Fabric Defect Detection using Computer Vision

# Summary

This report aims to how computer vision could be used to find defect in narrow fabric. Utilising the open source technologies, OpenCV and TensorFlow to create 3 different inspection techniques and Qt to build a fabric inspection GUI.

\*\*\* pre-processing maybe \*\*\*

The three inspection techniques were all created in python using OpenCV. The first used created and examined histograms generated form the pixel values of the images. The second utilised image morphology and contour finding to look for large objects present in the image. The last method leveraged TensorFlow to build a CNN (Convolutional Neural Network) that was trained on pre labelled defect data obtained from the aitex fabric image database.

A prototype graphical application was then created using the second and third inspection techniques and the report discusses how this would be implemented in a full inspection system. Finally, the report compares the inspection techniques created to human inspection, the current method most companies use. \*\* explain findings \*\*

The report concluded \*\* conclusion \*\*

# Acknowledgements

I would like to thank my assessor \*\*name\*\* and both of my supervisors Amy Lowe and David Head who provided indispensable guidance throughout the project.

I would also like to thank my friends who provided much needed data around human inspection.

Finally I would like dedicate this report to my late father Charles, who inspired the idea for the project. \*\* add more about dad \*\*

Table of Contents

[Summary 1](#_Toc133508948)

[Acknowledgements 1](#_Toc133508949)

[Chapter 1: Introduction and Background Research 3](#_Toc133508950)

[Introduction 3](#_Toc133508951)

[Fabric 3](#_Toc133508952)

[Spinning 3](#_Toc133508953)

[Weaving and Knitting 3](#_Toc133508954)

[Dyeing 4](#_Toc133508955)

[AITEX Fabric Image Database 4](#_Toc133508956)

[Types of Defects 4](#_Toc133508957)

[Importance of Inspection 6](#_Toc133508958)

[Current Inspection Techniques 6](#_Toc133508959)

[Computer Vision 6](#_Toc133508960)

[Deep Learning 6](#_Toc133508961)

[Background research summary 6](#_Toc133508962)

[Chapter 2: Methods 6](#_Toc133508963)

[Initial Project Decisions 6](#_Toc133508964)

[Sprint 1: Data Preparation and Analysis 7](#_Toc133508965)

[Goals 7](#_Toc133508966)

[implementation 8](#_Toc133508967)

[Tiling the image 9](#_Toc133508968)

[Preparing the images 10](#_Toc133508969)

[Saving and labelling tiles 11](#_Toc133508970)

[Sprint Review 12](#_Toc133508971)

[Sprint 2: OpenCV Contour Finding 12](#_Toc133508972)

[Goals (\*\*maybe re do\*\*) 12](#_Toc133508973)

[Implementation 12](#_Toc133508974)

[Sprint Review 15](#_Toc133508975)

[Sprint 3: TensorFlow 15](#_Toc133508976)

[Goals 15](#_Toc133508977)

[Implementation 15](#_Toc133508978)

[Sprint Review 15](#_Toc133508979)

[Sprint 4: Gui 15](#_Toc133508980)

[Goals 15](#_Toc133508981)

[Implementation 15](#_Toc133508982)

[Sprint Review 15](#_Toc133508983)

[Chapter 3: Results 15](#_Toc133508984)

[Chapter 4: Discussion 15](#_Toc133508985)

[Bibliography 15](#_Toc133508986)

[Appendix A Self-appraisal 16](#_Toc133508987)

[Appendix B External Materials 16](#_Toc133508988)

[Appendix D User Testing Consent Form 16](#_Toc133508989)

[Appendix F User Manual 16](#_Toc133508990)

# Chapter 1: Introduction and Background Research

## Introduction

The purpose of this project was to explore how computer vision could be used in fabric defect detection and what advantages it could offer over traditional methods.

This project and report will investigate the effectiveness of different computer vision techniques at locating defects in images of fabrics. To carry out this investigation three different inspection techniques were created, the first two using OpenCV and the last using TensorFlow.

Lastly, a prototype application was created that would be used to automatically inspect a user supplied images using the most effective inspection methods found in the initial stages of the project.

## Fabric

Most fabric / textile is produced through one of a number of processes weaving, knitting, felting, bonding or turfing. Out of these the most common are weaving and knitting which produces most conventional fabrics. Both involve very similar steps, first fibres (synthetic or natural) are spun into yarn which is then converted into fabric using either weaving or knitting (Shaker,2016).

As these two production methods are the most widespread the projects definition of a fabric or textile was limited to those produced by either of these two methods. To understand the problem and what defects can occur, it was important to have a surface level understand of how these production technique’s function.

### Spinning

First fibres are harvested naturally for example flax or are produced synthetically such as polyester. These are then aligned and collected into yarn through the process of spinning, this process varies for natural and synthetic material. Natural fibres can be spun in many ways, but all involve them being twisted, this binds them together to form yarn (Smith, 1969, p. 1).

Synthetic fibres are spun differently, here liquid polymer is extruded though many, densely packed small holes. A cool “quench air” is passed over them to solidify the liquid polymer into long continuous fibres. Variations in the air passed over them also causes fibres to bunch in certain areas binging them together into yarn (Denn, 1983, pp. 179-180).

### Weaving and Knitting

After yarn is created, weaving and knitting are two processes that can be used to convert this yarn into fabric. Weaving is the process of interlacing yarn perpendicularly to each other. The patterns this is done in determine the properties and appearance of the textile (Adanur, 2020, p. 1). Terms used later in the project are warp and weft and they are key when understanding the weaving process. All woven fabric consists of warp and weft yarns, warp yarns are the yarns that run parallel to the edge / selvage of the fabric and are used to form its structure. Weft yarns are those running perpendicular to the edge of the fabric and are interlaced between the warp yarns using a loom (Lord,1982).

Knitting is the second most prevalent for of textile manufacturing. It is achieved by vertically intermeshing loops of yarn (Ray, 2012, p.2). Similarly, to weaving the way in which the loops mesh decides the properties and appearance of the final textile. While they processes differ Both methods aim to produce an ordered and reoccurring structure of yarn.

### Dyeing

Once these textiles are produced most if not all go through some level of processing. This may include dyeing a form of post processing that colours the fabric with a dye or pigment. There are many ways to dye a fabric but the common is the batch dyeing processes. The batch dyeing processes involves a textile being submerged in a large quantity of dye for an extended period, this allows the dye to transfer into the textile (Perkins, 1991, p. 23).

Many other finishing processes can also be applied such as coatings, but these are less prone to producing visible defects and so do merit discussion.

## AITEX Fabric Image Database

This project developed using the AITEX fabric image database( <https://www.aitex.es/afid/> ) and its accompanying article “A PUBLIC FABRIC DATABASE FOR DEFECT DETECTION METHODS AND RESULTS” (Silvestre-Blanes, 2019, pp.363-374). The article explains how the database was created and explores some detection methods. The database consists of 245 images ranging over 7 different types of woven fabric. It contains 140 images of defect free fabric, 20 each type of fabric. With 105 images of defective fabric spread between all 7 fabric and 12 defect types.

Usefully the database also contains 105 mask images for each image of defective fabric, theses mask images match the dimensions of the fabric images (4096×256 pixels). The masks contain only black pixels apart from the pixels that make up the defect, these are coloured white.

### Types of Defects

Now the project has a baseline understanding of how fabric is created and a database to refence from we can assess what defects are produced during the phases of its creation.

The twelve defect types present in the AITEX database are:

* Fuzzy balls. Fuzzy balls are the result of warp yarns that are too small lowering their abrasion resistance allowing fibres to break loose and form fuzz balls. According to cotton works list of defects.
* Neps. Neps are formed by an accumulation of fly and fluff on the machinery used and result in damage the fabrics appearance when dyeing (Nateri, 2014). Often caused in the spinning process.
* Knots. Knots occur in natural or synthetic warp yarns and damage the appearance and tensile strength of the textile. Knots can also refer dead or immature natural fibres (Lim, 2018).
* Crease. A crease is a valley or ridge present in the final fabric caused by folding or improper tension.
* Broken End. A broken end appears as an untied or broken warp end and manifests as horizontal lines in the lines in the fabric. The yarn is usually broken during weaving or finishing processes (Lim, 2018).
* Broken pick. A broken pick defect consists of a broken weft yarn and results in a sudden discontinuity in the weave (Lim, 2018).
* Broken yarn. Broken yarns are breaks in either the warp or weft yarn during the weaving process and result in elongated holes.
* Contamination. Contamination refers foreign objects suck too or woven into the textile. Even if the object is later removed it may have disturbed the weave or cause uneven dyeing.
* Warp ball. A warp ball is caused by a single or multiple warp yarns becoming clumped or entangled.
* Cut selvage. Selvage a is the densely woven edge of a piece of fabric a cut in the selvage can cause a separation in the weave (Lim, 2018).
* Weft Curling. Weft curling Is produced by the use of highly twisted weft yarn as a result it disturbs the pattern of the weave. According to Textile Tutorials.
* Weft Crack. A weft crack results in a narrow streak running in parallel with the weft caused by an absence of weft yarn. According to Textile Sphere.

Calendar

Description automatically generated with low confidenceFigure 1. ROI of 256 x 256 pixels from original images of defective fabrics, with the names used in the database. (a) broken end, (b) broken yarn, (c) broken pick, (d) weft curling, (e) fuzzy ball, (f) cut selvage, (g) crease, (h) warp ball, (i) knot, (j) contamination, (k) nep, and (l) weft craft. ROI, region of interest (Silvestre-Blanes, 2019).

Figure - Defect Images (Silvestre-Blanes, 2019)

### Importance of Inspection

Fabric inspection is important for a multitude of reasons, the upmost being the prevention of inferior quality product being sold to or used by consumers (Malek, 2013). Inferior or defective product being used or sold can cause a variety consequence for the producer.

Inspection at early stages, straight after weaving, can finically be benefit the producer it stops the offending sections being processed further or being sold to those who will apply finishing processes. Supplying defective fabric is likely to cause animosity directed towards the fabric supplier, worsening customer satisfaction and possibly affecting future revenue.

The garment industry represents one largest purchasers of textiles, they need assurance the raw materials they purchase are defect free. Many of the designer brands will expect or enforce that textiles be inspected before purchase and are unafraid to hold producers accountable, as their clientele expect a level of quality.

Defects are not only cosmetically undesirable many effect the structure of the weave and so can degrade its tensile strength. Ultimately leading to textiles being distributed with a lower tensile strength than advertised, creating the potential for catastrophic failures. For example, it is key that seat belts or ratchet straps have no structural defects.

## Current Inspection Techniques

Currently fabric inspection techniques vary, the majority use human inspection. Human inspection uses workers to scrutinise fabric by hand. The accuracy of human inspection declines over time due monotonous nature of the task. This results in slower, more expensive, and erratic inspection (Chin, 1982).

Many in the industry paired a use of human inspection with inspection using DSP (digital signal processing). DSP would analyse the waveform produced by a sensor pointed at the fabric in controlled lighting. However, most are now moving towards computer vision approaches as they easier to implement and are more capable at inspecting a variety if fabrics.

## Computer Vision

Computer vision is defined as the processes of allowing computers to see the real world (Learned-Miller, 2011). Computer vision has proven key when automating the workplace.

### Deep Learning

## Background research summary

# Chapter 2: Methods

The goal of this chapter is to address the design and implementation of the three inspection techniques mentioned earlier and create the prototype inspection application. All were completed in python utilising OpenCV, TensorFlow and PyQt5.

## Initial Project Decisions

This projected was developed using the agile methodology, using sprints to manage milestones. This method suited the project as in any form of inspection results and the times in which they are achieved are not guaranteed. This allowed some area of the project to always be in active development, even during long model training periods or when certain avenues of thought lead to dead ends.

Version controlled was used throughout and was facilitated with a GitLab repository. <https://gitlab.com/TomSchofield/personalproject>

The decision to use the aitex image fabric defect database informed what detection methods would be perused and how they would be evaluated. There was another choice, the University of Moratuwa fabric defect database (Belkhir, 2020). This included a far greater amount defect of data however it was split between only 5 defect types: colour, cut, hole, thread, metal contamination. All images from this dataset are also all from one type of fabric (figure 2). For these reasons and the fact the type of fabric is less widespread than that found in the aitex database, it was decided that while accuracy might suffer the inspection methods produced would be more general if the project used the aitex database.

Figure - Sample Fabric (Belkhir, 2020)

One of the initial ideas for the project was to include some form of defect categorisation, this would separative images already deemed as defective into individual categories. However once development commence it was decided that this was out of the scope of the project, the primary reason for this is the lack of large enough datasets. The dataset chosen contained at most 20 images of each defect type and so does contain enough data to train a defect categorisation algorithm and achieve a reasonable accuracy. Hence the decision to focuses efforts on defect detection.

## Sprint 1: Data Preparation and Analysis

### Goals and Design Decisions

The goals for the first sprint were to prepare and investigate the dataset.

Figure - image 0001\_002\_00



Figure 3 shows an example image form the dataset; all images are standardised to a size of 4096×256 pixels. This presents two main problems the first is that the pixels that make up the defect represent a very small proportion of the overall image. The second is that any operations on such a large image would take a substantial amount of time.

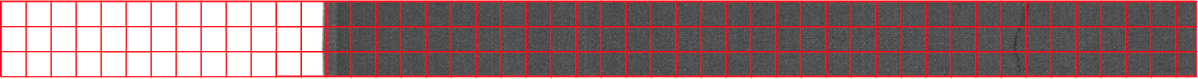


Figure - Tiled 0001\_002\_00

Figure 4 shows the solution to this issue, to segment the image. Once the image is segmented these tiles could be processed individually. furthermore, defective pixels represented a much larger proportion of the tile, they reside in.

However, when examining figure 4 a flaw in this methodology appears, we can see that approximately the first quarter of the image is the background behind the fabric. Hence the tiles in the section contain no meaning full information and would probably be miss classified as detective. To remedy this the background must be removed.

The last goal will be to store the created tiles as either a defective or normal. However this presents some issues as not all tiles from a defective image are themselves defective, the mask images from the database will likely be useful here.

### implementation

#### Tile Class

Once the data was downloaded to implement the tiling routing was done by creating a class to represent the tile.

class Tile:

    x = 0

    y = 0

    width = 0

    height = 0

    imagex = 0

    imagey = 0

    roi = None

    overlap = 0

Code Snippet 1 – Tile Class Attributes

This Tile class holds all the key attributes needed when describing the contents and location of the segment in relation to the master image. The attributes “x” and “y” hold the position of the tiles in relation to the other tiles. Whereas “imagex” and “imagey” hold the position of the tile in relation to the pixels of the master image. For example a the second tile on the second row would have “x” and “y” value of 1. However, assuming a tile “width” and “height” of 64 and a “overlap” of 0 the same tile would have the value of 64 for both the “imagex” and “imagey”. Overlap simply describes the amount the tiles overlap and is calculated from the right most corner of the previous tile. Figure 5 shows of overlap is visualised.

A picture containing rectangle

Description automatically generated

Figure - Overlap Visualisation

    def populate(self,master):

        self.roi = master[y:y+height, x:x+width]

        if (self.roi.shape[1] < width):

            new\_x = x - (width-self.roi.shape[1])

            self.roi = master[y:y+height, new\_x:new\_x+width]

            self.imagex = new\_x

        if (self.roi.shape[0] <height):

            new\_y = y - (height-self.roi.shape[0])

            self.roi = master[new\_y:new\_y+height, x:x+width]

            self.imagey = new\_y

Code Snippet 2 – population method

Code snippet 2 shows the implementation of the populate method of the tile class. It takes the master image as a parameter and calculates which segment of the image should be copied into the tile’s “roi” (region of interest). The rest of the snippet relaxes the overlap constraint if the tile is not the correct size. This happens at the edge of the master image where there aren’t enough pixels to fill an entire tile.

### Tiling the image

\*\*insert code if space\*\*

Tiling the image is quite simple and is done inside the “tileImage()” function. First the number of tiles that should be created is calculated this is done by finding the number of tiles in bot the x and y axis. The number of tiles in the x direction is found by rounding up to the nearest integer the width of the image divided by the width of the tile minus the overlap. The same can be done to find the number of tiles along the y axis using the image height and tile height.

The number of tiles in either direction is the looped over and each tile is created and populated, the x and y coordinates if the tile in relation to the pixels in the master image is calculated by multiplying the x and y coordinates in relation to the other tiles times the height or width of the tile minus the overlap. These were then placed into a 2d array so they can be easily accessed.

### Preparing the images

As described in the goals of this sprint the images needed to be cropped to remove the background. Unfortunately, the exact location of the edge of the fabric was not in the same location for each image. First the images needed to be loaded in from the memory to do the OpenCV function “imread()” when supplied with a file path loads the image into a NumPy array. A NumPy array functions very similarly to a C or C++ array, they are fast accesses and have strict requirements on the homogeneity of the objects. However, they do allow some functionality of a python list such as the ability to append to them.

A picture containing timeline

Description automatically generatedOnce the image was loaded in the subroutine “findEdge()” was created. “findEdge()” takes an image in as a parameter and applies three processes to determine its edge first a light blur is applied to the image, then a threshold is applied and finally Canny edge detection is used to find all vertical edges present in the image.

Figure - Edge Finding Steps

Figure 6 demonstrates the steps involved when finding the edge of the fabric. (b) displays the conversion of the image to greyscale, this is needed to apply thresholding. Conversion to greyscale is simple achieved using a build in OpenCV function, it changes the given image from three channel colour (blue, green, red) to a single channel. This single channel ranges from 0 (black) to 255 (white). Blurring was also achieved using the OpenCV function “blur()”, this function receives the image to be blurred and the size of the kernel that should be used to blur and uses the widely know box blur algorithm. Box blur was used as it is substantially faster than other methods such as the Gaussian blur. The projected used a kernel of size 3 by 3, meaning each pixel was replaced by the average of the eight pixels surrounding and itself (Szeliski, 2022). The purpose of this blur is to reduce is to reduce noise or in this chase obscure the weave of the textile. This is needed as light spots can confuse the thresholding algorithm and vertical lines in the weave can confuse the canny algorithm.

(c) displays the output of the thresholding algorithm, once again this functionality is offered by an OpenCV function. The “threshold()” function again takes the parameters: the greyscale image to be processed, the threshold value, a maximum value and a tag representing the type of thresholding to be applied. The project uses binary inverse thresholding, any pixel in the image greater than the thresh hold value is set to 0 and any value less than the threshold value is set to the maximum value. The maximum value used in the project was 255 but any value would have been suitable and a threshold value of 240 as while the pixel values of the background were 255 some leeway should be left for inconsistency in the lighting of the images. The purpose of applying a threshold is to remove any vertical lines form the weave of the fabric leaving only the line between the background and fabric, removing any noise that may confuse the canny edge detection algorithm.

(d) the Canny edge detection algorithm again is offer by an OpenCV function. Canny edge detection works in stages, it first smooths the image to remove noise then computes the edge strength and edge direction. In simpler terms the edge strength and direction are found though the gradient of the image, a transition from black to white or vice versa is considered a positive or negative gradient. Edges usually produce a high magnitude gradient and so can be recognised as an edge (Xu, 2017). The OpenCV function uses further processing to clean up the edge. The “Canny()” function returns a list of pixels that constitute the edge, the innermost pixels location is then used to crop the image lengthwise. (c) The dimensions of the cropped images were then found using the “.shape” attribute of a NumPy array and the corresponding mask image was cropped to the same dimensions (f).

### Saving and labelling tiles

Now the images had been cropped the individual tile needed to be labelled and saved. Labelling is the processes of designating a class to each data item, in the context of this project this would be designating as either a defective or normal. As each image is broken down into approximately 540 individual tiles and the database consists of 245 images hand labelling would be infeasible. However, the cropped mask images can be used to determine which tiles include defects. All the mask images consist of only black and white pixels, the black pixels substitute any pixel in the original image that is not part of the defect. White pixels take the place of the pixels that do make up the defect.

As the mask were cropped along with the original images when segmented the tiles of both match. Figure 7 shows this relation.

Chart

Description automatically generated

Figure - Mask and Tile

Hence, if any mask tile contains any white pixels, its corresponding unmasked tile could safely be labelled as defective and those containing no white pixels as normal. In practice this method had one problem even those masks which contained only a single white pixel would be treated as defective, this is far too strict and could confuse later inspection methods. To remedy this only mask tiles that contained over 500 white pixels would be treated as defective.

This increased the number amount of data available from 245 large images to 909 defective tiles and 113,307 normal tiles at a more manageable size. This segmentation will allow for more detailed testing and training of inspection methods in later sprints.

### Sprint Review

At the end of this sprint all goals were achieved. There now existed functions that can successfully read images form the database, crop them, tile them, and save the tiled for later use.

During this sprint no detection methods were produced however all pre-processing was now complete making development of more detection methods faster.

## Sprint 2: Beginning Inspection

### Goals and Design Decisions

The goal of this sprit was to develop an analytical approach to inspection by comparing the properties of tiles to a known good tile.

The main goal for this sprint was to develop an inspection technique using the principle of object detection. To achieve this, a way to reduce noise in the image must be developed as well to find groups of similarly coloured pixels.

### Implementation

#### Histogram Generation

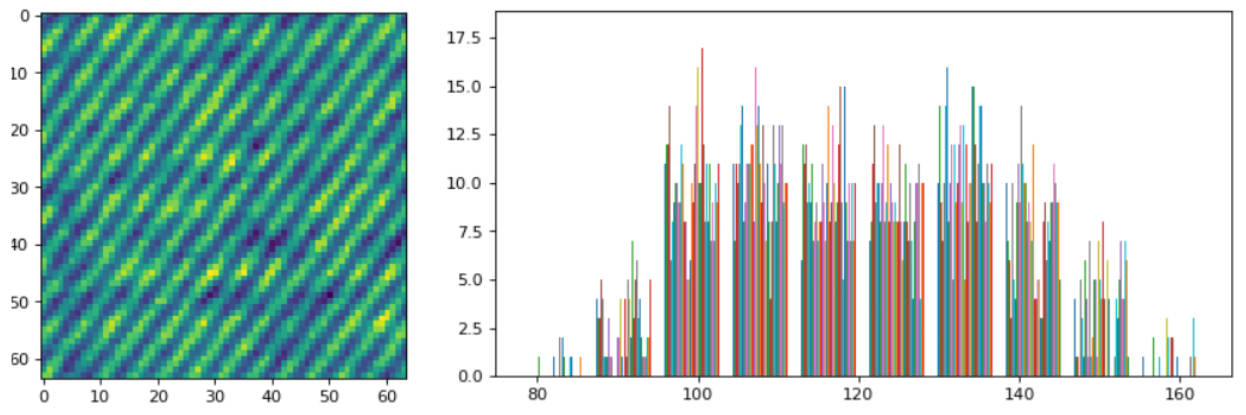
The first form of defect detection attempted was to analyse the pixel values of each tile of the image and compare them to a know good tile. A histogram of the pixel values from a greyscale conversion of the tile was created using the matplotlib “hist()” function.

Figure - Histogram of Non-Defective Tile

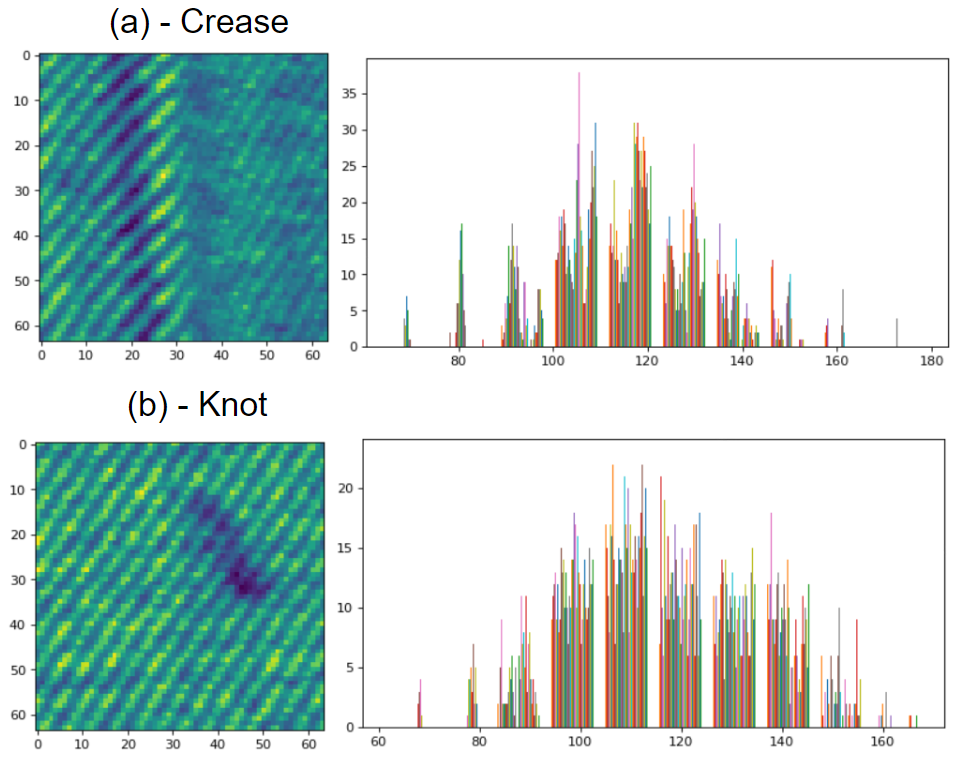
Figure 8 shows the histogram of a non-defective tile.

Figure - Histograms of Defective Tiles

Figure 9 displays the two defective tiles a (a) crease and a (b) knot and their associated histograms.

The histograms illustrate that some defective tiles differ in key metrics such as mean pixel colour or the range of the pixel values from ideal tiles. These could be used to identify defective tiles.

#### Statistical Analysis

Two forms of statistical analysis were attempted, the first was an attempted to use the range and standard deviation of the pixel values within a tile to deduce whether its defective. The tiles greyscale image data inside the tiles are stored as NumPy array and so the standard deviation can be simply found using a the NumPy function “np.std()”. Range was found using a “findRange()” function created during the project. The function loops through all the pixel in the array and stores both the lowest and highest values found. The lowest can then be subtracted from the highest to return the range.

Example ranges and standard deviations of example non-defective tiles are as follows:

|  |  |  |
| --- | --- | --- |
| Normal Tile Number | Standard Deviation | Range |
| 1 | 16.448 | 81 |
| 2 | 16.736 | 85 |
| 3 | 11.917 | 70 |
| 4 | 18.832 | 94 |
| 5 | 27.395 | 128 |
| Average: | 18.266 | 91.6 |

Example ranges and standard deviations of example defective tiles are as follows:

|  |  |  |
| --- | --- | --- |
| Defective Tile Number | Standard Deviation | Range |
| 1 | 16.289 | 114 |
| 2 | 16.839 | 107 |
| 3 | 17.569 | 107 |
| 4 | 17.099 | 113 |
| 5 | 14.642 | 99 |
| Average: | 16.488 | 108 |

These are discussed further in results section however it’s clear that comparing the range and standard deviation of tiles would not prove an effective form of inspection.

The next approach attempted was to reduce the colour space of the image. The idea was to reduce the colour space to a set of intervals and count the number of instances inside each interval with the hope that defective tiles will have abnormally high pixel counts in certain intervals. This was achieved by rounding pixel values to the closest interval.

The “roundTile()” functions is supplied with the tile to be processed and the intervals the pixels should be rounded too. This then calls the “roundPixel()” function on each pixel un the tile, this takes the absolute difference between the pixel value and each interval and returns the interval that produced the lowest difference.

Polygon

Description automatically generated

Figure - Colour Space Reduction and Plot

Figure 10 displays an original tile (a) the result of the colour reduction processes (b) and the plot of pixel colour against the occurrences of pixels of that colour in the colour reduced tile. It illustrates that the majority of pixels have a value of either 100 or 120.

Chart, line chart

Description automatically generated

Figure - Normal against Defective Plot

Figure 11 plots the occurrence of pixel colours from five good tiles and six defective tiles. It shows that while the example defective tiles do contain a larger number of there pixels towards the centre of the colour space, unfortunately so do many normal tiles.

#### Blob and Line Detection

OpenCV has many functions designed to assist in object detection, many of these functions will help when looking for defects as most if not all present as a large mass similarly coloured pixel. First the “SimpleBlobDetector” class provided by OpenCV was used but after many attempts and many different permutations of the “SimpleBlobDetector” parameters it could not detect even the most obvious defects.

#### Noise Reduction

Contour finding algorithms first need an image with very little noise. However as stated previously in the project images of fabric are inherently noisy do to the weave of the fabric. In order to reduce the noise a number od steps can be taken.

def findDefect(img, threshHold, pixThresh, lightBlur, errode, blur):

    exitCode = 0 # an exit code of 0 means a blob has been detected

    greyMaster = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # grey blob defect

    # Light blur to reduce size of small shadows and highlights

    lightBlur = cv2.blur(greyMaster, (lightBlur,lightBlur))

    # adjusted = setLims(lightBlur, 0 ,110) # massively increases time to run

    ret, adjusted = cv2.threshold(lightBlur, pixThresh,pixThresh,cv2.THRESH\_TRUNC)

    kernel = np.ones((errode,errode),np.uint8) # forms the matrix used when eroding

    erosion = cv2.erode(adjusted,kernel,iterations = 1)

    # large blur to hide background weave and increase the size of defects

    greyBlur = cv2.blur(erosion, (blur,blur))

    #threshold on Gray image

    thresh = cv2.adaptiveThreshold(greyBlur,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 101, 3)

    # Apply morphology open then close

    kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5,5))

    blob = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel)

    kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (9,9))

    blob = cv2.morphologyEx(blob, cv2.MORPH\_CLOSE, kernel)

    # invert blob

    blob = (255 - blob)

    # Get contours

    cnts = cv2.findContours(blob, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

Code Snippet 3, first half of the “findDefcet()” function.

First a light blur is a applied using the “blur()” function shown in code snippet 3 to shrink highlights and shadows and obscure the weave. This blur is achieved in the same way as blurring in the first sprit was and the factor the image was blurred by is controlled by the “lightBlur” parameter.

Highlights and shadows produced by inconsistent lighting needed to be removed. This was done by the function “setLims()” this function is supplied with an image and a lower and upper bound. The image is then iterated through, and new image array is created each pixels is appended to the new image array. However, if any pixel has a lower value of than the lower bound supplied it is appended as the lower bound, if any pixel has a value higher than the upper bound it is appended as the upper bound. Once every pixel had been processed the new image array is returned. The maximum and pixel threshold are both determined by the parameter “pixThresh”.

While this implementation worked its sequential implementation proved too slow and instead OpenCV thresholding was used to remove highlights. The thresholding was done using the same “threshold()” function, using a max and threshold value of 110 and the “THRESH\_TRUNC” tag. This tag changes how the function functions, values lower than the threshold are kept and not set to 0.

A picture containing arrow

Description automatically generated

Figure - Morphology Examples (https://docs.opencv.org/4.x/d9/d61/tutorial\_py\_morphological\_ops.html)

The next step to reduce noise was to start using image morphology. Several morphological transformations were used starting with erosion. Figure 12 demonstrates how erosion effects an image, the erosion algorithm attacks the boundaries of an object and leads to small object being completely removed. It does this by moving a kernel (2d matrix) across the image the anchor pixel, usually the centre pixel, is replaced by the minimum pixel value inside the area described by the kernel. The size of the kernel decides the amount the of erosion that takes place. Dilation, also shown in figure 12, works in a very similar way however the anchor pixel is replaced by the maximum pixel value instead increasing the size of objects. A heavier blur is then applied to reduce or even eliminate any remaining background noise and increase the size of defects that were just shrunk by erosion.

Adaptive thresholding is then applied, this is very similar to the thresholding done previously in the project, but instead of using a global threshold value chucks of the image are processed individually, and a local threshold value is computed for each.

To forms of morphology are then applied consecutively Opening and Closing. Opening applies erosion followed by dilation this again reduces noise and then increases the size of the defect. Closing applies dilation followed by erosion and fills in any gaps inside the defect that have been created from previous operation.

OpenCV contour finding is then used outline or bound the shape of the defect or defects. It works by finding all points on the boundary of an object if these points then form a loop a contour is found. This works best when used on binary images, where the object is white, and the background is black. This is another reason why adaptive thresholding was applied and why the image is inverted after the morphological transformations.

The end of the function simply checks if the size of any of the contours found is over a given value and is so the tile is deemed defective and an “exitCode” of 0 is returned.

A picture containing text

Description automatically generatedHowever, the was on major problem with this implementation, lighter defects would not be found. The initial thresholding was needed to remove highlights but would also remove lighter defects. This was easily fixed by ruining “findDefect()” function twice each tile, inverting the tile after the first call transforming any light defects into darker defects which are more easily detected. This was implemented in the “twoPassInspection()” function.

Figure - Inspection Steps

Figure 13 displays the result of each stage of the inspection function, while some steps may seem unnecessary, they become more necessary in smaller or more complex defects.

### Sprint Review

This sprint was moderately successful many of the goal were achieved. There now existed a method to analytically compare the segments of an image using several pixel intervals. However, this method could not accurately determine whether an image included a defect.

This was expected, the progress was made on the more complex inspection technique using image morphology and contour finding proved more effective at determining defects. However, its accuracy could improved upon by tuning many of the internal parameters that control the factor of blurring or eroding. This tuning would need to be done in a later sprint.

## Sprint 3: Parameter Tuning and TensorFlow

### Goals and Design Decisions

For this sprint the goals were to tune the contour finding inspection technique to increase it accuracy and start development on an inspection technique leveraging deep learning. The tuning will be done by exhausting lists of parameters until the highest accuracy is found. While this technique will likely be slow it will find the exact combination of parameters needed to find the maximum accuracy the method is capable of.

For the third and last inspection techniques developed in the project it was decided that some form of neural network would be trained to categorise image segments / tile as either defective or normal. A convolution neural network was the exact method chose. Hence second goal for this sprint is to develop an appropriate layer structure for the CNN and then train it. The CNN would be trained on the tile / segments of an image as the defects represent a larger proportion of the image. It also would increase the number of each label in training set. This will likely increase the inference time however accuracy was the key metric in this project.

### Implementation

#### Parameter Tuning

\*\*Insert code snippet if space\*\*

To make sure the second inspection technique was as accurate as possible multiple sets of parameters were looped through. So that this optimising could be done in a reasonable amount of time it was split up in to two sets of nested for loops. The first set was consisted of two for loops that iterated over two lists [50,100,150,200,250,300] and [50,60,70,80,90,100,110,120,140]. The first list decides the minimum size of a defect and the second contains what pixel value should be used when thresholding. The upper and lower bounds of the list was found through manual testing.

The Second set of for loops consisted of three nested for loops. The first two iterated over values 1-10 and controlled the factor of the first blurring and erosion filters. The third iterated over values 1-20 and controlled the amount of the heavier blur.

For each set of parameters 1000 randomly selected normal tiles and all 900 defective tiles were put through the inspection function and the accuracy was calculated. The set of parameters that produced the highest accuracy was then saved for later use.

#### CNN

### Sprint Review

## Sprint 4: Inspection Application and Human Testing Application

### Goals and Design Decisions

The goals of this sprint were developing a prototype inspection application and application to test the accuracy of human inspection. The inspection application will use the morphology inspection technique and the convolutional neural network. It must allow the user too:

* Import a set of images to be inspected.
* Choose which inspection technique is used.
* If the neural network inspection is chosen, the user can choose which model to use.
* Control the parameters of the chosen inspection technique.
* Graphical user interface, application

  Description automatically generatedView the defects found.

Figure - Inspection Application Wireframe

I chose to import images rather than connect to a live camera feed as it enables the user to quickly inspect a variety of fabric without having to reset a machine. While this would not be ideal in a production environment it is optimal for testing.

The application to test human inspection should be simple, it should display several images in a random order some with defects and some without. For each image the user should select whether they believe the image to include a defect or not. The application should display how the user performed so it can be recorded. This application should be as minimal as possible as to not take up development time that could be used form more important tasks.

Graphical user interface

Description automatically generated with medium confidence

Figure - Human Testing Application Wireframe

I used PyQt to develop both as the use of the designer tool allows for rapid development.

### Implementation

#### Inspection Application

Graphical user interface

Description automatically generatedQT designer was used to create a basic layout for the application, consisting of a number of Qt widgets: labels (used to display both images and text), check boxes, spin boxes (allow the users to input numerical data) and button. Once converted into a python file these buttons, check boxes and spin boxes could be connected to functions.

Figure - Inspection Application User Interface

The image data is loaded after the user specifies the folder they are stored , no background removal is incorporated into this prototype so any images must have been pre-processed beforehand. The images are moved into main memory by listing all the file names in the specified folder and calling the OpenCv “imread()” function. This loads all the image at once enabling faster accesses when inspecting at the cost hardware utilisation.

The model used for the neural network inspection technique loaded at the applications startup. Two models are loaded both trained during the third sprint, one is trained on only a set of tiles and the other trained using the same set of tiles with data augmentation applied.

Inspection begins once the start button is pressed, an image is tiled and each tile run through the selected inspection techniques, an array of the defective tiles found is then returned and displayed to the user. Whenever the “next” button is pressed the next defect found is displayed to the user until all the defects in the list have been displayed. “Start” can then be repeatedly pressed to inspect images in turn each time displaying the defective tiles found.

#### Human Testing Application

This application was developed very similarly to the last. The filenames from two predetermined folders are stored in a list, they contain names of 8 defective images and 8 normal images. The user is then presented with a simple screen only contain a start button. Once pressed the application uses the time library of python to obtain the current time.

Timeline

Description automatically generated with low confidenceThe start button is then hidden, the first image to be inspected by the user is displayed and the application wait for the user to press either the “defect” or “normal” button. When pressed the application checks whether the file name of the image being displayed belongs to the folder of defective or normal images and increments the appropriate variable out of “normalCorrect”, “normalIncorrect”, “defectCorrect”, “defectIncorrect”. Once all 16 images have been inspected these variables are displayed to the user along with an accuracy calculated by:

### Sprint Review

# Chapter 3: Results

Example ranges and standard deviations of example non-defective tiles are as follows:

|  |  |  |
| --- | --- | --- |
| Normal Tile Number | Standard Deviation | Range |
| 1 | 16.448 | 81 |
| 2 | 16.736 | 85 |
| 3 | 11.917 | 70 |
| 4 | 18.832 | 94 |
| 5 | 27.395 | 128 |
| Average: | 18.266 | 91.6 |

Example ranges and standard deviations of example defective tiles are as follows:

|  |  |  |
| --- | --- | --- |
| Defective Tile Number | Standard Deviation | Range |
| 1 | 16.289 | 114 |
| 2 | 16.839 | 107 |
| 3 | 17.569 | 107 |
| 4 | 17.099 | 113 |
| 5 | 14.642 | 99 |
| Average: | 16.488 | 108 |

\*\* Talk about, how converting to greyscale in the CNN and maybe a size reduction layer. This could massively reduce the training and inference time.

Explain different defect metrics, explain difference between tile and image accuracy.

# Chapter 4: Discussion

# Bibliography

Shaker, K., Umair, M., Ashraf, W. and Nawab, Y. (2016) Fabric manufacturing. Physical Sciences Reviews, Vol. 1 (Issue 7), pp. 20160024. <https://doi.org/10.1515/psr-2016-0024>

P. A. Smith B.Sc., Ph.D., F.T.I. (1969) YARN PRODUCTION AND PROPERTIES, Textile Progress, 1:2, 1-117, DOI: [10.1080/00405166908688985](https://doi.org/10.1080/00405166908688985)

Adanur, S., 2020. Handbook of weaving. CRC press.

Ray, S.C. ed., 2012. Fundamentals and advances in knitting technology. CRC Press.

Perkins, W.S., 1991. A Review of Textile Dyeing Processes. Textile Chemist & Colorist, 23(8).

Silvestre-Blanes, J., Albero-Albero, T., Miralles, I., Pérez-Llorens, R. and Moreno, J., 2019. A public fabric database for defect detection methods and results. *Aitex.* [Online]. [Date Accessed]. Available from: <https://www.aitex.es/afid/>

Silvestre-Blanes, J., Albero-Albero, T., Miralles, I., Pérez-Llorens, R. and Moreno, J., 2019. A public fabric database for defect detection methods and results. *Autex Research Journal*, *19*(4), pp.363-374

Nateri, A.S., Ebrahimi, F. and Sadeghzade, N., 2014. Evaluation of yarn defects by image processing technique. *Optik*, *125*(20), pp.5998-6002.

Malek, A.S., Drean, J.Y., Bigue, L. and Osselin, J.F., 2013. Optimization of automated online fabric inspection by fast Fourier transform (FFT) and cross-correlation. *Textile Research Journal*, *83*(3), pp.256-268.

2023. Cotton Works. [online]. [Accesses 20April 2023]. Available from: <https://www.cottonworks.com/en/defect/0121/>

Lim, S.L. 2018. 23 Fabric Defects To Look Out For During Fabric Inspection. [online]. [Accesses 20April 2023]. Available from: <https://www.intouch-quality.com/blog/5-common-fabric-defects-prevent>

Textile-tutorials. 2018. Types of Fabric Defects with Causes and Remedies. [online]. [Accesses 20April 2023]. Available from: <https://textiletutorials.com/types-of-fabric-defects-with-causes-and-remedies/#:~:text=The%20weft%20curling%20defect%20is,the%20surface%20of%20woven%20fabric>

Textile Sphere. [online]. [Accesses 20April 2023]. Available from: <https://www.textilesphere.com/2021/03/weaving-yarn-processing-fabric-defects.html#:~:text=Weft%20Crack%3A,due%20to%20absence%20of%20weft.&text=Broken%20filament%3A,the%20main%20yarn%20are%20broken>

Lord, P.R., 1982. *Weaving: Conversion of yarn to fabric* (Vol. 12). Woodhead Publishing.

Chin, R.T. and Harlow, C.A., 1982. Automated visual inspection: A survey. *IEEE transactions on pattern analysis and machine intelligence*, (6), pp.557-573.

Learned-Miller, E.G., 2011. Introduction to computer vision. University of Massachusetts, Amherst.

Belkhir, N.B. 2020. Textile Defect Detection. Kaggle. [Online]. [Date Accessed]. Available from: <https://www.kaggle.com/datasets/belkhirnacim/textiledefectdetection>

Szeliski, R., 2022. *Computer vision: algorithms and applications*. Springer Nature.

Xu, Z., Baojie, X. and Guoxin, W., 2017, October. Canny edge detection based on Open CV. In *2017 13th IEEE international conference on electronic measurement & instruments (ICEMI)* (pp. 53-56). IEEE.

# Appendix A Self-appraisal

# Appendix B External Materials

# Appendix D User Testing Consent Form

# Appendix F User Manual