         67.238
      2         240        00:09:53        0.001         69.465
      3         260        00:12:09        0.001         57.962
      3         280        00:12:42        0.001         58.428
```

Figure 15) Top results of the output of the training results

```
     68        8400        05:40:47        0.001        0.34114         0.040164
     68        8420        05:41:32        0.001        0.098022
     69        8440        05:43:23        0.001        0.28226
     69        8460        05:44:03        0.001         0.5683
     69        8480        05:44:30        0.001        0.84813
     69        8500        05:44:59        0.001        0.41292         0.038197
     69        8520        05:45:41        0.001        0.86225
     69        8540        05:46:10        0.001        0.23892
     70        8560        05:48:05        0.001        0.15426
     70        8580        05:48:33        0.001         2.2088
     70        8600        05:48:56        0.001         1.3434         0.032705
     70        8620        05:49:40        0.001         2.3703
     70        8640        05:50:10        0.001         1.3292
     70        8660        05:50:51        0.001         1.8895
     70        8680        05:51:17        0.001          1.299         0.031385

************************************************************************
Detector training complete.
************************************************************************
```

Figure 16) Bottom results of the output of the training results (Full results in Appendix B)

When training a network, you need to keep an eye on the training loss and validation loss. Good signs to look for is the training loss and the validation loss being less than one. The network took 5 hours and 51 minutes to complete, and the final training loss and validation loss was 1.299 and 0.031385 respectively.

4.2 Evaluating the Network

We analyse the data using the function "detect" with the trained network and test data to better understand the training results. The function returns a table with the bounding boxes, detection score, and label after it has been run. The precision accuracy of the data is then calculated, which determines how exact the trained network is. This is accomplished by using the "evaluateDetectionPrecision" function, which returns three variables: recall, precision, and ap.Evaluating the detector using test dataset

Run the detector on all the test images.

```
load trainingDetector.mat

detectionResults = detect(trainingDetector,testData);
```

Evaluate the object detector using average precision metric.

```
 [ap,recall,precision] =
evaluateDetectionPrecision(detectionResults,testData);
```

Calculating mean Average Precision (AP)

```
 ap
```

```
ap = 4×1

    0.2708
    0.8274
    0.1848
    0.2991
```

```
 mean(ap)
```

```
ans = 0.3955
```

The average precision was calculated from the variable ap, which came to 0.3955. Variable ap contained 3 values less than 0.3 and only one greater than 0.8

Creating PR curve plot

The precision/recall (PR) curve highlights how precise a detector is at varying levels of recall. The ideal precision is 1 at all recall levels. The use of more data can help improve the average precision but might require more training time.

```
 figure
 plot(recall,precision)
```

```
Error using plot
Not enough input arguments.
```

```
 xlabel("Recall")
 ylabel("Precision")
 grid on
 title(sprintf("Average Precision = %.2f",ap))
```

No.	Boxes	Scores	Label
1	[]	[]	[]
2	[]	[]	[]
3	[26.186035,49.721802,1721.9805,1674.2135]	0.98115045	1x1 categorical
4	[801.79590,54.395935,230.26953,1666.3494]	0.83285952	1x1 categorical
5	[8.4257202,42.802185,1756.4504,1687.9500]	0.99753439	1x1 categorical
6	[601.13342,390.31488,377.09790,381.24127]	0.56982297	1x1 categorical
7	[820.08014,1013.3676,719.09320,415.01727]	0.98195606	1x1 categorical
8	[]	[]	[]
9	[444.41803,459.84891,694.80194,420.93466]	0.97853196	1x1 categorical
10	[474.84741,587.54407,364.94189,384.44971]	0.70719683	1x1 categorical

11	[]	[]	[]
12	[]	[]	[]
13	[]	[]	[]

Table 1) First 13 results of the detectionResults (Full results in Appendix A)

After running the plot function, there was an error message that declared there wasn't enough input arguments, which led to analysing the detectionResults variable

Table 1 is a subset of the overall table of detectionResults. The majority of the table is made up of rows of data with or without blank spaces (rows with squared brackets). The blank areas represent photos that have not been discovered, but the other rows do not.

5 Discussion

5.1 Interpretation of Results

5.1.1 Training Progress

The results of training the network as shown in figure 15 and 16, suggests that the training went well. The final training loss and validation loss was 1.299 and 0.031385, which showed good signs overfitting didn't occur and the detection accuracy of the network is well.

5.1.2 Error message in using plot function

In section 4.2, when trying to plot a PR curve, the output returned and error. The error mentioned a lack on input arguments. To further understand the meaning of this error, table 1 was investigated. As shown in the table, there are blank spaces. Which represented undetected images. Therefore, it was decided to analyse the images that weren't detected by running the code below. The function of this code is to use the trained object detector on the undetected and detected images. The variable n was changed to retrieve 5 different samples.

```
n = 1
img = readimage(imdsTest,n);

[bboxes,scores,labels] = detect(trainingDetector,img);

detectedImg = insertObjectAnnotation(img,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```

Figure 17) Undetected image where n = 1

Figure 18) Undetected image where n = 231

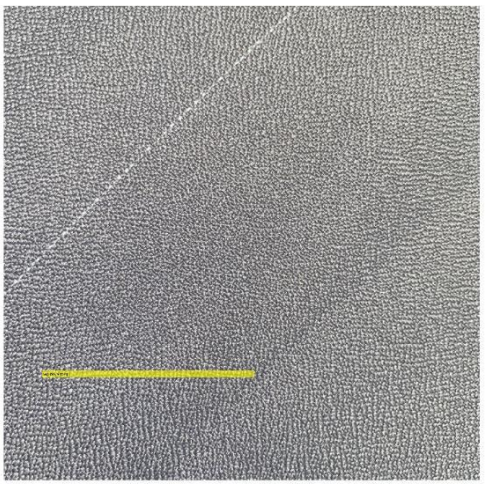


Figure 19) Detected fault where n = 210

Figure 20) Detected fault where n = 37

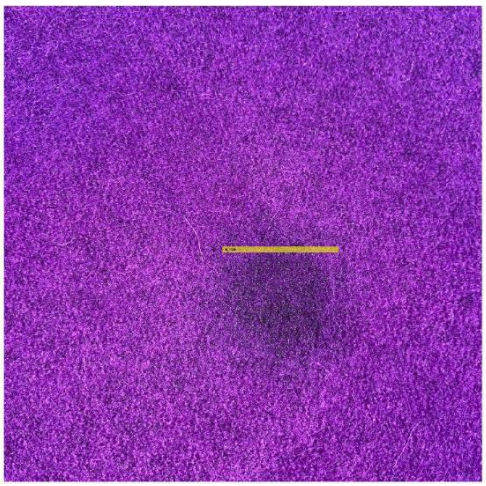


Figure 21) Detected fault where n = 150

Figures 17 and 18 show the photos that were not identified and are stored in Table 1. Figures 17 and 18 are row 1 and 231 of the detectionResults table, respectively. The rows 1 and 231 in the table are blank, however the rows 210, 37, and 150 where the pictures of figure 19, 20, and 21 are stored, are filled with data (these photographs are fabrics with faults). This indicates that the network has difficulty detecting fabrics that are free of flaws. This topic will be discussed in further depth in Section 5.1.3.

5.1.3 Fault in dataset

Since the network can't detect fabrics with no defects, this suggests there is a problem with the bounding boxes for these images. Therefore, a future change would be to use the Image Labeler application in MATLAB to relabel the images with no defects. This would mean that the current datasets made would have to be updated as well as the trained object detector.

5.1.4 Accuracy in detecting faults

Despite the fact that the AP of the tested data was 0.3955 (39.55 percent precise), the object detectors' inability to recognise fabrics with no flaw is a representation of the object detectors' inability to detect fabrics with no defect. The photos with no faults are the main cause of this problem, as 479 of the 903 images in the dataset had no defects. Furthermore, the fabrics with no faults are represented by the blank data in the detectionResults table. Because the project's goal is to detect fabric problems, it was decided to test the accuracy of detecting fabrics with defects rather than those without. This is due to the significant influence of non-defected photos, which resulted in findings that did not accurately reflect the object detector's accuracy.

From the detectionResults table, the average of the score's column was taken from rows that included data (these rows representing fabrics with defects). The outcome was a score of 0.882 on average (3 d.p.). This score indicates that the object detector has a confidence level of 88.2 percent in detecting faults.

Referring back to sections 2.1.1 and 2.1.2 of the literature review, the accuracy for the object detectors DefectNet and YOLO-LFD are 97.7% and 97.2% respectively. However, you should note that these datasets are trained on 2000-3000 images whereas this project uses 903. The accuracy of this project is around 9-10 percent less, suggesting there is more room for improvement.

5.2 Limitations

The object detector struggles with detecting fabrics with the no defects. This can be worked on by altering the labelled bonding boxes for those set of images. In addition, another limit is its small range of defects it can detect. The object detector can only recognise up to 3 different defects which limits itself to have multiple uses in the industry.

6 Conclusions

6.1 Conclusions for the Project

The project overall was a success in some ways. Such as having an 88.2 percent accuracy for detecting defects. However, that's where the object detector falls short. This is due to the fact that the project fails to recognise fabrics with no defects, resulting in poor precision. The project, however, detects these defects well but is only limited to three kinds. This is a fairly small amount but is a good start for a prototype. Therefore, relating back to the aims and objectives, which are: create a database of images of the fabrics without and with faults, train a network using the database of images, validate results of the program by testing the network. The project successfully met the last two listed objectives and partially met the first.

6.2 Suggestions for Further Work

The object detector has several limits that can be worked on for further improvement. One of them being the size of the dataset is used to train the network. The dataset currently contains 903 images with 3 types of defect but ideally 3000 with around 7 types of defects would be better, as a larger dataset would increase the accuracy of the detector.

Secondly, the base network of the YOLO v4 detector could be changed to a better more in depth pretrained network. This would mean the object detector could analyse the features of the fabrics with finer detail, resulting in more accurate readings.

Lastly, the non-defected images should be labelled properly so that the object detector can recognise those images which will result in a higher average precision and accuracy of the object detector.

7 References

[1] A. Kumar, "Computer-Vision-Based Fabric Defect Detection: A Survey," *IEEE Transactions on Industrial Electronics,* vol. 55, no. 1, pp. 348-363, 2008.

[2] S. Lim, "23 FABRIC DEFECTS TO LOOK OUT FOR DURING FABRIC INSPECTION," Asia Quality Focus, 23 October 2018. [Online]. Available: https://www.intouch-quality.com/blog/5-common-fabric-defects-prevent.

[3] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang and S. Tang, "Using Deep Learning to Detect Defects in Manufacturing: A Comprehensive Survey and Current Challenges," *Materials,* vol. 13, p. 5755, 2020.

[4] A. Rasheed, B. Zafar, A. Rasheed, N. Ali, M. Sajid, S. H. Dar, U. Habib, T. Shehryar and M. T. Mahmood, "Fabric Defect Detection Using Computer Vision Techniques: A Comprehensive Review," *Mathematical Problems in Engineering,* 2020.

[5] W. Wong and J. Jiang, "Computer vision techniques for detecting fabric defects," *Textile Institute Book Series,* pp. 47-60, 2018.

[6] Z. Liu, J. Cui, C. Li, S. Ding and Q. Xu, "Real-time Fabric Defect Detection based on Lightweight Convolutional Neural Network," *Proceedings of the 2019 8th International Conference on Computing and Pattern Recognition,* pp. 122-127, 2019.

[7] R. Orac, "What's new in YOLOv4?," Towards Data Science, 19 May 2020. [Online]. Available: https://towardsdatascience.com/whats-new-in-yolov4-323364bb3ad3. [Accessed 2022].

[8] "Getting Started with YOLO v4," The MathWorks, 2022. [Online]. Available: https://uk.mathworks.com/help/vision/ug/getting-started-with-yolo-v4.html.

Appendices

Appendix A – Detection Results Table

No.	Boxes	Scores	Label
1	[]	[]	[]
2	[]	[]	[]
3	[26.186035,49.721802,1721.9805,1674.2135]	0.98115045	1x1 categorical
4	[801.79590,54.395935,230.26953,1666.3494]	0.83285952	1x1 categorical
5	[8.4257202,42.802185,1756.4504,1687.9500]	0.99753439	1x1 categorical
6	[601.13342,390.31488,377.09790,381.24127]	0.56982297	1x1 categorical
7	[820.08014,1013.3676,719.09320,415.01727]	0.98195606	1x1 categorical
8	[]	[]	[]
9	[444.41803,459.84891,694.80194,420.93466]	0.97853196	1x1 categorical
10	[474.84741,587.54407,364.94189,384.44971]	0.70719683	1x1 categorical
11	[]	[]	[]
12	[]	[]	[]
13	[]	[]	[]
14	[479.82458,445.36838,756.80139,443.95694]	0.97636521	1x1 categorical
15	[506.78021,393.53955,713.35406,408.89490]	0.99447614	1x1 categorical
16	[3.1870728,29.616577,1767.0042,1714.4371]	0.99812502	1x1 categorical
17	[]	[]	[]
18	[]	[]	[]
19	[]	[]	[]
20	[18.274841,41.745850,1737.4587,1690.7593]	0.99263877	1x1 categorical

21	[345.31836,368.33304,580.57397,882.62280;442.97937,347.26215,893.74072,1192.7844]	[0.57558066;0.89464736]	2x1 categorical
22	[]	[]	[]
23	[]	[]	[]
24	[853.49756,628.83893,159.81165,244.42859]	0.75125587	1x1 categorical
25	[583.90869,133.02335,609.19873,884.99536;1,9.1092224,627.78357,534.27771]	[0.79330873;0.60826701]	2x1 categorical
26	[8.6921997,41.016907,1756.0920,1691.6670]	0.99717081	1x1 categorical
27	[621.20593,394.98981,448.51868,374.19128]	0.53863984	1x1 categorical
28	[376.79431,328.91330,712.61169,404.62393]	0.94823325	1x1 categorical
29	[]	[]	[]
30	[457.20459,575.61871,665.29126,439.04785]	0.79922938	1x1 categorical
31	[712.94904,504.33344,412.53961,428.77484]	0.6527018	1x1 categorical
32	[]	[]	[]
33	[]	[]	[]
34	[27.674622,44.635193,1719.9673,1684.6580]	0.94034278	1x1 categorical
35	[1095.7252,25.587952,230.08350,1722.7466]	0.93995696	1x1 categorical
36	[]	[]	[]
37	[855.20856,836.28479,144.54333,142.88068]	0.73070604	1x1 categorical
38	[2.3231812,33.699524,1768.4941,1706.0193]	0.9985922	1x1 categorical
39	[]	[]	[]
40	[]	[]	[]
41	[200.85324,35.162659,240.12717,1703.7473]	0.90575469	1x1 categorical
42	[263.47141,958.59406,686.00366,434.18524]	0.97744328	1x1 categorical

43	[]	[]	[]
44	[]	[]	[]
45	[552.62561,467.51770,619.14807,412.42041]	0.81985861	1x1 categoric al
46	[]	[]	[]
47	[]	[]	[]
48	[]	[]	[]
49	[497.55365,457.18140,714.15240,405.75391]	0.90166599	1x1 categoric al
50	[996.15491,744.92682,621.65417,435.86163]	0.92305923	1x1 categoric al
51	[]	[]	[]
52	[]	[]	[]
53	[511.87286,531.17181,402.37140,352.84143]	0.95445281	1x1 categoric al
54	[]	[]	[]
55	[577.45569,126.97394,622.12610,887.24512;1,11.185364,616.74048,527.61591]	[0.71759415;0.71041721]	2x1 categoric al
56	[]	[]	[]
57	[12.681152,22.713379,1747.9075,1728.2568]	0.99864191	1x1 categoric al
58	[1.7979126,27.829834,1770.2021,1718.2759]	0.98251045	1x1 categoric al
59	[506.37646,194.53581,720.43164,428.98920]	0.99914014	1x1 categoric al
60	[]	[]	[]
61	[579.31165,515.54077,680.74207,402.92407]	0.80359924	1x1 categoric al
62	[]	[]	[]
63	[637.23096,393.73492,700.58301,405.96185]	0.95857781	1x1 categoric al
64	[]	[]	[]
65	[722.16650,501.40714,412.77319,455.16782]	0.57704985	1x1 categoric al
66	[]	[]	[]
67	[11.756348,31.553711,1750.1204,1710.9238]	0.99660003	1x1 categoric al

68	[123.61496,455.46292,718.98792,438.65402]	0.99581909	1x1 categoric al
69	[]	[]	[]
70	[]	[]	[]
71	[]	[]	[]
72	[]	[]	[]
73	[582.34198,642.81055,684.00739,398.32764]	0.92974466	1x1 categoric al
74	[]	[]	[]
75	[627.00500,341.13904,708.43677,388.33130]	0.84780651	1x1 categoric al
76	[]	[]	[]
77	[]	[]	[]
78	[]	[]	[]
79	[]	[]	[]
80	[]	[]	[]
81	[437.01065,319.05481,710.47571,413.93298]	0.95778006	1x1 categoric al
82	[456.01904,990.44446,385.16748,335.89709]	0.71643895	1x1 categoric al
83	[3.2662354,29.406189,1767.2079,1715.0430]	0.99020952	1x1 categoric al
84	[8.7711792,23.964905,1755.9448,1725.9204]	0.9980498	1x1 categoric al
85	[878.01270,467.63889,432.34741,378.65311]	0.6995005	1x1 categoric al
86	[515.88287,398.15027,685.36444,412.57251]	0.95900697	1x1 categoric al
87	[1.3796387,38.294922,1770.5438,1696.9902]	0.99639893	1x1 categoric al
88	[]	[]	[]
89	[]	[]	[]
90	[]	[]	[]
91	[]	[]	[]
92	[552.41571,503.42743,733.76300,415.63849]	0.72611111	1x1 categoric al
93	[]	[]	[]

94	[857.03418,788.40460,107.85590,157.58771]	0.52690196	1x1 categoric al
95	[495.43018,41.928833,237.07861,1691.9382]	0.88704711	1x1 categoric al
96	[]	[]	[]
97	[369.61810,581.46460,453.20856,391.36755]	0.588085	1x1 categoric al
98	[552.93311,510.13000,742.73242,401.13525]	0.97412646	1x1 categoric al
99	[]	[]	[]
100	[594.08972,318.70343,663.58777,431.63403]	0.98984033	1x1 categoric al
101	[540.38977,469.33548,632.32239,421.19467]	0.82778412	1x1 categoric al
102	[]	[]	[]
103	[]	[]	[]
104	[]	[]	[]
105	[]	[]	[]
106	[]	[]	[]
107	[15.147766,30.938599,1743.4009,1712.0763]	0.99735248	1x1 categoric al
108	[532.84778,785.58502,648.86401,369.93964]	0.67865247	1x1 categoric al
109	[]	[]	[]
110	[]	[]	[]
111	[469.13498,388.55862,650.59949,429.05588]	0.94775283	1x1 categoric al
112	[499.99435,274.23392,712.80151,381.47702]	0.89091986	1x1 categoric al
113	[]	[]	[]
114	[]	[]	[]

115	[768.55072,964.59589,694.94537,398.75812]	0.98782551	1x1 categorical
116	[901.14758,460.77295,679.84094,405.21814]	0.98953825	1x1 categorical
117	[1,26.974731,1771,1719.4534]	0.99935549	1x1 categorical
118	[486.97897,889.66315,615.08850,337.22516]	0.5353322	1x1 categorical
119	[598.38855,666.39807,1173.6115,1105.6019]	0.66061598	1x1 categorical
120	[]	[]	[]
121	[25.101196,17.790710,1725.5052,1740.7026]	0.98124737	1x1 categorical
122	[]	[]	[]
123	[]	[]	[]
124	[911.87787,769.24316,374.96234,387.33044]	0.96415102	1x1 categorical
125	[36.208923,45.377808,1704.4626,1683.2765]	0.75230348	1x1 categorical
126	[8.4993286,34.816772,1756.8950,1704.3644]	0.98878413	1x1 categorical
127	[]	[]	[]
128	[16.305481,26.866333,1740.6487,1719.8708]	0.9988789	1x1 categorical
129	[]	[]	[]
130	[]	[]	[]
131	[414.73032,467.49731,637.83276,386.72754]	0.92907584	1x1 categorical
132	[]	[]	[]
133	[]	[]	[]
134	[]	[]	[]

135	[610.06909,660.72974,201.36523,179.22498]	0.66892117	1x1 categorical
136	[]	[]	[]
137	[785.61578,341.72064,374.36285,357.87274]	0.93419403	1x1 categorical
138	[609.66272,461.24945,744.19421,407.00787]	0.90542734	1x1 categorical
139	[]	[]	[]
140	[]	[]	[]
141	[12.311768,19.741577,1749.4459,1735.1692]	0.99564898	1x1 categorical
142	[481.56354,467.26340,451.52740,392.28738]	0.90664989	1x1 categorical
143	[]	[]	[]
144	[]	[]	[]
145	[]	[]	[]
146	[16.587341,38.748352,1740.2295,1696.2710]	0.99782938	1x1 categorical
147	[496.66113,1046.1105,425.89832,342.74036]	0.76656038	1x1 categorical
148	[]	[]	[]
149	[]	[]	[]
150	[809.62781,918.69342,431.78735,334.87628]	0.8805353	1x1 categorical
151	[]	[]	[]
152	[333.45486,331.60248,782.17896,439.61725;444.65424,452.51596,707.92462,421.59854]	[0.61502582;0.98992831]	2x1 categorical
153	[]	[]	[]
154	[11.833313,29.133240,1749.7549,1715.5476]	0.99756277	1x1 categorical

155	[642.45648,200.26031,687.63214,417.29645]	0.98630685	1x1 categorical
156	[]	[]	[]
157	[1.9728394,37.430237,1769.4387,1699.0278]	0.99276376	1x1 categorical
158	[]	[]	[]
159	[]	[]	[]
160	[]	[]	[]
161	[868.26892,184.02437,725.90613,438.78857]	0.62399411	1x1 categorical
162	[77.454437,11.581360,231.08221,1751.3037]	0.71767592	1x1 categorical
163	[]	[]	[]
164	[]	[]	[]
165	[407.67261,1035.9596,374.56934,370.37903]	0.72075814	1x1 categorical
166	[1.7778931,20.060303,1769.9465,1733.9979]	0.99637204	1x1 categorical
167	[42.500854,50.111511,1689.2621,1674.0244]	0.99447125	1x1 categorical
168	[]	[]	[]
169	[2.5169678,25.391602,1768.3602,1722.8732]	0.99893695	1x1 categorical
170	[4.6848145,38.212708,1763.9806,1697.1401]	0.99613094	1x1 categorical
171	[3.8959351,30.028564,1765.4248,1713.4351]	0.99859661	1x1 categorical
172	[]	[]	[]
173	[]	[]	[]
174	[]	[]	[]

175	[18.437561,35.908813,1736.8623,1702.2793]	0.99646509	1x1 categorical
176	[]	[]	[]
177	[507.50836,1,761.72406,979.48145]	0.61022419	1x1 categorical
178	[16.690125,35.021667,1740.0518,1703.7292]	0.99850529	1x1 categorical
179	[]	[]	[]
180	[1286.0469,39.088074,225.02209,1695.8203]	0.93608999	1x1 categorical
181	[774.02100,338.36285,398.16125,376.93744]	0.74503458	1x1 categorical
182	[781.04816,206.82938,663.23358,405.13229]	0.96950889	1x1 categorical
183	[]	[]	[]
184	[]	[]	[]
185	[460.90027,49.807617,228.91309,1674.4969]	0.8732633	1x1 categorical
186	[]	[]	[]
187	[]	[]	[]
188	[]	[]	[]
189	[266.99472,607.30927,670.88257,359.27094]	0.72662354	1x1 categorical
190	[]	[]	[]
191	[54.172729,68.141541,1666.9003,1638.1292]	0.9884364	1x1 categorical
192	[]	[]	[]
193	[]	[]	[]
194	[]	[]	[]
195	[]	[]	[]

196	[]	[]	[]
197	[]	[]	[]
198	[]	[]	[]
199	[]	[]	[]
200	[]	[]	[]
201	[]	[]	[]
202	[]	[]	[]
203	[]	[]	[]
204	[]	[]	[]
205	[]	[]	[]
206	[]	[]	[]
207	[852.07495,572.87952,372.05933,399.75525]	0.84214658	1x1 categorical
208	[7.9414673,29.124390,1757.7041,1715.7113]	0.99799603	1x1 categorical
209	[]	[]	[]
210	[142.59134,1,791.12830,1364.5931]	0.64008284	1x1 categorical
211	[]	[]	[]
212	[]	[]	[]
213	[]	[]	[]
214	[5.8468628,29.830994,1761.6401,1714.0190]	0.99859875	1x1 categorical
215	[629.51318,441.54138,708.62354,462.54114]	0.94521981	1x1 categorical
216	[8.1690063,27.012329,1757.0232,1719.7007]	0.99837172	1x1 categorical
217	[]	[]	[]

218	[]	[]	[]
219	[]	[]	[]
220	[649.03204,265.45013,668.74042,396.35638]	0.52597117	1x1 categorical
221	[7.8793945,19.754639,1759.7241,1735.2759]	0.90751791	1x1 categorical
222	[]	[]	[]
223	[]	[]	[]
224	[]	[]	[]
225	[]	[]	[]
226	[]	[]	[]
227	[]	[]	[]
228	[764.20422,961.81677,701.12415,422.51477]	0.98928183	1x1 categorical
229	[983.65527,672.84680,352.29443,319.49512]	0.84716213	1x1 categorical
230	[684.35590,701.66895,428.74725,404.17224]	0.54491919	1x1 categorical
231	[]	[]	[]
232	[]	[]	[]
233	[224.65237,579.28625,745.91028,439.41034]	0.88390654	1x1 categorical
234	[310.03497,372.00488,714.37738,436.26282]	0.96816325	1x1 categorical
235	[]	[]	[]
236	[]	[]	[]
237	[]	[]	[]
238	[]	[]	[]

239	[430.06161,334.65100,734.22168,409.48499]	0.99710798	1x1 categorical
240	[]	[]	[]
241	[358.96832,328.68341,746.28143,426.60223]	0.9650026	1x1 categorical
242	[19.779602,35.206726,1736.4756,1704.6226]	0.98639596	1x1 categorical
243	[]	[]	[]
244	[]	[]	[]
245	[]	[]	[]
246	[]	[]	[]
247	[737.99133,818.57031,139.15161,114.34589]	0.89105117	1x1 categorical
248	[]	[]	[]
249	[]	[]	[]
250	[]	[]	[]
251	[76.736244,57.957153,237.61740,1658.3593]	0.85065323	1x1 categorical
252	[1093.0356,21.461853,228.17883,1730.9949]	0.93558073	1x1 categorical
253	[18.006958,43.282227,1738.0217,1687.0247]	0.97328138	1x1 categorical
254	[]	[]	[]
255	[13.937744,45.947083,1745.8917,1682.1401]	0.99363923	1x1 categorical
256	[]	[]	[]
257	[125.51056,364.37976,1026.2793,1161.3314]	0.78390402	1x1 categorical
258	[7.4281616,24.858215,1759.0869,1723.9219]	0.85150111	1x1 categorical

259	[]	[]	[]
260	[]	[]	[]
261	[]	[]	[]
262	[]	[]	[]
263	[17.290527,41.519836,1740.0968,1691.7632]	0.99472773	1x1 categorical
264	[]	[]	[]
265	[2.0087280,36.543091,1769.8628,1701.0592]	0.99235588	1x1 categorical
266	[]	[]	[]
267	[]	[]	[]
268	[]	[]	[]
269	[63.350128,445.22394,811.83447,1088.7761]	0.74316752	1x1 categorical
270	[535.41888,847.44952,645.16071,368.26263]	0.76279557	1x1 categorical

Appendix B – Training Progress output

****

Training a YOLO v4 Object Detector for the following object classes:

* Hole
* Oil_Stain
* Mutliple_Netting
* No_Defect

Epoch	Iteration	TimeElapsed	LearnRate	TrainingLoss	ValidationLoss
Starting parallel pool (parpool) using the 'local' profile ...					
Connected to the parallel pool (number of workers: 6).					
1	20	00:02:35	0.001	2796.2	
1	40	00:03:00	0.001	965.34	
1	60	00:03:43	0.001	577.41	
1	80	00:04:10	0.001	309.34	
1	100	00:04:39	0.001	223.74	7.2114
1	120	00:05:21	0.001	185.48	
2	140	00:07:16	0.001	149.67	
2	160	00:07:54	0.001	111.98	
2	180	00:08:20	0.001	93.467	
2	200	00:08:48	0.001	92.981	2.6487
2	220	00:09:28	0.001	67.238	
2	240	00:09:53	0.001	69.465	
3	260	00:12:09	0.001	57.962	
3	280	00:12:42	0.001	58.428	
3	300	00:13:21	0.001	59.327	1.4738
3	320	00:14:14	0.001	50.373	
3	340	00:14:52	0.001	42.507	
3	360	00:15:44	0.001	42.996	
4	380	00:17:51	0.001	30.459	
4	400	00:18:25	0.001	43.225	1.0005
4	420	00:19:15	0.001	32.91	
4	440	00:19:52	0.001	23.53	
4	460	00:20:45	0.001	26.673	
4	480	00:21:21	0.001	31.478	
5	500	00:23:23	0.001	24.471	0.71365
5	520	00:24:11	0.001	23.948	
5	540	00:24:41	0.001	22.74	
5	560	00:25:27	0.001	26.748	
5	580	00:26:03	0.001	20.368	
5	600	00:26:40	0.001	18.004	0.53732
5	620	00:27:27	0.001	21.504	0.51445

6	640	00:29:31	0.001	14.12	
6	660	00:30:11	0.001	16.898	
6	680	00:30:43	0.001	13.424	
6	700	00:31:14	0.001	15.288	0.4688
6	720	00:31:59	0.001	17.114	
6	740	00:32:26	0.001	14.409	
7	760	00:34:38	0.001	13.385	
7	780	00:35:04	0.001	13.955	
7	800	00:35:40	0.001	10.068	0.37643
7	820	00:36:22	0.001	16.24	
7	840	00:36:53	0.001	13.771	
7	860	00:37:34	0.001	13.47	
8	880	00:39:31	0.001	13.974	
8	900	00:39:59	0.001	11.509	0.3129
8	920	00:40:42	0.001	19.184	
8	940	00:41:12	0.001	11.333	
8	960	00:41:58	0.001	9.7962	
8	980	00:42:28	0.001	11.13	
9	1000	00:44:24	0.001	6.1459	0.29286
9	1020	00:45:06	0.001	7.9323	
9	1040	00:45:35	0.001	12.198	
9	1060	00:46:19	0.001	11.936	
9	1080	00:46:50	0.001	12.695	
9	1100	00:47:21	0.001	10.181	0.28402
10	1120	00:49:32	0.001	9.0113	
10	1140	00:50:05	0.001	7.911	
10	1160	00:50:44	0.001	8.5983	
10	1180	00:51:21	0.001	8.9733	
10	1200	00:51:54	0.001	8.9597	0.26821
10	1220	00:52:43	0.001	7.6647	
10	1240	00:53:12	0.001	11.016	0.23236
11	1260	00:55:29	0.001	10.269	
11	1280	00:55:56	0.001	12.658	
11	1300	00:56:34	0.001	9.3908	0.22135
11	1320	00:57:22	0.001	6.9296	
11	1340	00:57:56	0.001	5.6588	
11	1360	00:58:36	0.001	8.6257	
12	1380	01:00:39	0.001	6.0103	
12	1400	01:01:07	0.001	8.6104	0.20606
12	1420	01:01:57	0.001	5.3188	
12	1440	01:02:29	0.001	5.7503	
12	1460	01:03:17	0.001	3.8932	
12	1480	01:03:47	0.001	7.3943	
13	1500	01:05:48	0.001	7.9514	0.16302
13	1520	01:06:32	0.001	4.0996	
13	1540	01:07:06	0.001	9.5569	

13	1560	01:07:52	0.001	7.0839	
13	1580	01:08:27	0.001	5.4509	
13	1600	01:09:01	0.001	3.5601	0.19201
14	1620	01:11:15	0.001	5.7035	
14	1640	01:11:46	0.001	3.1829	
14	1660	01:12:31	0.001	9.8253	
14	1680	01:13:04	0.001	3.8945	
14	1700	01:13:38	0.001	4.7706	0.13749
14	1720	01:14:26	0.001	9.2571	
15	1740	01:16:23	0.001	6.5155	
15	1760	01:17:10	0.001	8.4638	
15	1780	01:17:40	0.001	5.2152	
15	1800	01:18:14	0.001	4.2266	0.12823
15	1820	01:19:04	0.001	6.0438	
15	1840	01:19:39	0.001	2.4243	
15	1860	01:20:23	0.001	11.307	0.13713
16	1880	01:22:22	0.001	5.3295	
16	1900	01:22:49	0.001	7.2934	0.14334
16	1920	01:23:41	0.001	3.0575	
16	1940	01:24:14	0.001	3.3403	
16	1960	01:25:02	0.001	5.1823	
16	1980	01:25:32	0.001	5.0287	
17	2000	01:27:32	0.001	3.2505	0.13476
17	2020	01:28:14	0.001	5.0486	
17	2040	01:28:51	0.001	5.3291	
17	2060	01:29:36	0.001	3.283	
17	2080	01:30:09	0.001	3.9639	
17	2100	01:30:39	0.001	6.1463	0.11433
18	2120	01:32:55	0.001	1.6398	
18	2140	01:33:26	0.001	7.023	
18	2160	01:34:12	0.001	6.5045	
18	2180	01:34:46	0.001	3.0663	
18	2200	01:35:21	0.001	4.8578	0.084665
18	2220	01:36:08	0.001	5.5383	
19	2240	01:38:09	0.001	5.8226	
19	2260	01:38:54	0.001	3.6918	
19	2280	01:39:28	0.001	3.608	
19	2300	01:40:03	0.001	5.514	0.13598
19	2320	01:40:51	0.001	2.3887	
19	2340	01:41:26	0.001	4.0416	
20	2360	01:43:35	0.001	3.7582	
20	2380	01:44:06	0.001	3.6397	
20	2400	01:44:30	0.001	5.8924	0.085909
20	2420	01:45:19	0.001	4.5353	
20	2440	01:45:52	0.001	6.946	
20	2460	01:46:38	0.001	1.5944	

20	2480	01:47:06	0.001	1.9647	0.054284
21	2500	01:49:01	0.001	5.2409	0.089354
21	2520	01:49:40	0.001	4.479	
21	2540	01:50:17	0.001	2.0022	
21	2560	01:51:02	0.001	3.1641	
21	2580	01:51:35	0.001	9.1748	
21	2600	01:52:02	0.001	6.2021	0.103
22	2620	01:54:14	0.001	1.7063	
22	2640	01:54:42	0.001	5.1243	
22	2660	01:55:26	0.001	3.5532	
22	2680	01:55:59	0.001	2.809	
22	2700	01:56:32	0.001	2.857	0.055033
22	2720	01:57:15	0.001	4.5426	
23	2740	01:59:12	0.001	4.8561	
23	2760	01:59:55	0.001	1.769	
23	2780	02:00:27	0.001	4.3422	
23	2800	02:00:57	0.001	1.5476	0.049049
23	2820	02:01:39	0.001	1.8321	
23	2840	02:02:07	0.001	2.986	
24	2860	02:04:06	0.001	3.8502	
24	2880	02:04:33	0.001	2.1099	
24	2900	02:05:01	0.001	1.744	0.072294
24	2920	02:05:41	0.001	1.1241	
24	2940	02:06:11	0.001	2.2648	
24	2960	02:06:54	0.001	0.91551	
25	2980	02:08:41	0.001	3.8915	
25	3000	02:09:10	0.001	7.0983	0.05777
25	3020	02:09:50	0.001	1.5651	
25	3040	02:10:20	0.001	3.1061	
25	3060	02:11:03	0.001	6.1286	
25	3080	02:11:33	0.001	2.0717	
25	3100	02:11:59	0.001	1.4486	0.043184
26	3120	02:13:48	0.001	3.9758	
26	3140	02:14:11	0.001	2.7436	
26	3160	02:14:54	0.001	3.6769	
26	3180	02:15:24	0.001	1.8636	
26	3200	02:15:54	0.001	1.0897	0.060495
26	3220	02:16:30	0.001	5.0981	
27	3240	02:18:22	0.001	4.1731	
27	3260	02:18:58	0.001	2.1865	
27	3280	02:19:30	0.001	3.1316	
27	3300	02:19:56	0.001	0.49426	0.049187
27	3320	02:20:39	0.001	5.1299	
27	3340	02:21:07	0.001	2.1392	
28	3360	02:23:10	0.001	1.6745	
28	3380	02:23:38	0.001	0.69046	

28	3400	02:24:09	0.001	3.7966	0.047529
28	3420	02:24:49	0.001	2.8902	
28	3440	02:25:19	0.001	3.3608	
28	3460	02:25:59	0.001	1.1561	
29	3480	02:27:46	0.001	0.92862	
29	3500	02:28:13	0.001	3.7015	0.060341
29	3520	02:28:53	0.001	3.616	
29	3540	02:29:21	0.001	2.6174	
29	3560	02:30:03	0.001	3.5974	
29	3580	02:30:32	0.001	4.9278	
30	3600	02:32:21	0.001	1.4637	0.059441
30	3620	02:33:00	0.001	2.577	
30	3640	02:33:26	0.001	0.47134	
30	3660	02:34:12	0.001	2.5587	
30	3680	02:34:45	0.001	1.6352	
30	3700	02:35:17	0.001	4.9983	0.053671
30	3720	02:35:54	0.001	3.1435	0.05289
31	3740	02:37:54	0.001	0.90516	
31	3760	02:38:28	0.001	3.3815	
31	3780	02:38:57	0.001	4.3234	
31	3800	02:39:25	0.001	0.98488	0.061547
31	3820	02:40:06	0.001	1.113	
31	3840	02:40:30	0.001	2.413	
32	3860	02:42:36	0.001	3.9275	
32	3880	02:42:59	0.001	5.2729	
32	3900	02:43:28	0.001	1.3168	0.060087
32	3920	02:44:09	0.001	0.64613	
32	3940	02:44:39	0.001	1.2805	
32	3960	02:45:18	0.001	1.6409	
33	3980	02:47:24	0.001	2.7039	
33	4000	02:47:54	0.001	1.7966	0.059843
33	4020	02:48:38	0.001	1.8662	
33	4040	02:49:09	0.001	1.2016	
33	4060	02:49:56	0.001	1.8881	
33	4080	02:50:28	0.001	1.8374	
34	4100	02:52:26	0.001	2.0892	0.076526
34	4120	02:53:05	0.001	1.6215	
34	4140	02:53:34	0.001	0.66143	
34	4160	02:54:20	0.001	5.3057	
34	4180	02:54:53	0.001	2.5928	
34	4200	02:55:25	0.001	3.5019	0.066969
35	4220	02:57:23	0.001	4.7085	
35	4240	02:57:51	0.001	0.84415	
35	4260	02:58:27	0.001	0.72068	
35	4280	02:58:56	0.001	1.065	
35	4300	02:59:24	0.001	1.2092	0.08482

35	4320	03:00:05	0.001	3.0749	
35	4340	03:00:30	0.001	0.53024	0.062644
36	4360	03:02:33	0.001	2.8641	
36	4380	03:02:54	0.001	2.252	
36	4400	03:03:26	0.001	3.6088	0.05836
36	4420	03:04:07	0.001	3.1418	
36	4440	03:04:38	0.001	4.3198	
36	4460	03:05:16	0.001	0.96973	
37	4480	03:07:09	0.001	3.1748	
37	4500	03:07:32	0.001	1.5376	0.027759
37	4520	03:08:12	0.001	2.522	
37	4540	03:08:37	0.001	0.74181	
37	4560	03:09:17	0.001	1.2414	
37	4580	03:09:42	0.001	2.5789	
38	4600	03:11:43	0.001	0.79813	0.06594
38	4620	03:12:27	0.001	0.2985	
38	4640	03:12:59	0.001	1.0037	
38	4660	03:13:44	0.001	1.7685	
38	4680	03:14:17	0.001	1.8852	
38	4700	03:14:48	0.001	2.1044	0.087812
39	4720	03:17:01	0.001	1.6398	
39	4740	03:17:28	0.001	0.75728	
39	4760	03:18:09	0.001	4.0161	
39	4780	03:18:36	0.001	2.0127	
39	4800	03:19:08	0.001	0.96368	0.076891
39	4820	03:19:53	0.001	1.5345	
40	4840	03:21:45	0.001	6.4754	
40	4860	03:22:30	0.001	1.8449	
40	4880	03:22:56	0.001	0.84271	
40	4900	03:23:27	0.001	0.66747	0.025133
40	4920	03:24:11	0.001	0.99518	
40	4940	03:24:43	0.001	1.8317	
40	4960	03:25:22	0.001	1.8742	0.027842
41	4980	03:27:20	0.001	3.067	
41	5000	03:27:43	0.001	2.273	0.043024
41	5020	03:28:31	0.001	0.86761	
41	5040	03:29:01	0.001	2.7927	
41	5060	03:29:46	0.001	1.0268	
41	5080	03:30:11	0.001	0.84214	
42	5100	03:32:12	0.001	0.19831	0.021675
42	5120	03:32:51	0.001	0.5959	
42	5140	03:33:23	0.001	3.6565	
42	5160	03:34:04	0.001	1.7454	
42	5180	03:34:35	0.001	2.8559	
42	5200	03:35:03	0.001	2.096	0.032305
43	5220	03:37:12	0.001	1.3313	

43	5240	03:37:39	0.001	1.6435	
43	5260	03:38:19	0.001	4.4223	
43	5280	03:38:49	0.001	0.78181	
43	5300	03:39:20	0.001	1.478	0.028586
43	5320	03:40:02	0.001	0.69817	
44	5340	03:41:57	0.001	0.15231	
44	5360	03:42:39	0.001	3.2444	
44	5380	03:43:07	0.001	1.4024	
44	5400	03:43:36	0.001	1.0412	0.043174
44	5420	03:44:16	0.001	2.1987	
44	5440	03:44:45	0.001	0.24371	
45	5460	03:46:39	0.001	0.6562	
45	5480	03:47:08	0.001	0.14307	
45	5500	03:47:30	0.001	1.5897	0.055329
45	5520	03:48:14	0.001	1.2852	
45	5540	03:48:44	0.001	0.83649	
45	5560	03:49:25	0.001	0.62485	
45	5580	03:49:51	0.001	1.2886	0.012806
46	5600	03:51:36	0.001	0.23439	0.043932
46	5620	03:52:12	0.001	2.2578	
46	5640	03:52:47	0.001	1.1535	
46	5660	03:53:26	0.001	1.2207	
46	5680	03:53:55	0.001	0.84253	
46	5700	03:54:20	0.001	1.5778	0.05779
47	5720	03:56:20	0.001	1.3092	
47	5740	03:56:45	0.001	1.0099	
47	5760	03:57:27	0.001	1.7998	
47	5780	03:57:56	0.001	0.961	
47	5800	03:58:25	0.001	2.8414	0.068643
47	5820	03:59:03	0.001	1.0772	
48	5840	04:00:51	0.001	0.1676	
48	5860	04:01:29	0.001	0.87883	
48	5880	04:01:58	0.001	1.1262	
48	5900	04:02:26	0.001	0.77069	0.052726
48	5920	04:03:07	0.001	3.2681	
48	5940	04:03:35	0.001	2.0654	
49	5960	04:05:32	0.001	1.1746	
49	5980	04:05:58	0.001	0.33127	
49	6000	04:06:26	0.001	0.7084	0.012394
49	6020	04:07:04	0.001	0.15042	
49	6040	04:07:33	0.001	1.0142	
49	6060	04:08:14	0.001	0.17131	
50	6080	04:10:00	0.001	0.73247	
50	6100	04:10:31	0.001	0.65472	0.02633
50	6120	04:11:11	0.001	1.1101	
50	6140	04:11:43	0.001	1.2219	

50	6160	04:12:27	0.001	1.0126	
50	6180	04:12:58	0.001	5.4733	
50	6200	04:13:25	0.001	0.42882	0.031223
51	6220	04:15:13	0.001	1.2624	
51	6240	04:15:35	0.001	3.8611	
51	6260	04:16:16	0.001	3.7046	
51	6280	04:16:44	0.001	0.10605	
51	6300	04:17:13	0.001	4.5391	0.01569
51	6320	04:17:47	0.001	2.0693	
52	6340	04:19:35	0.001	0.41566	
52	6360	04:20:11	0.001	1.2651	
52	6380	04:20:41	0.001	0.90353	
52	6400	04:21:08	0.001	1.3831	0.019305
52	6420	04:21:48	0.001	0.13455	
52	6440	04:22:13	0.001	1.8623	
53	6460	04:24:12	0.001	0.69918	
53	6480	04:24:37	0.001	0.51816	
53	6500	04:25:06	0.001	0.67303	0.039835
53	6520	04:25:44	0.001	3.2689	
53	6540	04:26:12	0.001	1.9475	
53	6560	04:26:57	0.001	3.8151	
54	6580	04:28:48	0.001	3.5454	
54	6600	04:29:17	0.001	2.7832	0.039801
54	6620	04:29:59	0.001	0.45749	
54	6640	04:30:30	0.001	1.194	
54	6660	04:31:15	0.001	0.11344	
54	6680	04:31:45	0.001	1.0949	
55	6700	04:33:33	0.001	1.5194	0.034876
55	6720	04:34:14	0.001	0.55945	
55	6740	04:34:41	0.001	2.4283	
55	6760	04:35:26	0.001	1.4922	
55	6780	04:35:56	0.001	2.23	
55	6800	04:36:28	0.001	3.0715	0.050646
55	6820	04:37:07	0.001	0.93225	0.060554
56	6840	04:38:57	0.001	3.5578	
56	6860	04:39:35	0.001	2.4927	
56	6880	04:40:06	0.001	0.45371	
56	6900	04:40:35	0.001	4.0324	0.0079823
56	6920	04:41:19	0.001	0.93688	
56	6940	04:41:45	0.001	2.4856	
57	6960	04:43:51	0.001	0.53036	
57	6980	04:44:18	0.001	0.46915	
57	7000	04:44:52	0.001	1.263	0.038364
57	7020	04:45:35	0.001	0.77278	
57	7040	04:46:07	0.001	1.2036	
57	7060	04:46:49	0.001	0.4602	

58	7080	04:48:40	0.001	4.2268	
58	7100	04:49:07	0.001	2.0762	0.04366
58	7120	04:49:52	0.001	2.5946	
58	7140	04:50:21	0.001	1.1086	
58	7160	04:51:06	0.001	0.5184	
58	7180	04:51:36	0.001	0.38849	
59	7200	04:53:29	0.001	0.53687	0.058115
59	7220	04:54:11	0.001	1.0915	
59	7240	04:54:41	0.001	1.131	
59	7260	04:55:26	0.001	3.2601	
59	7280	04:55:58	0.001	4.9514	
59	7300	04:56:30	0.001	0.23821	0.046424
60	7320	04:58:31	0.001	0.48432	
60	7340	04:59:03	0.001	1.6434	
60	7360	04:59:45	0.001	3.4284	
60	7380	05:00:18	0.001	1.9151	
60	7400	05:00:49	0.001	0.89042	0.023235
60	7420	05:01:34	0.001	0.11724	
60	7440	05:02:01	0.001	1.1156	0.036965
61	7460	05:04:07	0.001	5.0784	
61	7480	05:04:32	0.001	0.37621	
61	7500	05:05:07	0.001	8.1892	0.16656
61	7520	05:05:49	0.001	1.0216	
61	7540	05:06:22	0.001	2.1384	
61	7560	05:07:01	0.001	2.8295	
62	7580	05:08:54	0.001	0.49761	
62	7600	05:09:21	0.001	2.752	0.031269
62	7620	05:10:04	0.001	0.999	
62	7640	05:10:36	0.001	0.052994	
62	7660	05:11:20	0.001	2.347	
62	7680	05:11:50	0.001	0.40187	
63	7700	05:13:42	0.001	2.8958	0.05046
63	7720	05:14:23	0.001	0.60479	
63	7740	05:14:55	0.001	1.1053	
63	7760	05:15:39	0.001	3.0372	
63	7780	05:16:11	0.001	1.2873	
63	7800	05:16:41	0.001	0.75015	0.021109
64	7820	05:18:44	0.001	1.7201	
64	7840	05:19:12	0.001	0.70326	
64	7860	05:19:54	0.001	0.60658	
64	7880	05:20:23	0.001	1.4521	
64	7900	05:20:54	0.001	0.88882	0.038029
64	7920	05:21:38	0.001	0.86088	
65	7940	05:23:25	0.001	1.7274	
65	7960	05:24:08	0.001	0.53851	
65	7980	05:24:38	0.001	1.9175	

65	8000	05:25:10	0.001	3.3329	0.039299
65	8020	05:25:52	0.001	0.071479	
65	8040	05:26:23	0.001	0.44675	
65	8060	05:27:04	0.001	1.7581	0.058447
66	8080	05:28:53	0.001	0.11056	
66	8100	05:29:18	0.001	0.83558	0.044624
66	8120	05:30:02	0.001	0.29657	
66	8140	05:30:33	0.001	2.3907	
66	8160	05:31:17	0.001	0.056673	
66	8180	05:31:42	0.001	0.25378	
67	8200	05:33:34	0.001	1.8043	0.045579
67	8220	05:34:13	0.001	0.26736	
67	8240	05:34:47	0.001	0.67448	
67	8260	05:35:30	0.001	1.008	
67	8280	05:36:02	0.001	1.3946	
67	8300	05:36:32	0.001	3.2052	0.053524
68	8320	05:38:35	0.001	1.5586	
68	8340	05:39:03	0.001	2.5235	
68	8360	05:39:44	0.001	5.0726	
68	8380	05:40:15	0.001	0.68636	
68	8400	05:40:47	0.001	0.34114	0.040164
68	8420	05:41:32	0.001	0.098022	
69	8440	05:43:23	0.001	0.28226	
69	8460	05:44:03	0.001	0.5683	
69	8480	05:44:30	0.001	0.84813	
69	8500	05:44:59	0.001	0.41292	0.038197
69	8520	05:45:41	0.001	0.86225	
69	8540	05:46:10	0.001	0.23892	
70	8560	05:48:05	0.001	0.15426	
70	8580	05:48:33	0.001	2.2088	
70	8600	05:48:56	0.001	1.3434	0.032705
70	8620	05:49:40	0.001	2.3703	
70	8640	05:50:10	0.001	1.3292	
70	8660	05:50:51	0.001	1.8895	
70	8680	05:51:17	0.001	1.299	0.031385

****

Detector training complete.

****