The University of Sheffield

ACS61012 Machine Vision

Leander Stephen Desouza

Registration Number: 230118120

Department of Automatic Control Systems Engineering

Contents

1 Task 1 - Introduction to Machine Vision **1**

1.1 Part I: Introduction to Image Processing 1

1.1.1 RGB Histogram 1

1.2 Part II: Edge Detection and Segmentation of Static Objects 3

1.2.1 Static Objects Segmentation by Edge Detection 3

2 Task 2 - Object Motion Detection and Tracking **5**

2.1 Corner analysis using Optical Flow 5

2.2 Optical Flow test for GingerBreadMan 8

3 Task 3 - Automatic Detection of Moving Objects in a Sequence of Video Frames **10**

3.1 Frame Differencing Approach 10

3.2 Gaussian Mixture Models 11

4 Task 4 - Robot Treasure Hunting **13**

4.1 Easy Case 13

4.2 Medium Case 15

4.3 Hard Case 16

5 Task 5 - Convolutional Neural Networks for Image Classification **18**

5.1 Accuracy of Default LeNet5 18

5.2 Precision, Recall and F1 Score of Default LeNet5 Model 19

5.3 Improvement of the LeNet5 Model 20

5.4 Ethics in Computer Vision 23

5.4.1 Positives of using Computer Vision 23

5.4.2 Challenges and Mitigation of using Computer Vision 24

5.4.3 Equality and Diversity in Computer Vision 24

6 Appendix **25**

List of Figures

1 RGB Histogram of Mallard indicating low resolution and high intensity . . 2
2 RGB histogram of a dog indicating high resolution and low intensity 2
3 Comparison of various edge detection algorithms 4
4 Corners detected in the red square sample image 5
5 Estimated Trajectory using Lucas Kanade vs Ground Truth Data 6
6 RMSE comparison throughout the video frames 7
7 Corner points in the GingerBreadMan 8
8 Flow vectors superimposed on the second image of GingerBreadMan 9
9 Plot of the flow vectors observed after Lucas-Kanade optical estimation . . 9
10 Frame Difference with a binary thresholding of 10 and 25, at frame 140 . . 11
11 GMM with n_frames = [1, 10], and n_gaussians = [2, 20] at frame 140 . . 12
12 Intermediate steps in the Easy Case 13
13 Intermediate steps in the Medium Case 15
14 Intermediate treasure in Hard Case 16
15 Intermediate steps in Hard Case 17
16 Training Progress on LeNet5 19
17 Confusion Matrix of the LeNet5 model 19
18 Training Progress on Custom model 22
19 Confusion Matrix of the Custom model 22

1 Task 1 - Introduction to Machine Vision

1.1 Part I: Introduction to Image Processing

1.1.1 RGB Histogram

An elementary way to analyze the image composition is to describe its RGB contribution. This helps us to understand the overall wavelengths in the image and can help in masking out certain features of the image [1]. The basic idea is if a certain area of the image needs to be segmented, its RGB or HSV ranges can be studied, and a suitable mask can be applied.

Analysis

In the following figures, we have compared the RGB histograms of a dog and mallard. For more information on the RGB histogram embedding, refer to Task -1, part -1 of the Appendix. The clear inference is the intensity of the supplied images. That is observed in the shift in the intensity of the graph; the figure 3 has low intensity and hence has the plot shifted towards the origin, whereas the 1 has greater intensity and is away from the origin.

The second inference is in the pixel density of the images. The mallard image has a significantly lower resolution than that of a dog, and hence, the frequency of the pixels is much lower as well.

In addition, there is no clear domination of a particular channel in either image. However, the green colour dominates in figure 1; this is due to the increased pixels in the mallard's head and the green grass. In figure 3, the blue colour dominates as the image seems to be taken at night with a blueish-green tint.

Conclusion

By comparing the plots of the mallard and dog, we understand the differences in the intensity, RGB composition and image resolution. The mallard has a high intensity and lower image resolution and is green dominant, whereas the dog has a lower intensity and higher image resolution and is blue dominant as inferred from the plot.

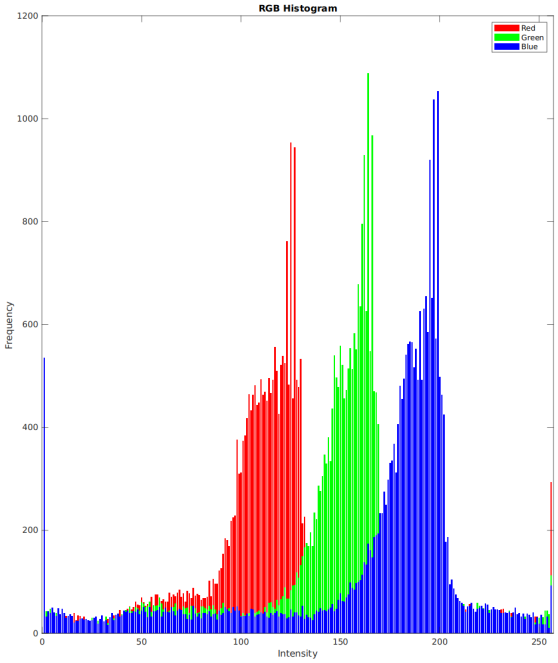

Figure 1: RGB Histogram of Mallard indicating low resolution and high intensity

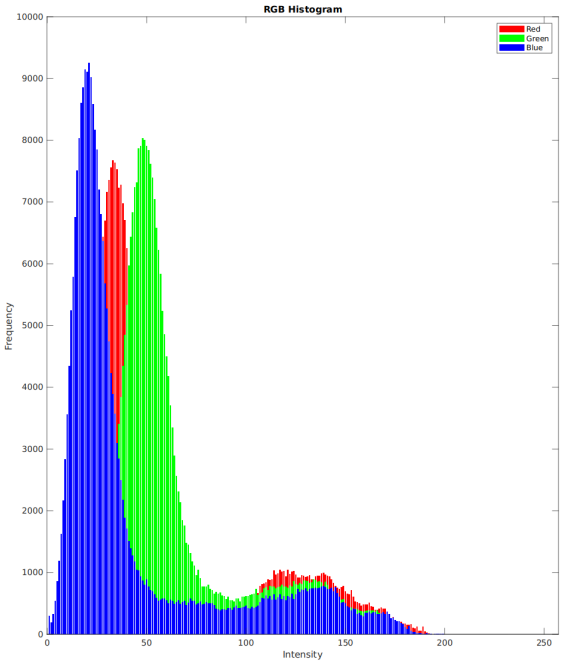

Figure 2: RGB histogram of a dog indicating high resolution and low intensity

1.2 Part II: Edge Detection and Segmentation of Static Objects

1.2.1 Static Objects Segmentation by Edge Detection

Here, we compare three popular edge detection algorithms, namely, Sobel, Canny and Prewitt methods. For all of these detection methods, the input needs to be converted into grayscale images as additional channels provide no data relevant to edge analysis [2]. For additional information about the subplots of the operators, refer to Task -1, part 2 of the Appendix. **Analysis**

1. **Sobel Operator**:

   This method of edge detection focuses on taking gradients of the image intensity function. The derivatives with respect to horizontal and vertical directions are taken and combined to find the edges in each direction. Finally, this is convoluted with the original image to highlight areas where the partial derivative is constant.

   This is incorporated in the form of edge detection kernels for each direction:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

2. **Canny Operator**:

   This method relies on finding the maximum intensity gradient of the image and suppressing the unwanted pixels after a full initial scan. Furthermore, a hysteresis thresholding step is applied to differentiate between false positives. Here, the initial step of Gaussian filtering proves to be essential as this filter tends to be more sensitive to noises that are Gaussian in nature.

3. **Prewitt Operator**:

   Similar to the Sobel operator, the Prewitt operator focuses on the approximation of the gradient of the intensity function of the image. However, the key difference lies in the computation of the gradient. This operator is applied on smaller filters that

span across horizontal and vertical directions.

The following are the corresponding kernels for each direction:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 3: Comparison of various edge detection algorithms

Conclusion

Various edge detection methods have been applied to the image of the mallard, as shown. All of the algorithms perform comparatively well. However, even after extensive tuning, there seems to be many false positives in the Canny edge detection image. This may be due to the initial Gaussian filtering step that is involved in the Canny operator, as the filter can have a tendency to nullify some image data as well.

2 Task 2 - Object Motion Detection and Tracking

2.1 Corner analysis using Optical Flow

In this task, we track a specified object of interest throughout its duration in a video frame. We find the optical flow of the vectors of importance and understand the overall trajectory that the object takes in its lifetime [3].

First, we estimate the corner points of the red square provided to us in the lab brief. The procedure used to detect corners here is the Harris Corner Detection Method. The final output is seen in 4, where the corners are detected by the + sign.

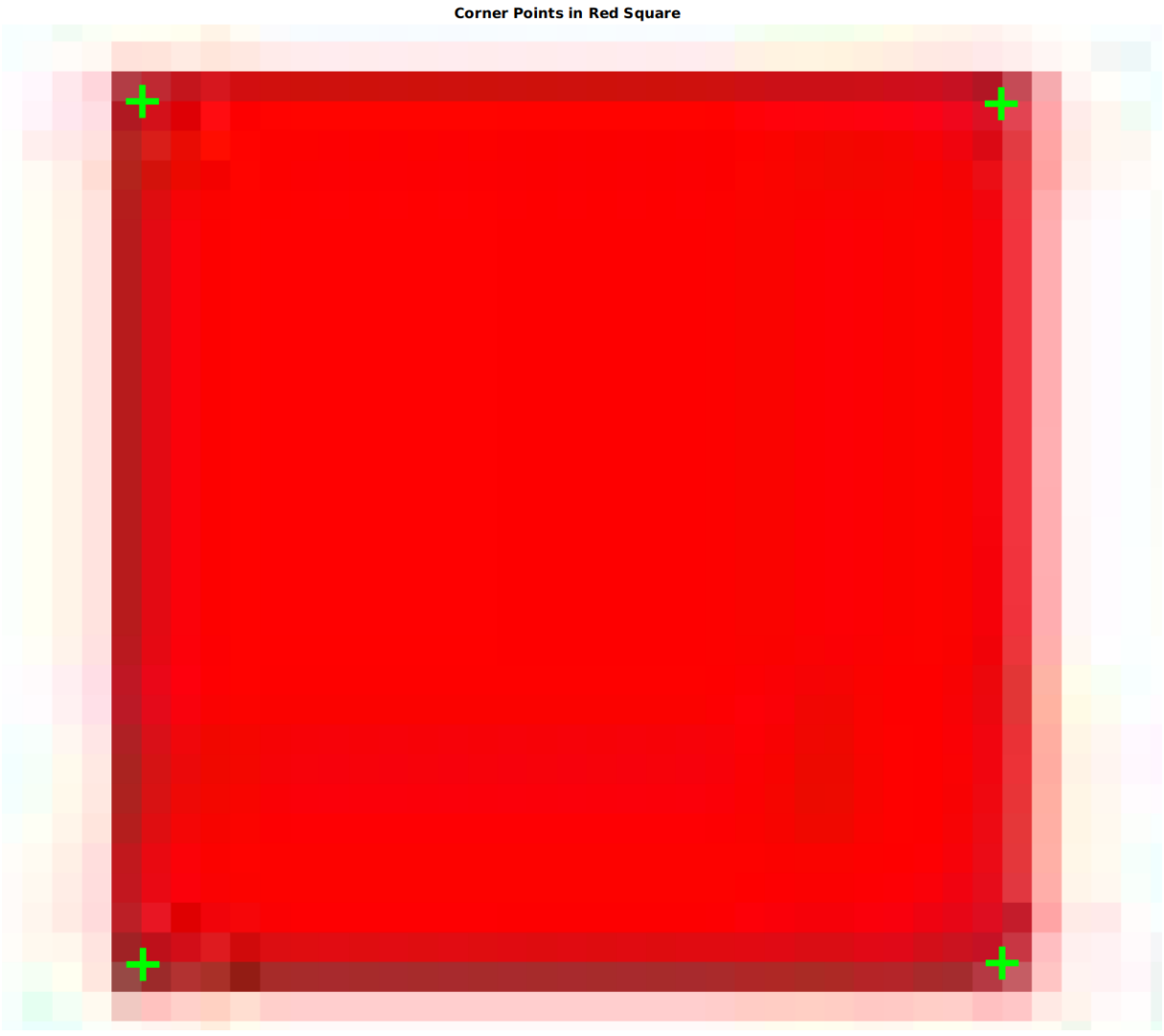

Figure 4: Corners detected in the red square sample image

We then proceed to track the top left corner of the square throughout the video by using the Harris method. The optical flow of the video is estimated using the Lucas

Kanade method, and the estimated new corner is interpolated using the flow calculated. This is then compared with the ground truth data provided while maintaining the RMSE score.

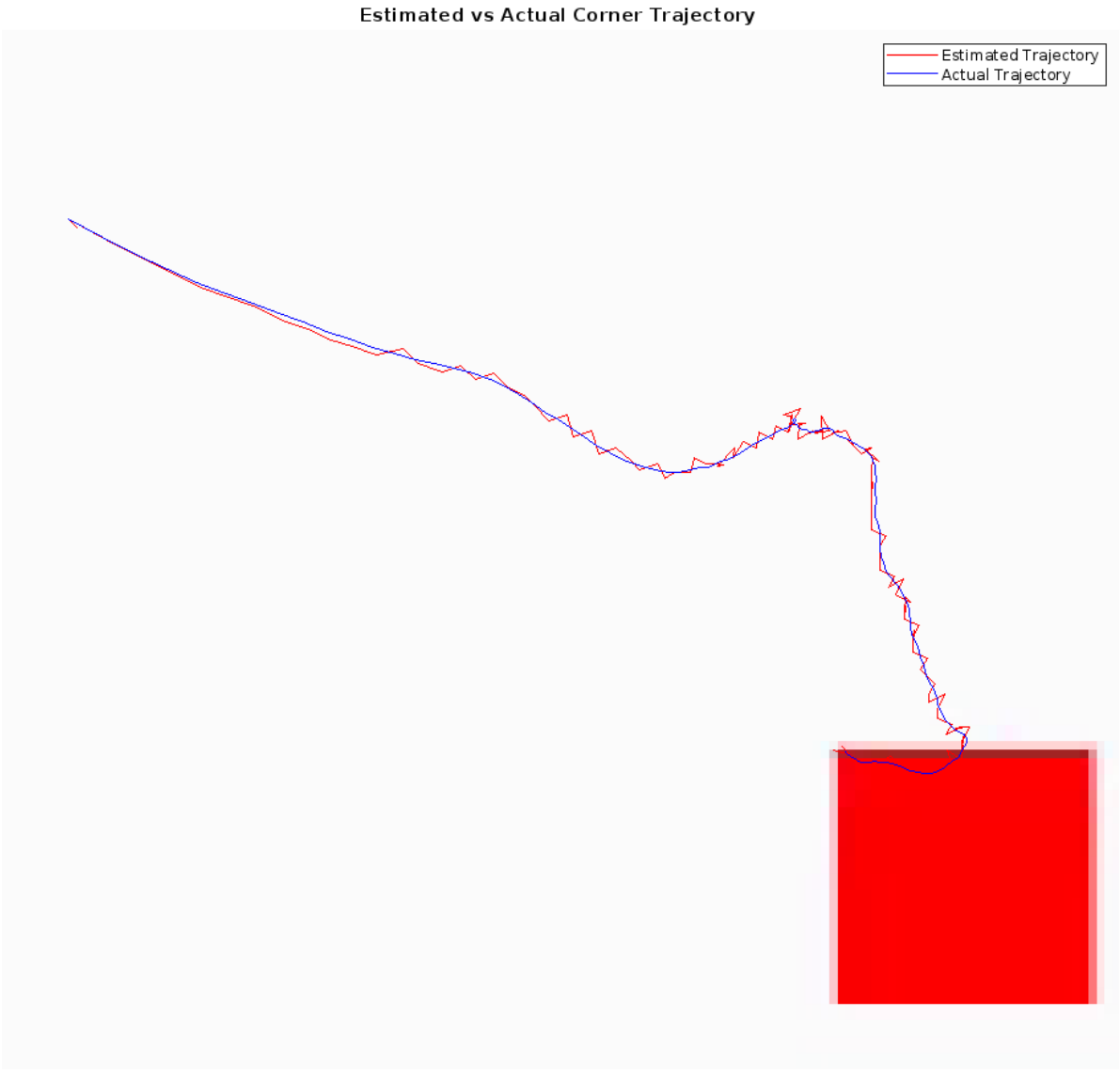

Figure 5: Estimated Trajectory using Lucas Kanade vs Ground Truth Data

As seen in the figure 5, the estimated trajectory closely follows the ground truth with some occasional spikes. This deviation is measured and calculated as follows:

$$RMSE_x = \frac{1}{n-1}\sum_{i=1}^{n-1}\sqrt{(x_{actual}(i+1) - x_{predicted}(i))^2}$$

$$RMSE_y = \frac{1}{n-1}\sum_{i=1}^{n-1}\sqrt{(y_{actual}(i+1) - y_{predicted}(i))^2}$$

$$RMSE_{xy} = \frac{1}{n-1} \sum_{i=1}^{n-1} \sqrt{(x_{actual}(i+1) - x_{predicted}(i))^2 + (y_{actual}(i+1) - y_{predicted}(i))^2}$$

Here, n is the number of frames, and i is each frame in the corresponding video stream. As you can see, the predicted frame is compared with the next future frame in order to determine the stability of the optical flow, hence the figure 6 starts from the second frame. All these formulas have been accordingly in Task 2, part 1 of the Appendix.

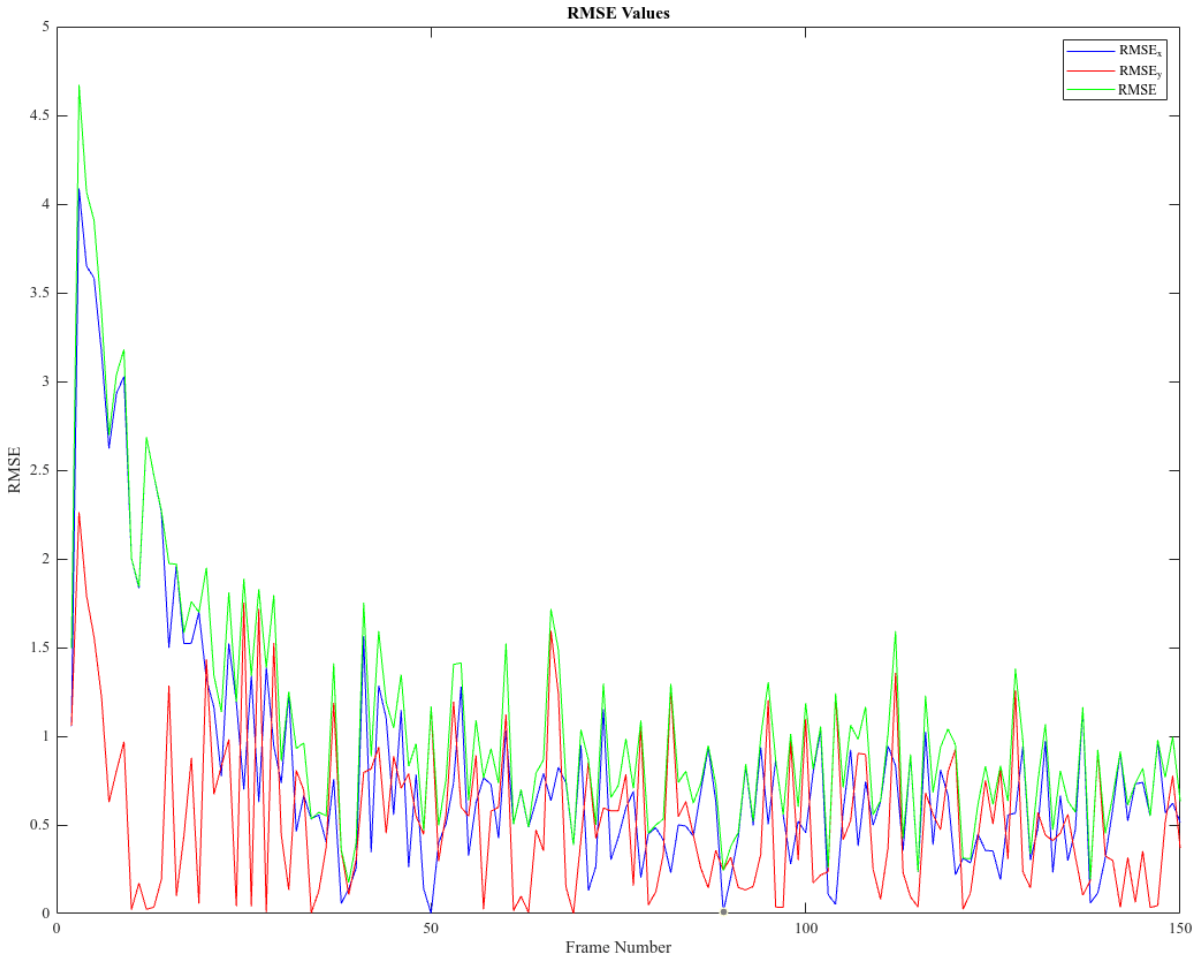

Figure 6: RMSE comparison throughout the video frames

From the figure, the RMSE starts at a high value but effectively converges as the video frames increase, resulting in a better prediction. The overall RMSE values are as follows:

$$RMSE_x = 0.8329$$

$$RMSE_y = 0.5483$$

$$RMSE_{xy} = 1.0891$$

2.2 Optical Flow test for GingerBreadMan

Similarly to the red square, first, we need to track the interesting points in the gingerbread man. Therefore, we extract the corner points using Harris method as seen in figure 7.

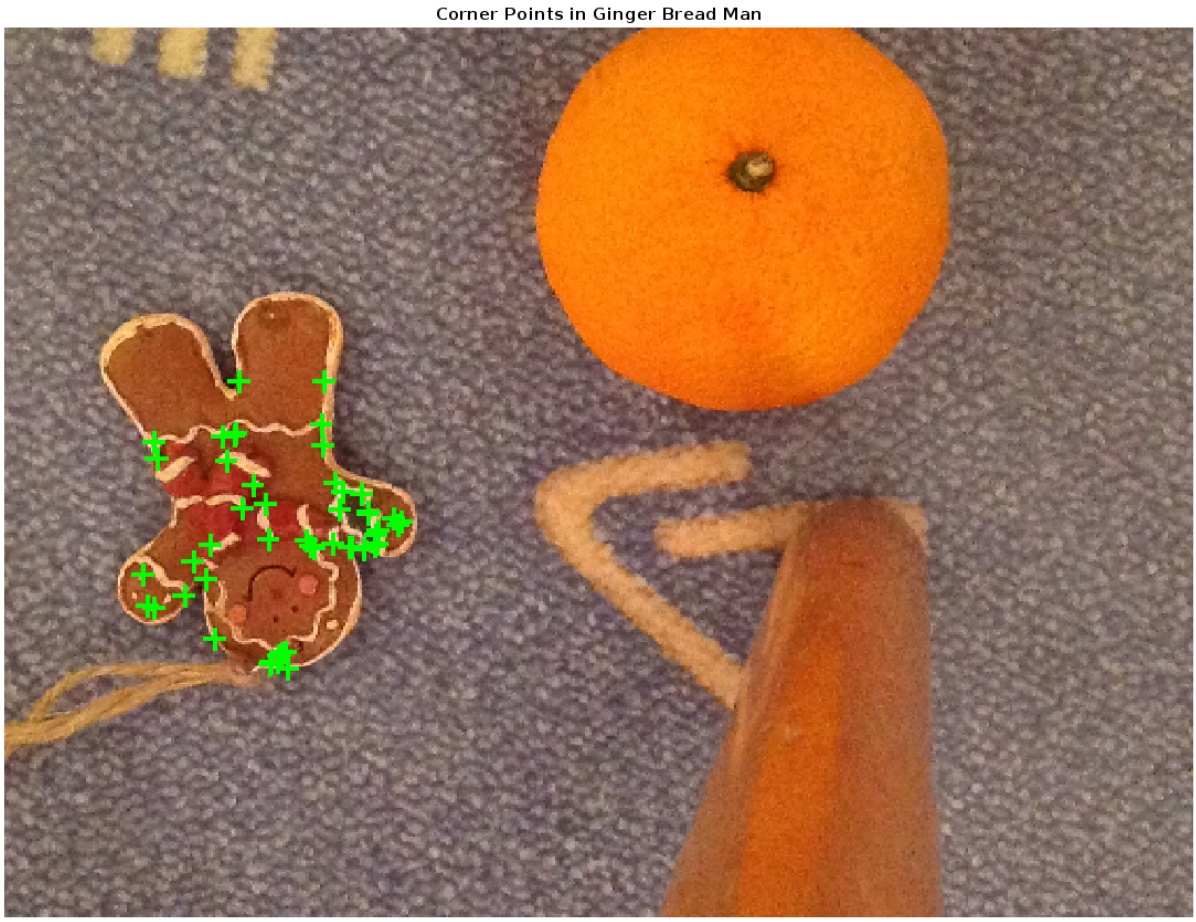

Figure 7: Corner points in the GingerBreadMan

We then proceed to save the optical flow vectors from the first image provided. This is then supplied to the second image for a better and updated determination of flow velocity vectors. As observed, the flow vectors centralise over the contours of the GingerBreadMan, indicating a successful flow estimation.

Figure 8: Flow vectors superimposed on the second image of GingerBreadMan

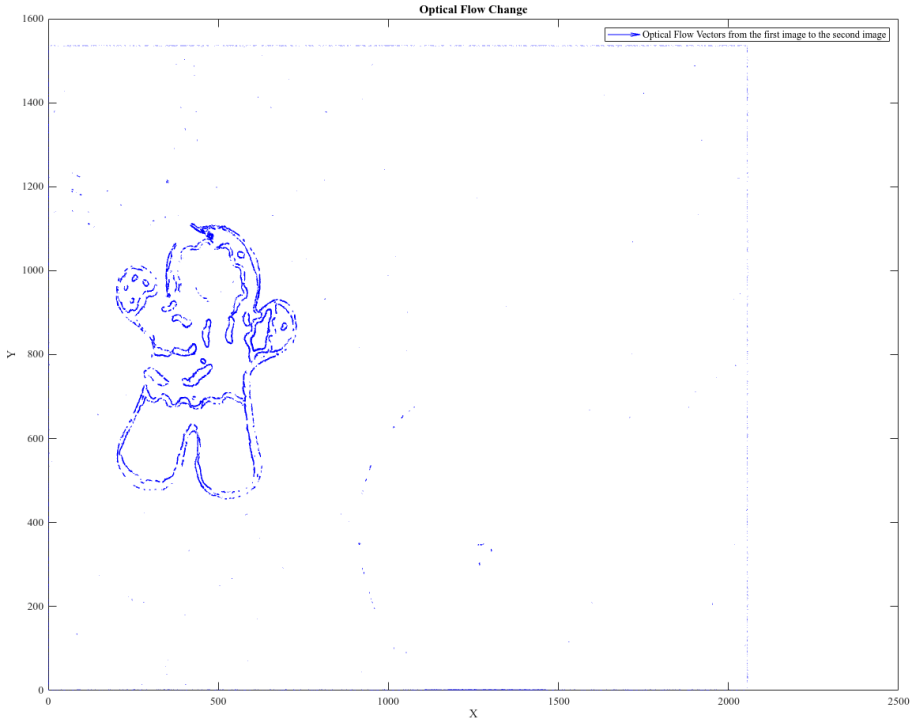

Figure 9: Plot of the flow vectors observed after Lucas-Kanade optical estimation

Conclusion

Here, we have tested the flow estimate for two videos, namely, the movement of a red square on a white background and a gingerbread cookie. Passing flow vectors from frame to frame has resulted in a better estimate of the next position of the object of interest, as seen in figure 6, as the initial estimate proved to contain high RMSE values at the start, but eventually converged.

The RMSE errors proved to be satisfactory in the case of the Red Square video and, indeed, were highly accurate as they matched the ground truth data very well. Additionally, the plot of the flow vectors from the gingerbreadman matched its contours, proving that the flow estimate is precise.

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

3.1 Frame Differencing Approach

Background subtraction is an essential tool in Image Processing to determine the object of interest. A very naive approach is to use basic frame subtraction in order to extract the moving frames. This method is extremely computationally light and can produce reasonable results [4].

We proceed by considering the first frame of the video as the background and subtracting each subsequent frame from the former. This is done by binary thresholding the result with a customizable threshold to vary. Additionally, the frame of reference is updated for each step. The procedure can easily be cross-referenced by taking Task 3, part 1 of the Appendix, into account.

As observed, thresholding is very crucial to obtaining accurate results. Otherwise, insignificant frames get overlooked in frame difference due to increased frame sensitivity.

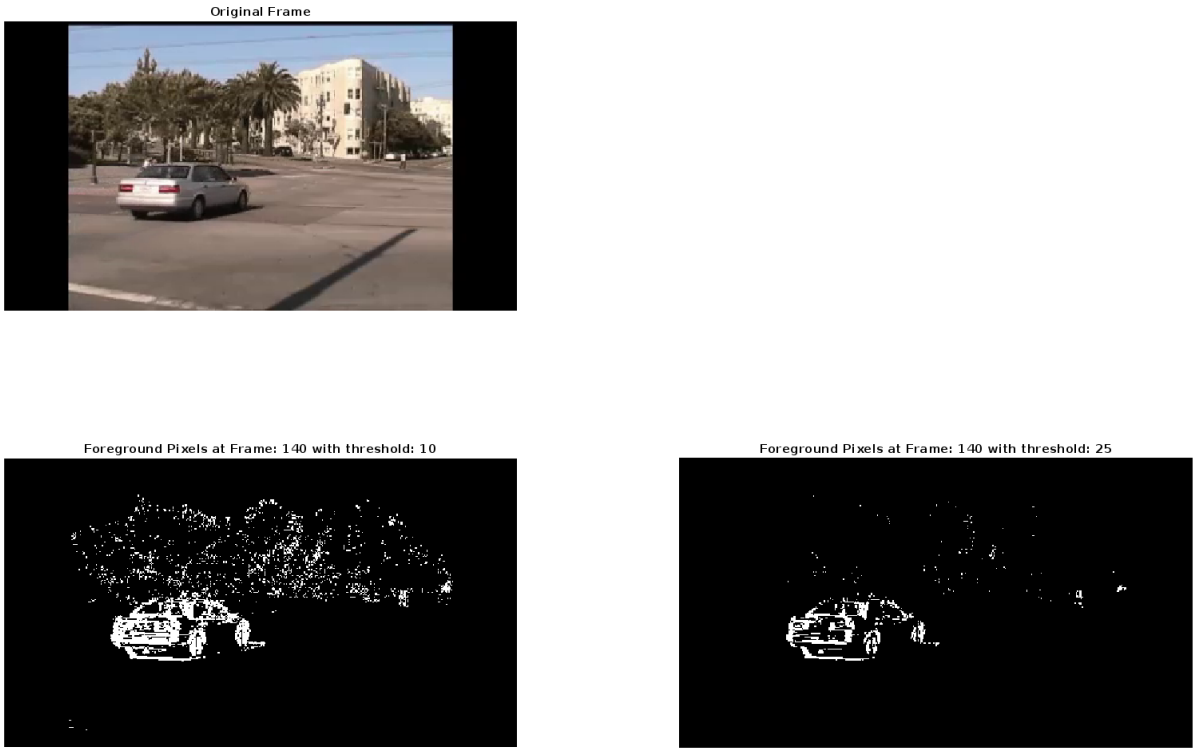

Figure 10: Frame Difference with a binary thresholding of 10 and 25, at frame 140

Analysis

The swaying of trees produces movable pixels and hence gets through the frame difference approach. However, the frame sensitivity can be reduced by varying the binary threshold. However, it may be difficult to capture subtle changes in the background while following this method. Moreover, there is no filtering step in frame differencing if noise is present in the video stream.

3.2 Gaussian Mixture Models

Gaussian Mixture Models are probabilistic models that represent small models that are probabilistic within an overall distribution. They break down complex multi-modal distributions into smaller Gaussian models in the aspects of mean and variance [5].

First, we supply an initial amount of frames to train the foreground detector. Hence, any additional training frames after a specific threshold do not affect the performance of the overall detector. A foreground mask is created after the model is trained using the Gaussian mixture models. This mask is then applied to each frame of the incoming video stream. Several tests can be made by varying the number of training frames and the number of Gaussian models by setting the initial parameters in T3 part2 of the Appendix.

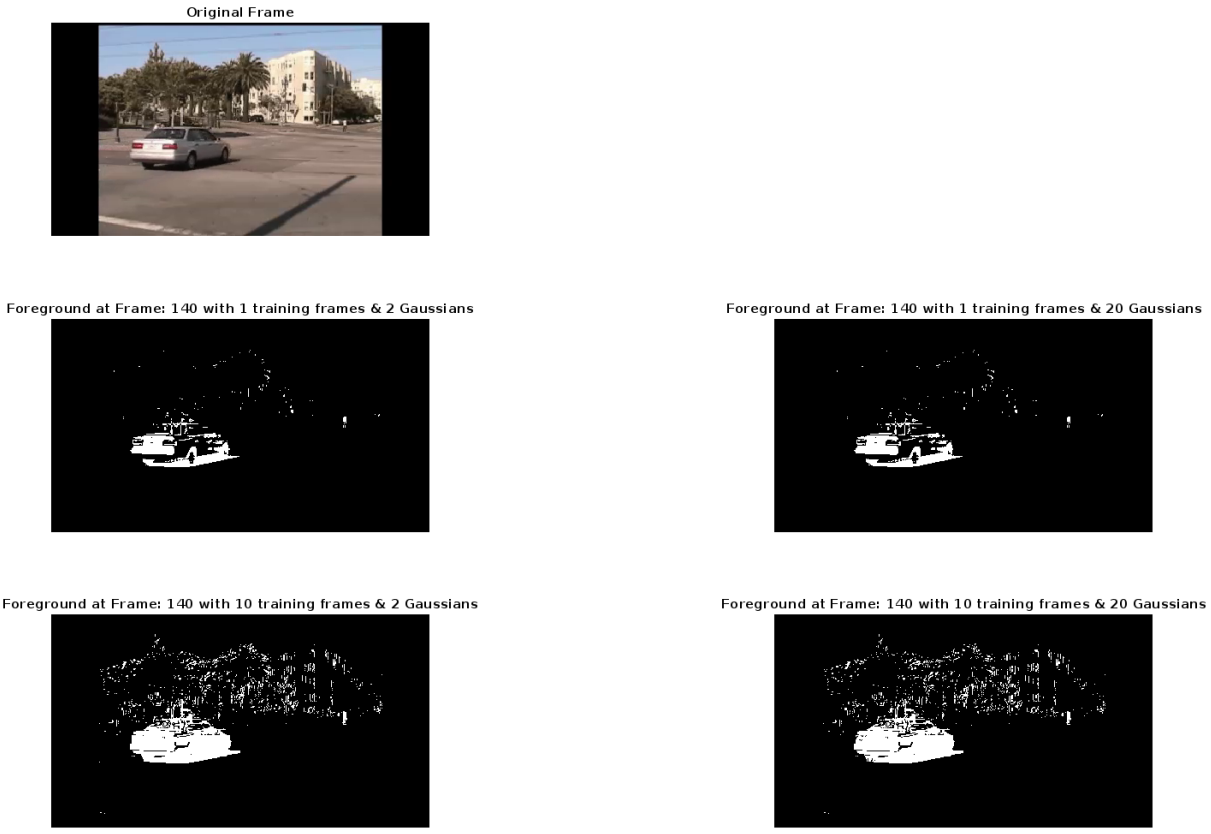

Figure 11: GMM with n_frames = [1, 10], and n_gaussians = [2, 20] at frame 140

Analysis

By analyzing the figure 11, we can observe that the most substantial parameter to vary is the number of training frames. As the training frames increase and saturate, the swaying of trees and the man walking in the background are observed as the filtered output. The Gaussian submodel split for 2 and 20 does not have a significant impact on the overall background subtraction.

4 Task 4 - Robot Treasure Hunting

4.1 Easy Case

We proceed with the underlying blueprint for all the cases as follows:

1. Converting the RGB image into a binary image
2. Analyzing the connected components in the image
3. Drawing and labelling the bounding boxes in the image

After getting the binary image, we label the connected clusters of pixels in an enumeration. Therefore, an index is assigned to every cluster of white pixels detected. A good way to visualize the detected clusters is by using the $label2rgb()$ function. This is clearly seen in the figure 12, the neon pixel clusters are the labelled components [6]. All subsequent functions mentioned here can be referenced in Task 4 of the Appendix.

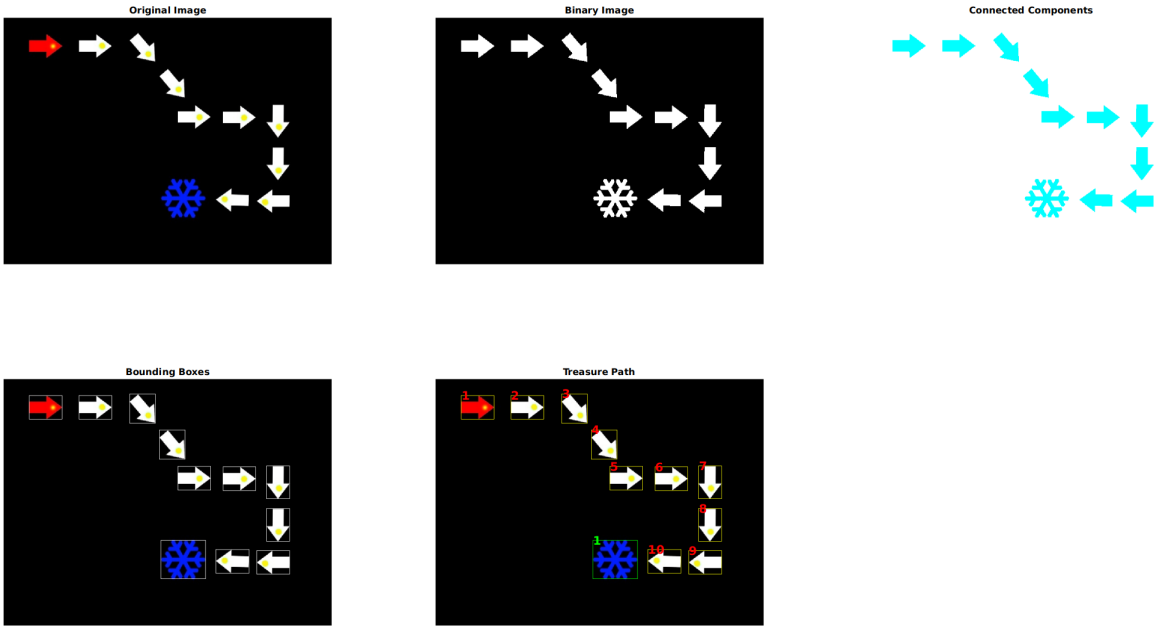

Figure 12: Intermediate steps in the Easy Case

This is done so we can get crisp and defined properties for the region properties. This helps us to get more accurate bounding boxes, centroids, and areas. After the bounding boxes are drawn and labelled, the main task is to label which boxes are arrows, treasures, or false detections. This is determined by the $arrow\_finder()$ function.

First, let us define the conditions for a bounding box to be an arrow:

1. The length and breadth of the bounding box need to be of a minimum length and breadth in order to be an effective detection.

2. The centroid of the bounding box has to be of white or yellow colour.

Then, let us define the suitable outlines for a bounding box to be a treasure:

1. The length and breadth of the bounding box need to be of a minimum length and breadth in order to be an effective detection.

2. The area of the bounding box is sufficiently larger than the rest of the boxes being detected.

Once we have extracted the indices of the treasure and arrows from the list of bounding boxes, the next task is to determine the starting point of the path to follow. This is easily determined as, in all of the three tasks, we only have a single red arrow as the starting point. Therefore, the starting index is determined by whether the centroid of the arrow bounding box is red.

Next, we need to determine the next arrow for traversal from the start. This is determined by the $next\_object\_finder()$ function. This returns the next object in the path, whether it is an arrow or a treasure.

1. Effectively, the intuitive way to travel to the treasure is to follow from the tip of the current arrow to the base of the next arrow. This is done by calculating the distance from the current yellow centroid to the centroid of the next object.

2. The yellow centroid is computed in the $get\_yellow\_centroid()$ function. This is done by cropping a region of interest and applying an HSV yellow mask to extract the bounding box of the yellow dot. The yellow centroid is then computed using region properties.

3. Next, we iterate through all the objects detected by $arrow\_finder()$ and omit objects already in the treasure path.

4. The centroid of the iterative object is computed, and the effective difference is monitored with the yellow centroid of the current arrow.

5. Once the index of the object is determined with the lowest effective distance, then the object's index is returned.

6. While iterating, the distance with the treasure is monitored, and a check for the area is done to prevent unwanted treasures.

7. After the index of the next object is determined, it gets appended in the path to follow. This is repeated until the object encounters a treasure.

We then proceed to draw yellow bounding boxes in the path if it is an arrow or green if it is a treasure. All the steps above can be inferred from the codebase in the appendix and the figure 12.

4.2 Medium Case

Here, the crux of this case is to determine which arrows are useful and which treasure is the destination. The earlier case for monitoring the distance between the current yellow centroid and the target object centroid still proves beneficial.

Additionally, proper thresholds for the treasure area need to be determined. This is easily found by monitoring the areas in the region properties of the bounding boxes detected. Hence, the path properly detects the fish instead of the iris flower.

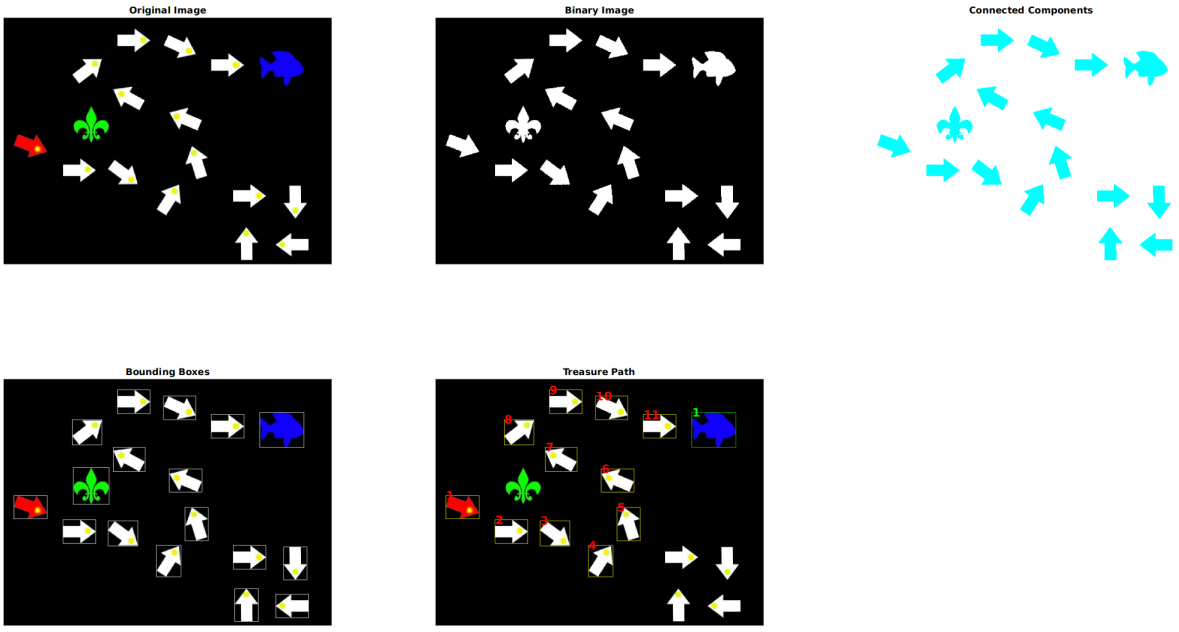

Figure 13: Intermediate steps in the Medium Case

The path does not choose the four arrows in the bottom right corner in the figure 13, as after the 5th arrow, the effective distance of the 6th labelled arrow is far smaller than the corresponding arrows.

$$effective\_distance = norm(current\_yellow\_centroid - target\_centroid)$$

4.3 Hard Case

Following the approach in the medium case, it segregates the treasures of interest by simply taking the greatest pair of areas in the image. This works because the question mark box in figure 14 is smaller than the clover and the sun object.

However, since we have two treasures here, we need to consider the next point to follow to be from the intermediate treasure rather than start from an arrow. Here, the effective distance does not work directly, as there is no yellow centroid from which to calculate.

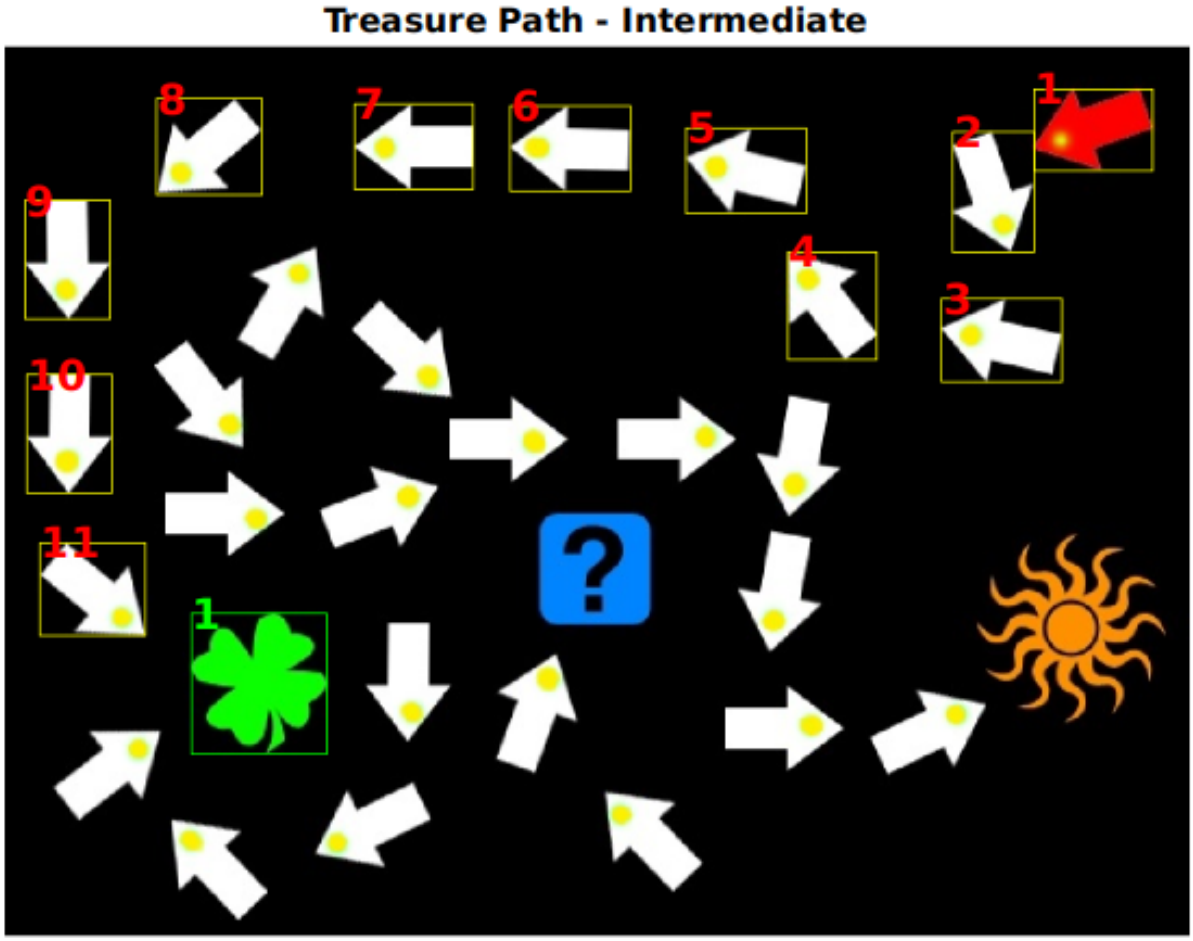

Figure 14: Intermediate treasure in Hard Case

Here, we skip all the next arrows lying in the width of the bounding box of the first clover treasure. This helps us to avoid "**jumps**" or "**diagonal movements**" while following the path to the second treasure. After following this simple trick, we can follow the effective distance defined in the easy and medium tasks, respectively.

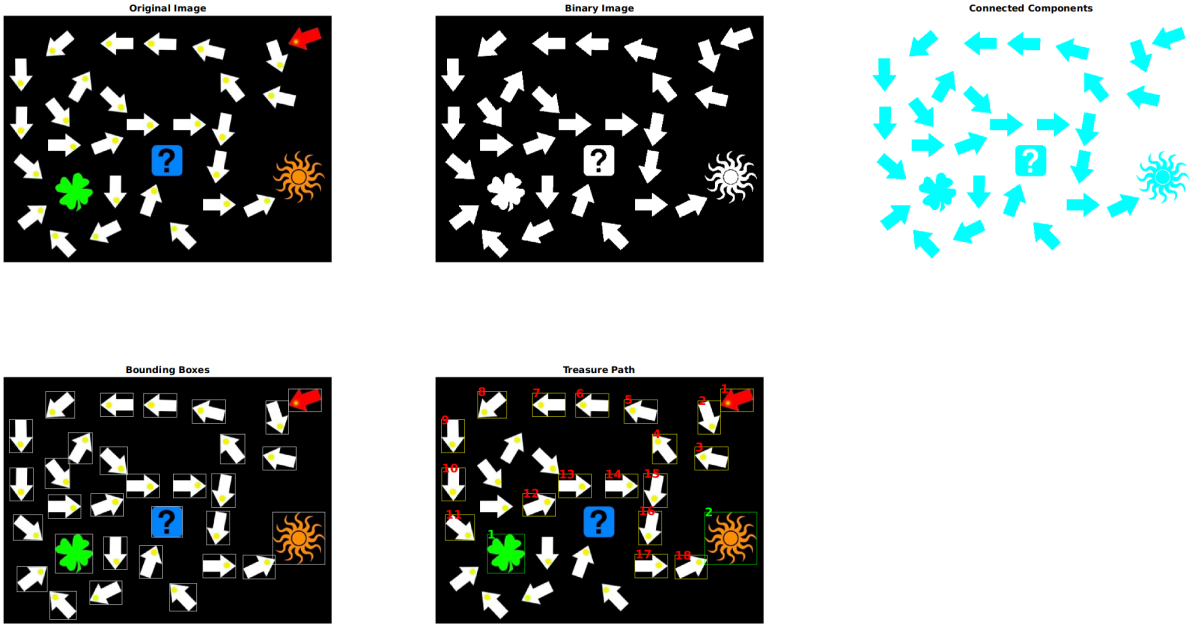

Figure 15: Intermediate steps in Hard Case

5 Task 5 - Convolutional Neural Networks for Image Classification

5.1 Accuracy of Default LeNet5

The default LeNet5 architecture is tested on the popular digits dataset, which consists of 10 classes. The layers and the training options have been pre-defined to match the architecture of Google's LeNet5 [7].

```
% Define the LeNet-5 architecture
layers = [
    imageInputLayer([image_size 1])

    convolution2dLayer(5,6)
    averagePooling2dLayer(2,'Stride',2)

    convolution2dLayer(5,16)
    averagePooling2dLayer(2,'Stride',2)

    fullyConnectedLayer(120)
    fullyConnectedLayer(84)

    fullyConnectedLayer(num_classes)
    softmaxLayer
    classificationLayer
];
```

As observed in the above code snippet, the architecture has two hidden layers and two fully connected layers. In addition, the training step uses a Stochastic Gradient with momentum as its optimizer. A custom change was introduced in the training options to use the GPU by default to speed up the model training. For more information on the GPU configuration, refer to Task 5 of the Appendix.

As seen from the figure 16, the accuracy of the model is about 63.26%.

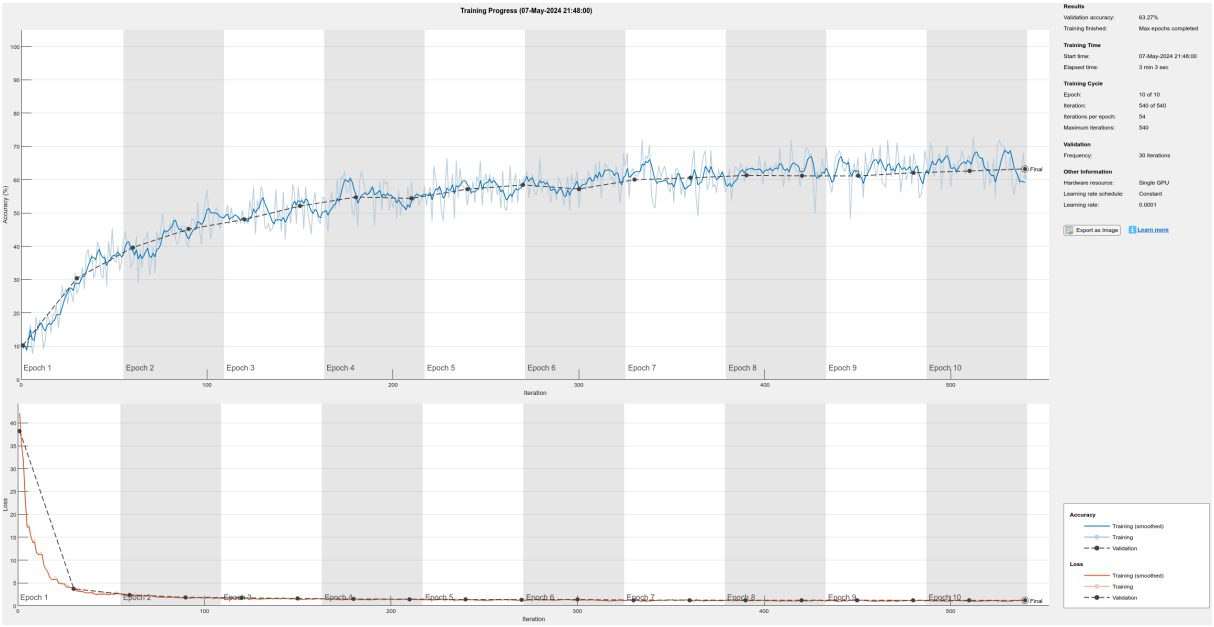

Figure 16: Training Progress on LeNet5

5.2 Precision, Recall and F1 Score of Default LeNet5 Model

In order to calculate the precision, recall and f1 score, we need to plot the confusion matrix from the training results.

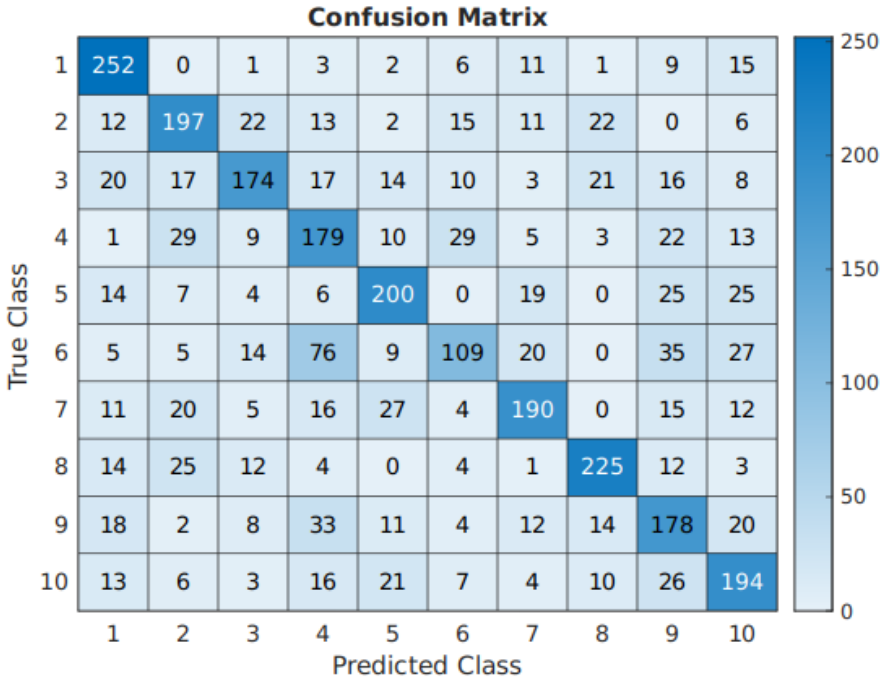

Figure 17: Confusion Matrix of the LeNet5 model

The following are the equations for calculating the precision, recall, and f1 score for each of the classes:

```
precision = diag(confusionMatrix) ./ sum(confusionMatrix, 2);
recall = diag(confusionMatrix) ./ sum(confusionMatrix, 1)';
f1Score = 2 * (precision .* recall) ./ (precision + recall);
```

Using the above equations, we get the following results:

$$precision = [0.84, 0.65, 0.58, 0.59, 0.66, 0.36, 0.63, 0.75, 0.59, 0.64]$$

$$recall = [0.7, 0.63, 0.69, 0.49, 0.67, 0.57, 0.68, 0.76, 0.52, 0.60]$$

$$f1\_score = [0.76, 0.64, 0.63, 0.53, 0.67, 0.44, 0.65, 0.75, 0.55, 0.62]$$

$$overall\_precision = 0.63$$

$$overall\_recall = 0.63$$

$$overall\_f1\_score = 0.62$$

5.3 Improvement of the LeNet5 Model

The following methods have been employed in order to increase the overall accuracy of the system:

1. **Regularization methods**: L2 regularization has been tuned for maximum performance and applied to the fully connected layer. In addition, to prevent overfitting, a dropout method of a tuned probability measure is also added after the fully connected layers.

2. **Activation function**: Two ReLU layers have been applied to the two hidden layers of the custom architecture in order to provide an accurate activation variation to the output of each layer.

3. **Custom Architecture Modifications**: A couple of hidden layers have been applied prior to fully connected layers. In addition, a batch normalization layer has been applied to the output of every 2D convolution filter in order to offer a base unit output to the next layers.

4. **Hyperparameter Tuning**: The filter size and the number of filters for each hidden layer have been tested and modified for maximum performance. Moreover, the optimizer has been changed from *sgdm* to *adam*, because of its robustness.

The newly formed layers look as follows:

```
layers = [
    imageInputLayer([image_size 1])

    % Hidden Layer 1
    convolution2dLayer(filterSize, numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    % Hidden Layer 2
    convolution2dLayer(filterSize, numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    fullyConnectedLayer(fullyConnectedLayerSize, ...
        'WeightL2Factor', l2_reg, 'BiasL2Factor', l2_reg)
    dropoutLayer(0.2)

    fullyConnectedLayer(num_classes)
    softmaxLayer

    classificationLayer
];
```

Now, we proceed to train the custom model with the *adam* optimizer and analyse its output metrics as well.

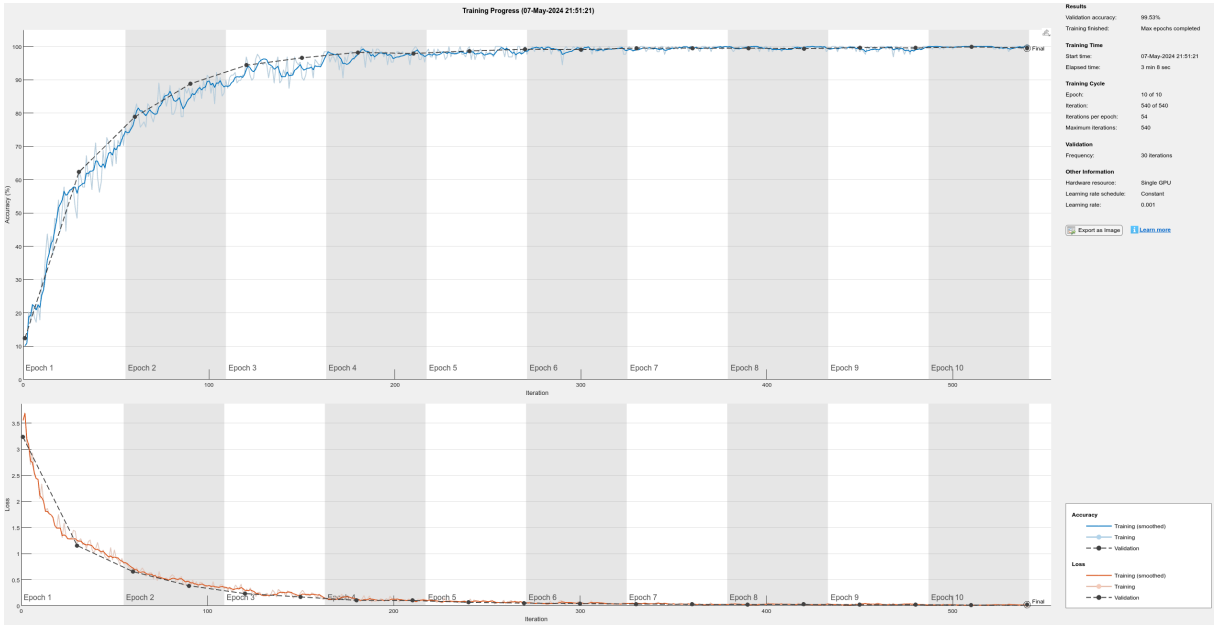

Figure 18: Training Progress on Custom model

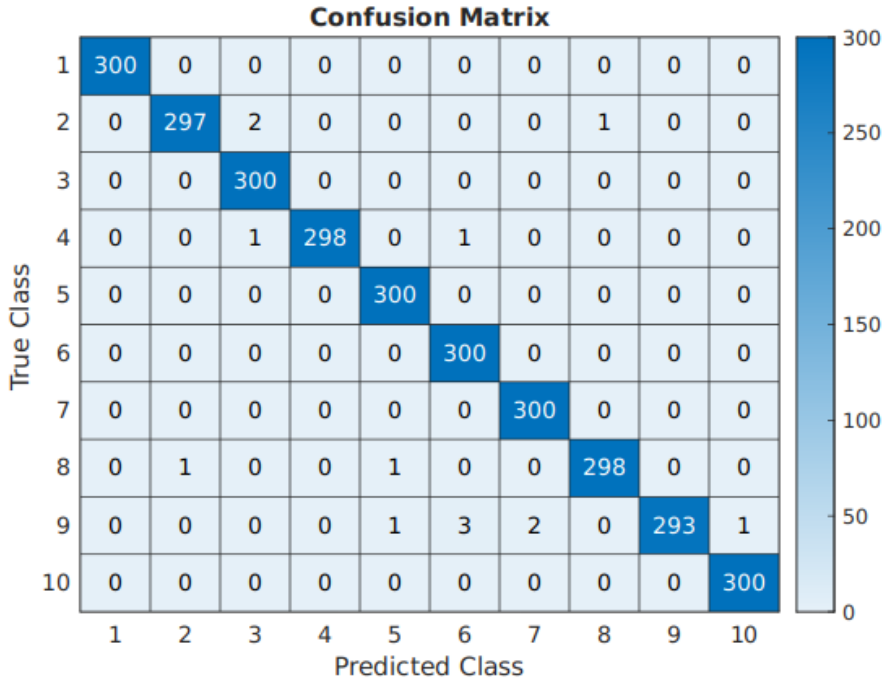

Figure 19: Confusion Matrix of the Custom model

The initial inference from both the training and loss and confusion plots is that the accuracy has bumped up to 99.5%, and all the diagonals of the confusion matrix have been populated.

$$precision = [1.00, 0.99, 1.00, 0.99, 1.00, 1.00, 1.00, 0.99, 0.97, 1.00]$$

$$recall = [1.0, 0.99, 0.99, 1.00, 0.99, 0.98, 0.99, 0.99, 1.00, 0.99]$$

$$f1\_score = [1.00, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.98, 0.99]$$

$$overall\_precision = 0.99$$

$$overall\_recall = 0.99$$

$$overall\_f1\_score = 0.99$$

5.4 Ethics in Computer Vision

5.4.1 Positives of using Computer Vision

The following are some of the benefits of using any computer vision library in our day-to-day lives:

1. **Healthcare**: Early detection of cancers and tumours, especially through MRI imaging, can be diagnosed early if pre-trained custom models are used. This can benefit the facility and patients by lowering costs.

2. **Security**: Popular doorbell security systems such as Ring has already been deployed in many households to reduce intruder entry. Simple motion models and HaarCascade face detectors can be employed in CCTV systems to prevent theft and robbery.

3. **Automating Tasks**: Industry lines can be employed with custom models for picking and placing operations, labelling, identifying defects in manufacturing and so on. This can reduce mundane labour and can improve quality of life.

5.4.2 Challenges and Mitigation of using Computer Vision

The following are some of the downsides of using machine vision techniques in our way of life [8]:

1. **Scarcity of Labelled Data**: Certain use cases require a large number of labelled datasets, which can prove to be computationally heavy and expensive to manage.
   Mitigation - Use of synthetic data generation methods such as changing the background, flipping and rotating the object of interest.

2. **Privacy**: Deepfakes have been recently introduced into social media platforms in great numbers due to AI image generation models such as Midjourney and DALL-E. This can harm one's reputation and can prove to be hazardous to one's mental health.
   Mitigation - The generation of these images must be limited to fictional people. Each image must be watermarked in advance, especially in the EXIF data, which is not apparently obvious.

3. **Greater number of dimensions in data**: It is hard to determine directly the leading behaviour in an outcome when presented with many features. This can lead to an unnecessary increase in training time and can prove to be cost-ineffective.
   Mitigation - Dimensional reduction techniques such as Principal Component Analysis can be applied to find the most significant components of the output vector.

5.4.3 Equality and Diversity in Computer Vision

1. **Equality**: A popular issue in image processing is the facial bias in detection techniques. This can usually happen when using a biased dataset to train the model. This can result in the model detecting only some of the faces well while neglecting the accuracy of detecting the people outside the particular region.

2. **Diversity**: This arises from the equality measure itself. Using data from diverse regions can lead to a greater number of generated data that is presented equally without any bias.

6 Appendix

- Task-1, Part-1: Introduction to Image Processing

```
% Copyright (c) 2024 Leander Stephen D'Souza
% Program to plot rgb histogram of an image

% Call the function with an example images
plot_rgb_histogram(imread('duckMallardDrake.jpg'));
plot_rgb_histogram(imread('Dog.jpg'));


% Function to plot rgb histogram of an image
function plot_rgb_histogram(img)
    % Get the size of the image
    [~, ~, channels] = size(img);

    % Check if the image is rgb
    if channels ~= 3
        disp('The image is not an rgb image');
        return;
    end

    % Initialize the histogram
    histogram = zeros(256, 3);

    % Calculate the histogram using imhist
    for k = 1:channels
        histogram(:, k) = imhist(img(:, :, k));
    end

    % Display the image and histogram in a single figure
    font_size = 12;
    figure;
    subplot(1, 2, 1);
    imshow(img);
    title('Original Image');

    subplot(1, 2, 2);
    bar(histogram(:, 1), 'r');
    hold on;
    bar(histogram(:, 2), 'g');
    bar(histogram(:, 3), 'b');
    title('RGB Histogram','FontSize',font_size);
    xlabel('Intensity','FontSize',font_size);
    ylabel('Frequency','FontSize',font_size);
    legend('Red', 'Green', 'Blue', 'Location', 'best');
end
```

- Task-1, Part-2: Edge Detection and Segmentation of Static Objects

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to test Edge Detection and Segmentation of Static Objects


og_image = imread('duckMallardDrake.jpg');

% Edge Detection
edge_detection_analysis(og_image);


function edge_detection_analysis(img)
    % 1. Apply Sobel Edge Detection
    sobel_edge_image = edge(rgb2gray(img), 'sobel', 0.1);

    % 2. Apply Canny Edge Detection
    canny_edge_image = edge(rgb2gray(img), 'canny', 0.15);

    % 3. Apply Prewitt Edge Detection
    prewitt_edge_image = edge(rgb2gray(img), 'prewitt', 0.05);

    % Plot all the images with appropriate titles
    figure;
    subplot(2, 2, 1);
    imshow(img);
    title('Original Image');

    subplot(2, 2, 2);
    imshow(sobel_edge_image);
    title('Sobel Edge Detection');

    subplot(2, 2, 3);
    imshow(canny_edge_image);
    title('Canny Edge Detection');

    subplot(2, 2, 4);
    imshow(prewitt_edge_image);
    title('Prewitt Edge Detection');
end
```

- Task-2: Object Motion Detection and Tracking

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to analyze Optical Flow in a Video Sequence
clc; clear; close all;

% Task 1: Find Corner Points
% find_corner_points(imread('red_square_static.jpg'), 'Red Square');
find_corner_points(imread('GingerBreadMan_first.jpg'), 'Ginger Bread Man');

% % Task 2: Optical Flow
first_image = imread('GingerBreadMan_first.jpg');
second_image = imread('GingerBreadMan_second.jpg');
estimate_optical_flow(first_image, second_image);

% % Task 3: Optical Flow in Video Sequence
video = VideoReader('red_square_video.mp4');
ground_truth = load('new_red_square_gt.mat');
% optical_flow_video(video, ground_truth);


% Function to analyze the optical flow in a video sequence
function optical_flow_video(video, ground_truth)
    % Read the video frames
    video_frames = read(video);
    num_frames = video.NumFrames;

    % Create an optical flow object
    opticalFlow = opticalFlowLK('NoiseThreshold', 0.009);

    % Load the actual corner trajectory
    actual_trajectory = cell2mat(struct2cell(ground_truth));

    % Save all the estimated trajectory of the corner points
    predicted_trajectory = zeros(num_frames - 1, 2);

    % Initialize the corner points
    first_frame = rgb2gray(video_frames(:,:,:,1));
    corners = corner(first_frame, 'Harris');
    corner_x = corners(1, 1);
    corner_y = corners(1, 2);

    % Initialize flow
    flow = estimateFlow(opticalFlow, first_frame);

    % Initialize corner trajectory
    vx = flow.Vx(round(corner_y), round(corner_x));
    vy = flow.Vy(round(corner_y), round(corner_x));

```

```
    % Compute the new position by adding the velocity vector to the current
    position
    x_new = corner_x + vx;
    y_new = corner_y + vy;
    predicted_trajectory(1, :) = [x_new, y_new];

    % Iterate through the video frames
    for i = 2:num_frames
        % Read the current frame
        current_frame = rgb2gray(video_frames(:,:,:,i));

        % Detect corners in the current frame
        corners = corner(current_frame, 'Harris');

        % Find the nearest corner to the previous corner
        min_dist = 1000000;
        for j = 1:size(corners, 1)
            dist = sqrt((corners(j, 1) - x_new)^2 + (corners(j, 2) - y_new)^2);
            if dist < min_dist
                min_dist = dist;
                nearest_corner = corners(j, :);
            end
        end

        % Update the corner position
        corner_x = nearest_corner(1);
        corner_y = nearest_corner(2);

        % Estimate the optical flow for the current frame
        flow = estimateFlow(opticalFlow, current_frame);

        % Compute the new position by adding the velocity vector to the current
    position
        vx = flow.Vx(round(corner_y), round(corner_x));
        vy = flow.Vy(round(corner_y), round(corner_x));

        % Update the track with the new position
        predicted_trajectory(i, :) = [corner_x + vx, corner_y + vy];
    end

    % Plot the estimated corner trajectory and the actual corner trajectory
    last_frame = video_frames(:,:,:,num_frames);

    imshow(last_frame);
    hold on
    plot(predicted_trajectory(:, 1), predicted_trajectory(:, 2), 'r');
    plot(actual_trajectory(:, 1), actual_trajectory(:, 2), 'b');
    title('Estimated vs Actual Corner Trajectory');
    xlabel('X');
```

```
    ylabel('Y');
    legend('Estimated Trajectory', 'Actual Trajectory');
    hold off

    % Get RMSEx, RMSEy, and RMSE
    for i = 1:num_frames-1
        RMSE_x(i) = sqrt((actual_trajectory(i+1, 1) - predicted_trajectory(i, 1)) ^ 2);
        RMSE_y(i) = sqrt((actual_trajectory(i+1, 2) - predicted_trajectory(i, 2)) ^ 2);
        RMSE(i) = sqrt((actual_trajectory(i+1, 1) - predicted_trajectory(i, 1)) ^ 2 + ...
                       (actual_trajectory(i+1, 2) - predicted_trajectory(i, 2)) ^ 2);
    end

    % Plot the RMSE values
    figure;
    plot(2:num_frames, RMSE_x, 'b');
    hold on;
    plot(2:num_frames, RMSE_y, 'r');
    plot(2:num_frames, RMSE, 'g');
    title('RMSE Values');
    xlabel('Frame Number');
    ylabel('RMSE');
    legend('RMSE_x', 'RMSE_y', 'RMSE');
    hold off;

    % Print the RMSE values
    fprintf('RMSE_x: %f\n', mean(RMSE_x));
    fprintf('RMSE_y: %f\n', mean(RMSE_y));
    fprintf('RMSE: %f\n', mean(RMSE));
end


% Function to estimate the optical flow between two images
function estimate_optical_flow(first_image, second_image)
    % Convert the images to grayscale
    first_image_gray = rgb2gray(first_image);
    second_image_gray = rgb2gray(second_image);

    % Find the optical flow of the pixels between the two images
    opticalFlow = opticalFlowLK('NoiseThreshold', 0.01);

    % The object stores the vectors from the first image to the second image
    estimateFlow(opticalFlow, first_image_gray);
    flow = estimateFlow(opticalFlow, second_image_gray);

    % Plot the optical flow on the second image
```

```
    figure;
    imshow(second_image);
    hold on;
    plot(flow, 'DecimationFactor', [5 5], 'ScaleFactor', 15);
    title('Optical Flow between the two images');
    hold off;

    % Plot the optical flow vectors
    figure;
    plot(flow);
    xlabel('X');
    ylabel('Y');
    legend('Optical Flow Vectors from the first image to the second image');
    title('Optical Flow Change');
end


% Function to find the corner points in an image
function find_corner_points(img, title_name)
    % Convert the image to grayscale
    img_gray = rgb2gray(img);

    % Find the corner points in the image
    corners = detectHarrisFeatures(img_gray);

    % Save the strongest corner points
    strongest_corners = corners.selectStrongest(50);

    % Plot the image with the corner points
    figure;
    imshow(img);
    hold on;
    plot(strongest_corners.Location(:, 1), strongest_corners.Location(:, 2), 'g+',
    'MarkerSize', 15, 'LineWidth', 3);
    title("Corner Points in " + title_name);
    hold off;
end
```

- Task-3, Part-1: Frame Differencing Approach

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to analyze background subtraction using frame difference

close all; clear; clc; % clear the workspace

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

% set the threshold
thresh_arr = [10, 25];
n_cols = 2;
n_rows = 1 + length(thresh_arr) / n_cols;

figure
set(gcf,'WindowState','maximized');

for i = 1:length(thresh_arr)
    % call the function to subtract the background
    source.CurrentTime = 0; % reset the video to the beginning
    frame_difference(source, thresh_arr(i), n_rows, n_cols, i); % call the function
     to subtract the background
end


% function to subtract the background
function frame_difference(source, thresh, n_rows, n_cols, i)
    % read the first frame of the video as a background model
    bg = readFrame(source);
    bg_bw = rgb2gray(bg);           % convert background to greyscale

    frame_counter = 0;              % initialize the frame counter
    frame_break = 140;

    % ---------------------- process frames ---------------------------------
    % loop all the frames
    while hasFrame(source)
        frame_counter = frame_counter + 1; % increment the frame counter

        if frame_counter > frame_break
            break;
        end

        fr = readFrame(source);     % read in frame
        fr_bw = rgb2gray(fr);       % convert frame to grayscale
        fr_diff = abs(double(fr_bw) - double(bg_bw));  % cast operands as double to
     avoid negative overflow

```

```
47          % if fr_diff > thresh pixel in foreground
48          fg = uint8(zeros(size(bg_bw)));
49          fg(fr_diff > thresh) = 255;
50
51          % update the background model
52          bg_bw = fr_bw;
53
54          % visualise the results and label the frame number with titles
55          subplot(n_rows, n_cols, 1), imshow(fr), title('Original Frame')
56          subplot(n_rows, n_cols, i + 2), imshow(fg), title('Foreground Pixels at
    Frame: ' + string(frame_counter) + ' with threshold: ' + string(thresh))
57          drawnow;
58      end
59  end
```

- Task-3, Part-2: Gaussian Mixture Models

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to analyze background subtraction using Gaussian mixture models

close all; clear; clc; % clear the workspace

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

% variable parameters
n_frames = [1, 10];
n_gaussians = [2, 20];

n_rows = length(n_frames);
n_cols = length(n_gaussians);

figure
set(gcf,'WindowState','maximized');

for i = 1:n_rows
    for j = 1:n_cols
        % call the function to subtract the background
        source.CurrentTime = 0; % reset the video to the beginning
        gaussian_mixture_models(source, n_frames(i), n_gaussians(j), n_rows + 1,
    n_cols, i, j); % call the function to subtract the background
    end
end


% Function to implement Gaussian Mixture Models
function gaussian_mixture_models(source, n_frames, n_gaussians, n_rows, n_cols, i,
    j)
    frame_counter = 0;              % initialize the frame counter
    frame_break = 140;

    detector = vision.ForegroundDetector('NumTrainingFrames', n_frames, '
    NumGaussians', n_gaussians);
    % make new figure window

    % --------------------- process frames -----------------------------------
    % loop all the frames
    while hasFrame(source)
        frame_counter = frame_counter + 1; % increment the frame counter

        if frame_counter > frame_break
            break;
        end

```

```
46          fr = readFrame(source);      % read in frame
47
48          fgMask = step(detector, fr);    % compute the foreground mask by Gaussian
    mixture models
49
50          % create frame with foreground detection
51          fg = uint8(zeros(size(fr, 1), size(fr, 2)));
52          fg(fgMask) = 255;
53
54          % visualise the results after labelling the frame number with titles
55          subplot(n_rows, n_cols, 1), imshow(fr), title('Original Frame')
56          subplot(n_rows, n_cols, i * n_cols + j), imshow(fg), title('Foreground at
    Frame: ' + string(frame_counter) + ' with ' + string(n_frames) + ' training
    frames & ' + string(n_gaussians) + ' Gaussians')
57
58          drawnow
59      end
60  end
```

- Task-4: Robot Treasure Hunting

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to implement Treasure Hunting using Arrow Detection
clear; clc; close all;

%% Treasure Hunting
treasure_hunter(imread('Treasure_easy.jpg'));
treasure_hunter(imread('Treasure_medium.jpg'));
treasure_hunter(imread('Treasure_hard.jpg'));


% Function to implement Treasure Hunting
function treasure_hunter(img)
    figure;

    % Show image
    subplot(2, 3, 1);
    imshow(img);
    title('Original Image');

    % Binarisation
    bin_img = imbinarize(rgb2gray(img), 0.1);
    subplot(2, 3, 2);
    imshow(bin_img);
    title('Binary Image');


    % Extracting connected components
    con_com = logical(bwlabel(bin_img));
    subplot(2, 3, 3);
    imshow(label2rgb(con_com));
    title('Connected Components');

    % Computing objects properties
    props = regionprops(con_com);

    % Drawing bounding boxes
    n_objects = numel(props);
    subplot(2, 3, 4);
    imshow(img);
    hold on;
    for object_id = 1 : n_objects
        rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'w');
    end
    hold off;
    title('Bounding Boxes');


```

```
49      % Finding arrows
50      [arrow_list, treasure_list] = arrow_finder(props, img);
51      all_boxes_list = [arrow_list, treasure_list];
52
53      % Finding red arrow (start point)
54      n_arrows = numel(arrow_list);
55      start_arrow_id = 0;
56      % check each arrow until find the red one
57      for arrow_num = 1 : n_arrows
58          object_id = arrow_list(arrow_num);    % determine the arrow id
59
60          % extract colour of the centroid point of the current arrow
61          centroid_colour = img(round(props(object_id).Centroid(2)), round(props(
    object_id).Centroid(1)), :);
62
63          red = centroid_colour(:, :, 1);
64          green = centroid_colour(:, :, 2);
65          blue = centroid_colour(:, :, 3);
66
67          if red > 240 && green < 10 && blue < 10
68          % the centroid point is red, memorise its id and break the loop
69              start_arrow_id = object_id;
70              break;
71          end
72      end
73
74      % Hunting
75      cur_object = start_arrow_id; % start from the red arrow
76      path = cur_object; % memorise the path
77
78      for treasure_idx = 1 : numel(treasure_list)
79          % while the current object is an arrow, continue to search
80          while ismember(cur_object, arrow_list)
81              % find the next closest arrow
82              prev_object = cur_object;
83              cur_object = next_object_finder(cur_object, props, all_boxes_list, img,
     path);
84              path(end + 1) = cur_object;
85          end
86          % Found treasure
87          cur_object = prev_object;
88          break;
89      end
90
91      % helper variable for drawing the treasure
92      treasure_counter = 0;
93      treasure_found = false;
94      treasure_offset = 0;
95
```

```
    % visualisation of the path
    subplot(2, 3, 5);
    imshow(img);
    hold on;

    for path_element = 1 : numel(path)
        object_id = path(path_element); % determine the object id

        % reset for treasure
        if ismember(object_id, treasure_list)
            treasure_counter = treasure_counter + 1;
            treasure_found = true;
            rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'g');
            str = num2str(treasure_counter);
            text(props(object_id).BoundingBox(1), props(object_id).BoundingBox(2),
    str, 'Color', 'g', 'FontWeight', 'bold', 'FontSize', 14);
            continue;
        end

        if treasure_found
            treasure_found = false;
            treasure_offset = 1;
            continue;
        end

        rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'y');
        str = num2str(path_element - treasure_counter - treasure_offset);
        text(props(object_id).BoundingBox(1), props(object_id).BoundingBox(2), str,
     'Color', 'r', 'FontWeight', 'bold', 'FontSize', 14);
    end

    hold off;
    title('Treasure Path - Intermediate');
end

% Function to find the arrow objects
function [arrow_list, treasure_list] = arrow_finder(props, img)
    n_objects = numel(props);
    arrow_list = zeros(1, n_objects);
    treasure_list = zeros(1, n_objects);
    min_length = 5;
    treasure_min_area = 3014;
    arrow_count = 0;
    treasure_count = 0;

    % iterate over all objects
    for object_id = 1 : n_objects
        % extract the bounding box of the current object
        bbox = props(object_id).BoundingBox;
```

```
143
144         % extract the length and breadth of the bounding box
145         length = bbox(3);
146         breadth = bbox(4);
147
148         % check if the object is of sufficient size
149         if breadth > min_length && length > min_length
150             % extract the centroid of the current object, if the colour of the
    centroid is white, then it is an arrow
151             centroid_colour = img(round(props(object_id).Centroid(2)), round(props(
    object_id).Centroid(1)), :);
152
153             red = centroid_colour(:, :, 1);
154             green = centroid_colour(:, :, 2);
155             blue = centroid_colour(:, :, 3);
156
157             % Use the fact that the arrow is red or white
158             if (red > 240 && green > 240 && blue > 240) || (red > 240 && green < 10
     && blue < 10)
159                 arrow_count = arrow_count + 1;
160                 arrow_list(arrow_count) = object_id;
161             elseif props(object_id).Area > treasure_min_area
162                 treasure_count = treasure_count + 1;
163                 treasure_list(treasure_count) = object_id;
164             end
165         end
166     end
167     % remove unnecessary zeros
168     arrow_list = arrow_list(1 : arrow_count);
169     treasure_list = treasure_list(1 : treasure_count);
170 end
171
172 % Function to find the next object in the path
173 function [cur_object] = next_object_finder(cur_object, props, all_boxes_list, img,
    path)
174     % get the bounding box of the current object
175     bbox_cur_object = props(cur_object).BoundingBox;
176
177     % calculate the current yellow centroid
178     cur_yellow_centroid = get_yellow_centroid(img, bbox_cur_object);
179
180     % find the closest object to the current object
181     min_dist = inf;
182     critical_dist = 100;
183     treasure_area = 3014;
184     cur_object = 0;
185
186     for object_idx = 1 : numel(all_boxes_list)
187         object = all_boxes_list(object_idx);
```

```
        % check if the object is not in the path
        if ~ismember(object, path)
            % get the centroid of the object
            object_centroid = props(object).Centroid;

            % get the distance between the current object and the object
            dist = norm(cur_yellow_centroid - object_centroid);

            % get area
            if props(object).Area > treasure_area
                % monitor intermediate treasure distance
                if dist < critical_dist
                    critical_dist = dist;
                    cur_object = object;
                end
            end

            % check if the object is the closest object
            if dist < min_dist
                min_dist = dist;
                cur_object = object;
            end
        end
    end
end

% Function to extract yellow centroid from a bounding box
function yellow_centroid = get_yellow_centroid(img, bbox)
    % extract the bounding box of the current object
    bbox = round(bbox);
    x = bbox(1);
    y = bbox(2);
    width = bbox(3);
    height = bbox(4);
    visualize = false;

    % extract the region of interest
    roi = img(y : y + height, x : x + width, :);

    % convert the region of interest to the HSV colour space
    hsv_roi = rgb2hsv(roi);

    % extract the hue, saturation and value channels
    hue = hsv_roi(:, :, 1);
    saturation = hsv_roi(:, :, 2);
    value = hsv_roi(:, :, 3);

    % threshold the hue channel to extract the yellow pixels
    yellow_mask = hue > 0.1 & hue < 0.2;
```

```
237
238     % threshold the saturation channel to extract the yellow pixels
239     yellow_mask = yellow_mask & saturation > 0.5;
240
241     % threshold the value channel to extract the yellow pixels
242     yellow_mask = yellow_mask & value > 0.5;
243
244     % check if the yellow mask is empty
245     if ~any(yellow_mask(:))
246         yellow_centroid = [0, 0];
247     else
248         % find the centroid of the yellow pixels
249         yellow_centroid = regionprops(yellow_mask, 'Centroid');
250         yellow_centroid = yellow_centroid.Centroid;
251         yellow_centroid = [yellow_centroid(1) + x, yellow_centroid(2) + y];
252
253         % visualisation
254         if visualize
255             imshow(img);
256             hold on;
257             rectangle('Position', bbox, 'EdgeColor', 'r');
258             plot(yellow_centroid(1), yellow_centroid(2), 'ro');
259             hold off;
260             pause(0.5);
261         end
262     end
263 end
```

- Task-5: Convolutional Neural Networks for Image Classification

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to implement Google LeNet CNN for Image Classification for the Digit
    Dataset
close all; clear; clc;

% define constants
image_size = [32, 32];
num_classes = 10;

% Load the digit dataset
digitDatasetPath = fullfile(toolboxdir('nnet'), 'nndemos', ...
    'nndatasets', 'DigitDataset');
%
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders', true, 'LabelSource', 'foldernames');

imds.ReadFcn = @(loc)imresize(imread(loc), image_size);

% Split the data into training and validation datasets
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.7, 'randomized');

% Define the LeNet-5 architecture
[lenet_layers, lenet_options] = lenet_architecture(image_size, num_classes,
    imdsValidation);

% Define custom architecture
[custom_layers, custom_options] = custom_architecture(image_size, num_classes,
    imdsValidation);


for i = 1:2
    if i == 1
        layers = lenet_layers;
        options = lenet_options;
    else
        layers = custom_layers;
        options = custom_options;
    end

    % Train the network
    net = trainNetwork(imdsTrain, layers, options);

    % Classify validation images and compute accuracy
    YPred = classify(net, imdsValidation);
    YValidation = imdsValidation.Labels;
    accuracy = sum(YPred == YValidation) / numel(YValidation);
    fprintf('Accuracy of the network on the validation images: %f\n', accuracy);
```

```
46
47      % Calculate the confusion matrix
48      confusionMatrix = confusionmat(YValidation, YPred);
49
50      % Display the confusion matrix as a heatmap
51      figure;
52      heatmap(confusionMatrix);
53      title('Confusion Matrix');
54      xlabel('Predicted Class');
55      ylabel('True Class');
56
57      % Calculate precision, recall, and F1 score for each class
58      precision = diag(confusionMatrix) ./ sum(confusionMatrix, 2);
59      recall = diag(confusionMatrix) ./ sum(confusionMatrix, 1)';
60      f1Score = 2 * (precision .* recall) ./ (precision + recall);
61
62      % Display the precision, recall, and F1 score
63      fprintf('Precision: %f\n', mean(precision));
64      fprintf('Recall: %f\n', mean(recall));
65      fprintf('F1 Score: %f\n', mean(f1Score));
66
67      % Display the precision, recall, and F1 score for each class
68      disp("precision" + precision);
69      disp("recall" + recall);
70      disp("f1Score" + f1Score);
71
72  end
73
74  % Function to define the LeNet-5 architecture
75  function [layers, options] = lenet_architecture(image_size, num_classes,
        validation_data)
76      % Define the LeNet-5 architecture
77      layers = [
78          imageInputLayer([image_size 1],'Name','input')
79
80          convolution2dLayer(5,6,'Padding','same','Name','conv_1')
81          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_1')
82
83          convolution2dLayer(5,16,'Padding','same','Name','conv_2')
84          averagePooling2dLayer(2,'Stride',2,'Name','avgpool_2')
85
86          fullyConnectedLayer(120,'Name','fc_1')
87          fullyConnectedLayer(84,'Name','fc_2')
88
89          fullyConnectedLayer(num_classes, 'Name', 'fc_3')
90          softmaxLayer('Name','softmax')
91          classificationLayer('Name','output')
92      ];
93
```

```
    % Specify the training options
    options = trainingOptions('sgdm', ...
        'InitialLearnRate',0.0001, ...
        'MaxEpochs',10, ...
        'Shuffle','every-epoch', ...
        'ValidationData', validation_data, ...
        'ValidationFrequency',30, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'ExecutionEnvironment', 'gpu');
end

% Function to define a custom architecture
function [layers, options] = custom_architecture(image_size, num_classes,
    validation_data)
    % Define the custom architecture
    filterSize = 5;
    numFilters = 16;
    fullyConnectedLayerSize = 64;
    l2_reg = 0.0001;

    layers = [
        imageInputLayer([image_size 1])

        % Hidden Layer 1
        convolution2dLayer(filterSize, numFilters,'Padding','same')
        batchNormalizationLayer
        reluLayer
        maxPooling2dLayer(2,'Stride',2)

        % Hidden Layer 2
        convolution2dLayer(filterSize, numFilters,'Padding','same')
        batchNormalizationLayer
        reluLayer
        maxPooling2dLayer(2,'Stride',2)

        fullyConnectedLayer(fullyConnectedLayerSize, ...
            'WeightL2Factor', l2_reg, 'BiasL2Factor', l2_reg)
        dropoutLayer(0.2)

        fullyConnectedLayer(num_classes)
        softmaxLayer

        classificationLayer
    ];

    % Specify the training options
    options = trainingOptions('adam', ...
        'InitialLearnRate',0.001, ...
```

```
        'MaxEpochs',10, ...
        'Shuffle','every-epoch', ...
        'ValidationData', validation_data, ...
        'ValidationFrequency',30, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'ExecutionEnvironment', 'gpu');

end
```

References

[1] S. Basar, M. Ali, G. Ochoa-Ruiz, M. Zareei, A. Waheed, and A. Adnan, "Unsupervised color image segmentation: A case of rgb histogram based k-means clustering initialization," *PLoS ONE*, vol. 15, no. 10, p. e0240015, 2020.

[2] A. Kardoost, "Optimizing edge detection for image segmentation with multicut penalties," in *Pattern Recognition(DAGM GCPR 2022)*, ser. Lecture Notes in Computer Science, vol. 13485. Springer, 2022.

[3] C. Xiao, Q. Cao, Y. Zhong, L. Lan, X. Zhang, Z. Luo, and D. Tao, "Motiontrack: Learning motion predictor for multiple object tracking," *arXiv preprint arXiv:2306.02585*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.02585

[4] T. Biswas, D. Bhattacharya, G. Mandal, and T. Das, "Motion detection using three frame differencing and cnn," in *Computational Intelligence in Communications and Business Analytics*, ser. Communications in Computer and Information Science, K. Dasgupta, S. Mukhopadhyay, J. Mandal, and P. Dutta, Eds., vol. 1955, 2024. [Online]. Available: https://doi.org/10.1007/978-3-031-48876-4_5

[5] D. Reynolds, *Gaussian Mixture Models*. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-1-4899-7488-$4_1$96

[6] M. Gröger, V. Borisov, and G. Kasneci, "Boxshrink: From bounding boxes to segmentation masks," *arXiv preprint arXiv:2208.03142*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2208.03142

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[8] R. A. Waelen, "The ethics of computer vision: an overview in terms of power," *AI and Ethics*, 2023. [Online]. Available: https://doi.org/10.1007/s43681-023-00272-x