# Machine Vision Coursework

Ruairi McElhatton

May 2024

# Contents

# 1 Task 1: Introduction to Machine Vision

## 1.1 Lab Session 1 - Part I

In this task, we were required to create a red, green and blue (RGB) image histogram of any picture we wanted and include the original image in that too. This is depicted in figures 1 and 2. The histograms in figure 2 show us the spread of each colour's intensity. From the blue histogram, we can see that blue is underexposed in the picture in question. We can tell this is the case due to there not being any information for the whites on the right side of the blue graph. We can describe the blue histogram here as being "lowkey" [1].

The red and green histograms are a bit more spread out, but red is still a little underexposed with most of the information here being towards the left hand side. Right is more central and spread out but has more information missing than red towards the most intense part of the histogram (the right).

Figure 1: Original image

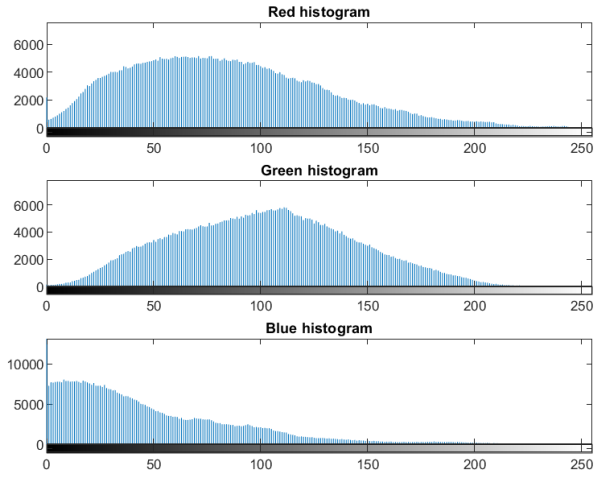


Figure 2: RGB histograms of original image

Overall, based off the histograms we can tell that there is less blue in the original image compared to the other two colours.

## 1.2 Lab Session 1 - Part II

This section goes into more detail around edge detection algorithms including Sobel, Prewitt and Canny. I decided to use a new image in this section which shows a fruit stall. This can be seen in figure 3.

Figure 3: Fruit stall image

### Enhancement contrast

The first part of this section was to analyse the original image histogram, seen in the first row of figure 4 and decide which method of enhancement would be best to increase contrast. Due to 4 hinting that the image may be a little underexposed, I thought *histogram equalisation* would be the best option for image enhancement due to the method being known for its ability to improve image contrast. I thought this would be a better option than *gamma correction* for example, which is more useful when trying to change the overall image intensity, without making a difference to contrast.

The second part of this section was to see if image enhancement using *histogram equalisation* worked well. To use this method, the image first had to be converted to a greyscale image. The results of using the method can be seen in figure 4.

As we can see in 4, the intensity of the image histogram is more spread out after *histogram equalisation* which means it has enhanced image quality and it will be better fit to the purposes of edge detection and segmentation. Whether it is a better image in general is a subjective view when it comes to photographs.

The next part of comparing image enhancement techniques was to look at *gamma correction*. The results of this are shown in the same format as the *histogram equalisation* method and are seen in figure 5. I settled on gamma = 0.7 as any lower and the image intensity would be too high, creating an image which looked "overexposed".

After seeing these two types of image enhancement in action, my initial belief that *histogram equalisation* would be better for image enhancement was confrimed as correct.

### Images with different types of noise and image denoising

The next variable to test was how different types of noise could influence edge detection. The two types of noise we were to synthesise the images with were Salt and Pepper noise and

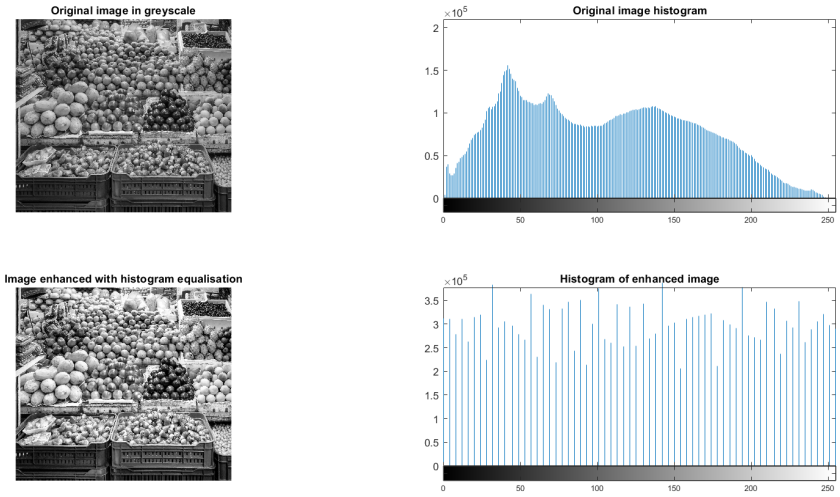


Figure 4: Image and histogram compared to histogram equalisation

Gaussian noise.

In figure 6 we can see the effect of noise on the image of the fruit (figure 3). The Gaussian noise makes the photograph look a bit like it has been painted, whilst it is clear why salt and pepper noise was given this name, with black (pepper) and white (salt) spots appearing on the image. The synthesised images are magnified to make the impact of the noise more visible.

We can then see what the Gaussian image filter does to both synthesised images. It makes them look almost identical to the original, but a little smoother, so the edges aren't as defined. Both of these used a Gaussian filter with standard deviation of 2. This was chosen because a $s.d$ that was too low meant not all the noise was removed, and a $s.d$ that was too high made the image very blurry.

Next, we were asked to use a median filter on the salt and pepper synthesised image. The results of before and after can be seen in figure 7. The default filter window size of [3 3] was optimal here. At [9 9], the image was blurry, and at [1 1], the image still had the salt and pepper noise in it.

The reason the salt and pepper noise stays there when the filter window is set so small is because when the window is small, only a very small number of neighbouring pixels are considered when filtering out noise. In this case with [1 1], the salt and pepper spots may be larger than the filter window, meaning the filter window sees the rest of the noise and only takes this into account, meaning the noise is not filtered out.

When the filter window is large, like [9 9], the image becomes blurry because all the pixels in this area filter similar colours/intensities through, meaning edges are less defined, or even lost.

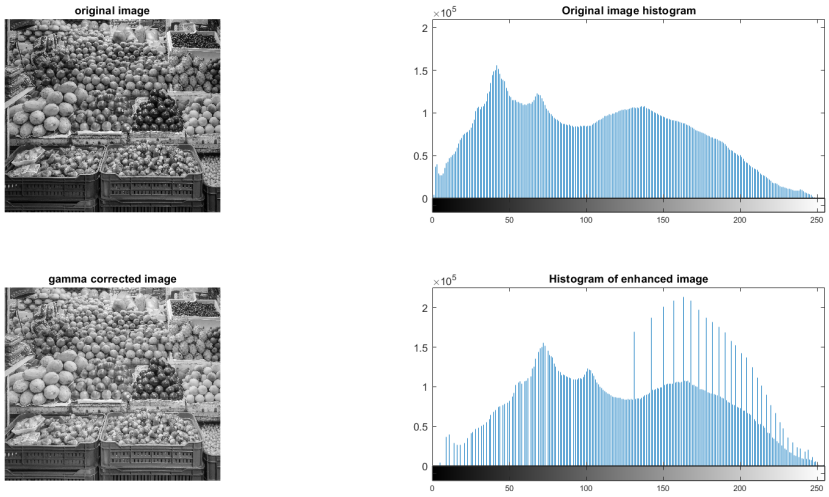


Figure 5: Image and histogram compared to gamma correction

**Static objects segmentation by edge detection**

The task for this part of the lab was to show results of different edge detection algorithms and see how different parameters would affect the outcome. "An edge in an image is a significant local change in the image intensity, usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity" [2].

The first parameter I looked at changing was the threshold. This parameter can be adjusted to ignore edges in a picture which are not stronger than the threshold. The lower this threshold parameter is, the more white spots we will see on our image of edges.

In figure 8 I have tried to optimise the thresholds of the Sobel and Prewitt algorithms to get a good output binary image. Both thresholds fared well when set low (0.03 out of a maximum 1) and used alongside "nothinning", a hyperparameter available to use in MATLAB [3] which means the edge detection algorithms skip out the edge-thinning stage. The images on the first column of the figure show the binary edge image in its default state. The second column is with the optimised threshold, and the right-most column is with the optimised threshold and the "nothinning" variable enabled.

It is quite clear looking at figure 8 that the last column is the best as we can actually see the outline of shapes in the image.

Lastly, I looked at the Canny edge detection technique along the bottom row, which required different hyperparameters to the other two. It did not yield results as good as the other two for this image, despite attempts to optimise. I chose sigma, which represents the Gaussian filter with reduced noise, to be 0.1, which is a low value. This is because setting it too high made the edges too smooth and our edge detector image almost fully black.

I then had to select two thresholds rather than one (as is the case for the other two edge detection methods). The difference being that pixels with gradient magnitudes in between the two selected thresholds are considered weak edge pixels and are included in the edge image only if they are connected to strong edge pixels (pixels which have gradient magnitudes higher

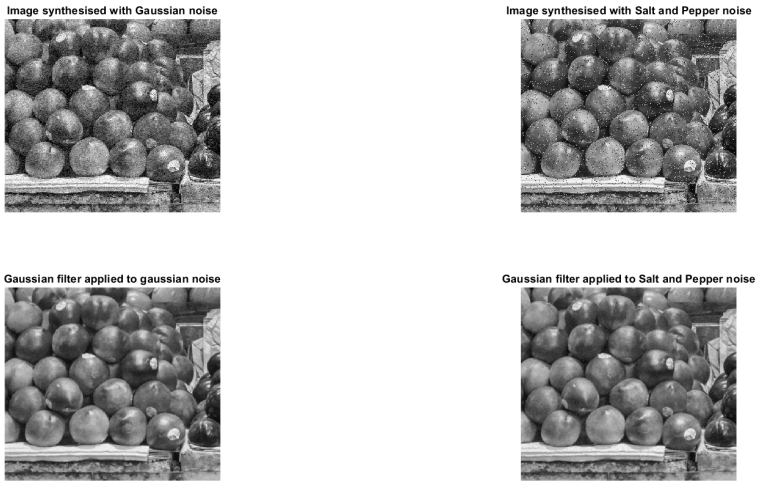


Figure 6: Fruit image with noise

than the upper threshold).


Figure 7: Fruit image with noise

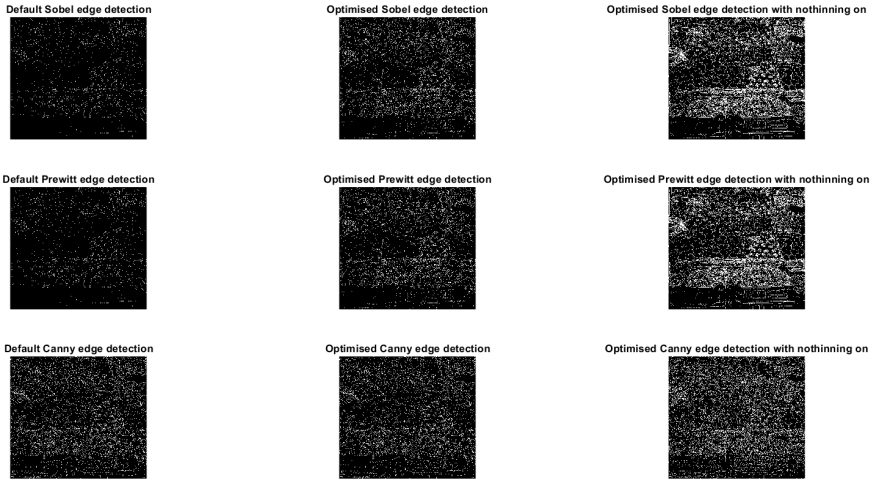


Figure 8: Optimising segmentation algorithms

# 2 Task 2: Optical Flow Estimation Algorithm

## 2.1 Finding Corner Points and Applying the Optical Flow Estimation Algorithm

To begin this lab session, the ability to find corner points of an image was tested. The results of finding and highlighting a single corner from two images is seen in figure 9 which is programmed from the start of listing 1. As can be seen the images, which were initially in RGB format, had to be converted into greyscale in order to find the corner points on them. I used a red square to highlight the area where the corner appears and then used the green cross to show the exact point.

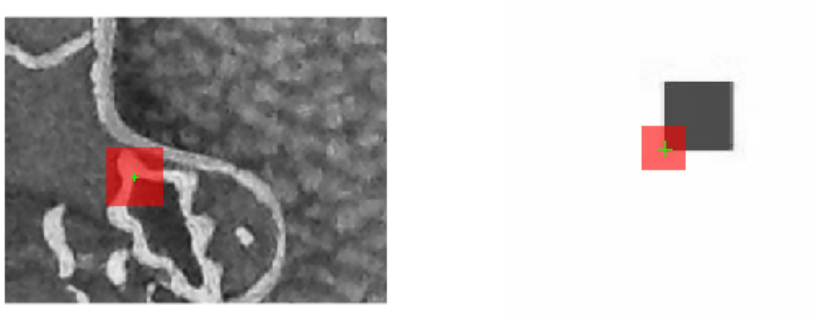

Figure 9: Square and gingerbread man marked corners

## 2.2 Finding optical flow of the pixels between two frames

This task involved taking two images, assumed to be consecutive frames from a video, and finding the optical flow between them, then visualising this. The result can be seen in figure 10. As with the task visualised in figure 9, the image had to be converted to gray-scale format in order to perform the task. This can be seen in lines 39-54 of 1.

The optical flow is depicted by the blue arrows on figure 10.

## 2.3 Tracking of single point using optical flow algorithm in a video

### Visualising the track and the ground truth of red square

The tracking throughout all 150 frames of the video compared with the ground truth can be seen in figure 11. The green dots represent the ground truth of where the top left corner is in the video. The red dots represent the tracked location of the top left corner of the red square. On the face of it, the tracked trajectory looks quite accurate. The green dots and red dots are close together throughout. This can be seen in lines 56-112 of 1.

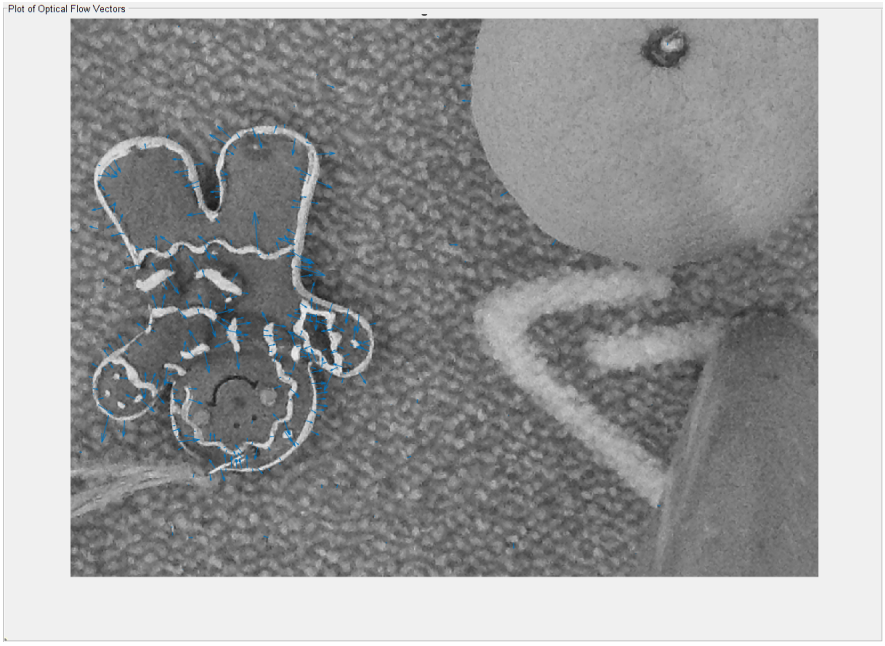


Figure 10: Optical flow in second gingerbread man image

**Compute and visualise the RMSE of tracking over frames and the RMSE average value**

To see how accurate the tracked trajectory really is, the Root Mean Squared Error (RMSE) of the coordinates must be found. The basic equation for RMSE 1

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{N} \|A_\mathrm{i} - F_\mathrm{i}|^2} \tag{1}$$

is applied to both the $x$ and $y$ coordinates in our data by the MATLAB function "rmse(F, A)". $A_\mathrm{i}$ represents the tracked trajectory and $F_\mathrm{i}$ represents the ground truth. To find the total RMSE we then combine $RMSE_\mathrm{x}$ and $RMSE_\mathrm{y}$ in the following way:

$$RMSE_\mathrm{total} = \sqrt{RMSE_\mathrm{x}^2 + RMSE_\mathrm{y}^2} \tag{2}$$

My results for this part of the lab are seen in table 1.

| | |
|---|---|
| $RMSE_\mathrm{x}$ | 1.0748 |
| $RMSE_\mathrm{y}$ | 0.6779 |
| $RMSE_\mathrm{total}$ | 1.2707 |

Table 1: Corner tracking RMSEs

For RMSE to be useful, we need the context. In this case, the values we measure for the $x$ coordinates are, on average, 1.0748 pixels away from their actual point, the ground truth. When looked at in this way, it becomes clear that the tracking method used in this task is

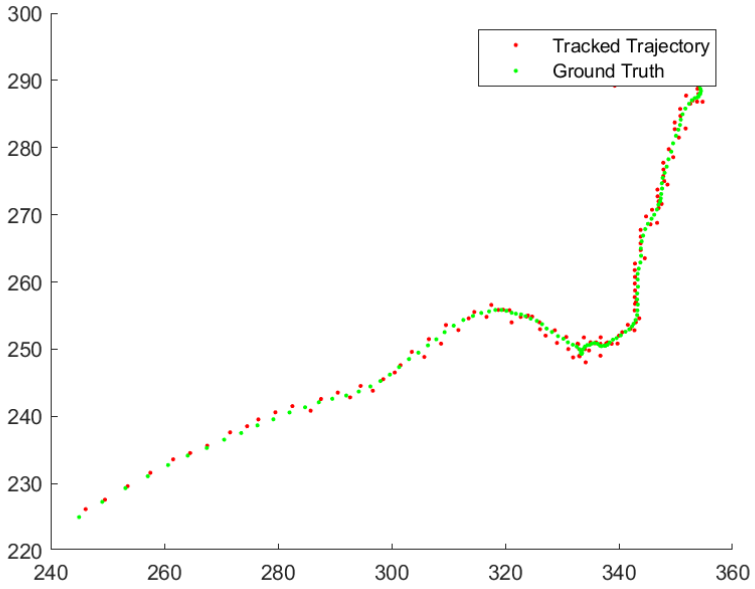


Figure 11: Tracked path of top left corner of square vs ground truth path

very accurate. The $y$ coordinates are 0.6779 pixels away from their actual point, which is even better than for $x$. The combined inaccuracy is 1.2707 which is still very small. The impact of this inaccuracy will also be minimal in terms of optical flow estimation. This part of the lab is seen in MATLAB in lines 114-128 of 1.

# 3 Task 3: Automatic Detection of Moving Objects in a Sequence of Video Frames

## 3.1 Part I: Frame differencing approach

The frame differencing approach works by comparing consecutive frames in a video and seeing the difference between them. If the difference in pixel intensity values for a given pixel is greater than a pre-specified threshold $T_s$, the pixel is considered as being a part of the foreground. One of the first steps is to make the images in question greyscale.

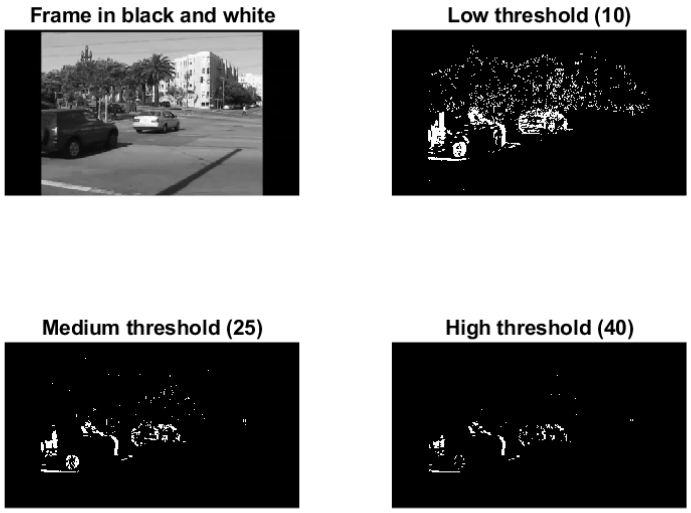


Figure 12: Impact of threshold on foreground image

Figure 12 shows the aforementioned foreground in the same frame for three different thresholds. As we can see, the foreground created using a low threshold gives us more white spots than the medium and high thresholds. The reason for this is most likely because the noise within the frames of the video has an intensity greater in magnitude than 10. A partial explanation for this effect of white spots is that the trees seen on the left of the image are rustling and this leads to a change in pixel intensity from one frame to the next.

One positive with this low threshold is that when viewed in video format, we can clearly see the person crossing the road ahead of where the cars are. This cannot be seen so clearly in the higher thresholds. It may be possible to get a better result than the three seen here if we apply a filter on the foreground image created by the low threshold.

The difference seen between the medium threshold and the high threshold is less stark. There still looks to be noise in both, but the noise is more pronounced in the medium threshold foreground, as would be expected. This part of the lab is seen in lines 1-129 of 2.

### 3.2 Part II: with the Gaussian mixture approach

In this section, the effect of Gaussian filters on images was explored. Varying different parameters on Gaussian mixtures was shown to leave us with differing results as will be shown. Setting the number of Gaussian components to one causes a black screen and it wasn't conducive to the discussion to include this, so I decided to make the "low nGauss" image have two Gaussian components.

Gaussian mixtures detect movement between frames by having multiple Gaussian distributions across the pixel intensity spectrum. Each pixel is allocated to a Gaussian component based on its intensity. If a pixel's intensity changes from one frame to the next by more than $2\sigma$ from the mean (with $\sigma$ being one standard deviation) then the foreground image shows this as movement. It is similar to the frame differencing approach but the threshold here is just $2\sigma$.

**Impact of varying number of Gaussian components**

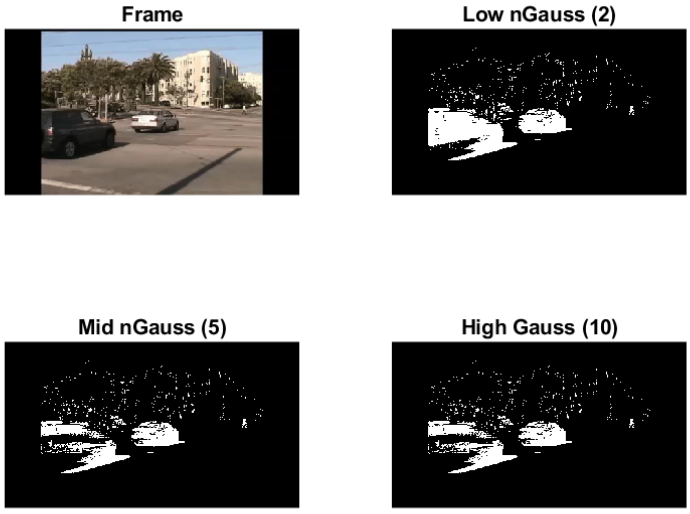


Figure 13: Impact of number of Gaussian modes on the detection results

As can be seen in figure 13, there are changes as we vary the number of Gaussian components when detecting motion from frame to frame. When changing the number of Gaussian components from two to five, the amount of noise in the foreground image is almost identical. The difference comes in the actual detection of motion. Looking at the cars in the two images, we can see that in the low nGauss image, the space that the black car takes is lighter than for the mid nGauss image. The MATLAB guidance [4] for the hyperparameter is minimal, but what it does say is that to model multiple background modes, the number of Gaussian components would have to be 3 or greater.

This part of the lab is seen in lines 276-419 of 2.

**Impact of varying number of frames learnt on**

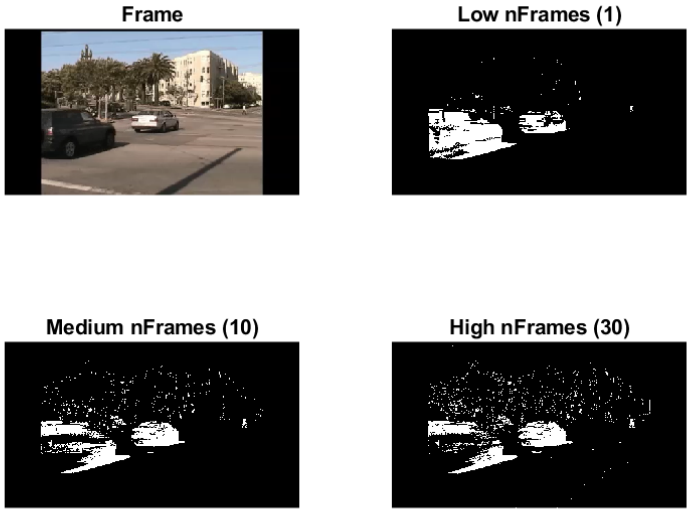


Figure 14: Impact of changing number of frames learnt on detection results

It can be seen in figure 14 that as the number of frames learnt on increases, so does the noise in the image. There is a lot more white spotting towards the top of the image in the foreground image "High nFrames" than in "Low nFrames". No matter the number of frames used to train, the cars can still be seen clearly. However, a difference is seen when looking at the person crossing the road ahead of the cars. As the number of frames used to train the background model increases, so does the clarity of that figure. So there is a trafe-off. Is it preferable to have less noise but potentially miss a moving figure, or to have more noise but a clearer picture of a moving object?

In my opinion the latter, image detection with more noise but a clearer picture of the moving object, is preferable. I am confident that with some further image processing, the noise in the image could be removed whereas it would be more difficult to make a moving object in an image clearer.

This part of the code can be seen in lines 132-274 of 2.

# 4 Task 4: Robot Treasure Hunting

In this task we are given the task of "finding the treasure" in three separate images which are seen in figure 15 by using image processing techniques to follow the arrows on the images.

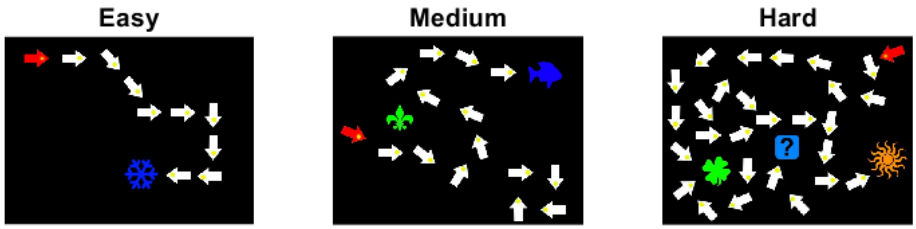


Figure 15: The three images to perform the tasks on

## 4.1 Binarisation and object detection

One of the first tasks that needed to be completed was to binarise the images in preparation for using a built-in MATLAB function which identifies objects in an image. In figure 16 it can be seen what happens when we vary the threshold between 0 and 0.2. Clearly, the optimal threshold is at 0.1. This is seen implemented in line 9 of listing 3. Having the threshold too

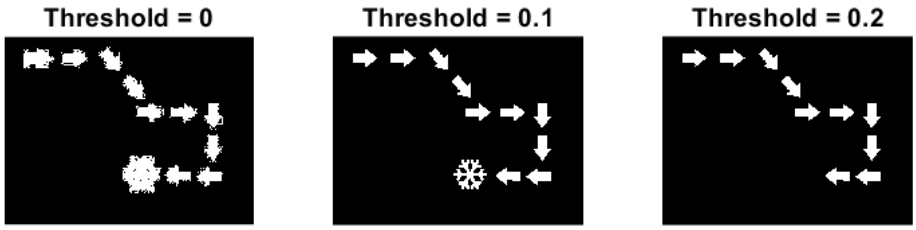


Figure 16: Varying thresholds lead to the following

high means some darker (less intense) objects within the image are not above the threshold and are therefore lost in the binary image. Having the threshold too low means that the edges of shapes are not made out very well and also noise is allowed to make it through to the binarised image. This would mean we end up with more objects than we actually have when getting onto the next couple of steps.

Once the shapes in the images have been identified, we can use the function "bwlabel()" in line 15 of listing 3 to give each one their own number. This function outputs a matrix the same size as the image so the pixels in each shape are given a number corresponding to the shape number. This number replaces what was the intensity of that pixel on the image.

Now that each image has its own number, we can use another function "regionprops()", which computes different characteristics of the objects identified. This is implemented in line 19 of listing 3

## 4.2 Identifying arrows

The next part of the task was to identify the indicies of the shapes which were arrows. I set about doing this by looking at the characteristics of the different shapes and seeing what was common between the arrows and what stood out from the other shapes. The areas of the arrows were between a range ($1430 < area < 1600$) which the other shapes seen in the three images in figure 15 were not. The outcome of this can be seen in figure 17. The function I made for this subtask can be seen in listing 4. The next step in the code was to find the red arrow which was our starting point by inputting the arrow indicies into the code which had been written for us.

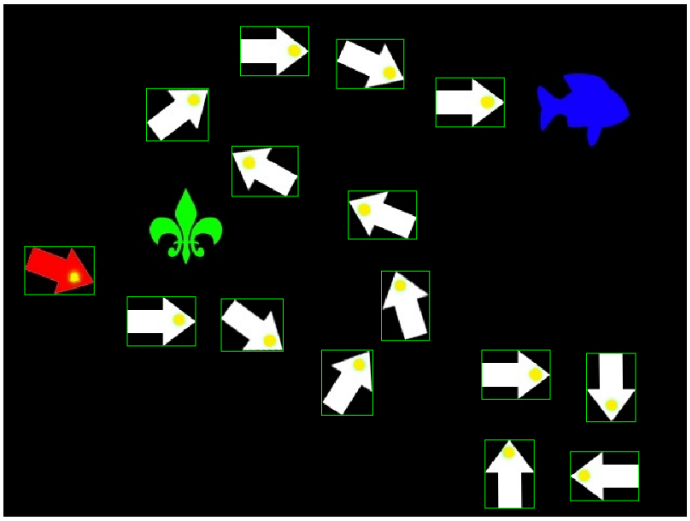

Figure 17: Arrows identified on image

## 4.3 Find the next object

In this stage we needed to find the next object along by working out, programmatically, which way the arrows were pointing. We would follow this path until we reached the treasure. I decided to go with a solution which involved using the yellow points which can be seen on all of the arrows to create a vector pointing to the next arrow.

As can be seen in listing 5, I have created a new field within the props class for the objects. The new field represents the centroids of the yellows spots seen on the arrows. These are matched up to the corresponding arrow struct in the correct order within the function in listing 5.

In 4, a line is drawn between the centroid of the arrow and the centroid of the yellow spot. This is then extended until it reaches the next object. If the next object is an arrow, this process is repeated. This goes on until we reach an object which is not an arrow and is therefore the treasure.

### 4.4 Visualisation of the solution

Finally, once the path of the solution had been found and stored in an array, the visualisation code had been written for us. So all I had to do at this point was run it. The solutions for the easy, medium and hard images can be seen in figures 18, 19 and 20 respectively.

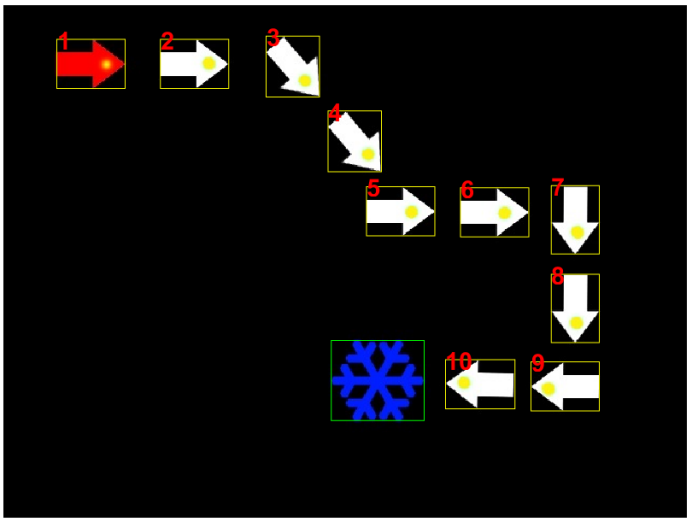


Figure 18: Visualisation of easy treasure hunting

As can be seen in 20, the path is numbered in order and then the treasure is highlighted with a rectangle and no number. In this case, the treasure is a four leaf clover.

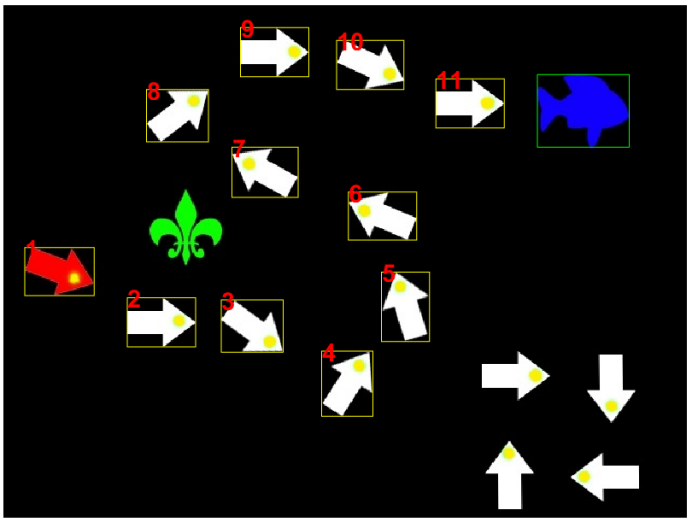


Figure 19: Visualisation of medium treasure hunting

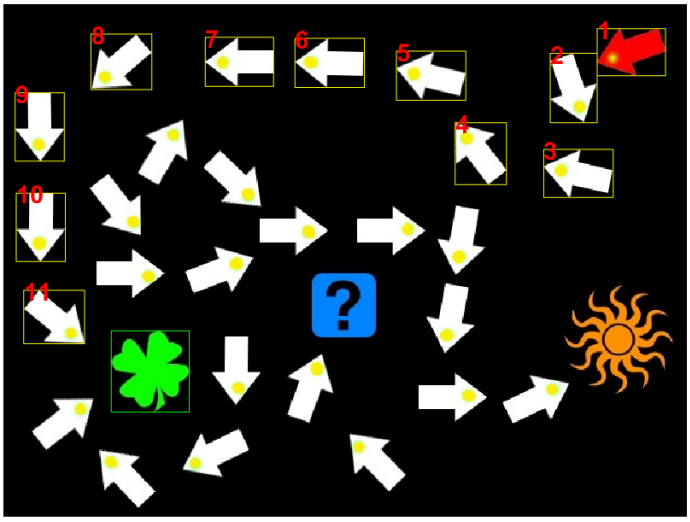


Figure 20: Visualisation of hard treasure hunting

# 5 Task 5: Image Classification with a Convolutional Neural Network (CNN)

## 5.1 Designing and Training a CNN

The predominant CNN architecture in this task is based off *LeNet-5*, which has played a significant role in the development of deep learning, particularly in the field of computer vision. We were given a sample *LeNet-5* architecture in the lab sheet of which the training progress can be seen in figure 21. The performance metrics from this baseline model can be seen in table 2.

Table 2: Table of performance metrics from baseline model

| Metric | Score |
|---|---|
| Accuracy | 0.6270 |
| Precision | 0.6294 |
| Recall | 0.6270 |
| F1 score | 0.6282 |

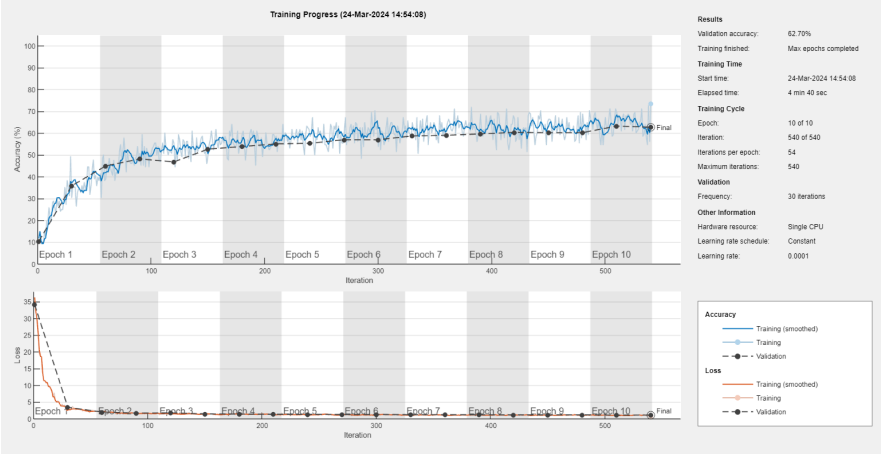


Figure 21: Training progress seen from unaltered CNN architecture

As can be seen in figure 21 and table 2, the performance metrics of the baseline model point to this model not being very good.

## 5.2 Improving the performance of the CNN Algorithm

The next part of this task involved improving the performance of the CNN through a number of different methods.

### Regularisation techniques

The suggested approach in the lab sheet for this part of the task was to combine *L1* and *L2* regularisation. This is called *elastic net regularisation* and is supposed to improve the

generalisation capability of the model. However, it is not used very often as it introduces an extra hyperparameter which demands meticulous tuning. With this is mind, the first method I chose to try to improve the performance of the classifier was *L2 regularisation*. The main reason I chose this over *L1 regularisation* is because *L2 regularisation* is more stable than *L1 regularisation*.

Having chosen *L2 regularisation*, I had to decide on what value the "WeightL2Factor" hyperparameter would take for each fully connected layer of the CNN. I decided to use Latin Hypercube Sampling [5] to help me choose the best combination. This led to results which were not as good as hoped for, with a very slight decrease in accuracy as can be seen in table 3. The best lambda values were: 0.313383, 0.423196, 0.634686. These values were used in the "fullyConnectedLayer()" functions seen in the base version of the CNN architecture respectively. The optimisation can be seen in listing 7.

Table 3: Table of performance metrics from L2 Regularisation model

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.6157 |
| Precision | 0.6173 |
| Recall | 0.6157 |
| F1 score | 0.6165 |

The next regularisation technique I tried to implement was *dropout*. This method initially gave me awful accuracy but I then decreased the dropout rate of the dropout layers from 0.5 to 0.1 and performance then only dropped a little from how it was pre-regularisation techniques. These performance metrics can be seen in 4.

Table 4: Table of performance metrics from dropout regularisation model

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.5857 |
| Precision | 0.5862 |
| Recall | 0.5857 |
| F1 score | 0.5859 |

Given both of these regularisation techniques implemented in different ways led to the model being less accurate than before they were implemented, I decided to leave them out of future models. They did not seem to be effective for this problem.

### Activation functions

Using this method to optimise the CNN architecture was much easier to implement and it also led to better results immediately. For the first activation function all I did was add a *Rectified Linear Units* (ReLU) activation function after each convolution layer. The training progress graph for this can be seen in figure 22. The performance metrics from this can be seen in table 5

Table 5: Table of performance metrics from ReLU activation function model

| Metric | Score |
|---|---|
| Accuracy | 0.7123 |
| Precision | 0.7214 |
| Recall | 0.7123 |
| F1 score | 0.7124 |

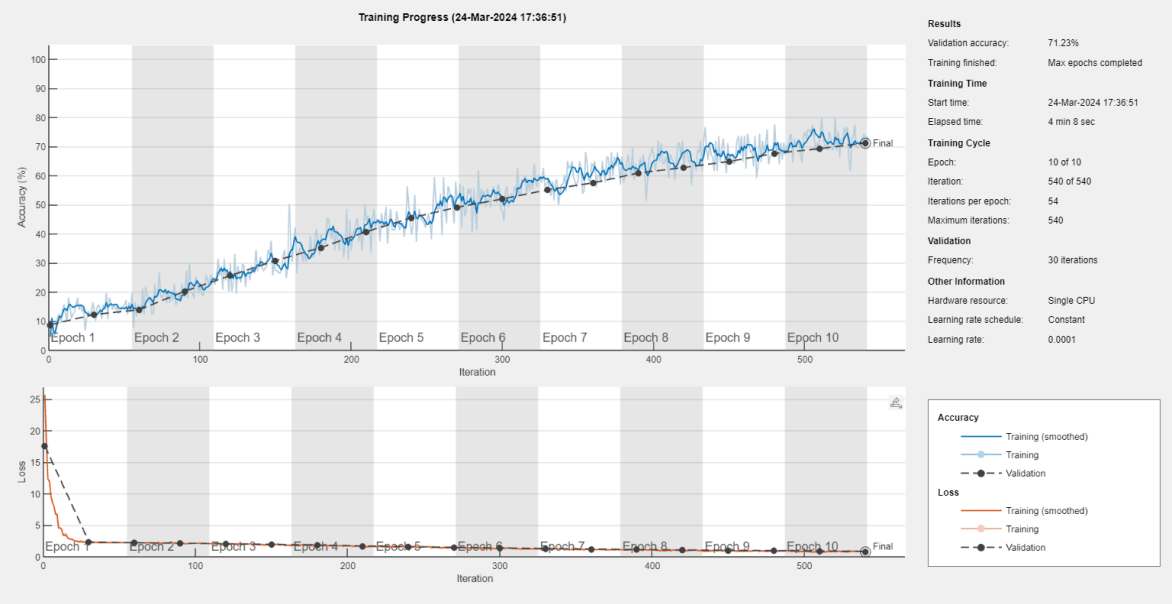


Figure 22: Training Progress graph from implementing activation function after each convolution layer

One trend we see in figure 22 is that the accuracy doesn't flatten out after 3 or 4 epochs as it did with previous training progress graphs. This could be attributed to several things, but overall is down to the *ReLU* activation function enhancing the power of neural networks through non-linearity and promoting more stable and efficient training dynamics [6]. This facilitates better representation learning.

For the second activation function I decided to use the *Sigmoid* activation function which led to a dramatic fall in accuracy, as seen in table 6. Based off the performance of the two activation functions seen here, I opted to add the *ReLU* activation function to my final CNN architecture seen in listing 10 due to the significant increase in model accuracy it achieved.

### CNN architecture exploration

In this section my attention was drawn to the *AlexNet* architecture mentioned in the lab sheet. Upon further reading [7], [8], I learnt that it is regarded as a big part of why CNN has become such a popular deep learning architecture.

When making my *AlexNet* model there were a number of hyperparameters I decided to

Table 6: Table of performance metrics from Sigmoid activation function model

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.16 |
| Precision | 0.1337 |
| Recall | 0.16 |
| F1 score | 0.1457 |

adjust from the model architecture seen in [7]. The original *AlexNet* architecture was designed for handling an input much larger than the one in this lab. Therefore, I decided to reduce some of the hyperparameters which came after this too, including the size of the filters in the first convolutional layer. The code I wrote to build the model is seen in listing 8.

Aside from there being more layers in the *AlexNet* than the *LeNet-5* design, the pooling method used was also different. In *LeNet-5*, *average pooling* is used whereas *max pooling* [9] is used in *AlexNet*. *Max pooling* is the most popular type of pooling and it just takes the max value in the pooling window. This window is slid over the input and the max value in the window at each position is taken. We only need to specify the window size and stride. With the pooling function used as it is in code seen in listing 8, each time it is used, the size of the feature map is halved. Downsampling the feature map whilst keeping the important information is the main use of pooling. For this task, *max pooling* is better due to the data being composed of black and white images, and *max pooling* being better at preserving important features.

Whilst training the model, I noticed the *AlexNet* model I had built took a lot longer than the original *LeNet-5* model. This is due to model complexity and the fact that *AlexNet* was actually built to take advantage of parallel computing resources, which I did not have access to when training the network.

The results that the *AlexNet* model produced were far superior to what we'd seen up to this point, so it could be argued that it was worth the long wait for the model to train.

Table 7: Table of performance metrics from AlexNet model

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.9907 |
| Precision | 0.9907 |
| Recall | 0.9907 |
| F1 score | 0.9907 |

### Hyperparameter tuning

For this section, I systematically varied the hyperparameters on the orginal model we were given. The first model I built was less accurate than the second, the structure of it can be seen in the code in figure 9. My main takeaway from this part of the task was no matter how much training of a model is done, it doesn't matter if the architecture is unsuitable. This

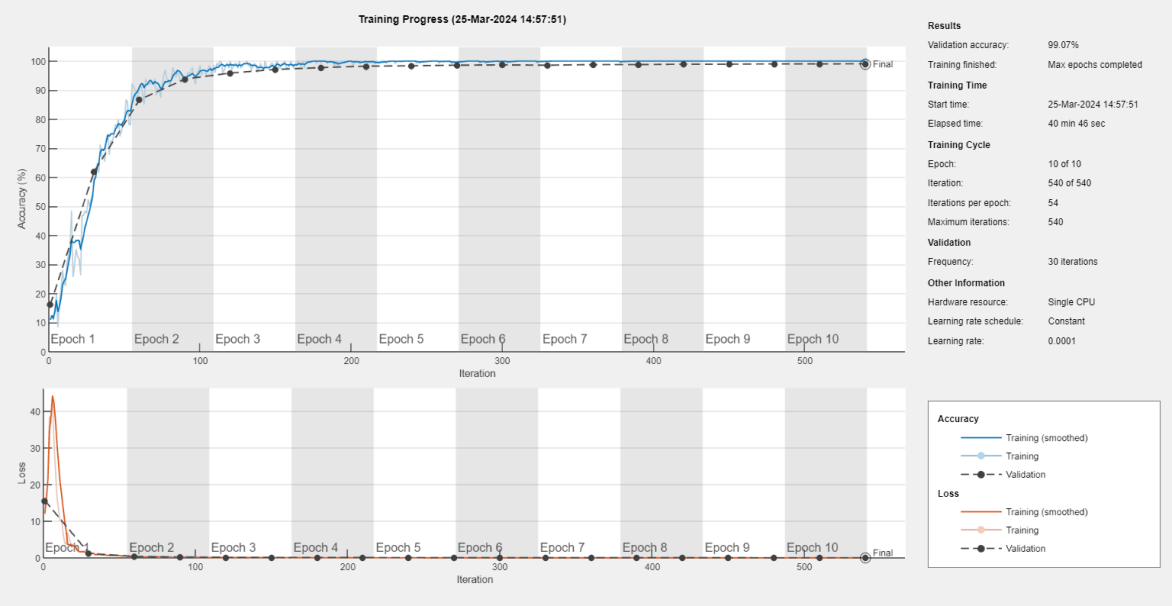


Figure 23: Training Progress of original AlexNet model

can be seen in figure 24. The model accuracy is seen to level out after four or five epochs. We end with performance metrics seen in table 8.

Table 8: Table of performance metrics from neural network with sub-optimal hyperparameters

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.6680 |
| Precision | 0.6729 |
| Recall | 0.6680 |
| F1 score | 0.6704 |

Due to the accuracy metric levelling off after 5 epochs as seen in figure 24, we can put the increase in accuracy relative to our original model down to the other hyperparameters being changed. Upon experimenting with many different combinations of hyperparameters, I found there to be a low ceiling without using a *ReLu* activation function. This ceiling is illustrated by table 8 and figure 24.

The optimised model I have presented in listing 10 has a very high accuracy and takes a relatively short time to train. I designed it off the back of the original model given to us in the lab and added a few features, as well as optimising some of the hyperparameters using Latin hypercube sampling [5]. This gave me the results seen in figure 25 and in table 9.

Regarding the filter count of the different layers seen in the network, the theory [8] states that "using more filters results in a more powerful model but at the risk of overfitting due to increased parameter count". Again, there is no single correct place to start as previously mentioned, but after testing a dozen different combinations, I settled with what is seen in listing 10. When I ran the model with these parameters, the accuracy was very high.

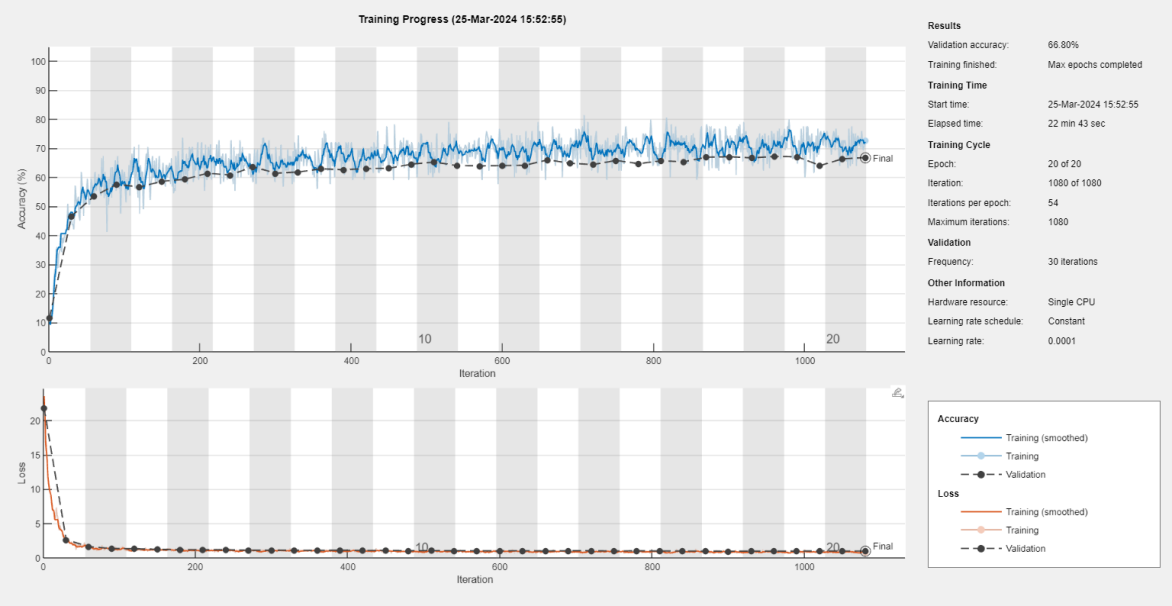


Figure 24: Training process of neural network with sub-optimal hyperparameters

Table 9: Table of performance metrics from edited LeNet-5 model

| **Metric** | **Score** |
|---|---|
| Accuracy | 0.9807 |
| Precision | 0.9809 |
| Recall | 0.9807 |
| F1 score | 0.9808 |

The results seen in table 9 and figure 25 were very good compared to the orginal model and came as a result of some simple changes. Two of these, the learning rate and number of epochs, come in the training options. Increasing the number of epochs from ten to 20 led to an increase in accuracy of about 8%. This led me to the conclusion that the training hyperparameters will only make a difference if the model is structured well to begin with. One very important addition to the architecture was the activation function. This increased accuracy drastically from when it was not there.

The accuracy is still slightly less than that seen using the enhanced *AlexNet* network seen in listing 8. However, this could easily be the other way round if a different random seed was used. Also, the *AlexNet* model took a lot longer to train, despite being trained over half as many epochs and at a lower learning rate.

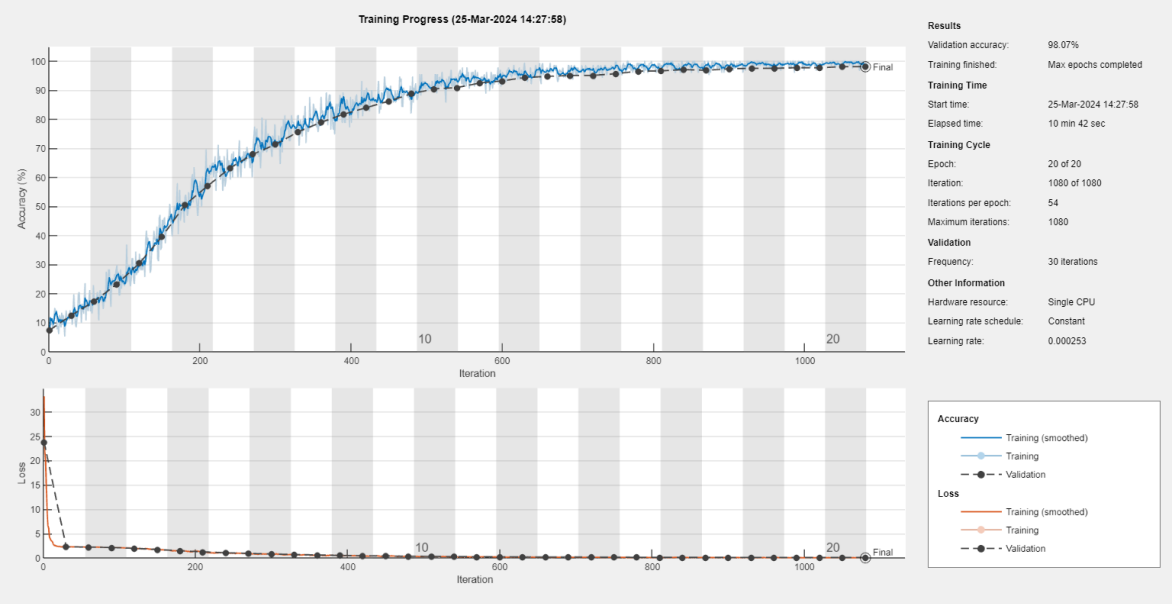


Figure 25: Training progress seen for optimised parameters model using Latin Hypercube Sampling

### 5.3 Ethics in Computer Vision

Ethics is a very important part of computer vision which needs to be considered every time computer vision is used in real-life scenarios, whether that be in image classification, detection, segmentation, or other areas. Ethics issues become very apparent when we consider equality, diversity and inclusion (EDI) in particular. Discussed here are the challenges machine vision pose, why we benefit from identifying these challenges, and potential solutions.

#### Bad data

After consulting the relevant literature it became apparent that many popular facial recognition models have varying performance capabilities across sex and ethnic groups. Some of these performance gaps are so large they lead to false positives being up to twice as likely for some demographics compared to others [10]. Another more specific example of this is seen with a Nikon camera, where a Taiwanese-American family found that their camera could not correctly identify whether they were blinking or not [11].

There is limited literature on the matter but most of the literature seen put the discrimination down to "bad data", or more specifically "non-representative training data" [12]. The general consensus is that image classification models regarding people are predominantly trained on caucasians and therefore struggle to classify other ethnicities accurately. Perhaps the most alarming example of discrimination due to training data is the case seen in a study to investigate object detection inequity among different skin tones, with a higher success rate found detecting lighter skin tones [13]. This bias and discrimination problem is not limited to machine vision but can also be seen in other areas of machine learning [14].

So, perhaps one way to help solve the issue of bias in machine vision is to have models which are put to use in real-life scenarios be trained on "good", representative data. This can come partly through data-processing and cleaning up training data. Another potential solution to this problem is to use discrimination-aware training methods like *sensitive-loss*. An example of this being put to use sees the model with *sensitive-loss* added outperform the models without both in terms of accuracy and in fairness metrics [10].

#### Lack of algorithmic transparency

Another reason why certain parts of machine vision could pose an ethical problem is due to the lack of algorithmic transparency and explainability. Neural networks like those seen in the task under section 5 are often referred to as "black box" because it is difficult to understand how they make their decisions. There are a number of reasons why this is a concern. One reason is that it can be difficult to trust these models if we don't understand how they work, especially in important applications where the stakes are high. An example of this in machine vision could be in driverless cars. This problem is one of the main factors contributing to why driverless cars aren't being rolled out on a commercial level just yet.

To bring this argument to life, how does a driverless car decide, if these are the only two possible outcomes, whether to run over an elderly person or a school child? Is the child's life more valuable than the elderly person's? It is an ethical concern that this kind of decision would have to be made partly by a computer vision sub-system on a car where nobody fully understands the machine learning algorithm on board. The lack of algorithmic transparency is a well-known issue in machine learning in general and there are many ideas for a solution.

One possible solution is to use methods that can provide explanations on how they work and how they produce certain results. This can be split into two categories, *intrinsic* and *post-hoc*. *Intrinsic* are those with transparent model design, this could be done through using simple or interpretable algorithms and avoiding black-box techniques. *Post-hoc* refers to using explanations after the model is trained such as visualisation or feature importance methods.

Another popular solution for this problem among industry leading experts is known as model lifecycle documentation. This is what it sounds like, and involves documenting the model lifecycle from data collention to deployment and monitoring. Documentation can help when assessing the quality of the data used and the model itself. It can also help stakeholders not from a technical background to understand what the model is doing along with any associated risks and benefits on society.

### Inadequate regulation and oversight

This final issue is an all-encompassing one. Machine vision models need to be regulated in order for the general population to be safe. Just like there is the Driver and Vehicle Standards Agency in the UK to protect people on the road, there must be an agency to protect people from machine vision algorithms which they may be interacting with.

There has been much international dicussion including the AI safety summit at Bletchley Park in 2023 [15] surrounding regulation of AI risks, which machine vision comes under. The regulation could be used to enforce improvements to bias models to ensure they are not biased.

### Conclusion

There is still a lot of work to be done to ensure that all ethical concerns surrounding machine vision and equality can be met. If the right people in the right positions take ownership of the issues then all this technology can be put to good use. This should be straightforward but there is a large risk if the authorities don't act correctly on the matter.

# 6 Bibliography

## References


[1] V. HADAI, *Histograms: How to read them and use them to take better photos*, Accessed: (21/02/24), 2019. [Online]. Available: https://phlearn.com/magazine/histograms-better-photos/.

[2] R. Jain and R. Kasturi, *Machine Vision*. McGraw-Hill, 1995.

[3] MATLAB, *Find edges in 2-d grayscale image*, Accessed: (14/02/2024), 2023. [Online]. Available: https://uk.mathworks.com/help/images/ref/edge.html#buo5g3w-1-BW.

[4] MATLAB, *Foreground detection using gaussian mixture models*, Accessed: (01/03/2024), 2023. [Online]. Available: https://uk.mathworks.com/help/vision/ref/vision.foregrounddetector-system-object.html.

[5] A. Olsson, G. Sandberg, and O. Dahlblom, "On latin hypercube sampling for structural reliability analysis," *Structural Safety*, vol. 25, no. 1, pp. 47–68, 2003. DOI: 10.1016/S0167-4730(02)00039-5. [Online]. Available: https://doi.org/10.1016/S0167-4730(02)00039-5.

[6] J. Brownlee, *A gentle introduction to the rectified linear unit (relu)*, Accessed: (24/04/2024), 2020. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386. [Online]. Available: http://dx.doi.org/10.1145/3065386.

[8] A. Dertat, *Applied deep learning - part 4: Convolutional neural networks*, Accessed: (25/03/2024), 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[9] MATLAB, *Max pooling layer*, Accessed: (25/03/2024), 2024. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.maxpooling2dlayer.html.

[10] I. Serna, A. Morales, J. Fierrez, and N. Obradovich, "Sensitive loss: Improving accuracy and fairness of face representations with discrimination-aware deep learning," *Artificial Intelligence*, vol. 305, p. 103 682, 2022, ISSN: 0004-3702. DOI: 10.1016/j.artint.2022.103682. [Online]. Available: https://doi.org/10.1016/j.artint.2022.103682.

[11] E. Foundation, *Nikon's face detection bias*, Accessed: (26/03/2024), 2021. [Online]. Available: https://eticasfoundation.org/nikons-face-detection-bias/#:~:text=Nikon%20has%20failed%20to%20comment,thus%20confuses%20them%20for%20blinking..

[12] S. Agrawal, *Characteristics of 'bad data' for machine learning and potential solutions?* Accessed: (26/03/2024), 2020. [Online]. Available: https://medium.com/@sanidhyaagrawal08/characteristics-of-bad-data-for-machine-learning-and-potential-solutions-88760bfa1532.

[13] B. Wilson, J. Hoffman, and J. Morgenstern, "Predictive inequity in object detection," *ArXiv*, vol. abs/1902.11097, 2019. DOI: 10.48550/arXiv.1902.11097. [Online]. Available: https://doi.org/10.48550/arXiv.1902.11097.

[14] A. Yapo and J. W. Weiss, "Ethical implications of bias in machine learning," H. I. C. on System Sciences, Ed., 2018. DOI: 10.24251/HICSS.2018.668. [Online]. Available: http://dx.doi.org/10.24251/HICSS.2018.668.

[15] D. Milmo and K. Stacey, *Five takeaways from uks ai safety summit at bletchley park*, The Guardian, 2023.

# A Exercise 2 code

Listing 1: Script for exercise 2

```
% Lab 2
clear
close all;

Ginger1 = imread('GingerBreadMan_first.jpg'); % Read in first and second gingerbread man images
Ginger2 = imread('GingerBreadMan_second.jpg');

redSquare = imread('red_square_static.jpg'); % Read in red square image

Ginger1grey = rgb2gray(Ginger1); % Convert all to grayscale to allow operations on them
Ginger2grey = rgb2gray(Ginger2);
redSquareGrey = rgb2gray(redSquare);

Ginger1Corner = corner(Ginger1grey, 1); % Find corners on all images
% Ginger2Corner = corner(Ginger2grey(Ginger1Corner:Ginger1Corner+SizeGinger), 1);
redSquareCorner = corner(redSquareGrey);
redSquareCorner = redSquareCorner(1,:); % Pick out the corner I want


SizeGinger = 50; % Size of bounding boxes I wanted for my figure to make it clearer
SizeSquare = 20;

bboxGinger1 = [Ginger1Corner(1,1)-SizeGinger/2, Ginger1Corner(1,2)-SizeGinger/2, SizeGinger, SizeGinger];
bboxSquare = [redSquareCorner-SizeSquare/2, SizeSquare, SizeSquare]; % Creating bounding boxes

markedGinger1 = insertShape(Ginger1grey, "filled-rectangle", bboxGinger1, 'Color', 'red');
markedGinger1 = insertMarker(markedGinger1, Ginger1Corner); % adds marks to the images so it can be seen in figures
markedSquare = insertShape(redSquareGrey, "filled-rectangle", bboxSquare, "Color", "red");
markedSquare = insertMarker(markedSquare, redSquareCorner);

markedCorners = figure; % Necessary step to save image where I want it
subplot(1,2,1), imshow(markedGinger1), zoom(2)
subplot(1,2,2), imshow(markedSquare), zoom(4)
saveas(markedCorners,'markedCorners.png')

Ginger2Corner = corner(Ginger2grey(Ginger1Corner(1,1):Ginger1Corner(1,1)+SizeGinger, ...
    Ginger1Corner(1,2):Ginger1Corner(1,2)+SizeGinger), 1);

%% Trying to visualise optical flow
opticFlow = opticalFlowLK("NoiseThreshold",0.005); %  initializes an object that can be used to compute optical flow using the LK algorithm
flow1 = estimateFlow(opticFlow, Ginger1grey); % computes the optical flow
```

```
flow2 = estimateFlow(opticFlow, Ginger2grey);

% Second gingerbread man
h2 = figure;
movegui(h2);
hViewPanel = uipanel(h2,'Position',[0 0 1 1],'Title','Plot of Optical Flow Vectors');
hPlot = axes(hViewPanel);

imshow(Ginger2grey), title('flow of Gingerbread man first')
hold on
plot(flow2,'DecimationFactor',[10 10],'ScaleFactor',10,'Parent',hPlot)
hold off
saveas(h2,'OpticalFlow.png')

%% Corner of sqaure
clear

redSquare = imread('red_square_static.jpg');
redSquareGrey = rgb2gray(redSquare);
redSquareCorner = corner(redSquareGrey);

MinA = min(redSquareCorner(:,1,1));
MinB = min(redSquareCorner(:,2,1));
TopLeftCorner = [MinA,MinB];

SizeSquare = 20;
bboxSquare = [TopLeftCorner(1,1)-SizeSquare/2, TopLeftCorner(1,2)-SizeSquare/2, SizeSquare, SizeSquare];

markedSquare = insertShape(redSquareGrey, "filled-rectangle", bboxSquare, "Color", "green");
imshow(markedSquare), title('Box shows region of corner')

%% Optical flow for video format
clear

videoReader = VideoReader('red_square_video.mp4'); % Creates video reader object
numFrames = 0;
while hasFrame(videoReader) % Works out how many frames there are so I know how many times to run for loop later
    readFrame(videoReader);
    numFrames = numFrames + 1;
end

opticFlow = opticalFlowLK("NoiseThreshold",0.005); %  initializes an object that can be used to compute optical flow using the LK algorithm

videoReader = VideoReader('red_square_video.mp4'); % overwrites old videoReader object

% % Need to put this in loop to get the last one

% First frame
video = readFrame(videoReader);
```

```
videoFrame = rgb2gray(video);
SquareCorner = corner(videoFrame);
MinA = min(SquareCorner(:,1,1));
MinB = min(SquareCorner(:,2,1));
TopLeftCorner = [MinA,MinB]; % Works out top left corner, this is the point
    to be tracked

position = zeros(numFrames,2); % initialising array of positions which will
    be filled later
position(1,:) = TopLeftCorner;

for i = 1:numFrames-1 % goes through video frame by frame then marks the
    coordinates of the top left corner
     video = readFrame(videoReader);
     videoFrame = rgb2gray(video);
     SquareCorner = corner(videoFrame);
     NearestCorner = dsearchn(SquareCorner,position(i,:));
     NearestCorner = SquareCorner(NearestCorner,:);
     flow = estimateFlow(opticFlow,videoFrame);
     corner_x = NearestCorner(1,1);
     corner_y = NearestCorner(1,2);
     x_new = corner_x + flow.Vx(round(corner_y), round(corner_x));
     y_new = corner_y + flow.Vy(round(corner_y), round(corner_x));
     position(i+1,:) = [x_new,y_new]; % Filling out array with positions
end

%% Compare trajectory to real data

load("red_square_gt.mat")

% plot created data
trackCorner = figure;
scatter(position(:,1),position(:,2),".","red")
hold on
scatter(gt_track_spatial(:,1),gt_track_spatial(:,2),".","green")
hold off
legend('Tracked Trajectory','Ground Truth')
saveas(trackCorner,'trackCorner.png')

RMSE = rmse(gt_track_spatial,position); % Calculating RMSE between my results
     and the actual position
RMSEt = sqrt(RMSE(1,1)^2 + RMSE(1,2)^2);

% Data in folder is not accurate to video but each data point is accurate
% relative to the last one
```

# B Exercise 3 code

Listing 2: Script for exercise 3

```
% Lab 3 script
%% Comparing thresholds at a point in time
clear
close all
% read the video
source = VideoReader('car-tracking.mp4');  % Create video file reader object

% create and open the object to write the results
outputLowThresh = VideoWriter('frame_difference_output.mp4', 'MPEG-4'); % Create video file writer object
open(outputLowThresh);

thresh = 10;      % A parameter to vary - intensity threshold decides whether it will be in the new image foreground or not

% read the first frame of the video as a background model
low.bg = readFrame(source);
low.bg_bw = rgb2gray(low.bg);           % convert background to greyscale

for i = 1:200 % do this for 200 frames
    low.fr = readFrame(source);     % read in frame
    low.fr_bw = rgb2gray(low.fr);       % convert frame to grayscale
    low.fr_diff = abs(double(low.fr_bw) - double(low.bg_bw));  % cast operands as double to avoid negative overflow

    % if fr_diff > thresh pixel in foreground
    low.fg = uint8(zeros(size(low.bg_bw)));
    low.fg(low.fr_diff > thresh) = 255;

    % update the background model
    low.bg_bw = low.fr_bw;

    % visualise the results
    figure(1),subplot(3,1,1), imshow(low.fr)
    subplot(3,1,2), imshow(low.fr_bw)
    subplot(3,1,3), imshow(low.fg)
    drawnow

    writeVideo(outputLowThresh, low.fg);           % save frame into the output video
end

% Create array of frames, compare the same frames whilst varying thresholds

close(outputLowThresh); % save video

%% medium threshold
% read the video
source = VideoReader('car-tracking.mp4');

% create and open the object to write the results
outputMedThresh = VideoWriter('frame_difference_output.mp4', 'MPEG-4');
```

```
open(outputMedThresh);

thresh = 25;      % This is to plot different thresholds on the same figure

% read the first frame of the video as a background model
med.bg = readFrame(source);
med.bg_bw = rgb2gray(med.bg);           % convert background to greyscale

for i = 1:200
    med.fr = readFrame(source);     % read in frame
    med.fr_bw = rgb2gray(med.fr);       % convert frame to grayscale
    med.fr_diff = abs(double(med.fr_bw) - double(med.bg_bw));  % cast
        operands as double to avoid negative overflow

    % if fr_diff > thresh pixel in foreground
    med.fg = uint8(zeros(size(med.bg_bw)));
    med.fg(med.fr_diff > thresh) = 255;

    % update the background model
    med.bg_bw = med.fr_bw;

    % visualise the results simultaneously
    figure(1),subplot(3,1,1), imshow(med.fr)
    subplot(3,1,2), imshow(med.fr_bw)
    subplot(3,1,3), imshow(med.fg)
    drawnow

    writeVideo(outputMedThresh, med.fg);           % save frame into the
        output video
end

% Create array of frames, compare the same frames whilst varying thresholds

close(outputMedThresh); % save video

%% High Threshold
% read the video
source = VideoReader('car-tracking.mp4');

% create and open the object to write the results
outputHighThresh = VideoWriter('frame_difference_output.mp4', 'MPEG-4');
open(outputHighThresh);

thresh = 40;      % A parameter to vary

% read the first frame of the video as a background model
high.bg = readFrame(source);
high.bg_bw = rgb2gray(high.bg);           % convert background to greyscale

for i = 1:200
    high.fr = readFrame(source);     % read in frame
    high.fr_bw = rgb2gray(high.fr);       % convert frame to grayscale
    high.fr_diff = abs(double(high.fr_bw) - double(high.bg_bw));  % cast
        operands as double to avoid negative overflow

    % if fr_diff > thresh pixel in foreground
```

```
102     high.fg = uint8(zeros(size(high.bg_bw)));
103     high.fg(high.fr_diff > thresh) = 255;
104
105     % update the background model
106     high.bg_bw = high.fr_bw;
107
108     % visualise the results
109     figure(1),subplot(3,1,1), imshow(high.fr)
110     subplot(3,1,2), imshow(high.fr_bw)
111     subplot(3,1,3), imshow(high.fg)
112     drawnow
113
114     writeVideo(outputHighThresh, high.fg);           % save frame into the
            output video
115 end
116
117 % Create array of frames, compare the same frames whilst varying thresholds
118
119 close(outputHighThresh); % save video
120
121 %% Plotting alongside eachother
122
123 CompareThresholds = figure;
124 title('Pixels exceeding threshold in foreground')
125 subplot(2,2,1), imshow(high.fr_bw), title('Frame in black and white')
126 subplot(2,2,2), imshow(low.fg), title('Low threshold (10)')
127 subplot(2,2,3), imshow(med.fg), title('Medium threshold (25)')
128 subplot(2,2,4), imshow(high.fg), title('High threshold (40)')
129 saveas(CompareThresholds,'Figures/CompareThresholds.png')
130
131
132 %% Gaussian part - nFrames comparison
133
134 clear all
135 close all
136
137 videoReader = VideoReader('car-tracking.mp4');
138 numFrames = 0;
139 while hasFrame(videoReader)
140     readFrame(videoReader);
141     numFrames = numFrames + 1;
142 end
143
144 %% Low n_frames to train on
145
146 % read the video
147 source = VideoReader('car-tracking.mp4');
148
149 % create and open the object to write the results
150 output = VideoWriter('gmm_output.mp4', 'MPEG-4');
151 open(output);
152
153 % create foreground detector object
154 % a parameter to vary - Number of frames used to compute foreground mask
155 n_frames = 1;
156
```

```
% a parameter to vary - Number of Gaussian modes in the mixture model,
% specified as a positive integer
n_gaussians = 5;

detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
    NumGaussians', n_gaussians);

% -------------------- process frames ----------------------------------
% loop all the frames
for i = 1:200
    lowNF.fr = readFrame(source);     % read in frame

    lowNF.fgMask = step(detector, lowNF.fr);    % compute the foreground mask
         by Gaussian mixture models

    % create frame with foreground detection
    lowNF.fg = uint8(zeros(size(lowNF.fr, 1), size(lowNF.fr, 2)));
    lowNF.fg(lowNF.fgMask) = 255;

    % visualise the results
    figure(1),subplot(2,1,1), imshow(lowNF.fr)
    subplot(2,1,2), imshow(lowNF.fg)
    drawnow

    writeVideo(output, lowNF.fg);    % save frame into the output video
end


close(output); % save video

%% mid n_frames to train on

% read the video
source = VideoReader('car-tracking.mp4');

% create and open the object to write the results
output = VideoWriter('gmm_output.mp4', 'MPEG-4');
open(output);

% create foreground detector object
% a parameter to vary - Number of frames used to compute foreground mask
n_frames = 10;

% a parameter to vary - Number of Gaussian modes in the mixture model,
% specified as a positive integer
n_gaussians = 5;

detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
    NumGaussians', n_gaussians);

% -------------------- process frames ----------------------------------
% loop all the frames
for i = 1:200
    midNF.fr = readFrame(source);     % read in frame

    midNF.fgMask = step(detector, midNF.fr);    % compute the foreground mask
```

```
            by Gaussian mixture models
210
211     % create frame with foreground detection
212     midNF.fg = uint8(zeros(size(midNF.fr, 1), size(midNF.fr, 2)));
213     midNF.fg(midNF.fgMask) = 255;
214
215     % visualise the results
216     figure(1),subplot(2,1,1), imshow(midNF.fr)
217     subplot(2,1,2), imshow(midNF.fg)
218     drawnow
219
220     writeVideo(output, midNF.fg);    % save frame into the output video
221 end
222
223
224 close(output); % save video
225
226 %% High n_frames to train on
227
228 % read the video
229 source = VideoReader('car-tracking.mp4');
230
231 % create and open the object to write the results
232 output = VideoWriter('gmm_output.mp4', 'MPEG-4');
233 open(output);
234
235 % create foreground detector object
236 % a parameter to vary - Number of frames used to compute foreground mask
237 n_frames = 30;
238
239 % a parameter to vary - Number of Gaussian modes in the mixture model,
240 % specified as a positive integer
241 n_gaussians = 5;
242
243 detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
        NumGaussians', n_gaussians);
244
245 % --------------------- process frames ----------------------------------
246 % loop all the frames
247 for i = 1:200
248     highNF.fr = readFrame(source);     % read in frame
249
250     highNF.fgMask = step(detector, highNF.fr);    % compute the foreground
            mask by Gaussian mixture models
251
252     % create frame with foreground detection
253     highNF.fg = uint8(zeros(size(highNF.fr, 1), size(highNF.fr, 2)));
254     highNF.fg(highNF.fgMask) = 255;
255
256     % visualise the results
257     figure(1),subplot(2,1,1), imshow(highNF.fr)
258     subplot(2,1,2), imshow(highNF.fg)
259     drawnow
260
261     writeVideo(output, highNF.fg);    % save frame into the output video
262 end
```

```
263
264
265 close(output); % save video
266
267 %% Compare n_frames
268 CompareNFrames = figure;
269 title('Pixels exceeding threshold in foreground')
270 subplot(2,2,1), imshow(highNF.fr), title('Frame')
271 subplot(2,2,2), imshow(lowNF.fg), title('Low nFrames (1)')
272 subplot(2,2,3), imshow(midNF.fg), title('Medium nFrames (10)')
273 subplot(2,2,4), imshow(highNF.fg), title('High nFrames (30)')
274 saveas(CompareNFrames,'Figures/CompareNFrames.png')
275
276 %% Gaussian part - nGauss comparison
277
278 clear all
279 close all
280
281 videoReader = VideoReader('car-tracking.mp4');
282 numFrames = 0;
283 while hasFrame(videoReader)
284     readFrame(videoReader);
285     numFrames = numFrames + 1;
286 end
287
288 %% Low n_gaussians to train on
289
290 % read the video
291 source = VideoReader('car-tracking.mp4');
292
293 % create and open the object to write the results
294 output = VideoWriter('gmm_output.mp4', 'MPEG-4');
295 open(output);
296
297 % create foreground detector object
298 % a parameter to vary - Number of frames used to compute foreground mask
299 n_frames = 10;
300
301 % a parameter to vary - Number of Gaussian modes in the mixture model,
302 % specified as a positive integer
303 % n_gaussians = 1;   % This creates a black screen
304 n_gaussians = 2;
305
306 detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
       NumGaussians', n_gaussians);
307
308 % -------------------- process frames ----------------------------------
309 % loop all the frames
310 for i = 1:200
311     lowGauss.fr = readFrame(source);     % read in frame
312
313     lowGauss.fgMask = step(detector, lowGauss.fr);    % compute the
          foreground mask by Gaussian mixture models
314
315     % create frame with foreground detection
316     lowGauss.fg = uint8(zeros(size(lowGauss.fr, 1), size(lowGauss.fr, 2)));
```

```
317      lowGauss.fg(lowGauss.fgMask) = 255;
318
319      % visualise the results
320      figure(1),subplot(2,1,1), imshow(lowGauss.fr)
321      subplot(2,1,2), imshow(lowGauss.fg)
322      drawnow
323
324      writeVideo(output, lowGauss.fg);    % save frame into the output video
325  end
326
327
328  close(output); % save video
329
330  %% mid n_gaussians to train on
331
332  % read the video
333  source = VideoReader('car-tracking.mp4');
334
335  % create and open the object to write the results
336  output = VideoWriter('gmm_output.mp4', 'MPEG-4');
337  open(output);
338
339  % create foreground detector object
340  % a parameter to vary - Number of frames used to compute foreground mask
341  n_frames = 10;
342
343  % a parameter to vary - Number of Gaussian modes in the mixture model,
344  % specified as a positive integer
345  n_gaussians = 3;
346
347  detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
         NumGaussians', n_gaussians);
348
349  % --------------------- process frames ----------------------------------
350  % loop all the frames
351  for i = 1:200
352      midGauss.fr = readFrame(source);     % read in frame
353
354      midGauss.fgMask = step(detector, midGauss.fr);    % compute the
             foreground mask by Gaussian mixture models
355
356      % create frame with foreground detection
357      midGauss.fg = uint8(zeros(size(midGauss.fr, 1), size(midGauss.fr, 2)));
358      midGauss.fg(midGauss.fgMask) = 255;
359
360      % visualise the results
361      figure(1),subplot(2,1,1), imshow(midGauss.fr)
362      subplot(2,1,2), imshow(midGauss.fg)
363      drawnow
364
365      writeVideo(output, midGauss.fg);    % save frame into the output video
366  end
367
368
369  close(output); % save video
370
```

```
%% High n_gaussians to train on

% read the video
source = VideoReader('car-tracking.mp4');

% create and open the object to write the results
output = VideoWriter('gmm_output.mp4', 'MPEG-4');
open(output);

% create foreground detector object
% a parameter to vary - Number of frames used to compute foreground mask
n_frames = 10;

% a parameter to vary - Number of Gaussian modes in the mixture model,
% specified as a positive integer
n_gaussians = 5;

detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
    NumGaussians', n_gaussians);

% --------------------- process frames ----------------------------------
% loop all the frames
for i = 1:200
    highGauss.fr = readFrame(source);     % read in frame

    highGauss.fgMask = step(detector, highGauss.fr);    % compute the
        foreground mask by Gaussian mixture models

    % create frame with foreground detection
    highGauss.fg = uint8(zeros(size(highGauss.fr, 1), size(highGauss.fr, 2)))
        ;
    highGauss.fg(highGauss.fgMask) = 255;

    % visualise the results
    figure(1),subplot(2,1,1), imshow(highGauss.fr)
    subplot(2,1,2), imshow(highGauss.fg)
    drawnow

    writeVideo(output, highGauss.fg);    % save frame into the output video
end


close(output); % save video

%% Compare n_Gauss
CompareNGauss = figure;
title('Comparing foreground masks based off changing NumGaussian modes')
subplot(2,2,1), imshow(highGauss.fr), title('Frame')
subplot(2,2,2), imshow(lowGauss.fg), title('Low nGauss (2)')
subplot(2,2,3), imshow(midGauss.fg), title('Mid nGauss (5)')
subplot(2,2,4), imshow(highGauss.fg), title('High Gauss (10)')
saveas(CompareNGauss,'Figures/CompareNGauss.png')
```

# C Exercise 4 code - Arrow finding

Listing 3: Main script for exercise 4

```
close all;
clear all;

%% Reading image
im = imread('Treasure_easy.jpg'); % change  file name to process other images - this works for all
% imshow(im);

%% Binarisation
bin_threshold = 0.1; % parameter to vary
bin_im = im2bw(im, bin_threshold); % Separates shapes from background
% imshow(bin_im);


%% Extracting connected components
con_com = bwlabel(bin_im); % Puts labels on the shapes
% imshow(label2rgb(con_com));

%% Computing objects properties
props = regionprops(con_com); % Function computes some different shape properties, including area

%% Drawing bounding boxes
n_objects = numel(props);
% imshow(im);
hold on;
for object_id = 1 : n_objects
    rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'b');
end
hold off;

%% Arrow/non-arrow determination
% You should develop a function arrow_finder, which returns the IDs of the arror objects.
% IDs are from the connected component analysis order. You may use any parameters for your function.

arrow_ind = arrow_finder(props); % returns indicies of objects which are arrows

arrows = props(arrow_ind);
% arrowFinder = figure; % commented out, but this was to check everything was going well
% imshow(im);
% hold on;
% for i = 1 : numel(arrow_ind)
%     rectangle('Position', arrows(i).BoundingBox, 'EdgeColor', 'g');
% end
% hold off;
% saveas(arrowFinder,'Figures/arrowFinder.png')

%% Finding red arrow
```

```
n_arrows = numel(arrow_ind);
start_arrow_id = 0;
% check each arrow until find the red one
for arrow_num = 1 : n_arrows
    object_id = arrow_ind(arrow_num);    % determine the arrow id

    % extract colour of the centroid point of the current arrow
    centroid_colour = im(round(props(object_id).Centroid(2)), round(props(
        object_id).Centroid(1)), :);
    if centroid_colour(:, :, 1) > 240 && centroid_colour(:, :, 2) < 10 &&
        centroid_colour(:, :, 3) < 10
        % the centroid point is red, memorise its id and break the loop
        start_arrow_id = object_id; % object ID of arrow
        break;
    end
end

%% Find yellow spots to allow us to tell direction of arrow

% This function adds a new field (which has the centroid of the yellow point)
     to the props struct
props = yellowFinder(props, arrow_ind, im);

%% show if yellows and arrows are matched - another check
% imshow(im);
% bboxes = zeros(n_arrows,4);
% hold on;
% for i = 1 : numel(props)
%     if props(i).yellowCentroid ~= 0
%         text(props(i).yellowCentroid(1), props(i).yellowCentroid(2),
    num2str(i), 'Color', 'r')
%         text(props(i).Centroid(1), props(i).Centroid(2), num2str(i), 'Color
    ', 'b')
%     end
% end
% hold off;
%% Hunting
cur_object = start_arrow_id; % start from the red arrow
path = cur_object;

% while the current object is an arrow, continue to search
while ismember(cur_object, arrow_ind)
    cur_object = next_object_finder(props, con_com, cur_object);
    path(end + 1) = cur_object;
end
% hold off

%% visualisation of the path
HardVisualSolution = figure;
imshow(im);
hold on;
for path_element = 1 : numel(path) - 1
    object_id = path(path_element); % determine the object id
    rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'y');
    str = num2str(path_element);
    text(props(object_id).BoundingBox(1), props(object_id).BoundingBox(2),
```

```
        str, 'Color', 'r', 'FontWeight', 'bold', 'FontSize', 14);
end

% visualisation of the treasure
treasure_id = path(end);
rectangle('Position', props(treasure_id).BoundingBox, 'EdgeColor', 'g');
saveas(HardVisualSolution,'Figures/HardVisualSolution.png')
```

Listing 4: Arrow finder function

```
function arrow_ind = arrow_finder(props) % takes struct of all shapes with
    characteristics as input
    n_objects = numel(props);
    arrow_ind = zeros(size(n_objects));
    for object_id = 1 : n_objects % For the number of objects identified,
        this is carried out
        if props(object_id).Area > 1430 && props(object_id).Area < 1600 %
            Arrows fall within this area, other objects don't
            arrow_ind(object_id) = object_id;
        end
    end
    arrow_ind = nonzeros(arrow_ind); % outputs the indicies of the arrows in
        an order which is not useful to us
end
```

Listing 5: Yellow spot finder function

```
function  props = yellowFinder(props, arrow_ind, im)
    [nr,nc,np]= size(im);
    newIm2= zeros(nr,nc,np);
    newIm2= uint8(newIm2);
    % had to do image processing to eliminate other yellowish colours to
    % leave us only with the yellow spots
    for r= 1:nr
        for c= 1:nc
            if im(r,c,1)>25 && im(r,c,2)>25 && im(r,c,3)>25 % if white, turn
                black
                newIm2(r,c,1)= 0;
                newIm2(r,c,2)= 0;
                newIm2(r,c,3)= 0;
            elseif im(r,c,1)<35 && im(r,c,2)<150 && im(r,c,3)>35  % if blue,
                turn black
                newIm2(r,c,1)= 0;
                newIm2(r,c,2)= 0;
                newIm2(r,c,3)= 0;
            elseif im(r,c,1)>25 && im(r,c,2)<100 && im(r,c,3)<100  % if red,
                turn black
                newIm2(r,c,1)= 0;
                newIm2(r,c,2)= 0;
                newIm2(r,c,3)= 0;
            elseif im(r,c,1)<100 && im(r,c,2)>25 && im(r,c,3)<100  % if green
                , turn black
                newIm2(r,c,1)= 0;
                newIm2(r,c,2)= 0;
                newIm2(r,c,3)= 0;
            else % the rest of the picture; no change
                for p= 1:np
```

```
27                      newIm2(r,c,p)= im(r,c,p);
28                  end
29              end
30          end
31      end
32      % figure
33  %     imshow(newIm2)
34
35      % Make yellow spots black and white for next step ----------------------
36      bin_threshold2 = 0.2; % parameter to vary 0.2 is better than 0.1 and
            better than 0.3
37      bin_im2 = im2bw(newIm2, bin_threshold2);
38      % figure
39  %     imshow(bin_im2);
40
41      % gives each object detected a number ----------------------
42      con_comYellow = bwlabel(bin_im2);
43
44      % Computing object properties ----------------------------
45      yellows = regionprops(con_comYellow);
46
47      % separating true yellow spots
48      % Needed becasue the star is very similar to the yellow dots within
            arrows
49
50      for i = 1 : numel(yellows)
51          if yellows(i).Area > 30 && yellows(i).Area < 100
52              yellows(i) = yellows(i);
53              yellow_ind(i) = i;
54          end
55      end
56
57      % Create a logical index to exclude the specified row
58      validRows = yellow_ind(yellow_ind ~= 0);
59
60      % Remove the specified row from each field
61      yellows = yellows(validRows); % This gives us a new struct without the
            old empty row
62
63      yellowCentroids = zeros(numel(yellows),2);
64      propsCentroids = zeros(numel(yellows),2);
65      % Match yellows up with props
66      % Extract the variable used for ordering from the first struct
67      for i = 1:numel(yellows) % may have to put blank rows in after this step
68          if yellows(i).Centroid == 0
69              yellowCentroids(i,:) = [0,0];
70          else
71              yellowCentroids(i,1) = (yellows(i).Centroid(1,1));
72              yellowCentroids(i,2) = (yellows(i).Centroid(1,2));
73          end
74      end
75
76      for i = 1:numel(props) % may have to put blank rows in after this step
77          propsCentroids(i,1) = (props(i).Centroid(1,1));
78          propsCentroids(i,2) = (props(i).Centroid(1,2));
79      end
```

```
80
81      % Find the differences between consecutive elements
82      differences = diff(arrow_ind);
83
84      % Find the index of the first difference larger than 1
85      missing_indices = find(differences > 1);
86
87      % Initialize an array to store missing numbers
88      missing_numbers = [];
89
90      % Iterate through the missing indices and determine the missing numbers,
91      % compares arrow_ind to props_ind so we know which lines to draw to
            yellows
92      for i = 1:length(missing_indices)
93          gap_start = arrow_ind(missing_indices(i));
94          gap_end = arrow_ind(missing_indices(i) + 1);
95          missing_numbers = [missing_numbers, (gap_start + 1):(gap_end - 1)];
96      end
97
98      % adding blank rows to yellows struct
99      % Number of blank rows to add
100     numBlankRows = numel(missing_numbers);
101     % Create a blank struct with empty fields
102     % Position to insert the blank rows
103     positions_to_insert = missing_indices; % could be "missing_indices"
104     positions_to_insert = sort(positions_to_insert, 'descend');
105
106     % Initialize a new array to store the result
107     newArray = zeros(size(yellowCentroids, 1) + numBlankRows, size(
            yellowCentroids, 2));
108
109     % Initialize insert index for the new array
110     insertIndex = 1;
111
112     % Loop through the original array and insert blank rows
113     for i = 1:size(yellowCentroids, 1)
114         % Insert the current row from the original array into the new array
115         newArray(insertIndex, :) = yellowCentroids(i, :);
116
117         % Check if a blank row needs to be inserted after this row
118         if any(positions_to_insert == i)
119             % Insert a blank row
120             insertIndex = insertIndex + 1;
121         end
122
123         % Move to the next insert index in the new array
124         insertIndex = insertIndex + 1;
125     end
126
127     yellowCentroids = newArray;
128
129     % Reorder the remaining rows to maintain the original order
130     propsCentroids = [propsCentroids(1, :); propsCentroids(setdiff(1:size(
            propsCentroids, 1), 1), :)];
131
132     % Display the updated array
```

```
    disp(propsCentroids);

    % Make them match up order wise so the yellows are matched with the
        arrows
    % Calculate similarity
    distances = pdist2(propsCentroids, yellowCentroids, 'euclidean');

    % Sort rows based on similarity
    [~, indices] = min(distances, [], 2);

    % Reorder matrix based on sorted indices
    yellowCentroids = yellowCentroids(indices(:,1), :);

    % replace duplicates with 0s
    yellowCentroids(missing_numbers, :) = 0;
    propsCentroids(missing_numbers, :) = 0;


    % add variable to props
    % props.yellowCentroid = 'coordinate';
    for i = 1:numel(props)
        if yellowCentroids(i,1) ~= 0
            props(i).Centroid = [propsCentroids(i,1), propsCentroids(i,2)];
            props(i).yellowCentroid = [yellowCentroids(i,1), yellowCentroids(
                i,2)];
        else
            props(i).yellowCentroid = [];
            props(i).Centroid = [];
        end
    end
end
```

Listing 6: Next object finder function

```
function cur_object = next_object_finder(props, con_com, cur_object)

    x1 = round(props(cur_object).Centroid(1,1));
    x2 = round(props(cur_object).yellowCentroid(1,1)); % currently one less
        yellow point than shapes and need a way of making this right
    y1 = round(props(cur_object).Centroid(1,2));
    y2 = round(props(cur_object).yellowCentroid(1,2));

%     plot([x1,x2], [y1,y2],'g')

    point1 = [x1,y1];
    point2 = [x2,y2];

    % Iteratively extend the line until the condition is satisfied

    % Calculate the next point on the line (you may need to define the line
        equation)
    next_point = point2 + (point2 - point1) / norm(point2 - point1);

    % Check if the pixel at the next point is the same object ID or 0
    x = round(next_point(1));
    y = round(next_point(2));
    pixel_value = con_com(y, x);
```

```
22
23
24      % Plot the next line segment
25  %     if pixel_value == 0 || pixel_value == cur_object
26      while pixel_value == 0 || pixel_value == cur_object
27
28          next_point = point2 + (point2 - point1) / norm(point2 - point1);
29          % Check if the pixel at the next point is the same object ID or 0
30          x = round(next_point(1));
31          y = round(next_point(2));
32          pixel_value = con_com(y, x);
33
34  %          plot([point2(1), next_point(1)], [point2(2), next_point(2)], 'Color
        ', 'green');
35
36          % Update points for the next iteration
37          point1 = point2;
38          point2 = next_point; % point2 is final point up to here
39      end
40
41  %     text(props(cur_object).BoundingBox(1), props(cur_object).BoundingBox(2)
        , num2str(i), 'Color', 'g')
42  %     rectangle('Position', props(cur_object).BoundingBox, 'EdgeColor', 'b');
         % error once it gets to blank element in struct
43
44      % Calculate the next point on the line (you may need to define the line
           equation)
45      next_point = point2 + (point2 - point1) / norm(point2 - point1);
46
47      % Check if the pixel at the next point is the same object ID or 0
48      x = round(next_point(1));
49      y = round(next_point(2));
50      pixel_value = con_com(y, x);
51
52      cur_object = pixel_value;
53  end
```

# D Exercise 5 code - CNNs

Listing 7: LHS optimisation on L2 regularisation

```
% LHS optimisation

% L2 regularisation optimisation code
clear
close all;

rng(123); % Set the random number generator seed to 0 (or any other desired
    seed)

%% Getting data in

% Load the Digits dataset
digitDatasetPath = fullfile(toolboxdir('nnet'), 'nndemos', ...
 'nndatasets', 'DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
imds.ReadFcn = @(loc)imresize(imread(loc), [32, 32]);
% Split the data into training and validation datasets
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomized');

%% Finding optimal hyperparameters for the model

% Define parameter ranges (adjust these as needed)
lambdaRange = [0.0001, 0.01]; % Range for number of filters

% Define the number of samples
numSamples = 10; % Adjust as needed

% Perform Latin Hypercube Sampling
parameterSamples = lhsdesign(numSamples, 3); % 3 parameters to sample

for i =1:numSamples

    lambda1 = parameterSamples(i, 1);
    lambda2 = parameterSamples(i, 2);
    lambda3 = parameterSamples(i, 3);

    % Define the LeNet-5 architecture
    layers1 = [imageInputLayer([32 32 1],'Name','input') % Input layer for 32
        x32 grayscale images
          convolution2dLayer(5,6,'Padding','same','Name','conv_1') %
              second number is number of filters, first is the size of
              these filters
          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_1') % reduces
               the spatial dimensions of the feature maps, 2x2 average
              pooling with stride 2

          convolution2dLayer(5,16,'Padding','same','Name','conv_2')
          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_2')

          fullyConnectedLayer(120,'Name','fc_1','WeightL2Factor', lambda1)
          fullyConnectedLayer(84,'Name','fc_2', 'WeightL2Factor', lambda2)
```

```
47              fullyConnectedLayer(10,'Name','fc_3', 'WeightL2Factor', lambda3)
48
49              softmaxLayer('Name','softmax') % Softmax activation for
                    classification
50              classificationLayer('Name','output')];
51
52      % Specify the training options
53      options1 = trainingOptions('sgdm', ...
54       'InitialLearnRate',0.0001, ...
55       'MaxEpochs',10, ...
56       'Shuffle','every-epoch', ...
57       'ValidationData',imdsValidation, ...
58       'ValidationFrequency',30, ...
59       'Verbose',false, ...
60       'Plots','training-progress');
61
62      % Train the network
63      net1 = trainNetwork(imdsTrain,layers1,options1);
64
65      % Classify validation images and compute accuracy
66      YPred = classify(net1, imdsValidation);
67      YValidation = imdsValidation.Labels;
68      accuracy(i) = sum(YPred == YValidation) / numel(YValidation);
69
70  end
71
72  % Identify the best set of parameters
73  bestAccuracy = max(accuracy);
74  bestParametersIndex = find(accuracy == bestAccuracy);
75  bestParameters = parameterSamples(bestParametersIndex, :);
76  fprintf('Best Accuracy: %f\n', bestAccuracy);
77  fprintf('Best Parameters: %f, %f, %f, %f\n', bestParameters);
78
79  %% Model accuracy
80
81  % Classify validation images and compute accuracy
82  YPred = classify(net1,imdsValidation);
83  YValidation = imdsValidation.Labels;
84  accuracy = sum(YPred == YValidation)/numel(YValidation);
85  fprintf('Accuracy of the network on the validation images: %f\n', accuracy);
86
87  % Working out the other metrics of accuracy
88  % Calculate confusion matrix
89  C = confusionmat(YValidation, YPred);
90
91  % Calculate true positive (TP), false positive (FP), and false negative (FN)
        counts
92  TP = diag(C); % True positives (correctly predicted)
93  FP = sum(C, 1)' - TP; % False positives
94  FN = sum(C, 2) - TP; % False negatives
95
96  % Calculate precision and recall
97  precision = TP ./ (TP + FP);
98  recall = TP ./ (TP + FN);
99
100 % Display precision and recall for each class
```

```
for i = 1:numel(unique(YValidation))
    fprintf('Class %d - Precision: %.2f, Recall: %.2f\n', i, precision(i), recall(i));
end

% Calculate overall precision and recall (macro-average)
precision = mean(precision);
recall = mean(recall);

% F1 score
F1score = 2*(recall*precision)/(recall+precision);
```

Listing 8: AlexNet code with hyperparameters

```
% AlexNet code with hyperparameters
clear
close all;

rng(123); % Set the random number generator seed to 0 (or any other desired seed)

%% Getting data in

% Load the Digits dataset
digitDatasetPath = fullfile(toolboxdir('nnet'), 'nndemos', ...
 'nndatasets', 'DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
imds.ReadFcn = @(loc)imresize(imread(loc), [32, 32]);
% Split the data into training and validation datasets
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomized');

%% Define the architecture

layers2 = [imageInputLayer([32 32 1],'Name','input') % Input layer for 32x32 grayscale images
      convolution2dLayer(3,96,'Padding','same','Name','conv_1') % second number is number of filters, first is the size of these filters
      maxPooling2dLayer(2,'Stride',2,'Name','maxpool_1') % reduces the spatial dimensions of the feature maps, 2x2 average pooling with stride 2

      convolution2dLayer(5,256,'Padding','same','Name','conv_2')
      maxPooling2dLayer(2,'Stride',2,'Name','maxpool_2')

      convolution2dLayer(3,384,'Padding','same','Name','conv_3')
      convolution2dLayer(3,384,'Padding','same','Name','conv_4')
      convolution2dLayer(3,256,'Padding','same','Name','conv_5')
      maxPooling2dLayer(2,'Stride',2,'Name','maxpool_3')

      fullyConnectedLayer(4096,'Name','fc_1') % connects every neuron in previous layer to current one
      fullyConnectedLayer(4096,'Name','fc_2')

      fullyConnectedLayer(10)

```

```
37          softmaxLayer('Name','softmax') % Softmax activation for
                classification
38          classificationLayer('Name','output')];
39
40 % Specify the training options
41 options2 = trainingOptions('sgdm', ...
42  'InitialLearnRate',0.0001, ...
43  'MaxEpochs',10, ...
44  'Shuffle','every-epoch', ...
45  'ValidationData',imdsValidation, ...
46  'ValidationFrequency',30, ...
47  'Verbose',false, ...
48  'Plots','training-progress');
49
50 % Train the network
51 net2 = trainNetwork(imdsTrain,layers2,options2);
52
53 %% Model accuracy
54
55 % Classify validation images and compute accuracy
56 YPred = classify(net2,imdsValidation);
57 YValidation = imdsValidation.Labels;
58 accuracy = sum(YPred == YValidation)/numel(YValidation);
59 fprintf('Accuracy of the network on the validation images: %f\n', accuracy);
60
61 % Working out the other metrics of accuracy
62 % Calculate confusion matrix
63 C = confusionmat(YValidation, YPred);
64
65 % Calculate true positive (TP), false positive (FP), and false negative (FN)
       counts
66 TP = diag(C); % True positives (correctly predicted)
67 FP = sum(C, 1)' - TP; % False positives
68 FN = sum(C, 2) - TP; % False negatives
69
70 % Calculate precision and recall
71 precision = TP ./ (TP + FP);
72 recall = TP ./ (TP + FN);
73
74 % Display precision and recall for each class
75 for i = 1:numel(unique(YValidation))
76     fprintf('Class %d - Precision: %.2f, Recall: %.2f\n', i, precision(i),
          recall(i));
77 end
78
79 % Calculate overall precision and recall (macro-average)
80 precision = mean(precision);
81 recall = mean(recall);
82
83 % F1 score
84 F1score = 2*(recall*precision)/(recall+precision);
```

Listing 9: Code for sub-optimal LeNet-5 CNN

```
1 % Sub-optimal LeNet-5 CNN
2 clear
3 close all;
```

```
4
5  rng(123); % Set the random number generator seed to 0 (or any other desired
       seed)
6
7  %% Getting data in
8
9  % Load the Digits dataset
10 digitDatasetPath = fullfile(toolboxdir('nnet'), 'nndemos', ...
11  'nndatasets', 'DigitDataset');
12 imds = imageDatastore(digitDatasetPath, ...
13  'IncludeSubfolders', true, 'LabelSource', 'foldernames');
14 imds.ReadFcn = @(loc)imresize(imread(loc), [32, 32]);
15 % Split the data into training and validation datasets
16 [imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomized');
17
18 %% Sub-optimal LeNet-5 CNN
19
20 % Define the LeNet-5 architecture
21 layers3 = [imageInputLayer([32 32 1],'Name','input') % Input layer for 32x32
       grayscale images
22           convolution2dLayer(3,64,'Padding','same','Name','conv_1') % second
                number is number of filters, first is the size of these filters
23           averagePooling2dLayer(2,'Stride',2,'Name','avgpool_1') % reduces the
                spatial dimensions of the feature maps, 2x2 average pooling with
                stride 2
24
25           convolution2dLayer(5,32,'Padding','same','Name','conv_2')
26           averagePooling2dLayer(2,'Stride',2,'Name','avgpool_2')
27
28           fullyConnectedLayer(256,'Name','fc_1') % connects every neuron in
                previous layer to current one
29           fullyConnectedLayer(256,'Name','fc_2')
30           fullyConnectedLayer(10,'Name','fc_3')
31           softmaxLayer('Name','softmax') % Softmax activation for
                classification
32           classificationLayer('Name','output')];
33
34 % Specify the training options
35 options3 = trainingOptions('sgdm', ...
36  'InitialLearnRate',0.0001, ...
37  'MaxEpochs',20, ...
38  'Shuffle','every-epoch', ...
39  'ValidationData',imdsValidation, ...
40  'ValidationFrequency',30, ...
41  'Verbose',false, ...
42  'Plots','training-progress');
43
44 % Train the network
45 net3 = trainNetwork(imdsTrain,layers3,options3);
46
47 %% Model accuracy
48
49 % Classify validation images and compute accuracy
50 YPred = classify(net3,imdsValidation);
51 YValidation = imdsValidation.Labels;
52 accuracy = sum(YPred == YValidation)/numel(YValidation);
```

```
fprintf('Accuracy of the network on the validation images: %f\n', accuracy);

% Working out the other metrics of accuracy
% Calculate confusion matrix
C = confusionmat(YValidation, YPred);

% Calculate true positive (TP), false positive (FP), and false negative (FN)
    counts
TP = diag(C); % True positives (correctly predicted)
FP = sum(C, 1)' - TP; % False positives
FN = sum(C, 2) - TP; % False negatives

% Calculate precision and recall
precision = TP ./ (TP + FP);
recall = TP ./ (TP + FN);

% Display precision and recall for each class
for i = 1:numel(unique(YValidation))
    fprintf('Class %d - Precision: %.2f, Recall: %.2f\n', i, precision(i),
        recall(i));
end

% Calculate overall precision and recall (macro-average)
precision = mean(precision);
recall = mean(recall);

% F1 score
F1score = 2*(recall*precision)/(recall+precision);
```

Listing 10: Code for optimal LeNet-5 CNN

```
% Optimal LeNet-5 CNN
clear
close all;

rng(123); % Set the random number generator seed to 0 (or any other desired
    seed)

%% Getting data in

% Load the Digits dataset
digitDatasetPath = fullfile(toolboxdir('nnet'), 'nndemos', ...
 'nndatasets', 'DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
imds.ReadFcn = @(loc)imresize(imread(loc), [32, 32]);
% Split the data into training and validation datasets
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomized');

%% Optimal LeNet-5 CNN

% Define the LeNet-5 architecture
layers4 = [imageInputLayer([32 32 1],'Name','input') % Input layer for 32x32
    grayscale images
          convolution2dLayer(5,6,'Padding','same','Name','conv_1') % second
              number is number of filters, first is the size of these filters
          reluLayer()
```

```
24          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_1') % reduces the
                spatial dimensions of the feature maps, 2x2 average pooling with
                stride 2
25          convolution2dLayer(5,16,'Padding','same','Name','conv_2')
26          reluLayer()
27          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_2')
28          fullyConnectedLayer(120,'Name','fc_1') % connects every neuron in
                previous layer to current one
29          fullyConnectedLayer(84,'Name','fc_2')
30          fullyConnectedLayer(10,'Name','fc_3')
31          softmaxLayer('Name','softmax') % Softmax activation for
                classification
32          classificationLayer('Name','output')];
33
34  % Specify the training options
35  options4 = trainingOptions('sgdm', ...
36   'InitialLearnRate',0.0001, ...
37   'MaxEpochs',10, ...
38   'Shuffle','every-epoch', ...
39   'ValidationData',imdsValidation, ...
40   'ValidationFrequency',30, ...
41   'Verbose',false, ...
42   'Plots','training-progress');
43
44  % Train the network
45  net4 = trainNetwork(imdsTrain,layers4,options4);
46
47  %% Model accuracy
48
49  % Classify validation images and compute accuracy
50  YPred = classify(net4,imdsValidation);
51  YValidation = imdsValidation.Labels;
52  accuracy = sum(YPred == YValidation)/numel(YValidation);
53  fprintf('Accuracy of the network on the validation images: %f\n', accuracy);
54
55  % Working out the other metrics of accuracy
56  % Calculate confusion matrix
57  C = confusionmat(YValidation, YPred);
58
59  % Calculate true positive (TP), false positive (FP), and false negative (FN)
        counts
60  TP = diag(C); % True positives (correctly predicted)
61  FP = sum(C, 1)' - TP; % False positives
62  FN = sum(C, 2) - TP; % False negatives
63
64  % Calculate precision and recall
65  precision = TP ./ (TP + FP);
66  recall = TP ./ (TP + FN);
67
68  % Display precision and recall for each class
69  for i = 1:numel(unique(YValidation))
70      fprintf('Class %d - Precision: %.2f, Recall: %.2f\n', i, precision(i),
            recall(i));
71  end
72
73  % Calculate overall precision and recall (macro-average)
```

```
74 precision = mean(precision);
75 recall = mean(recall);
76
77 % F1 score
78 F1score = 2*(recall*precision)/(recall+precision);
```