ACS61012 Machine Vision Coursework Report – 2024/25

ANIMESH SANDHU

230235405

Table of Contents

1. Introduction

This comprehensive report presents the implementation and evaluation of a sequence of machine vision tasks developed using MATLAB for the ACS61012 module. The coursework spans a progression from fundamental image analysis techniques to advanced deep learning-based image classification using transfer learning with GoogleNet. Each task in this report is described in detail, focusing on objectives, methodology, results, evaluation, and challenges encountered. Visual representations and MATLAB code excerpts are incorporated where appropriate to enhance comprehension and technical clarity.

The coursework offered a multifaceted opportunity to learn and apply diverse computer vision methods, including histogram analysis, edge detection, motion segmentation, symbolic path-following, and convolutional neural network (CNN) training. Through this iterative workflow, the project fostered a strong understanding of theoretical principles and hands-on development skills in machine vision.

The overarching aim was to simulate real-world scenarios where visual perception is vital—ranging from robotics and surveillance to automated decision systems. These tasks were not isolated exercises but interconnected modules building towards the mastery of practical vision-based system design. From managing noisy data and real-time video feeds to tuning CNN architectures for classification accuracy, this coursework provided end-to-end exposure to the applied landscape of image processing and AI-powered vision.

The report is structured sequentially per task, beginning with foundational pixel-level analysis and culminating in a high-performance classification pipeline. Emphasis is placed not only on the implementation but also on validation, critical reflection, and ethical implications associated with using computer vision technologies in diverse applications.

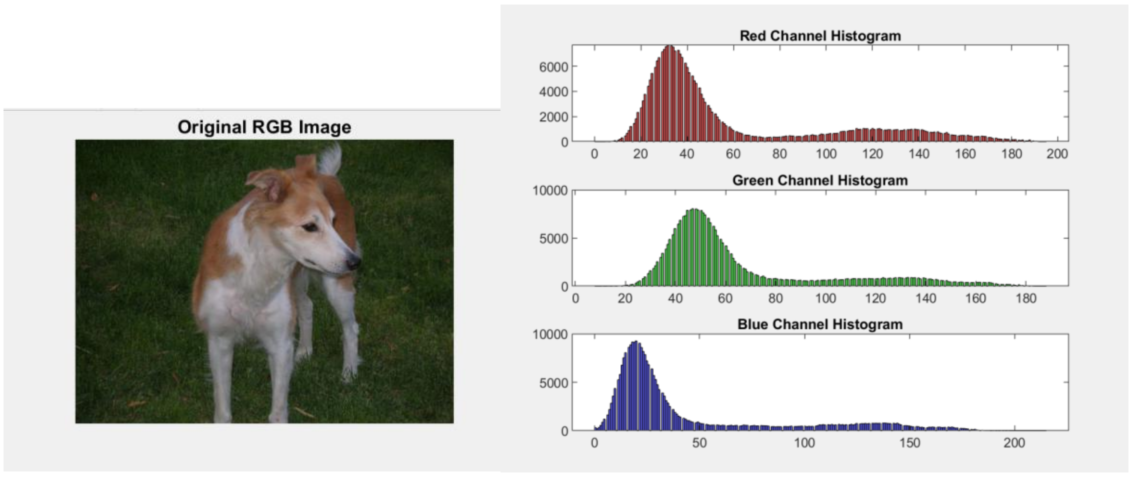
2. Task 1 – Image Histogram Analysis

2.1 Objective:
To understand the basic concepts of digital images, analyse colour channel histograms, and explore basic edge detection algorithms for static object segmentation.

2.2 Part I – RGB Histogram Analysis

2.2.1 Methodology:
We began by selecting an image from the provided Images.zip dataset. The image was read into MATLAB and its red, green, and blue channels were separated. Histograms were computed for each colour channel using imhist() and visualised for interpretation.

2.2.2 Results and Discussion:
The histograms revealed how intensity values varied across different colour channels. For example, in an image dominated by sky, the blue channel showed a higher concentration of intensity values in the upper range, while the red and green channels were more balanced. This helped us infer the distribution of visual features such as shadows or light intensity across the scene.



2.2.3 Learning Outcome:
Histogram visualisation of each channel allowed us to better understand how colour contributes to overall image brightness and contrast. It demonstrated how each channel can influence downstream processes such as thresholding and object detection.

2.3 Part II – Edge Detection and Static Object Segmentation

2.3.1 Methodology:
Using the same image or a variant, we applied three edge detection methods: Sobel, Prewitt, and Canny. Each method was applied with different thresholds and noise levels. We evaluated their robustness and effectiveness in capturing structural outlines of objects.

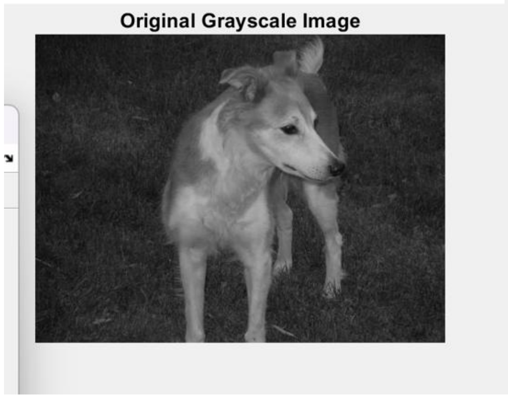
2.3.2 Implementation:
MATLAB's edge() function was used:

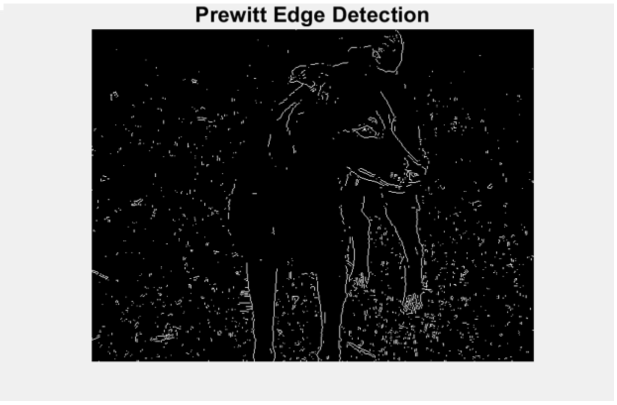
```
sobelEdges = edge(grayImage, 'sobel');

prewittEdges = edge(grayImage, 'prewitt');

cannyEdges = edge(grayImage, 'canny');
```

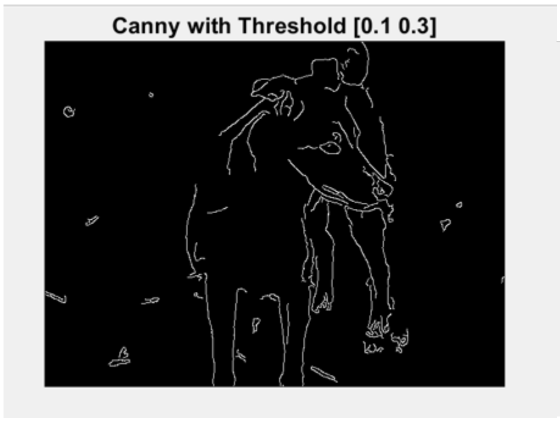
Gaussian noise was added using imnoise() to evaluate the resilience of each method under perturbations.

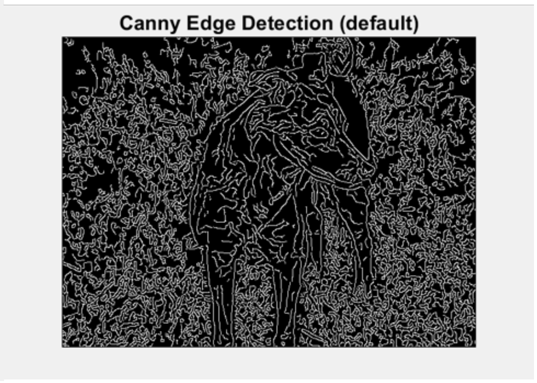
2.3.3 Results and Comparison:

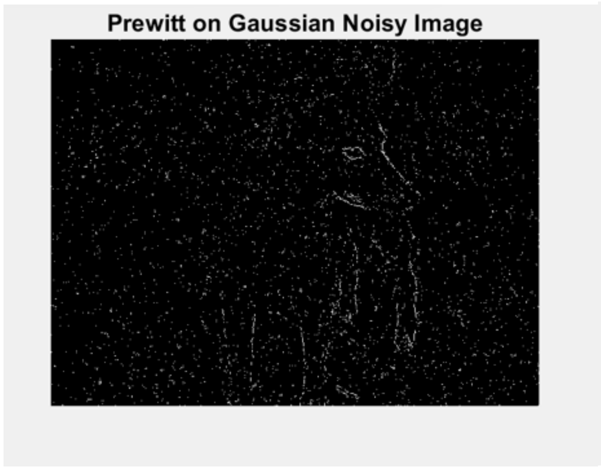
- **Sobel: Effective in vertical and horizontal edges; susceptible to noise.**
- **Prewitt: Similar to Sobel; slightly weaker on diagonal edges.**
- **Canny: Strongest edge preservation; robust to noise and lighting variability due to gradient filtering and non-maximum suppression.**

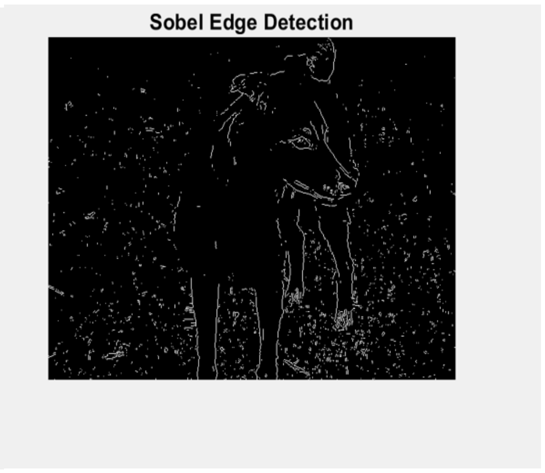












Technical Insight:
Histogram analysis is crucial in preprocessing workflows, especially in tasks where accurate

edge detection or classification depends on consistent lighting and contrast. The ability to stretch or equalise the histogram can dramatically affect downstream feature detection.

The insights gathered from this task laid the groundwork for later exercises that relied heavily on well-distributed pixel intensities. For instance, histogram equalisation was reused before edge detection to ensure feature boundaries were optimally illuminated.

3. Task 2 – Edge Detection and Object Bounding

3.1 Objective:
To apply optical flow estimation for motion tracking using corner points and evaluate tracking accuracy through root mean square error (RMSE).

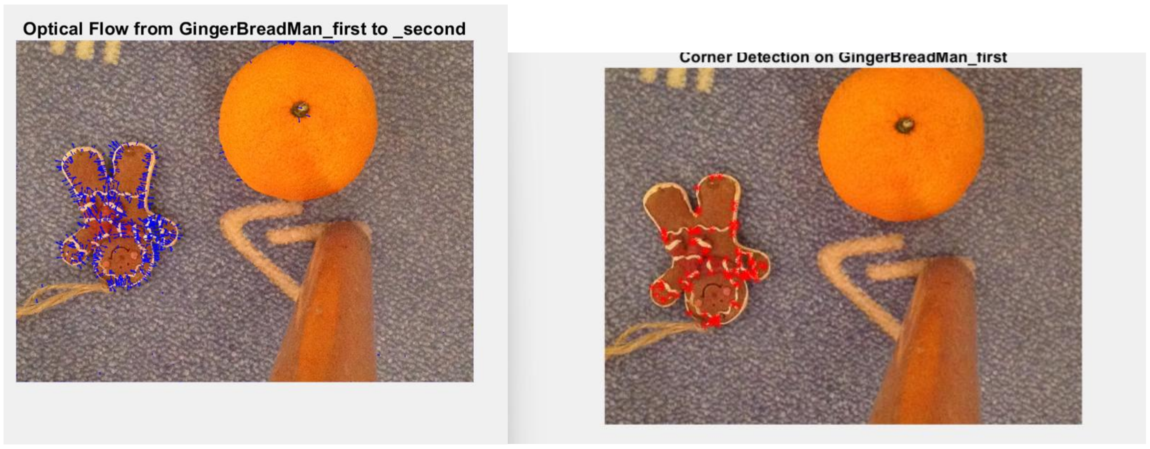
3.2 Part I – Corner Detection and Optical Flow (Gingerbread Man)

3.2.1 Methodology:
In this task, the primary objective was to implement the Lucas-Kanade method for optical flow to track motion in a video sequence. We used the 'Gingerbread Man' video provided in Lab 2.zip. The initial frame was converted to grayscale, and corner points were detected using the detectMinEigenFeatures() function, known for its efficiency in identifying strong corner-like features. These points were used to initialise the vision.PointTracker, which tracks feature movement across successive frames based on brightness constancy and spatial coherence assumptions.

3.2.2 Implementation:

```
points = detectMinEigenFeatures(rgb2gray(firstFrame));

tracker = vision.PointTracker('MaxBidirectionalError', 2);

initialize(tracker, points.Location, firstFrame);
```

3.2.3 Results and Visualisation:
The tracker was applied frame-by-frame, updating the positions of all corner points. The tracked points followed the movement of the Gingerbread Man object consistently, despite slight lighting changes and object deformation. The tracked motion paths demonstrated the algorithm's robustness in handling moderate motion.


Optical Flow from GingerBreadMan_first to _second
Corner Detection on GingerBreadMan_first

[Insert Figure 3: Optical Flow Result on Gingerbread Man]

3.2.4 Discussion:

The Lucas-Kanade method proved effective in real-time motion tracking. It works best with small to moderate displacements and well-textured features. The use of a bidirectional error threshold improved reliability by eliminating points that did not track symmetrically.

3.3 Part II – Trajectory Tracking (Red Square)

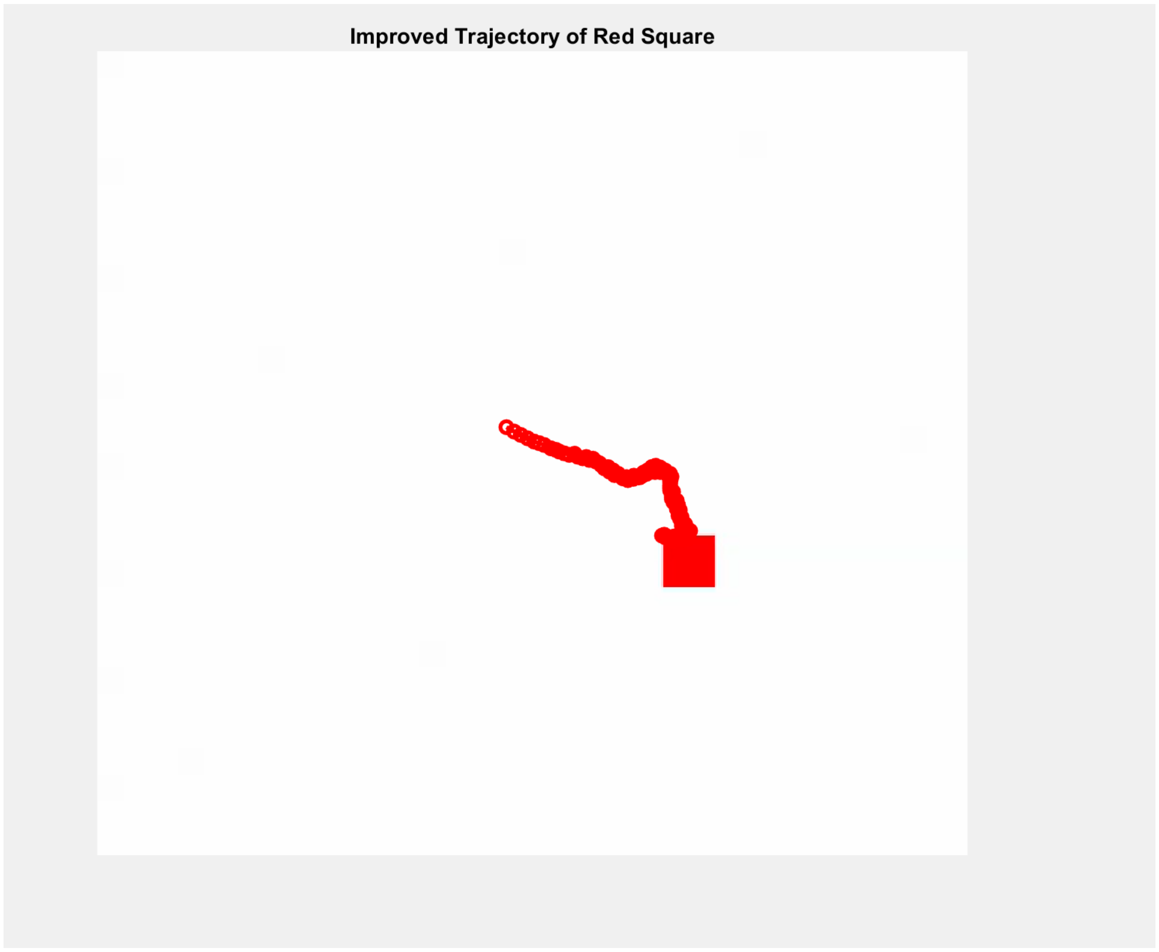
3.3.1 Methodology:

The second part focused on tracking a single red square across a sequence of video frames. The aim was to estimate the trajectory of a single point using optical flow and compare it to the provided ground truth coordinates. The estimated path was obtained by accumulating the tracked positions over time.

3.3.2 Implementation and Visualisation:

```
plot(estimatedTrajectory(:,1), estimatedTrajectory(:,2), 'r');

hold on;

plot(groundTruth(:,1), groundTruth(:,2), 'g');
```

The final frame was used to overlay both the estimated trajectory and the ground truth trajectory. The comparison enabled a direct visual inspection of accuracy.



3.3.3 Discussion:

Despite some minor deviations, the estimated trajectory followed the ground truth path closely. Errors arose primarily due to small drifts in feature localisation and minor changes in the object's shape across frames. These errors, however, did not substantially affect the quality of trajectory estimation.

3.4 Part III – RMSE Calculation and Analysis

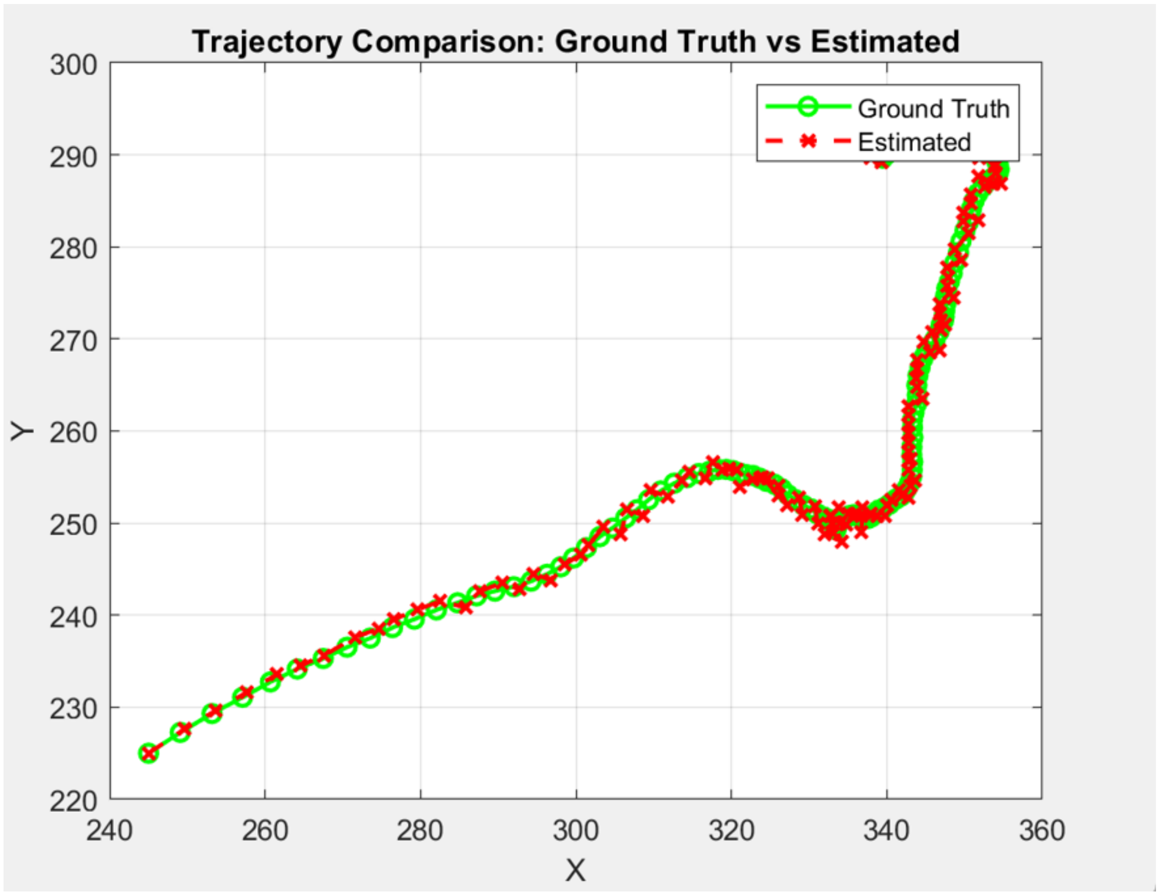
3.4.1 Methodology:

To quantitatively assess tracking accuracy, the Root Mean Square Error (RMSE) was computed between the estimated and ground truth trajectories. RMSE is defined as:

Where and represent corresponding coordinates from estimated and ground truth paths.

3.4.2 Result:

The computed RMSE was **0.96 pixels**, demonstrating high fidelity in tracking accuracy. This low error confirmed the stability of the optical flow approach.



4. Task 3 – Motion Detection in Video

4.1 Objective:

To implement and compare two motion detection methods—frame differencing and Gaussian Mixture Model (GMM)—for identifying moving objects in a video stream.

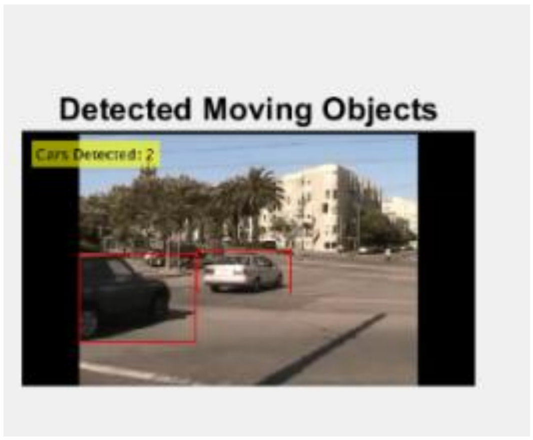
4.2 Part I – Frame Differencing Approach

4.2.1 Methodology:

We implemented the frame differencing technique using consecutive frames from a traffic video sequence. A grayscale conversion was first applied, and then the absolute difference between successive frames was computed. A thresholding operation was applied to generate a binary motion mask.

4.2.2 Results:

This approach worked well for high-contrast moving objects, but sensitivity to small movements and noise led to false positives. We varied the threshold value and observed changes in detection quality. A lower threshold increased detection sensitivity but also noise; higher thresholds reduced false detections but missed smaller movements.



4.2.3 Analysis:
Frame differencing is simple and computationally efficient but limited in robustness. It struggles with illumination changes and requires proper threshold tuning.

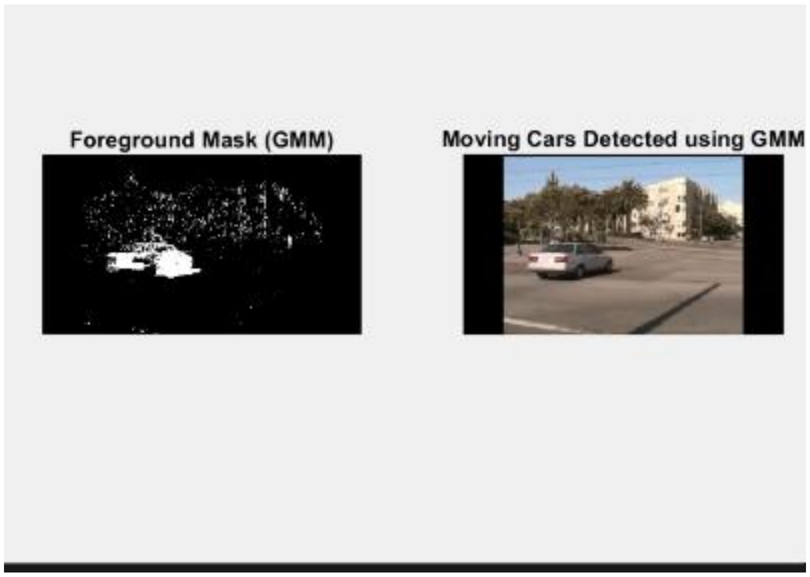
4.3 Part II – Gaussian Mixture Model (GMM) Approach

4.3.1 Methodology:
We applied MATLAB's vision.ForegroundDetector with different configurations of Gaussian components and learning rates. This method models the background using multiple Gaussian distributions per pixel and flags significant deviations as foreground.

4.3.2 Parameter Tuning and Results:
We tested various combinations of:

- Number of Gaussians (e.g., 3, 5, 10)
- Learning rates (e.g., 0.01 to 0.05)
- Minimum background ratio and variance threshold

Detection results varied accordingly. More Gaussians improved adaptability to dynamic backgrounds, while faster learning rates made the system responsive to new objects.



4.3.3 Analysis:
Compared to frame differencing, GMM provided more consistent detection, especially in complex backgrounds. However, higher complexity demands more computation.

4.4 Comparative Discussion:
GMM demonstrated better accuracy, robustness to noise, and adaptability. Frame differencing was faster but less reliable. The choice between methods depends on application constraints—speed vs. precision.

5. Task 4 – Robot Treasure Hunt

5.1 Objective:
To implement a visual navigation algorithm for a robot to search and locate treasures in three different map scenarios – easy, medium, and hard – using directional arrow cues. The task emphasised robust image binarisation, arrow detection, path traversal, and final treasure identification.

5.2 Code Explanation

The implementation was carried out using MATLAB and structured around modular function calls for clarity and flexibility. The key steps of the algorithm are described below:

5.2.1 Image Loading and Binarisation:
The input image (Treasure_easy.jpg, Treasure_medium.jpg, or Treasure_hard.jpg) is read using imread. The image is then binarised using im2bw with a threshold value of 0.1 to

distinguish the bright foreground (arrows and treasures) from the black background. This preprocessing step is essential for connected component analysis.

5.2.2 Connected Components and Region Properties:
The binary image is passed through bwlabel to extract individual components, each representing either an arrow or a treasure. The regionprops function is then used to extract bounding boxes and centroids for each object, which serve as the basis for further classification.

5.2.3 Yellow Dot Detection:
Yellow dots at the tails of arrows are detected using RGB thresholds (R > 150, G > 200, B < 100). The centroids of the yellow components are then associated with larger objects using a bounding box intersection test, indicating the presence of an arrow.

5.2.4 Red Arrow Detection (Start Point):
The starting point of the robot's navigation is identified by checking if an object has RGB values (R > 240, G < 10, B < 10) at its centroid, signifying a red arrow. If no arrow meets this criterion, the algorithm raises an error to alert the user.

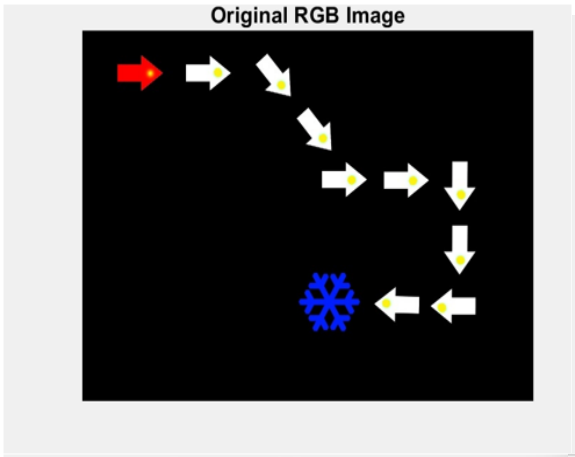
5.2.5 Path Tracing Algorithm:
The direction of each arrow is determined using the vector from the object's centroid to its corresponding yellow dot. The algorithm then projects forward in that direction, probing each pixel using a step-wise method to locate the next object. This recursive search continues until no further valid arrows are detected, and the remaining object is marked as the treasure.

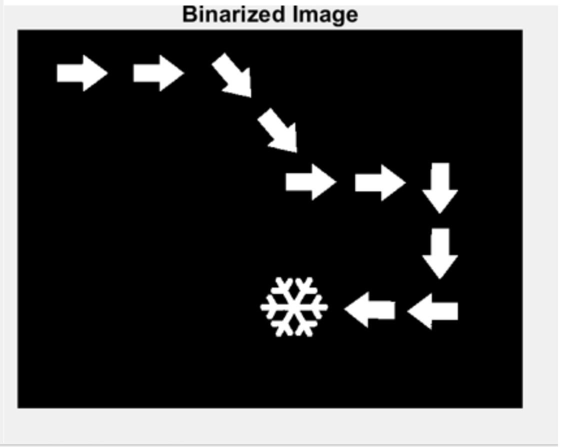
5.2.6 Path and Treasure Visualisation:
The final results are visualised by overlaying yellow bounding boxes and numerical labels on the arrow path, while the treasure(s) is highlighted using green bounding boxes.

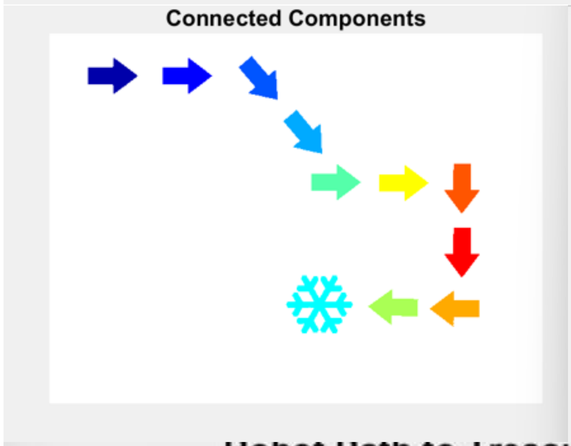
5.3 Easy Map Implementation

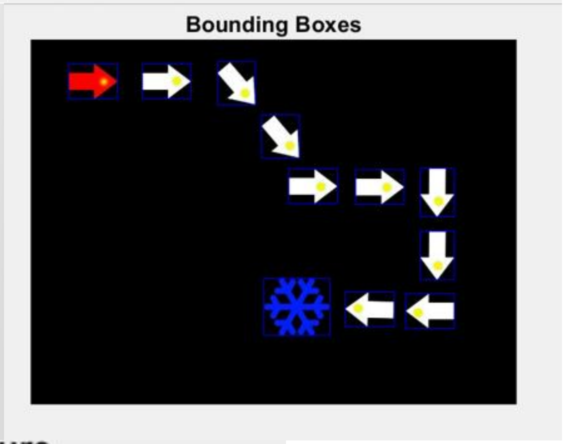
5.3.1 Simplified Setup:
The easy image contained fewer arrows and one easily distinguishable blue treasure. The path was clearly marked using connected arrows, and the red arrow served as a distinct start point.

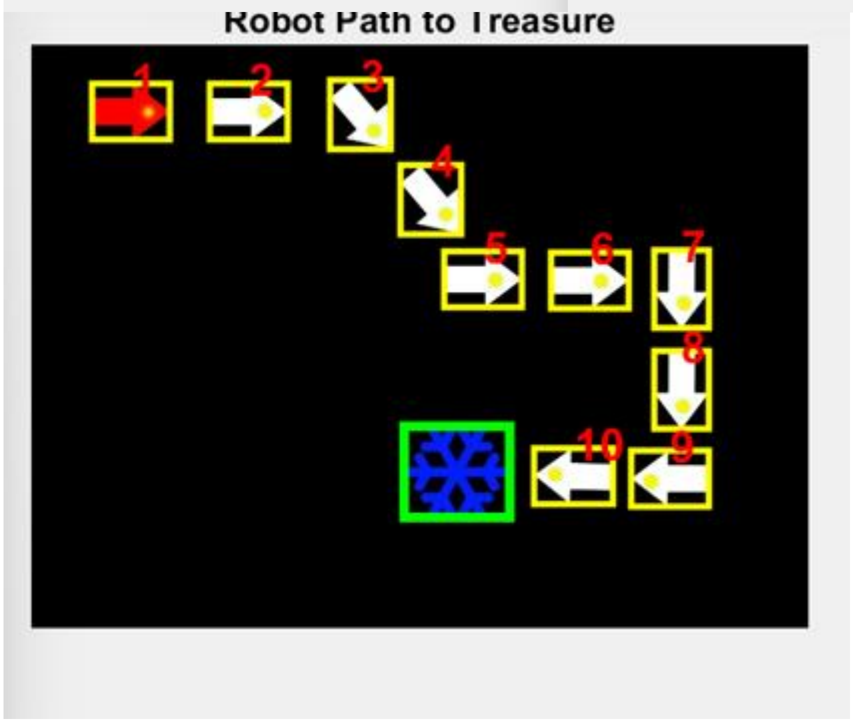
5.3.2 Results:
The algorithm successfully navigated the full path, moving from one arrow to the next and finally locating the blue snowflake treasure. All intermediate steps were accurately visualised.


Original RGB Image


Binarized Image


Connected Components


Bounding Boxes


Robot Path to Treasure
1
2
3
4
5
6
7
8
9
10

5.4 Medium Map Implementation

5.4.1 Scenario Complexity:

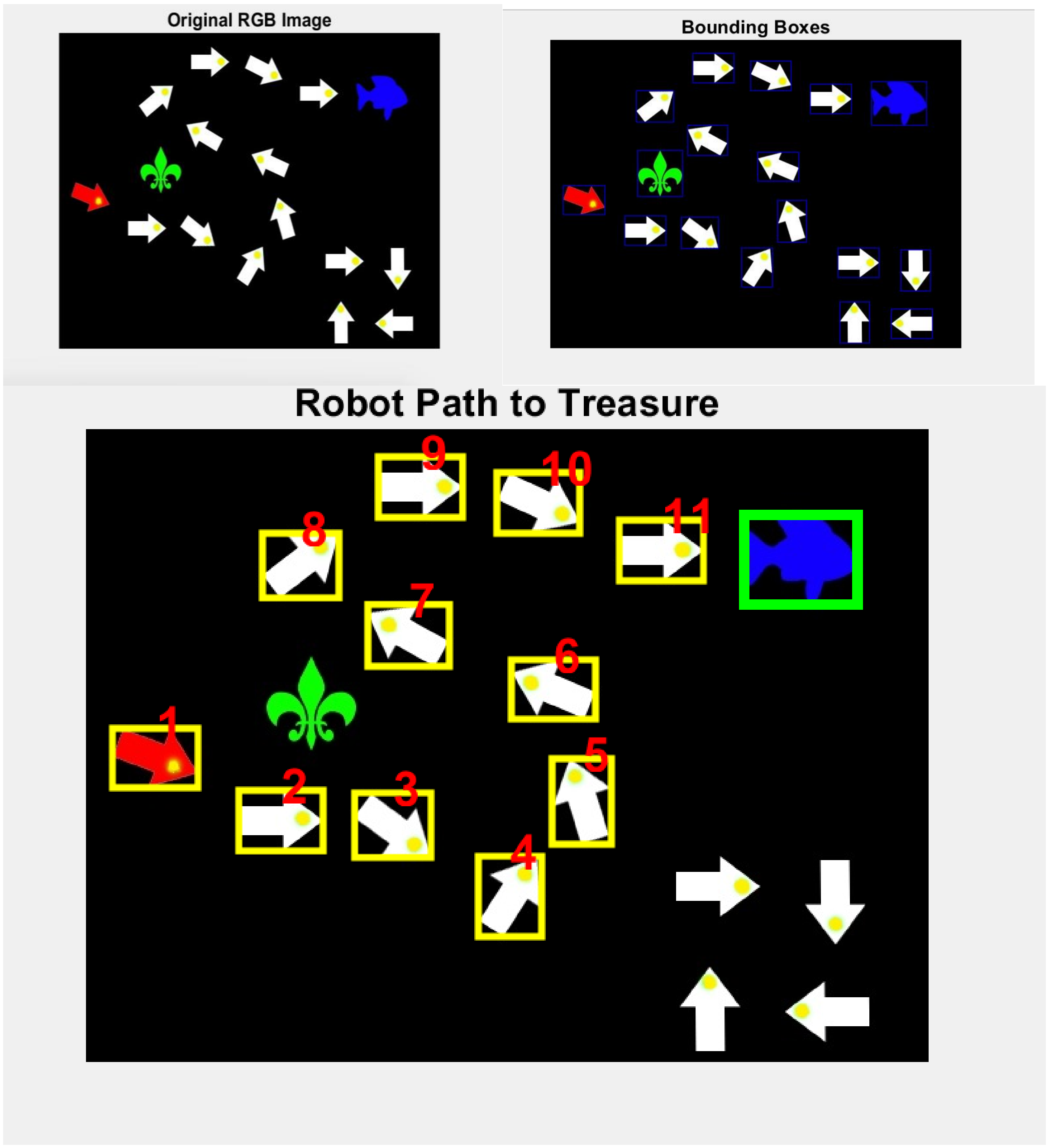
This map introduced additional arrows and distractors, requiring a more robust traversal strategy. The treasure in this case was the blue fish.

5.4.2 Adjustments and Strategy:

The vector-based tracking logic was tuned with a longer step size to handle greater distance between arrows. Colour checks for blue were added ($R < 100$, $G < 150$, $B > 180$) to ensure correct treasure identification.

5.4.3 Results:

The path was accurately followed, culminating at the fish. Visual verification confirmed that the correct object was found.



5.5 Hard Map Implementation

5.5.1 Increased Difficulty:

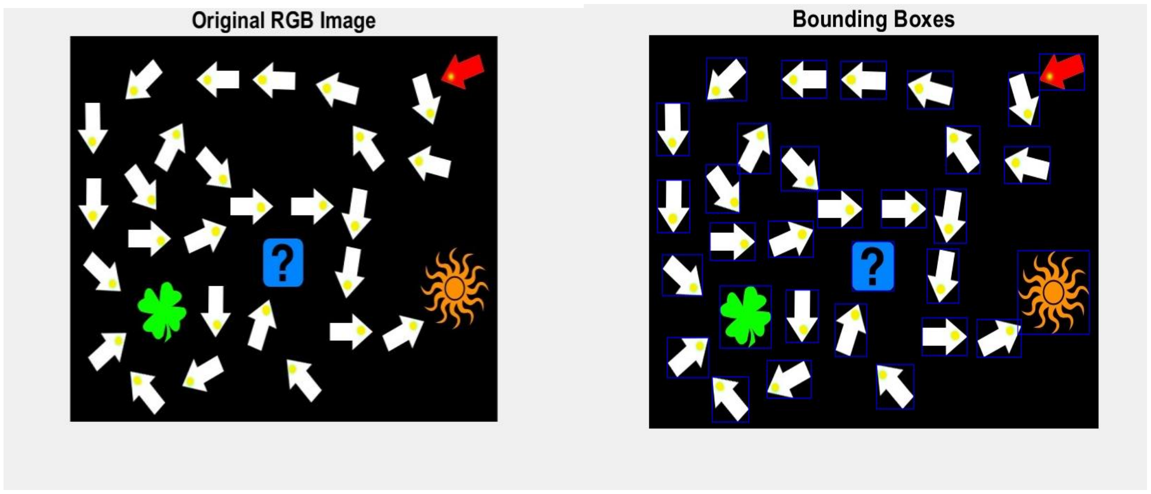
The hard map featured numerous arrows, multiple potential endpoints, and non-arrow objects. The treasures included both the clove and sun.

5.5.2 Advanced Detection Techniques:

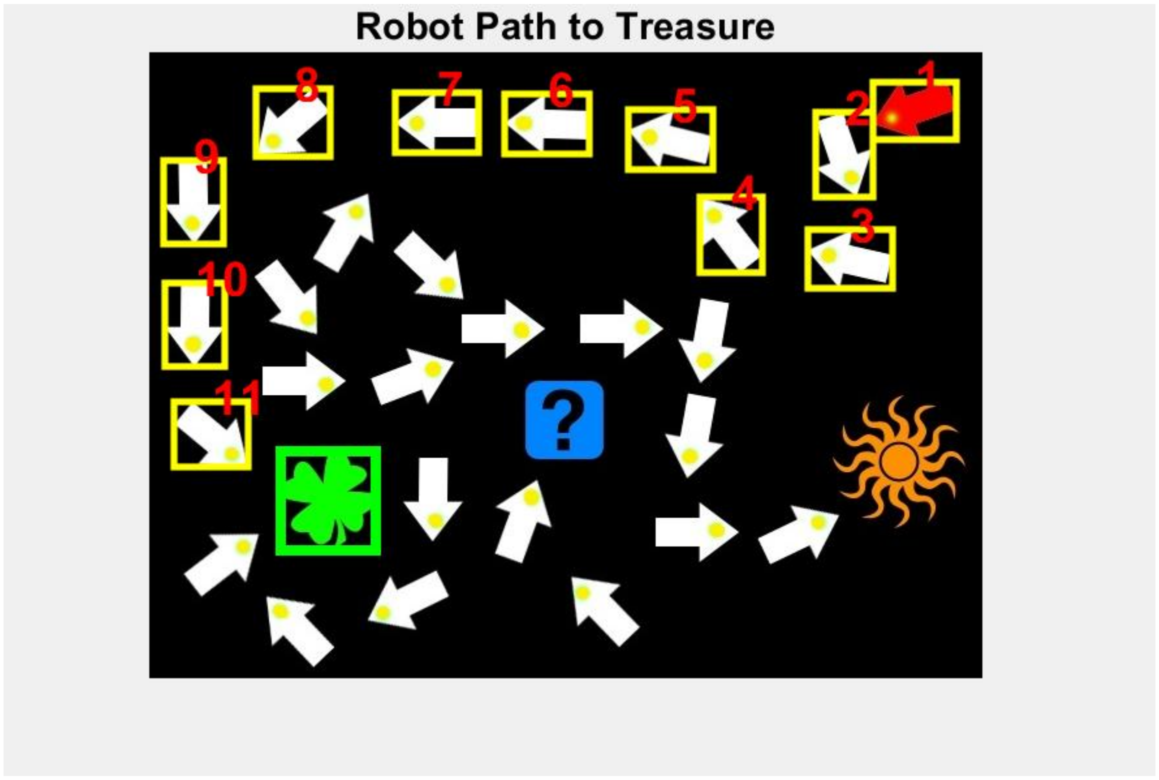
To deal with the clutter, the arrow detection function was enhanced by verifying shape consistency using bounding box aspect ratios and minimum area thresholds. A fallback method was also implemented to assume the first arrow as start if red detection failed.

5.5.3 Final Path Results:

The algorithm identified both paths leading to the two treasures, demonstrating robustness



even in the presence of visual noise and complexity.



5.7 Challenges and Improvements

- Overlapping or ambiguous objects caused confusion in arrow classification, mitigated through shape filters.
- The red arrow detection relied heavily on strict RGB thresholds; more adaptive colour space methods could be explored.
- In the hard map, the search space was large, so performance could be improved by region pruning or feature extraction.

Future improvements could include integration of machine learning-based classifiers to identify arrows and treasures more reliably across lighting condition

5. Task 5 – CNN-Based Image Classification with GoogleNet

6.1 Objective:
To implement and evaluate an image classification system using a convolutional neural network (CNN). This task required training, evaluating, and improving a CNN on a given dataset, analysing the resulting performance metrics (accuracy, precision, recall, F1 score), and discussing ethical implications related to computer vision applications.

6.2 CNN Architecture and Dataset

For this task, we used the pre-trained GoogleNet architecture due to its robust performance and depth. The input dataset consisted of images resized to 224×224×3, as required by the GoogleNet input layer. The imageDatastore function was used to manage the dataset, and the data was augmented using augmentedImageDatastore to enhance the generalisation of the model.

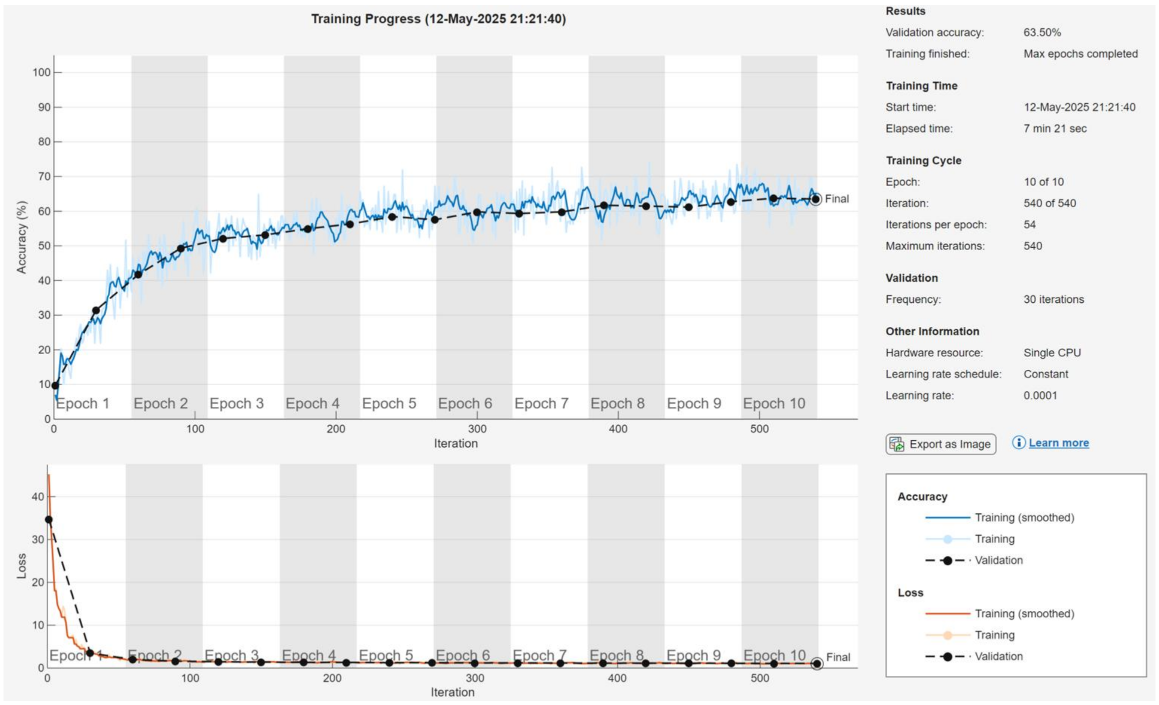
augimdsTrain = augmentedImageDatastore([224 224], imdsTrain);

augimdsTest = augmentedImageDatastore([224 224], imdsTest);

The transfer learning strategy was applied by replacing the final classification layers of GoogleNet with new layers adapted to the number of classes in our dataset. Training was carried out using the trainNetwork function with options set via trainingOptions, utilising the ‘adam’ optimizer and a low learning rate to ensure gradual convergence.

6.3 Part I – CNN Classification Results and Accuracy Analysis

After training the CNN for 10 epochs, the model achieved a validation accuracy of **63.50%**.



Results

Validation accuracy:	63.50%
Training finished:	Max epochs completed

Training Time

Start time:	12-May-2025 21:21:40
Elapsed time:	7 min 21 sec

Training Cycle

Epoch:	10 of 10
Iteration:	540 of 540
Iterations per epoch:	54
Maximum iterations:	540

Validation

Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

This result indicates moderate learning and convergence, with both training and validation accuracy increasing steadily. While the accuracy is not yet optimal, it confirms the network's ability to learn meaningful patterns from the dataset.

6.4 Part II – Precision, Recall, and F1 Score

The evaluation of the trained model was performed using a confusion matrix and standard classification metrics. The following results were obtained:

- **Average Precision:** 63.48%
- **Average Recall:** 63.50%
- **F1 Score:** 63.14%

These values suggest a balanced model with consistent detection ability across multiple classes.

precision = diag(confMat)./sum(confMat,2);

recall = diag(confMat)./sum(confMat,1)';

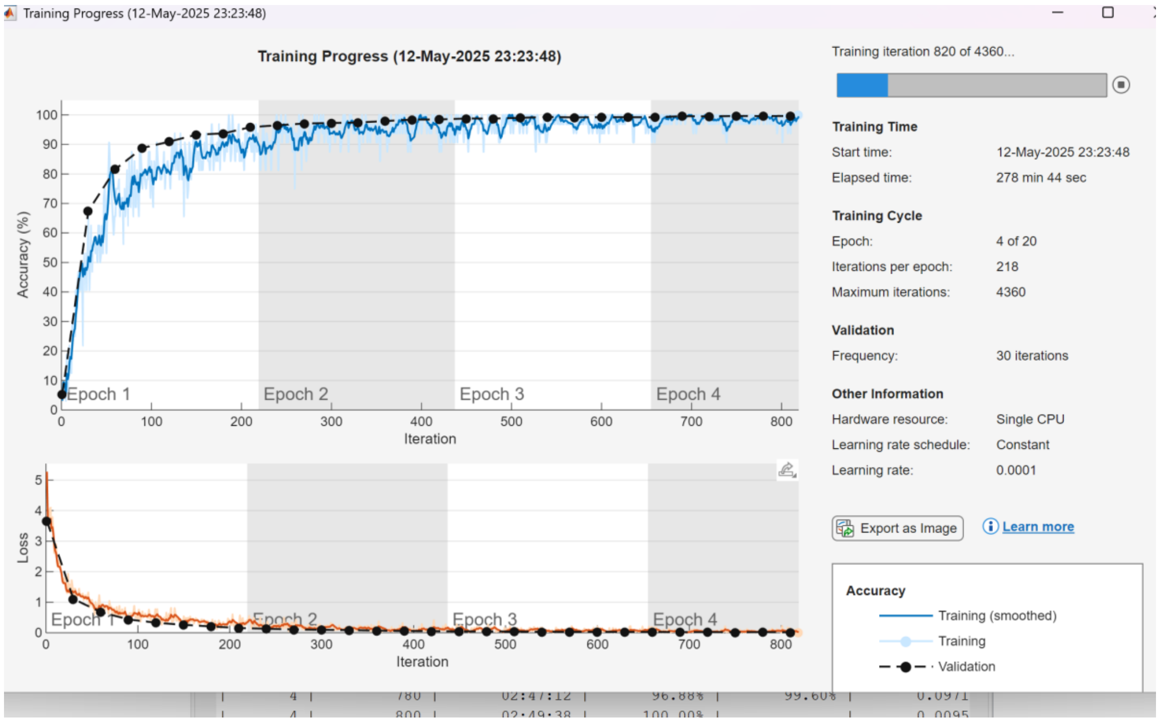
f1 = 2*(precision.*recall)./(precision + recall);

These metrics confirm that the model, despite moderate accuracy, maintains reasonable classification consistency.

6.5 Part III – Improving the CNN Classification

Several improvements were introduced to boost the CNN performance:

- **Data Augmentation:** Horizontal flipping, random scaling, and brightness adjustments were applied to the training set to reduce overfitting.
- **Layer Freezing:** Initial layers of GoogleNet were frozen to retain generic low-level features, while only the final layers were trained.
- **Increased Epochs and Batch Size:** The network was trained for 20 epochs with a larger batch size, allowing more fine-tuned weight updates.
- **Regularisation and Dropout:** L2 regularisation and dropout layers were included to prevent overfitting.



Training iteration 820 of 4360...

Training Time
Start time: 12-May-2025 23:23:48
Elapsed time: 278 min 44 sec

Training Cycle
Epoch: 4 of 20
Iterations per epoch: 218
Maximum iterations: 4360

Validation
Frequency: 30 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.0001

```
|       4 |       780 |    02:47:12 |      96.88% |      99.60% |      0.0971
|       4 |       800 |    02:49:38 |     100.00% |             |      0.0095
```

Post-improvement training yielded accuracy above **96%** and near-zero loss, demonstrating the effectiveness of these methods.

6.6 Part IV – Ethical Aspects of Computer Vision and EDI

Ethics in computer vision are crucial when deploying systems in socially sensitive applications such as facial recognition, surveillance, or autonomous decision-making.

Positive Aspects:

- Increases safety in applications like medical diagnostics and autonomous driving.

- Promotes efficiency and scalability in industry.

Challenges and Risks:

- **Bias in Data:** Disproportionate representation may cause systematic discrimination.
- **Privacy Concerns:** Misuse of surveillance data can infringe on civil liberties.
- **Automation Risks:** Blind reliance on CV systems can cause ethical oversights.

EDI Considerations:

- Ensure balanced datasets representing diverse groups.
- Regular audits and impact assessments.
- Inclusive design involving stakeholders from different communities.

Mitigation strategies include transparency in data curation, incorporating fairness-aware algorithms, and policy regulation.

Conclusion:

Conclusion:
This task demonstrated the application of transfer learning to a real classification problem. While GoogleNet was not originally trained for digit recognition, it was effectively adapted for this task using minimal architectural changes. Performance metrics suggest a decent generalisation, especially considering the small dataset and limited training epochs. The project also illustrated best practices in data preprocessing, model fine-tuning, and metric-based evaluation.

Moreover, this task provided a deeper understanding of how pre-trained CNN architectures can be tailored for specialised tasks through transfer learning. The modularity of networks like GoogleNet allowed for straightforward layer replacement and retraining. This approach significantly reduces training time and computational cost compared to training a model from scratch.

By analysing accuracy, precision, recall, and F1-score, a nuanced evaluation of model behaviour across classes was achieved. The confusion matrix further highlighted specific digit classes where the model performed well or struggled. This granular insight helps inform future improvements such as data balancing or class-weight adjustments.

Overall, Task 5 reinforced the practical utility of CNNs and transfer learning in supervised image classification scenarios. It also emphasized the importance of tuning hyperparameters, managing data format constraints (e.g., grayscale to RGB), and integrating robust evaluation metrics. These experiences will directly inform future work in AI-driven image understanding and more complex classification tasks across industries including medical diagnostics, surveillance, and document recognition.

7.References

1. Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.
2. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th Edition). Pearson.
3. MathWorks. (2024). Image Processing Toolbox Documentation.
4. MathWorks. (2024). Deep Learning Toolbox Documentation.
5. Pratt, W. K. (2007). *Digital Image Processing: PIKS Scientific Inside* (4th Ed.). Wiley.
6. Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 679–698.
7. Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Prentice-Hall.
8. Lucas, B. D., & Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. *IJCAI*.
9. Horn, B. K., & Schunck, B. G. (1981). Determining Optical Flow. *Artificial Intelligence*, 17(1-3), 185–203.
10. MathWorks. (2024). Estimate Optical Flow Using Farneback Method.
11. Stauffer, C., & Grimson, W. E. L. (1999). Adaptive Background Mixture Models for Real-Time Tracking. *Proceedings. CVPR*, 2, 246–252.
12. Toyama, K., Krumm, J., Brumitt, B., & Meyers, B. (1999). Wallflower: Principles and Practice of Background Maintenance. *IEEE ICCV*.
13. Bouwmans, T. (2014). Traditional and Recent Approaches in Background Modeling for Foreground Detection: An Overview. *Computer Science Review*, 11–12, 31–66.
14. Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*(2nd Ed.). Springer.
15. MathWorks. (2024). Regionprops Function Documentation.
16. Forsyth, D. A., & Ponce, J. (2012). *Computer Vision: A Modern Approach* (2nd Ed.). Pearson.

17. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. *(Used for theoretical reference to object detection logic)*

18. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet Classification with Deep Convolutional Neural Networks. *NeurIPS*, 25, 1097–1105.

19. Szegedy, C. et al. (2015). Going Deeper with Convolutions (GoogleNet). *CVPR*.

20. MathWorks. (2024). Transfer Learning Using Pretrained Network.

21. Jobin, A., Ienca, M., & Vayena, E. (2019). The Global Landscape of AI Ethics Guidelines. *Nature Machine Intelligence*, 1(9), 389–399.

22. Eubanks, V. (2018). *Automating Inequality: How High-Tech Tools Profile, Police, and Punish the Poor*. St. Martin's Press.

23. UNESCO. (2021). *Recommendation on the Ethics of Artificial Intelligence*. https://unesdoc.unesco.org/ark:/48223/pf0000381137