

# **A REPORT**

**ON**

## **Learning Based Computer Vision for Leather Surface Quality Discriminant Analysis**

**BY**

**Soundarya Krishnan**

**2016B5A70472G**

**MSc. Physics &  
B.E. Computer Science**

**Pujika Kumar**

**2016A3B30653G**

**B.E. Electrical and Electronics  
Engineering and MSc.  
Economics**

Prepared in partial fulfilment of the  
Practice School-I Course

**AT**

**Central Electronics Engineering Research Institute (CEERI), Chennai campus**

**A PRACTICE SCHOOL –I STATION OF**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI  
GOA CAMPUS  
JUNE, 2018**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE**

**PILANI (RAJASTHAN)**  
**Practice School Division**

**Station:** Central Electronics Engineering Research Institute

**Centre:**  
Chennai

**Duration:** 8 Weeks

**Date of Commencement:** 22-May-2018

**Date of Submission:** 18-June-2018

**ID No./Name:** 2016B5A70472G/ SOUNDARYA KRISHNAN

**Discipline of the student:** Msc. Physics + B.E. Computer Science

**Project Title:** Learning Based Computer Vision for Leather Surface Quality Discriminant Analysis

**Name and Designation**

**Of the expert** Mr. B. Sundaresan, Chief Technical Officer

**Name of the PS Faculty:** Ms. Akshaya Ganesan

**Keywords:** Leather, Defect analysis, Computer vision

**Project Areas:** Deep learning, Image processing

## **Abstract**

The leather industry is an important industry considering how extensively leather is used in clothes and accessories. Leather Defect Detection is the most important step before its sale and further processing. Defect inspection is done manually at present by experienced workers. However, the analysis is slow and unreliable. To avoid these drawbacks, we suggest an automated defect detection and analysis system to detect and classify defects, as well as find the percentage of leather affected. For this, we use C++ in Visual Studio for Image Processing and Python for Deep Learning. We also plan to achieve this in real time and create a form-based interface for ease of use. This method of automated defect analysis will be faster as well as more accurate than the manual checking being done at present.

**Signature of Student**

**Signature of PS Faculty**

**Date**

**Date**

## **Acknowledgements**

We would like to express our heartfelt gratitude to Dr. B. Sundaresan, who has guided us at every step of the way. We sincerely acknowledge his invaluable knowledge, expertise and motivation that has helped maintain our progress.

We express our deepest gratitude and special thanks to Ms. Malathy Jawahar, Principal Scientist, Central Leather Research Institute, for allowing us to take pictures of leather samples. We would also like to acknowledge the crucial role of her assistant for imparting us knowledge about the several defects present on leather samples which were extremely valuable for our project.

We also extend our sincere gratitude to Ms. Akshaya Ganesan, our PS1 faculty for her invaluable guidance throughout the course and ensuring our comfort in the working environment.

Lastly, we would like to thank PSD for providing us with industry exposure and be a part of a professional organization.

## **PS-1 Organization Profile**

Central Electronics Engineering Research Institute (CEERI) is a part of the Council of Scientific and Industrial Research (CSIR) and is committed to research and Development in electronics to meet the requirements of its customers. CEERI has been continually identifying the thrust areas of key National/Industrial relevance and has been developing competitive technologies with a goal to achieve excellence and self-reliance in these areas.

CEERI Centre in Chennai, a pioneering institution in the field of Quality Control Instrumentation for process industries, is a regional centre of Central Electronics Engineering Research Institute (CEERI), Pilani, Rajasthan. The centre has a rich experience and expertise in this field and has developed many online monitoring systems for various process on indigenous development of special sensors and systems suited to the online measurement and control and these involve varying technologies such as Near Infra Red (NIR) Gauging, Beta Gauging, Optical, Electromechanical methods and the currently emerging Image Processing techniques. The centre has been extensively contributing its expertise in Lab VIEW for online apple sorting. The Centre has also unveiled a system for sorting plastics through NIR spectroscopy.

CEERI Centre has well equipped Optics laboratory for the development of electro optical systems, DSP and FPGA lab and rapid prototyping environment of electronic systems hardware and firmware and work stations with advanced imaging and computing software/hardware to develop machine vision technologies for online-inspection and grading. It also has CAD and drafting facilities to cater the needs of R & D activities.

# TABLE OF CONTENTS

1	INTRODUCTION	1
2	SCOPE/ AIM AND OBJECTIVES	1
3	METHODOLOGY	
3.1	Dataset Preparation	2
3.2	Pre-processing	3
3.3	Object Detection	5
3.4	Object Classification	7
3.5	Optimization	8
4	RESULTS AND DISCUSSION	
4.1	Basic Pre-processing	8
4.2	Advanced Pre-processing	10
4.3	Blob Detection	11
4.4	Support vector machines	11
4.5	Real time object detection	12
5	CONCLUSION	13
6	REFERENCES	14

## 1. Introduction:

Raw hides undergo several machining processes to qualify for market standards. These raw hides may have several defects from the lifetime of the animal. Further, defects may arise due to the production process as well. Leather Defect Detection is the most crucial process before the sale of leather, as it is the type, as well as extent of defects in the sample that determine its market value.

Currently, the inspection is done manually by highly experienced workers. However, this has many disadvantages. The speed of examining is slow. Moreover, defect detection by humans is a very nuanced process and can easily be affected by the mental status of the inspector.

In today's age when the leather industry is more competitive than ever, this may prove to be a serious disadvantage, and this is why we attempt to make an automated way to identify defects and determine the extent of affected leather, in order to fix the grade of the leather sample. For this, our initial plan was to use SVM to classify the defect and Bilateral filtering and Histogram Normalization methods, followed by blob detection using Determinant of the Hessian for calculating area of affected leather. However, these methods didn't provide satisfactory results so we switched to darkflow for defect detection and wrote a custom code to find area of fabric affected.

The YOLO algorithm works with a speed of 45 frames per second, even better than real-time. It understands generalized object representation and thus is especially useful for complex tasks like defect detection in leather. Thus, we present a robust, fast way to identify defects from leather and classify them quickly as well as easily.

Since most of our report revolves around OPEN-CV and Convolutional Neural Networks, in order to fully understand the project, following is a brief introduction to both.

### 1.1. OpenCV:

OpenCV, or Open Source Computer Vision Library is free for use as it is released under a BSD license. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV is written in C/C++ and thus can use parallel computing to utilize all the cores of the system, thus being very useful for real-time applications.

OpenCV has a staggering 47 thousand people in its user community and downloads exceeding 14 million. It is used all around the world for various tasks as diverse as collaborative art to cutting edge robotics. The OpenCV library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic

and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used for facial recognition, object detection, data augmentation, real time filters and so much more. OpenCV is a must for any practical image processing related work and the powerful library is the basis for the first half of our project, including facial recognition, object detection, as well as object classification using SVMs.

## 1.2. Convolutional Neural Networks:

Convolutional neural networks are neural networks that are used in image recognition and classification. There are three main operations performed in a convolutional neural network:

- Convolution
- Non-linearity
- Pooling or sub-sampling

### Convolution:

Any grayscale image can be represented as a 2D matrix with the entries corresponding to the pixel intensities. An RGB image can be visualized as a set of three 2D matrices stacked on top of each other. The primary purpose of a convolution is to extract the features of an image by preserving the spatial relation between pixels. In convolution, the input image is convolved with a kernel (filter) to produce the output image also called a Feature Map or Activation Map. An example of convolution is shown below in Figure I.

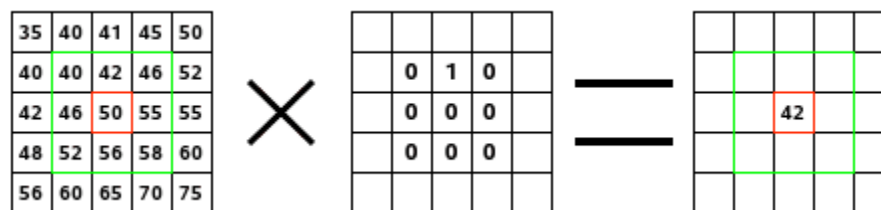


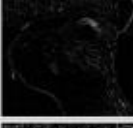






Figure (i): Example of convolution



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

**Table (i): Different convolution matrices**

Different values of the filter matrix will produce different feature maps of the same image. In the table shown above, we see the effect of convolution with different filters. Thus, different filters can detect different features from an image, for example edges, curves, contrast variation, etc.

CNNs learn the values of these filters in its own during the training process. The more filters we have, the more number of features are going to be extracted and the network become better in recognizing images.

### Non-linearity:

A non-linear layer is mostly applied to the output image from convolution because most of the real world data we deal with are non-linear. Some of the non-linear functions include tanh, sigmoid and ReLU(Rectified Linear Unit)

## Pooling:

Pooling (also called as subsampling or downsampling) reduced the dimensionality of the feature map but retains the most important feature. Pooling can be of different types: Max, Average, Sum, etc.

An example of maxpooling is shown below:

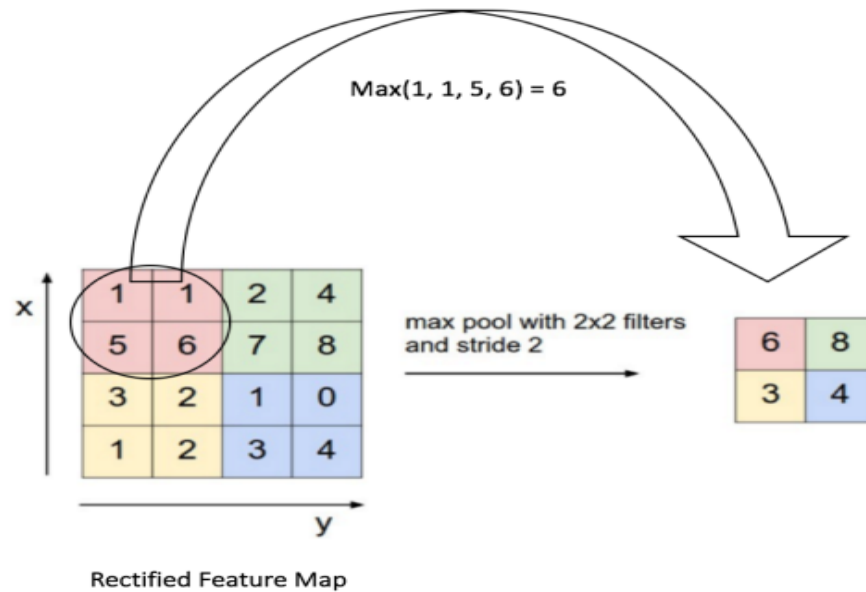


Figure (ii): Rectified Feature Map

Therefore, the whole CNN process can be visualized as below:

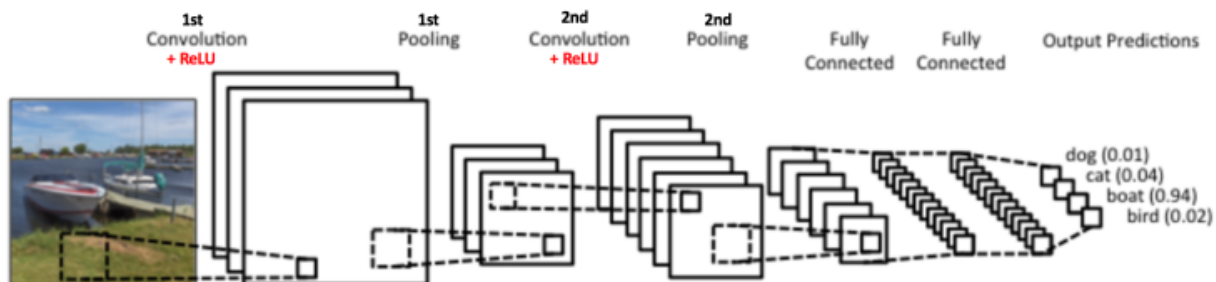


Figure (iii): CNN process

## 2. Scope and Aim:

This project, we hope, will help revolutionize the leather industry by providing automated real-time defect analysis. The scope of this project isn't restricted to just CLRI or leather manufacturers and can also be extended to analysing defects in textiles.

The aim of the project is to make a real time automated system for detecting and the defects present on leather. In addition to this, the severity of defects will also be determined by calculating the area of defect. We also wish to make a graphical interface for ease of use.

Before we proceed, we would like to state that mathematically rigorous proofs for the methods used are beyond the scope of this paper. What we wish to do is apply well-known algorithms and methods to new problems, and thus will be giving a qualitative overview of the methods being used.

### Objectives:

- Defect detection and calculation of area of defects in a given leather sample
- Making a Graphical User Interface for ease of use
- Usage of Intel Parallel Studio and Parallel Computing techniques to achieve this in real time

## 3. Methodology:

The initial plan was to classify defects using SVM and blob-detection, and the first part of the methodology describes in detail the various methods used. As will be discussed in following sections, the SVM classifier as well as blob detection methods failed for various reasons, and thus, we pivoted to defect detection using darkflow, which will be explained in Part (b).

### Part (a)- Initial Research:

We have identified 7 broad steps to fulfil our objective. These are as follows:

- Dataset Preparation
- Data Pre-processing
- Defect Detection
- Defect Classification
- Optimization

All the image processing codes are written in Visual Studio 2017 in C++ using OpenCV, and machine learning codes are written in Python using Keras. The reason for this is that C++ provides faster results for image processing, while Python with its extensive libraries for deep learning makes it easier to code and understand Machine Learning.

### 3.1. Dataset Preparation:

For collecting data, we had to visit the Central Leather Research Institute (CLRI) from which we collected images of unprocessed leather samples. This was done after contacting the Principal Scientist of CLRI, Dr. Malathy Jawahar, who was experienced in this area. After consulting a worker specializing in leather defects, we were able to classify the defects observed. Some of the common defects observed were growth marks, scars and grain off defects. Figure 1 shows a small snippet of our dataset obtained from CLRI, and Figure 2 shows some common types of defects found in leather.

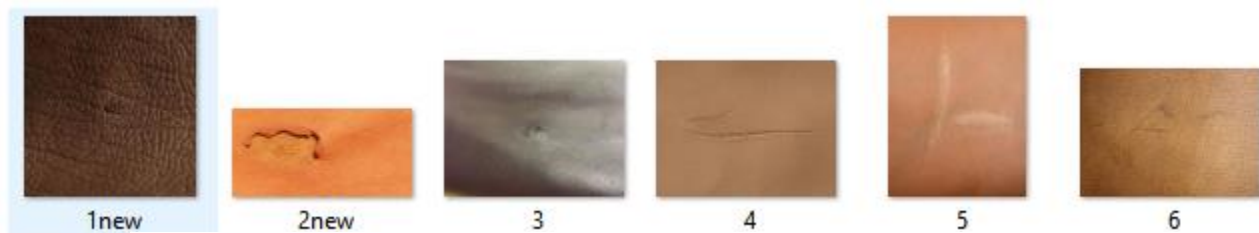


Figure 1: Leather Dataset



Figure 2: Defect types

We could get a total of 203 images of defects.

Since we took some time to obtain the leather dataset, most of the work we have done is on other data to test out the code, and these were tested for leather samples later.

### 3.2. Pre-processing:

- **Re-sizing:**

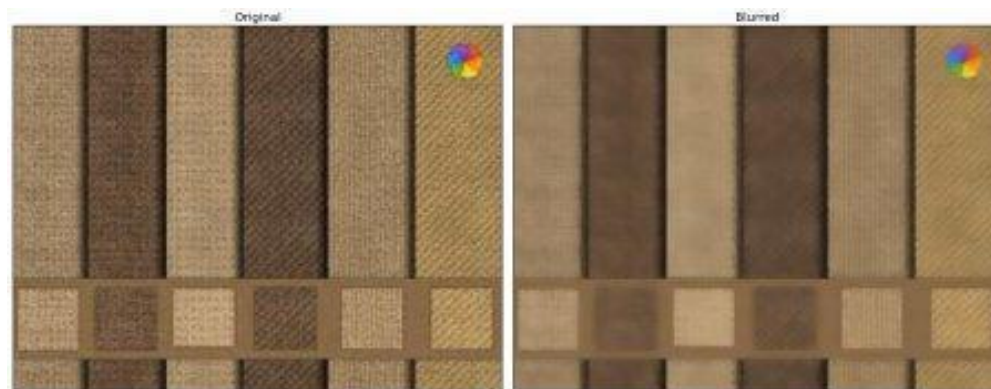
We had to write a code to manually resize the images to the desired size in a loop and save them automatically, as most machine learning algorithms require inputs to be of the same size.

- **Augmentation:**

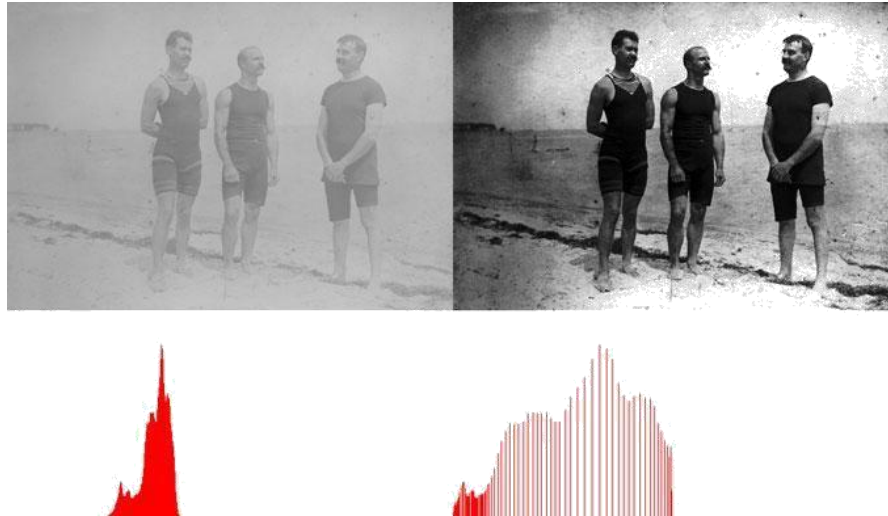
Any machine-learning based classification requires a large dataset for better performance. For this purpose, and to communicate to the machine that even a slightly rotated banana is still a banana, we used augmentation, i.e. applied a random rotation or mild noise to each picture in the dataset. This achieves a larger dataset and improves accuracy.

The second part of pre-processing was for the alternate project that we had in mind, to find the percentage of area affected by a defect. For this purpose, we needed to use blob detection methods that are described later. The pre-processing required for this included Bilateral Filtering and Histogram Normalization.

Without going into the mathematical specifics which are beyond the scope of this report, Bilateral Filtering is a way to smooth out the textural information without affecting the edges. Histogram Normalisation, or contrast stretching effectively makes areas of low contrast get higher contrast, by spreading out the most frequent intensity values. Examples of Bilateral Filtering and Histogram normalization are given in Figures 3 and 4 respectively.



**Figure 3: Bilateral Filtering**



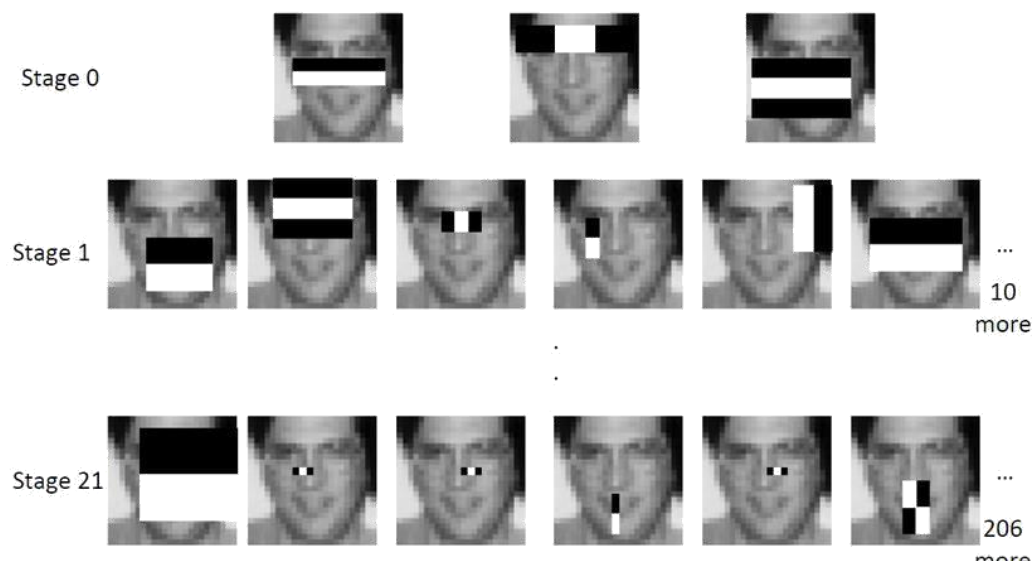
**Figure 4: Histogram Normalization**

Essentially, these pre-processing methods make the edges of blobs more obvious, hence improving blob detection. These 2 procedures were implemented using C++. The results after applying this to a leather sample are given in the results section.

### 3.3. Object Detection:

Initially, we attempted object detection by implementing facial and pedestrian detection, in order to get familiar with the code. We will attempt to use this code to detect leather defects too. We were able to implement real-time facial detection using Haar Cascades. For this, Haar features, shown below are used. Each feature is obtained by the difference of sum of pixels in a particular area. Cascading is a way to optimize this by checking the more important features first, followed by less important features.

However, we did not use this for leather defect detection as we realised that this is quite an outdated algorithm and decided to switch over to newer algorithms like YOLO which are more advanced as well as effective.



**Figure 5: Haar Cascades**

Pedestrian Detection was implemented using Histogram of Oriented Gradients.

Qualitatively, it captures the shape of structures in the region by capturing information about gradients. This is done by checking the x gradient and y gradient, i.e., the difference between pixels in the horizontal direction and vertical direction. The overall gradient magnitude is the root of sum of squares of the x and y gradients, and direction can easily be found by dividing the two and taking  $\tan^{-1}$ .

We did not apply this algorithm to leather defects as it had the disadvantage of needing objects to be of a particular ratio. While this works tremendously well for objects like pedestrians or pets or even objects like water bottles etcetera, we found that this doesn't perform well for images with uneven defects with no standard ratios. Thus, we needed to use different methods for leather defect detection.

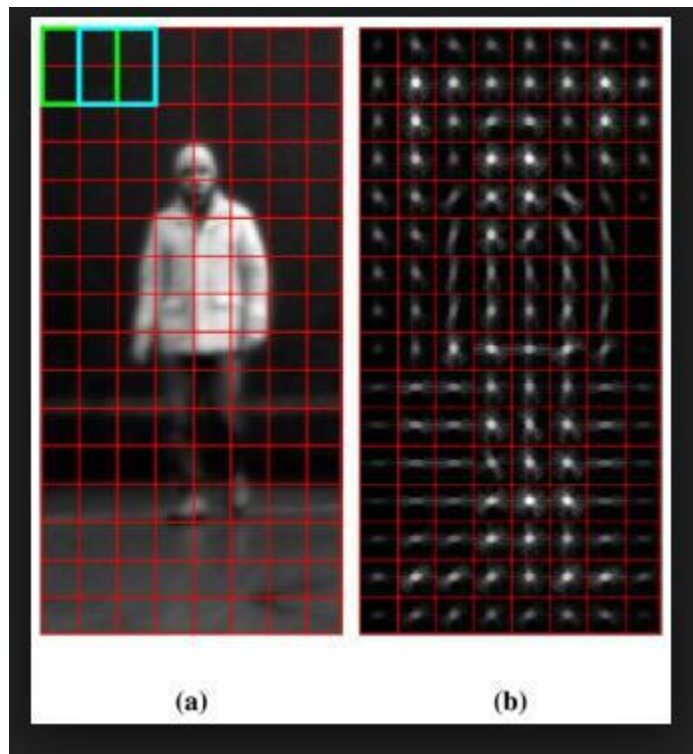


Figure 6: HOG



As was mentioned earlier, the side project planned requires us to detect the region of defects. For this, we use blob detection using a mathematical technique, determinant of Hessian. For our purposes, what this method does is evaluate regions of local maxima of intensity and mark those regions. With this method, we can find the percentage of the material affected by defects.

We applied these algorithms on C++. Our application of both of this is discussed in the results section. We found that these methods were not generalizable to leather and thus pivoted the focus of our project to darkflow.

### 3.4. Object Classification:

On reading several papers on object classification, one of the most popular methods is Support Vector Machines (SVM).

#### Support Vector Machines:

Support Vector Machines are a powerful way to classify objects. It draws a line, plane, or hyperplane, depending on the dimensionality, separating the various categories. A parameter controls whether we prioritize distance between clusters or fewer miss-classified examples.

We attempted Support Vector Machines with a self-prepared dataset of bananas, in order to classify objects as either a banana or not a banana, both before and after augmentation. As we will see in the results section, the accuracy wasn't sufficient for even an object with a clear shape like a banana, so we decided to switch over to dark flow that uses advanced algorithms.

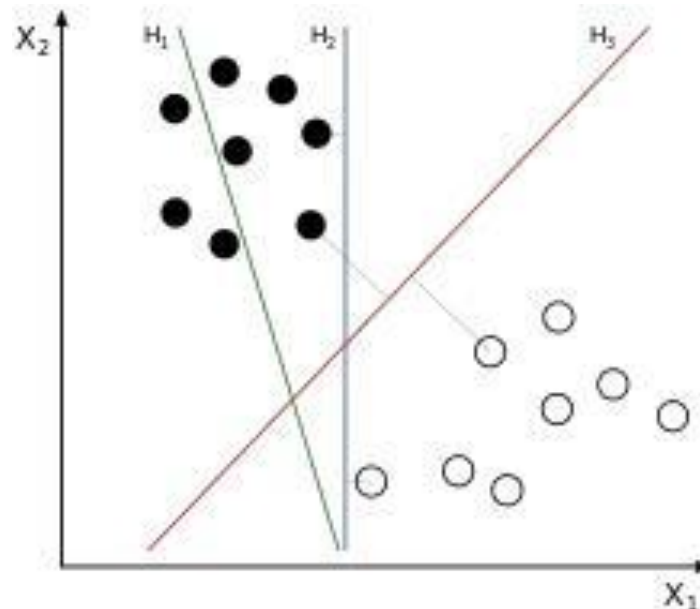


Figure 5: SVM Classifier



## **Part (b)- Dark flow:**

Darkflow is a network builder which allows building Tensorflow neural networks from configuration files and pre-trained weights. It requires Tensorflow, OpenCV, python, NumPy and skimage. The darkflow is installed and setup using the command `pip install`. We will use it to implement YOLO (You only look once) which is a state of the art, real time object detection algorithm.

### **YOLO (You Only Look Once) Algorithm:**

The traditional object detectors work on the sliding window approach. A small window slides across the image, and at each window predicts what sort of object it is using extracted features. This process repeats again and again with the window size increasing each time. Thus, for an image it gives thousands of predictions but only considers those which have a higher probability. YOLO on the other hand, is a convolutional neural network used for real time object detection.

Unlike traditional classifiers which is very time consuming, YOLO trains on full images and directly optimizes detection performance. It divides the image into 13x13 grid cells. Each of these cells is responsible for predicting bounding boxes. For each bounding box, the cell predicts a confidence score about how certain it is that the bounding box contains an object and its class. An example of the 13x13 grid is shown in Figure 8(a). The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object. Example is shown in Figure 8(b). We set a threshold value depending upon the accuracy we want, according to which the low scoring boxes get ignored and produces an image, as shown in Figure 8(c).

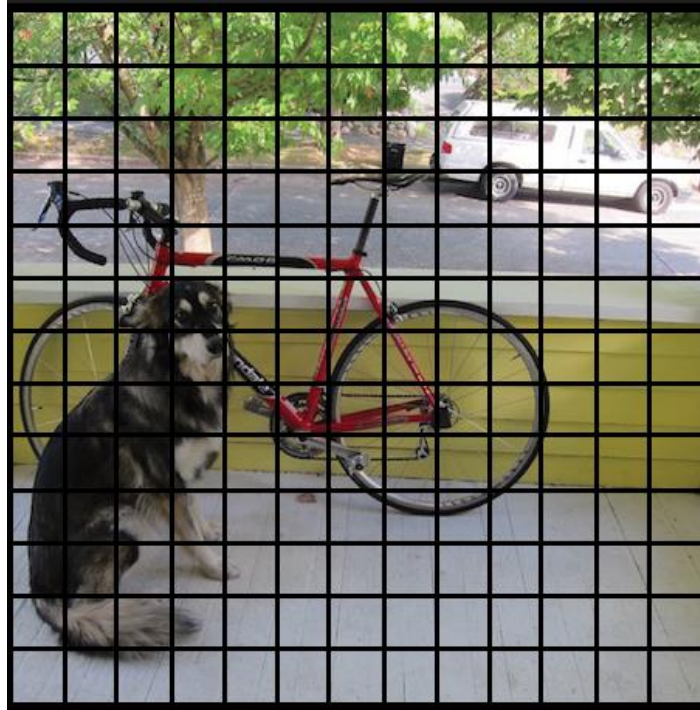


Figure 8 (a): YOLO Algorithm

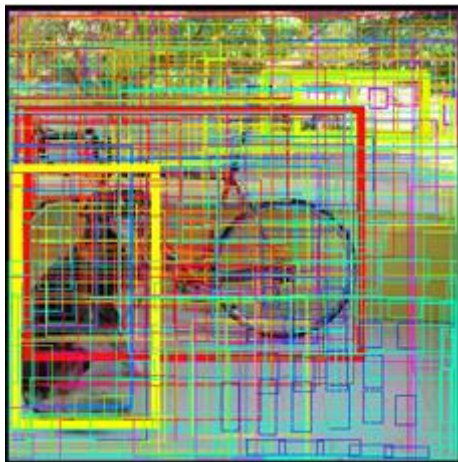


Figure 8 (b): YOLO Algorithm

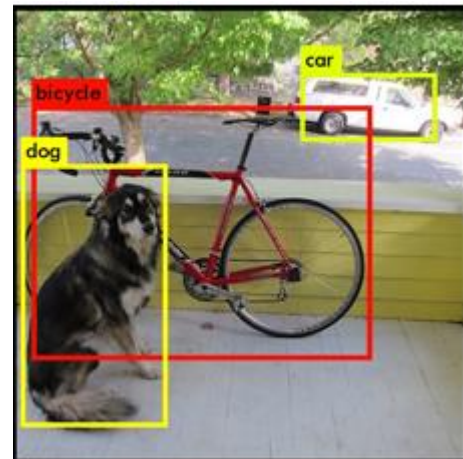


Figure 8 (c): YOLO Algorithm

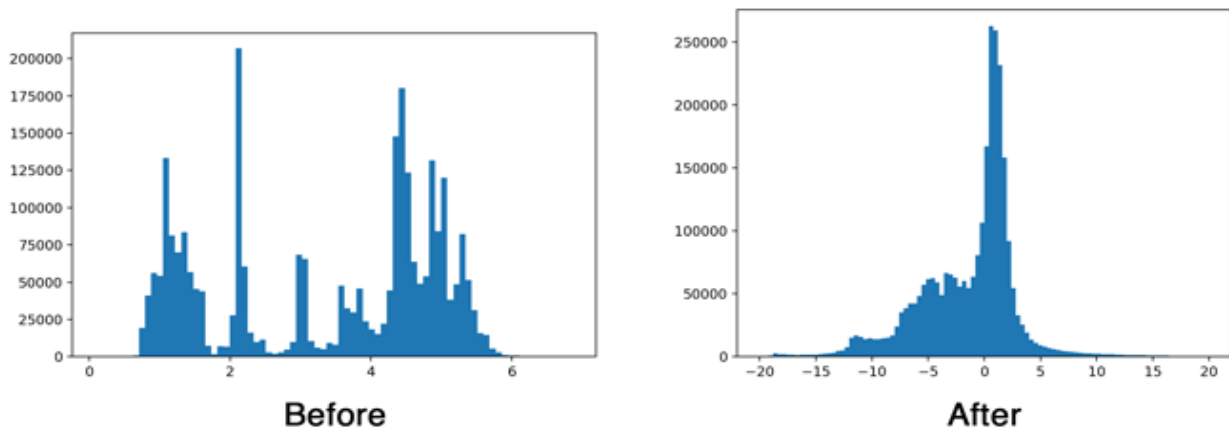
Layer	kernel	stride	output shape
Input			(416, 416, 3)
Convolution	3×3	1	(416, 416, 16)
MaxPooling	2×2	2	(208, 208, 16)
Convolution	3×3	1	(208, 208, 32)
MaxPooling	2×2	2	(104, 104, 32)
Convolution	3×3	1	(104, 104, 64)
MaxPooling	2×2	2	(52, 52, 64)
Convolution	3×3	1	(52, 52, 128)
MaxPooling	2×2	2	(26, 26, 128)
Convolution	3×3	1	(26, 26, 256)
MaxPooling	2×2	2	(13, 13, 256)
Convolution	3×3	1	(13, 13, 512)
MaxPooling	2×2	1	(13, 13, 512)
Convolution	3×3	1	(13, 13, 1024)
Convolution	3×3	1	(13, 13, 1024)
Convolution	1×1	1	(13, 13, 125)

Figure 9

This neural network only uses standard layer types: convolution with a 3×3 kernel and max-pooling with a 2×2 kernel. The very last convolutional layer has a 1×1 kernel and exists to reduce the data to the shape 13×13×125. We can specify the input size of the image by specifying the width and height in the configuration file. The input image goes through the convolutional network and produces a vector of size 13x13x125 describing the bounding boxes. The description includes:

- x, y, width, height for the bounding box's rectangle
- the confidence scores
- the probability distribution over the classes

YOLO also uses a regularization technique called batch normalization which basically helps the data from deteriorating as it passes through the network. Batch normalization is done before the activation layer is applied. Figure 10 is a histogram of the output of the first convolution layer without and with batch normalization:



**Figure 10: Histograms**

The parameters present in the configuration file include:

- the input size of the network: width, height and number of channels
- learning parameters: learning rate, number of steps after which the learning rate is changed, factor by which the learning rate is re-scaled and the maximum number of iterations

The configuration file also includes the convolutional neural network architecture along with the max-pooling layers. In the convolutional layer, the number of filters and the activation type is specified. Finally, in the last layer the number of predicting boxes per cell, the number of classes and the relative weights for the combined loss function are also specified. Thus, in this way the parameters can be customized according to the performance we want.

Therefore, YOLO is the ideal method for achieving objective as it is extremely fast and it learns generalizable representations of objects. When tested on images, it outperforms top detection algorithms like DPM and R-CNN.

### **3.6. Darkflow leather dataset:**

We again required a dataset for this purpose, and to suit this purpose, we had to sort out the previous dataset to include images with prominent defects that were localised, so that it would be easy to draw boxes around it. We decided to train the network to detect only scars as we were unable to find localised instances of other defects like, say, stretch marks. We selected images with prominent defects surrounded by good leather so that the network could learn defects properly. A sample set is shown in Figure 11.

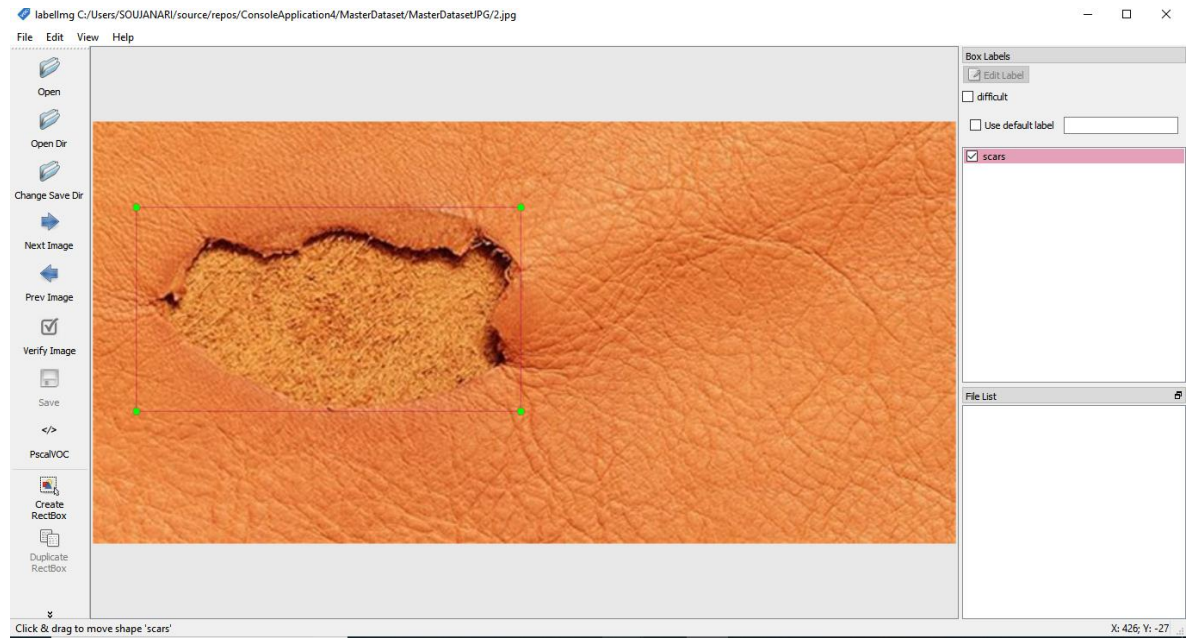


**Figure 11: Darkflow dataset**

### 3.7. Data-Pre-processing:

Initially, we flipped images in a loop to increase the dataset and quality of predictions. We wrote a code for flipping all the images in a directory automatically, and this is shown in the results section.

For putting data for training, we require the bounding boxes in an .xml format. For this, we used LabelImg, a graphical annotation tool. We marked the bounding boxes manually around each scar or scratch so that the model would be able to effectively identify defects, as shown in Figure 12.



**Figure 12: labelling**

The bounding boxes in each image get stored in a .xml format automatically and this input goes into training.

### 3.8. Training:

Since our objective was only to detect whether a given piece of leather has a defect or not, there was only one class in the configuration file. The number of filters in the second last layer also had to be changed according to the formula  $\text{num} * (\text{classes} + 5)$  where num is the number of bounding boxes per grid cell. Apart from this, a text file had to be made called labels.txt which contained the name of the classes. In this project, there is only one class named scars. After editing the configuration file, a suitable weights file has to be downloaded. For this project, we downloaded tiny-yolo-voc.weights

The training command requires the directory of training images as well as the annotations (xml files). The training can be done using cpu or by using both cpu and gpu. The percentage of gpu usage can also be specified. An example is shown in the figures below:

```
C:\Users\Lohia\Anaconda3\Lib\site-packages>python flow --model cfg/tiny-yolo-new.cfg --train --dataset "C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\training_images" --annotation "C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\training_xml"
```

**Figure 13**



```
C:\Users\Lohia\Anaconda3\Lib\site-packages>python flow --model cfg/tiny-yolo-new.cfg --train --dataset "C:\Users\Lohia\A
naconda3\Lib\site-packages\darkflow-master\training_images" --annotation "C:\Users\Lohia\Anaconda3\Lib\site-packages\dar
kflow-master\training_xml" --gpu 1.0
```

Figure 14

Once training starts, all the convolutional layers are initialized and loaded. The output size of the images after convolution are also specified. It computes the loss hyper-parameters such as the number of grids, boxes, classes and the scaling factor, as shown in Figure 15.

```
Building net ...
Source | Train? | Layer description | Output size
-----+-----+-----+-----
Init   | Yes!   | input            | (?, 416, 416, 3)
Load   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 416, 416, 16)
Load   | Yes!   | maxp 2x2p0_2     | (?, 208, 208, 16)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 208, 208, 32)
Load   | Yes!   | maxp 2x2p0_2     | (?, 104, 104, 32)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 104, 104, 64)
Load   | Yes!   | maxp 2x2p0_2     | (?, 52, 52, 64)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 52, 52, 128)
Load   | Yes!   | maxp 2x2p0_2     | (?, 26, 26, 128)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 26, 26, 256)
Load   | Yes!   | maxp 2x2p0_2     | (?, 13, 13, 256)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 13, 13, 512)
Load   | Yes!   | maxp 2x2p0_1     | (?, 13, 13, 512)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 13, 13, 1024)
Init   | Yes!   | conv 3x3p1_1 +bnorm leaky | (?, 13, 13, 1024)
Init   | Yes!   | conv 1x1p0_1     | (?, 13, 13, 30)
-----+-----+-----+-----
Running entirely on CPU
cfg/tiny-yolo-voc_1c.cfg loss hyper-parameters:
H      = 13
W      = 13
box    = 5
classes = 1
scales = [1.0, 5.0, 1.0, 1.0]
```

Figure 15

For our network according to the hyperparameters we have set, the moving average loss starts at 109 and after approximately 1500 steps reduces to a single digit number. The ideal time to stop the training is when the moving average loss is close to 0.067. The updated weights (checkpoints) are stored after every 125 steps in the ckpt folder and training stops automatically after 1000 epochs. Since the required loss wasn't achieved even after 1000 epochs, we started training again by loading the last saved checkpoint using the following command.

```
C:\Users\Lohia\Anaconda3\Lib\site-packages>>python flow --model cfg/tiny-yolo-voc_1c.cfg --load bin/tiny-yolo-voc.weights --train --dataset "C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\training_image" --annotation "C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\training_xml" --gpu 1.0 --load -1
```

**Figure 16**

For easy user accessibility, we have integrated this command with the windows Form application. Thus, with the click of a button, the user can train a customized object detector without having to worry about the internal working of the network.

### 3.9. Testing:

For testing the YOLO network, we have to specify the test images directory along with the checkpoint we wish to use. An example is shown below where the 3625<sup>th</sup> checkpoint is being used for testing, as shown in Figure 17.

```
C:\Users\Lohia\Anaconda3\Lib\site-packages>python flow --imgdir C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\test_images --model cfg/tiny-yolo-voc_1c.cfg --load 3625
```

**Figure 17**

After executing this command, the test images with the bounding boxes are found in an out directory inside the test images directory. If required, the coordinates of these boxes can also be obtained in a json file using the following command.

```
C:\Users\Lohia\Anaconda3\Lib\site-packages>python flow --imgdir C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\test_images --model cfg/tiny-yolo-voc_1c.cfg --load 3625 --json
```

**Figure 18**

All these commands are also integrated with the Visual Studio Windows form application. The user can test the images by clicking “test” button on the form. These commands are executed internally and the final output is displayed on the form.



## 4.0. Windows Form Application:

One of the last steps required for making our project important practically was making a Graphical User Interface for ease of use and distribution. For this purpose, we used Visual Studio's Windows Form Application using C++/CLI, so that users could get the bounding boxes around the defects using one swift click. The backend of this GUI can be summarised using a flowchart, as shown in Figure 19.

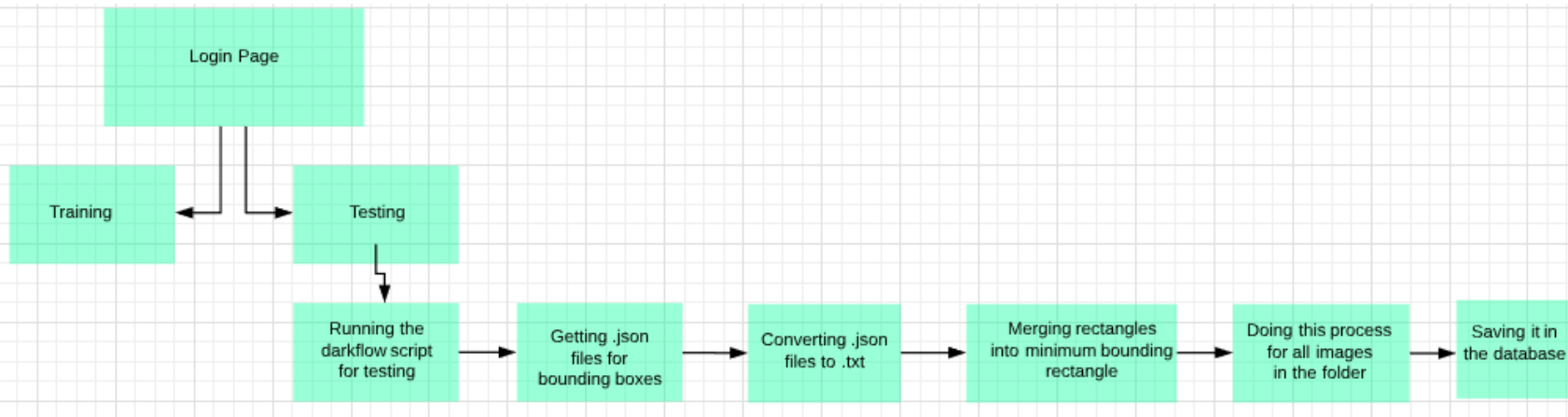


Figure 19: WFA Flowchart

The first page of the Windows Form application is the login page which has two levels of entry:

- Administrator
- Employee

In order to make a fully functional login page and to store data, we were required to link Visual Studio with MySQL.

MySQL is an open source software used for creating relational databases. A relational database is a set of multiple datasets organized in tables where all the datasets are linked to each other. The databases are connected to the MySQL server which provides user-interactive queries for creating databases. Therefore, it is necessary to start the MySQL server before creating a database.

For this project, we have used MySQL5.7 which is currently the latest version. Using MySQL, we have created two tables, one for storing login credentials and the other for storing the test results, both of which are accessed through the Visual Studio Windows Form application. The login table has three fields: Serial number, Username and Password. After the user enters his details in the form and pressed Submit, the database validates these details. If the details are correct the user is redirected to

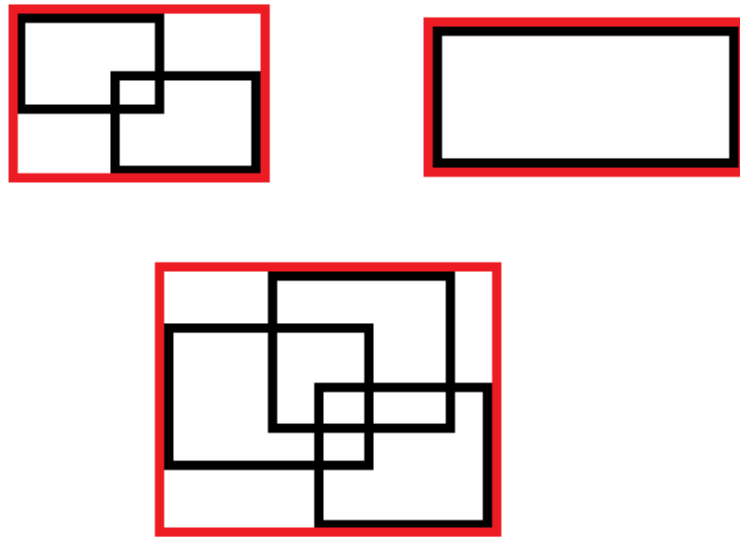
the next form where he/she can choose to either test, train or view the results, else if the details are incorrect a message box is displayed saying "Username and password is incorrect". The test results table has five fields: Serial number, Image ID, Type of defects, Number of defects and Area of defective region. The test results' table is updated from the form itself.

The administrative rights include altering the login credentials of the employees, training and testing the darkflow network, and viewing the test results. The administrator can train by using any images of his or her own, and this feature provides the user with the advantage of adding any other defects that may be relevant to them. The training page which can be accessed only by the administrator has the option to train the darkflow network by adding custom images and then identifying the defects using LabelImg. If training is done again, the checkpoints are altered and the results would be different. This is necessary as it helps achieve better accuracy and boost the leather industry standard.

On the other hand, the employee has rights to only test and view the results. While testing, there is also an option of storing the test results in the database. The visual look of the form is given in the results section. If the user presses view, he/she has the option of viewing all the records stored in the database.

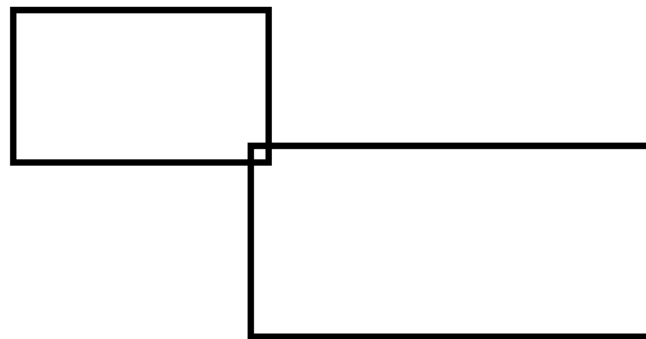
Coming to the testing part, we needed to run a python script to test the images using our trained yolo weights. We did this by opening command prompt from the form-based application using the 'system' command. The darkflow testing command outputs boxes around defective regions on leather images as well as stores the coordinates of the top-left and bottom-right coordinates of these boxes in a json file. We needed to get these files in so that we could send it as an input to the merge rectangles code, which is described shortly. We wrote a python script to read these coordinates along with the number of boxes, to be sent as an input to Merging rectangles code in a .txt format. The python script is called from the Form itself when the user presses test.

Once that step was complete, we could proceed to merging rectangles into the minimum bounding rectangle, so as to better represent the defective region. This is better explained through a picture. The black boxes represent the areas the model thinks contain scars, and the red rectangles is our output, which contains the entire defective region, shown in Figure 21.



**Figure 21**

This was so that we could capture the entire defective region and could easily calculate the area of leather that's actually usable. So, in case if the model predicts two instances of scars nearby, this code could ensure that the entire region covered by these 2 scars is shown as the defective region. For this, we wrote 3 nested loops to go through each of the boxes and check if the area of intersection of the two was above a certain threshold, 25% of the union of both the areas in our case. This was done to ensure that boxes like shown in Figure 22 are still treated as separate rectangles.



**Figure 22**

Now, we needed to do so for all images in a particular directory, so we looped over these steps, and put the increment condition in a button so that users can view the next images after pressing the button.

We also wrote a code to calculate the percentage of area of defective leather. We calculated the area of the merged rectangles and divided it by the area of the image to find the defective area. We also return the number of final merged rectangles as well as the image number in the form, which is later stored in the database for further use. Another side project being considered is cropping out the defective regions bounded by the coordinates and sending it to a classifier. With this, we can get the type of defect simultaneously as well. The visual look of the GUI is given in the results section.

#### **4.1. Usage of the camera:**

For continuously capturing leather samples in real time without losing any frames, we have used a ueye camera (Figure 23) compatible with USB 3.0 along with OpenCV API functions. Our objective is to use a ring image buffer sequence with at least 3 buffers. These buffers will be put into an image queue. Using the image queue, the oldest buffer can be fetched. Collecting the image buffer will be done in a detached thread. This thread will be independent of the GUI interaction and will allow collection of images without interruption. We are still working on the camera to achieve the aforementioned tasks.



**Figure 23**

#### **4.2. Optimization:**

This is the final method required to fully implement our project. Our end goal is to implement in real time, for which we needed to optimize the code. We have now shifted to dark flow which uses advanced parallel computing techniques and runs up to 5 times faster than a normal CPU. We checked the CPU usage and found that all the cores are indeed in use.



**Figure 24**

Here is a screenshot of the CPU usage of the different cores while training. This shows that darkflow when run on Intel Python makes use of advanced parallel computing techniques.

Other than this, there is also the `--gpu 1.0` command for darkflow that runs the training 100% on GPU.

However, for the parts of the project that don't involve dark flow, we can use parallel computing to utilize the idle cores of the system. We plan to use this in 2 ways.

- **Parallel Segmentation of images:**

For image processing uses like converting to grayscale or resizing the image, a way of optimization would be to segment the image into smaller blocks and send each block to a different core, since making different blocks grayscale is independent of each other.

- **Parallel For loops:**

For operations that contain for loops that do operations independent of each other, a good way to optimise would be to send different iterations to different cores to do computations in parallel, thus reducing runtime.

## 4. Results and Discussion

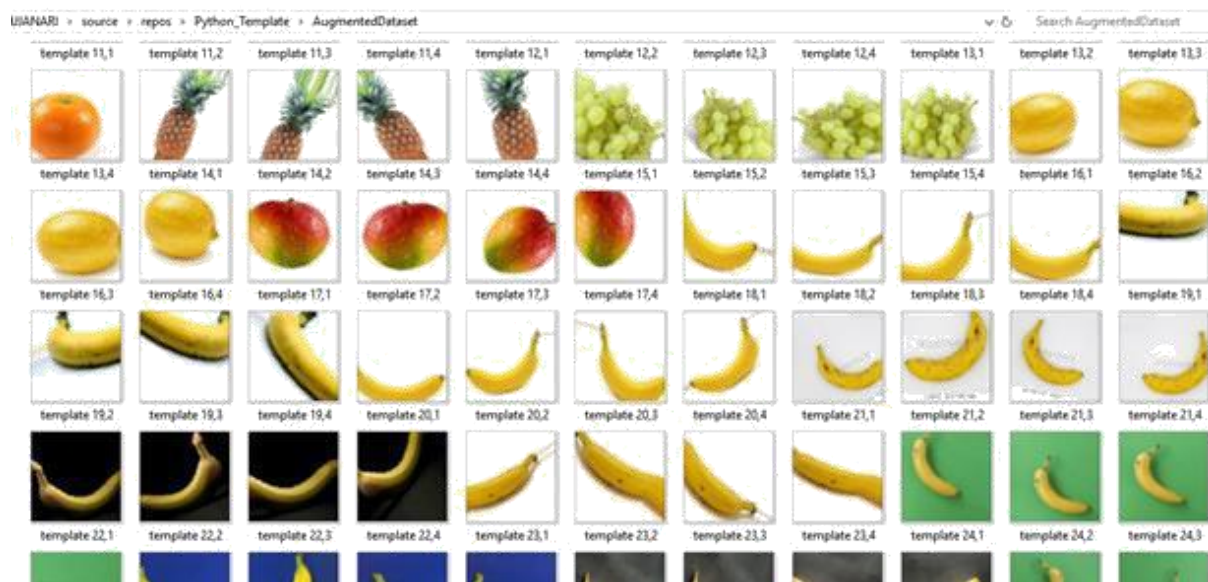
### (a) Initial Research

Here are the results that were obtained by using the methods described earlier

#### 4.1. Basic Pre-processing

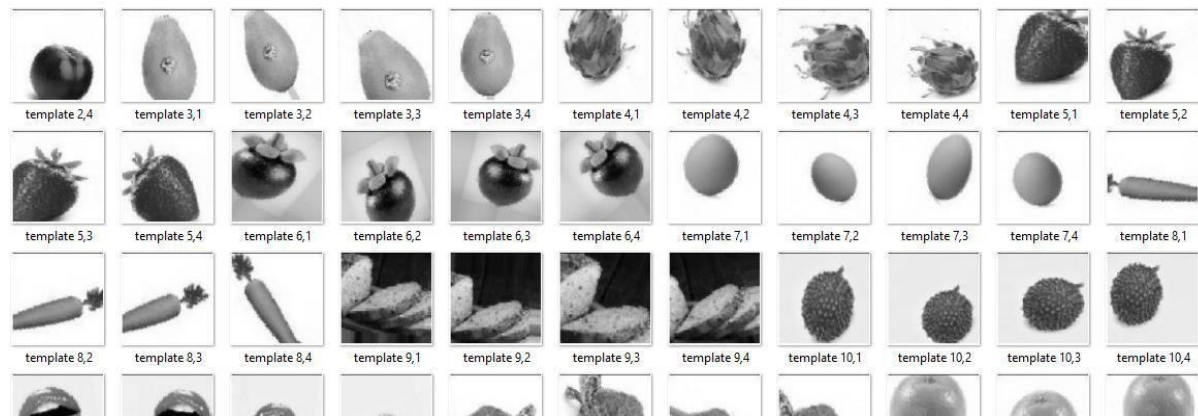
The images were first resized in a loop and saved. We were then able to make 4 different variants of the same image by applying different affine transformations on the image, thus expanding our dataset by 4x, and this can be increased to any number desired as the transformations are random.

Figure 25 is the augmented dataset for bananas:



**Figure 25: Augmented Banana Dataset**

The images were converted into grayscale using parallel segmentation as run-time decreased on using grayscale images, shown in Figure 26.



**Figure 26: Augmented dataset in B&W**

## 4.2 Advanced Pre-processing:

Leather pieces have an inherent texture. To extract defects from the textured image, it is necessary to apply filters which smoothen out the textures and increase the contrast so that the defects become prominent, making it easier for the machine to detect it. We applied Bilateral Filtering along with Histogram equalization on the images and achieved the following result:

Before:



**Figure 27: Leather Defect- Before**



After:



**Figure 28: Leather Defect- After**

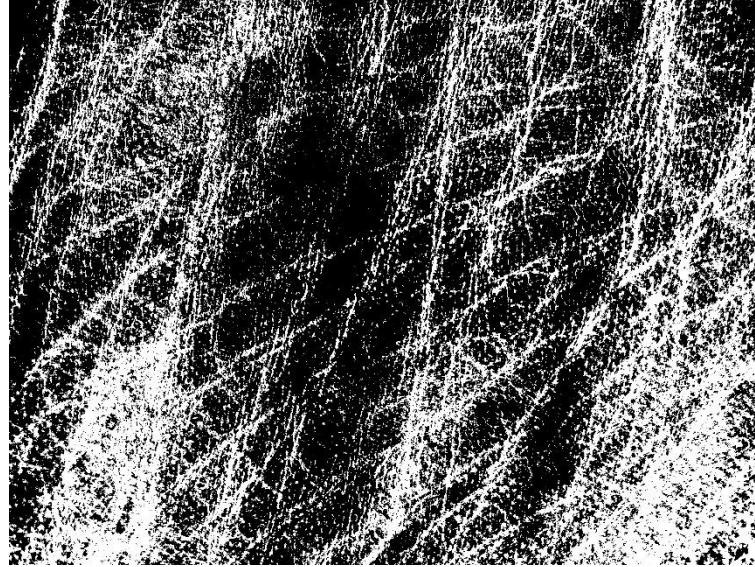
Before:



**Figure 29: Leather Defect 2- Before**



After:



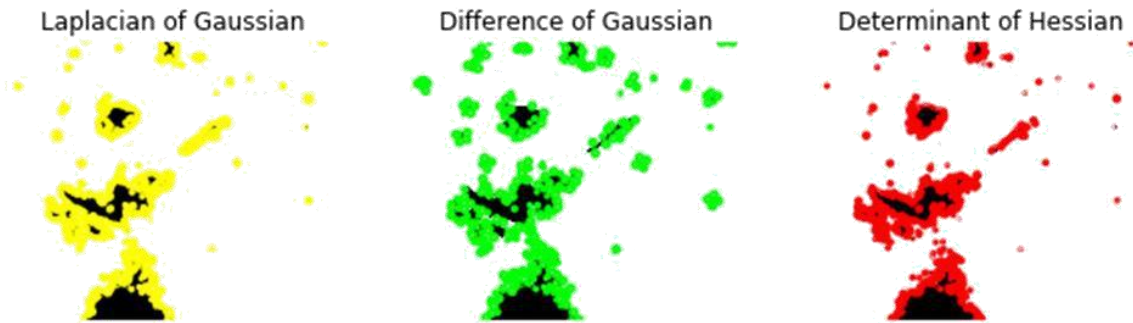
**Figure 30: Leather Defect 2- After**

We can see that the resultant image emphasizes the defective regions much more, smoothing out the textural information.

However, in some images with shadows, histogram equalization merely emphasizes the shadows more, leading to identification of defects wrongly. Another drawback is that Bilateral Filtering didn't work on subtle defects like pinholes and stretch marks which didn't stand out from the texture much.

### **4.3. Blob Detection**

The defects we are dealing with are mostly patches randomly spread on leather pieces. They can be treated as blobs after the pre-processing step. With this, our initial plan was to speed-up classification and find the area of fabric affected. The results of blob detection are shown in Figure 31.



**Figure 31: Blob Detection**

Though this method of blob extraction works indeed quite well for this particular image, the results weren't equally impressive in other defect images which had shadows in them, or images with defects not as prominent as scars. Even in the above image, the shadow is marked as a blob.

Another issue that we faced was that we needed to pre-process different types of defects differently. For example, we needed to extract the black regions of defects like scars and scratches, while we needed to extract the white regions of stretch marks. Thus, we would have to classify the defect before using this method, while we were trying to use this method to classify in the first place.

The biggest drawback, however, was how long this algorithm took. Running on spyder, the code takes close to 25 minutes for each image, and this is why we decided to shift to a different method.

## 4.4 Support Vector Machines

Support Vector Machines is a machine learning algorithm that is used to classify data. We implemented this algorithm for the augmented leather dataset with 50 images and were able to get 66.67% accuracy.

```

C:\Users\Lohia\source\repos\OPENCV3_TEMPLATE\Debug\OPENCV3_TEMPLATE.exe
User code starts
Train_HOG Size : 148
Descriptor Size : 8100
done
Kernel type      : 2
Type             : 100
C                : 3
Degree          : 0
Nu              : 0
Gamma           : 0.50625
the accuracy is :66.6667

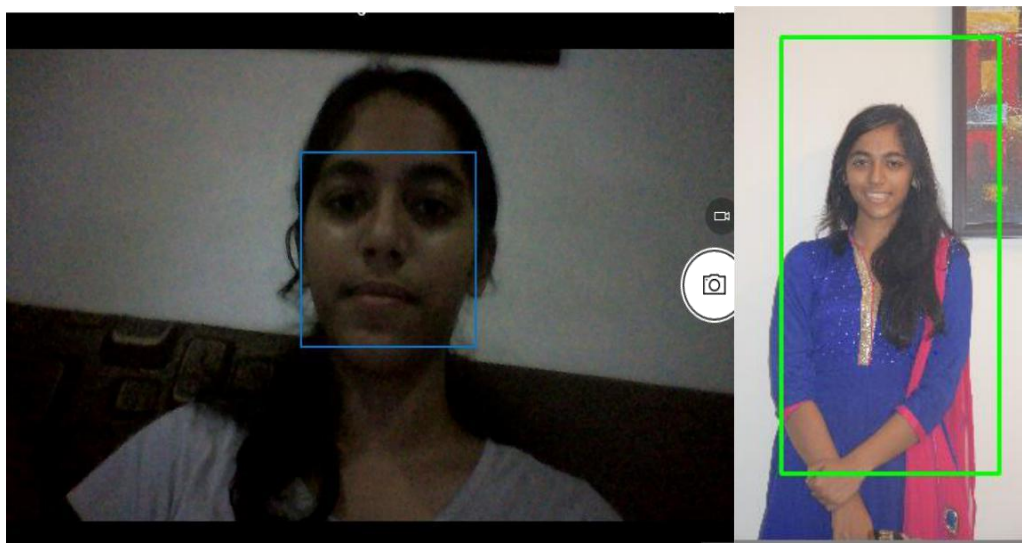
```

**Figure 32: Custom SVM result**

The main drawback of SVM was because of the small dataset. We were restricted by the dataset size. SVM could theoretically perform as well neural network algorithms given a big enough dataset, but since we couldn't obtain more images, we needed to switch to darkflow.

#### 4.5. Real time object detection

Since the detection of the defects will be in real time, we decided to implement an algorithm for face detection using a web camera. Although the code was slow, we were able to detect faces with 100% accuracy. Shown below are some of the screenshots of our result



**Figure 33: Facial recognition and Pedestrian detection**

However, this step couldn't be generalised to leather defects as making a custom Haar Cascade required an enormous dataset, and as mentioned before, HOG's also performed better on objects with a fixed aspect ratio which was absent in leather samples.

#### **(b) Darkflow Results:**

#### **4.6. Image Pre-processing:**

This is the result of the code to flip images automatically. The code takes only the images from a directory and flips them, as seen in Figure 34. We didn't use the augmentation with the rotation as the border reflection applied on leather distorted the defects too much to be of any practical importance.



**Figure 34: Flipped Leather dataset**

This augmented dataset proved to train the network much better than the initial dataset and the accuracy of the bounding boxes increased.

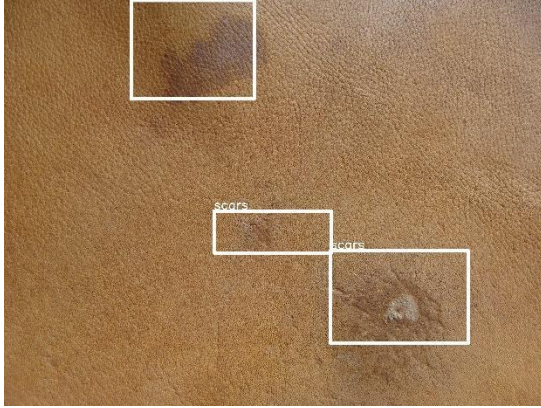
#### 4.7. Testing:

The results obtained after testing include the bounding boxes around the tested leather images and their coordinates in a json file. The json file has the top-left and bottom-right coordinates. It also includes the class (in this project, there is only one class, i.e. scars) and the confidence of its prediction. While testing, the threshold can also be specified to remove multiple detected instances. By default, the threshold value is set to 0.25. Thresholding picks the cells with the highest probabilities so that the more correct bounding boxes are selected. The command for applying threshold is given below

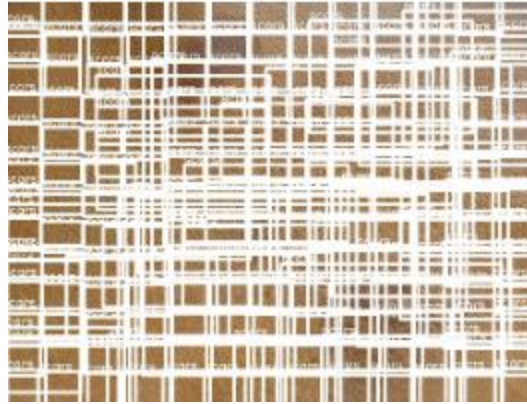
```
C:\Users\Lohia\Anaconda3\Lib\site-packages>python flow --imgdir C:\Users\Lohia\Anaconda3\Lib\site-packages\darkflow-master\test_images --model cfg/tiny-yolo-voc_1c.cfg --load 3625 --json --threshold 0.1
```

**Figure 35: Threshold**

The results obtained after applying two different thresholds are as shown in Figures 36 (a) and 36 (b)



**Figure 36 (a): Threshold - 0.19**



**Figure 36 (b): Threshold-0.05**

These test images are displayed on the visual studio form after the user presses the “test” button. Using the coordinates obtained in the json file, the area of the defective region is also calculated to infer the quality of the leather sample

#### 4.9. Windows Form Application:

The first page of the form asks the user to pick either administrator or employee depending upon what they wish to do. As explained earlier, the administrator can view the database, test, and train the model, but an employee can only view the database and test the model. Once the user picks a particular option, they are redirected to a login page where they need to enter their credentials. They go to the next form only if the username and password match.

**Figure 37: Login form**

The training form involves going to LabellImg and training the network using the command prompt (as seen earlier).

This is the look of the testing window. The user can select the folder in which the testing images are present and when he/she pressed the Test button, immediately we can see the defects with the bounding boxes, the number of defects, and defective area. When the user presses the 'next' button, the same process repeats for the next image and so on until no more images are left. When the user clicks 'Update in database', all the in information is immediately updated in the database, and both administrators and employees can view this data.


# Leather Defect Detection

Choose the test image

Browse

Test

Detected Defects



Next

Details

Image Name

No. of defects

2

Type of defects

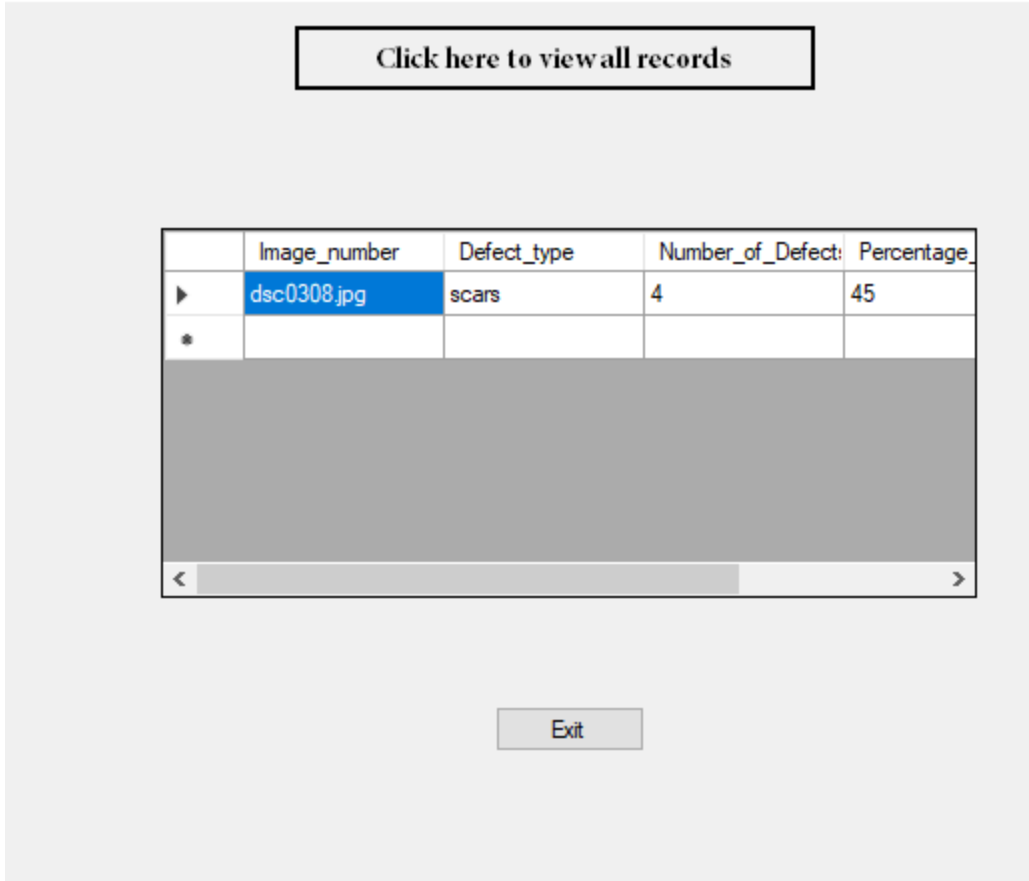
Defective area

4.488475

Update in Database

**Figure 38: Windows Form for testing**

The following figure shows the form to view the database. When the user pressed on the view button, information gets loaded from the database and is sent to the form in the form of a table.



**Figure 39: Windows Form for viewing**

## 5. Conclusion:

Leather Defect detection is not an easy task as it involves identifying defects which don't have a regular shape or size. Our initial plan was to use blob detection and SVM to classify leather defects but because of various reasons listed earlier, this didn't work out. Darkflow defect detection worked and we have made a GUI too for easy access. We do still have many plans for future work, like modifying hyperparameters on darkflow to obtain better results, optimising our GUI, design-wise and speed-wise, and maybe sending the detected regions to a deep learning network for classification too.

Other than these, the hardware-related work also remains. We have started working on capturing images from the camera with a code. We also plan to configure the hardware to make an automatically moving camera that captures images of the leather sample in real time. At the same time, we will keep looking for other methods to solve the problem of defect detection and see if any of these methods work better than the method we have proposed.



## 6. References:

<https://www.meccanismocomplesso.org/en/opencv-python-otsu-binarization-thresholding/>  
<https://github.com/detsikas/Texture-Segmentation-Using-Texture-Filters-and-OpenCV>  
<https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/index2.html>  
<https://www.bioss.ac.uk/people/chris/ch4.pdf>  
[https://docs.opencv.org/3.4/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html)