The University Of Sheffield.

Machine Vision ACS61012

Course Work Submission: Individual Project (ACS61012)

By

Ankur Tiwari

Registration Number 210158775

Table of Contents

List of Figures

Task 1: Introduction to Machine Vision

Part 1: Understanding different Image formats, analysis of image histogram.

Image Histograms for Grayscale Images:

- Histograms are used to plot the frequency and cumulative frequency of the intensity values of an image and are useful in enhancing the contrast and exposure of an image.
- Vertical bars in histogram represents number of pixels for each intensity values and helps in decisions regarding adjustments of image contrast as shown in Figure 1. This can be achieved by trimming the intensities at the extreme end of intensity distribution and redistributing them also known as Gamma Correction.

Figure 1 Representation of histogram of a grayscale image

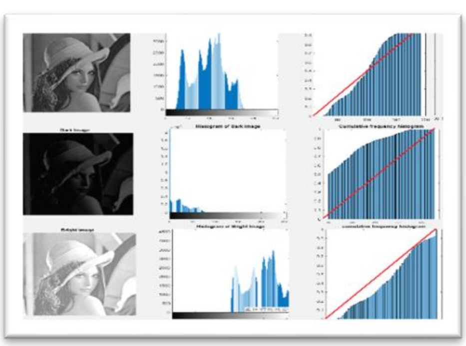

Figure 2 Comparison between cumulative frequency curve of properly exposed, over exposed, and underexposed image

- Images with cumulative frequency curve aligning with the red line in Figure 3 depicts good contrast or properly exposed image.

Image histogram for Colored Images:

The image used for studying the histograms of RGB image is shwon below.

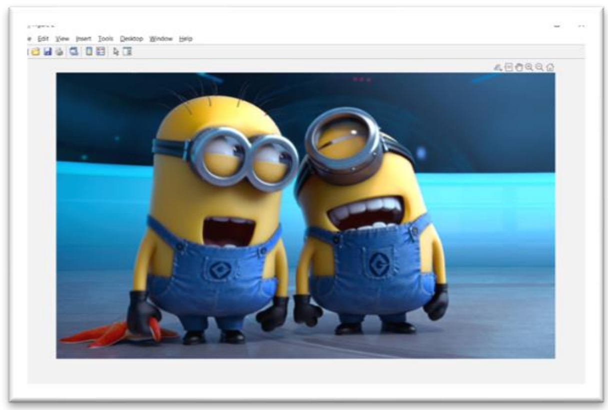

Figure 3 Original RGB image used

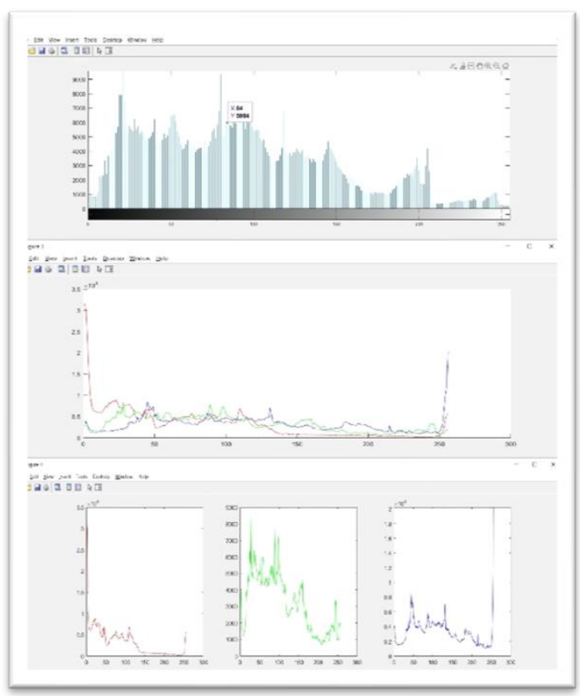

Figure 4 Histogram for original RGB image (a) Luminance histogram (b) All channels merged (c) All channels separate.

Analysis of histogram of RGB image:

- The luminance histogram (a) in Figure 4 is skewed towards slight left which indicates the presence of darker shades of colors.
- Analysis of red color channel indicates presence of fewer pixels of red color which is indicated by the lesser area covered by the red channel histogram. Moreover, it is tilted towards left which indicates its in darker tone and can be verified by seeing the color of red object held by one of the minions.
- A spike in blue channel towards right indicates blue pixels with bright color in large number and this can be verified by the background of the minions which is in a lighter shade of blue color.
- Although green color's presence is not visible in image, but its histogram is mostly shifted towards left indicative of darker tone, here green color contributes towards shadows in the image.
- The distribution of colors is uniform and not segregated at extreme ends indicating good contrast level.

Figure 5 (a) Red channel normalized (b) Green channel normalized (c) blue channel normalized

Identifying 2 colored image based on histogram

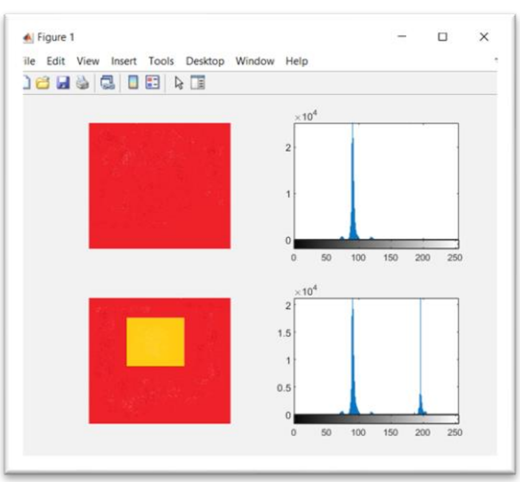

Figure 6 Comparison between histograms of One-color Image and two-colored Image

An interesting observation from **Figure 6** is that first figure with just red color shows one spike whereas the second figure with two colors has two spikes which is indicative of two different intensities. In addition to this the spike shown by the red color is thicker than the spike shown by the yellow color, which is indicative of the fact that red color is distributed over wider intensity range.

Part 2 Different types of image noise / image denoising, static object segmentation based on edge detection

In this section image is induced with two gaussian as well as salt and pepper noise and are denoised using various filters and in second part a intensive study is done on the same image first without any noise to find the optimal value of various filters and then the image is tempered with different types of noises and then filters are applied with different values of threshold and the best result are shown .

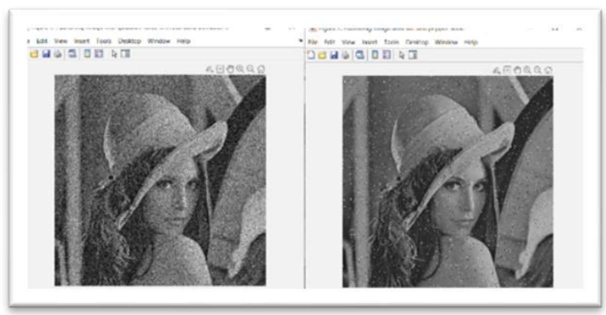

Figure 7 (a) Image with gaussian noise (b) Image with salt and pepper noise.

Noise Removal:

Results showing the use of above averaging filter and median filters for salt and pepper noise.

Figure 8 (a) Image with noise (b) averaging filter (c) median filter

Analysis:

Median filter gives better results and is more robust as a single and unrepresentative neighboring pixel does not affects the result in case of median filter.

Gaussian Filters:

These are low pass filters useful in reducing noise and blurring regions of an image.

Figure 9 Applying Gaussian filters with different values of standard deviation on image with gaussian noise

Figure 10 Applying Gaussian filters with different values of standard deviation on image with salt and pepper noise.

Part 2 Different types of Edge detection algorithms.

Edge detection with Sobel filters:

Figure 11 Output for Sobel filter for different values of threshold

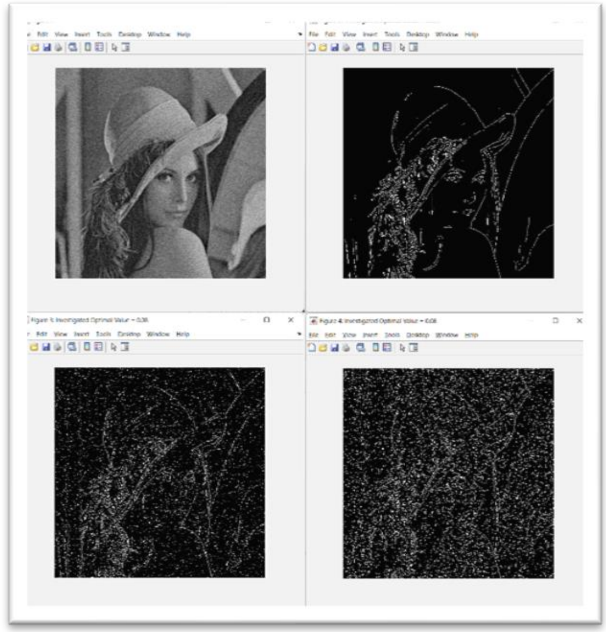

Figure 12 Edge detection using Sobel filter (a) Image with gaussian noise (b) On normal image (threshold = 0.055) (c) On image with gaussian noise (threshold=0.08) (d) image with salt and pepper noise (threshold=0.08)

Edge detection using Canny Filter

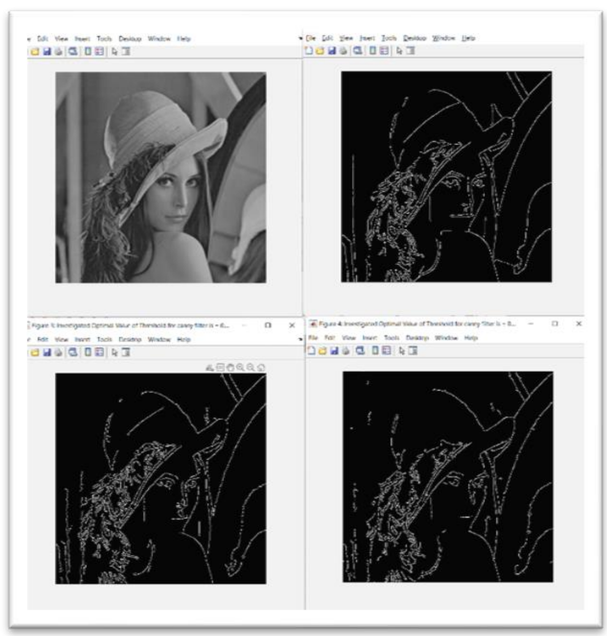

Figure 13 (a) Original image (b) Canny filter on normal image (threshold = 0.28) (c) Canny filter on image with gaussian noise (threshold = 0.32) (d) Canny filter on image with salt and pepper noise (threshold = 0.38)

Edge detection with Prewitt filter

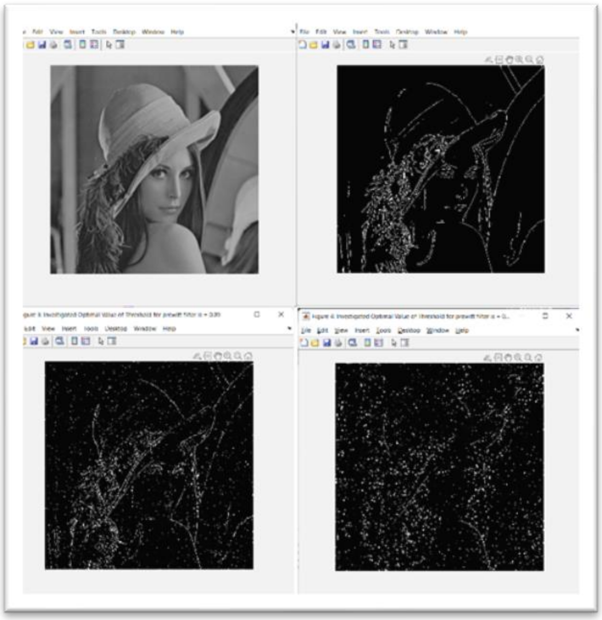

Figure 14 (a) Normal Image (b) Prewitt filter on normal Image (threshold =0.05) (c) Prewitt filter on image with gaussian noise (threshold = 0.09) (d) Prewitt filter on image with salt and pepper noise (threshold = 0.15)

Conclusion:

Filter	Normal Image	Image (Gaussian noise)	Image (Salt & pepper noise)
Sobel	0.055	0.08	0.08
Canny	0.28	0.32	0.38
prewitt	0.05	0.09	0.15

Figure 15 Table showing the optimal Values of different filters for Normal Image, Image with Gaussian noise and Image with salt and pepper noise.

- For image without any noise prewitt filter gives the best result.
- For image with Gaussian & Salt and pepper noise canny filter gives best result.
- Canny detector appears to be good at detecting even weaker edges.
- Out of the three sobel operator is most affected by the noise and gives least convincing results.
- Canny operator detects edges by detaching noise from the image prior to finding an edge.

Task 2: Optical flow estimation algorithm:

Optical Flow:

Optical flow algorithm is used to estimate where a point would move in the next image sequence by calculating velocity of points in the image.

Part 1: Find corner points and apply the optical flow estimation algorithm.

Corner points in Gingerbread Man.

Corner detection is an important aspect of computer vision and is used for feature extraction. It establishes a base for techniques such as object recognition, motion detection, image mosaicing, 3D reconstruction, panorama stitching, video tracking etc.

The corner function used performs nonmaximal suppression on candidate's corner and the corners are at least 2 pixels apart.

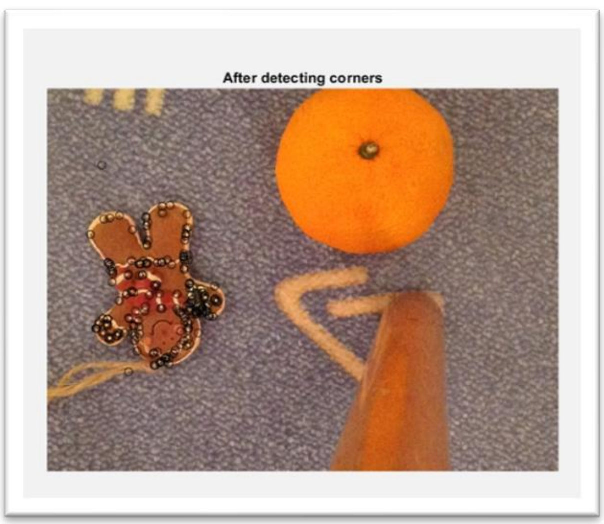

Figure 16 Corner detected in Gingerbread Man.

Optical Flow estimation

The optical flow is estimated as the motion between two consecutive frames of a video. Here two images are being used instead of a video therefore these are treated as two consecutive frames.

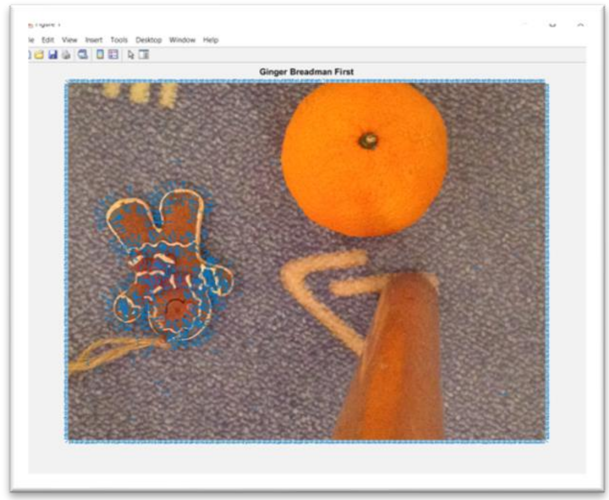

Figure 17 Optical flow object created for the first frame

On applying optical flow on image 1 gives velocity vector although there was no previous frame, so these vectors are derived randomly and as we pass the next frame the vectors are computed based on the difference between the points of two frames.

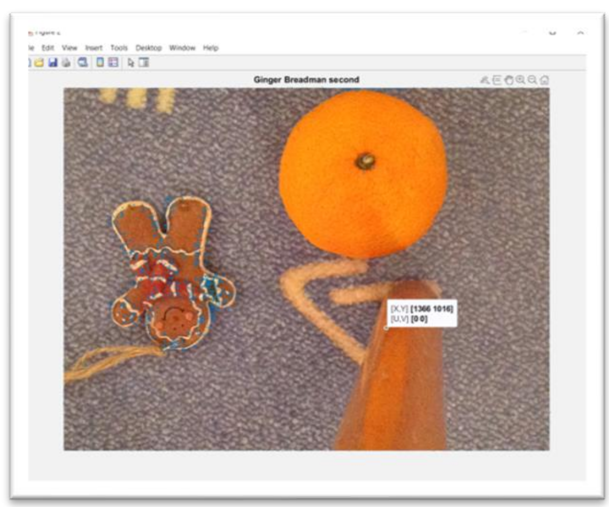

Figure 18 optical flow estimated for Gingerbread Man.

The velocity vectors around Gingerbread Man in second image shows its movement, also we can see velocity vector around the orange showing its movement in downward direction.

Part 2: Optimal flow estimation on red square video

The path of red colored box is estimated across different frames using Lucas-Kanade optical flow method.

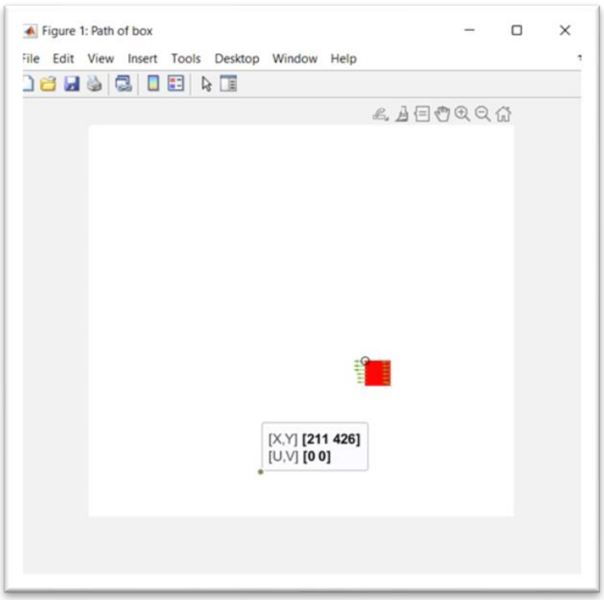

Figure 19 Screenshot of the final position of the box.

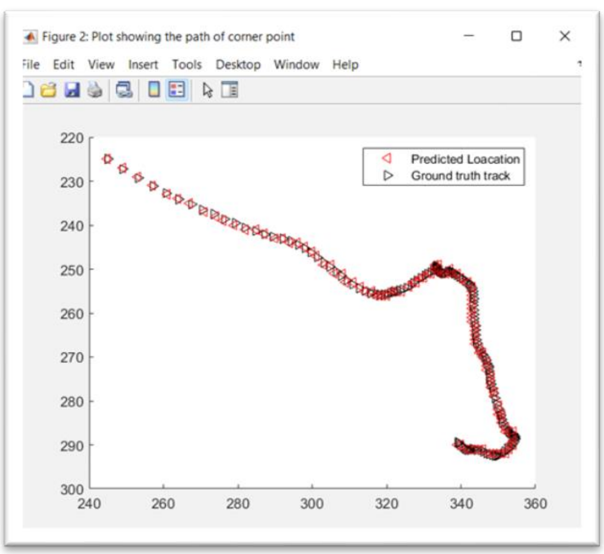

Figure 20 Screenshot showing the coordinates of the path of the box (Red) triangle represents the predicted location whereas (black) triangle represents the Ground truth track.

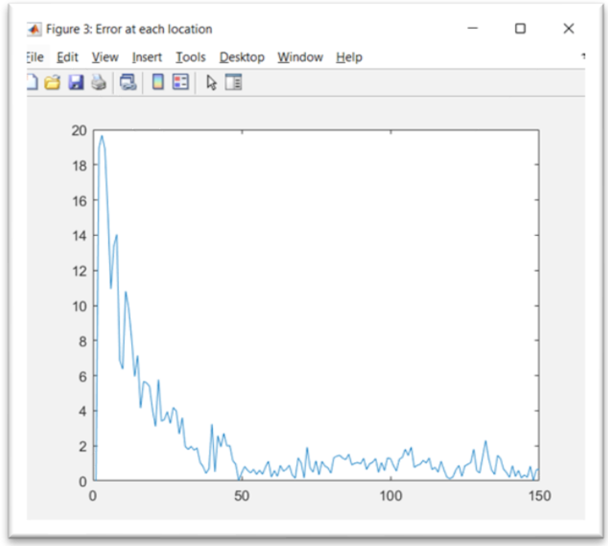

Figure 21 Error at each Frame.

Figure 21 shows the error value at each frame. It is evident that the value of error is high for the first few frames but then the error is reduced this is also evident from figure 18. The RSME value was found to be 1.5113 which indicates that the predictions obtained by Lucas-Kanade optical flow estimations are pretty accurate.

Task 3: Automatic detection of moving objects in a sequence of Video frames

Part 1: With the frame differencing approach:

Frame differencing:

It is a technique deployed in computer vision to detect the difference between two frames in a video. The change in pixels between two consecutive frames indicates motion. Moving object detection has a wide range of applications in real world situations ranging from surveillance to security. The complete procedure of frame differencing is shown in Figure 22.

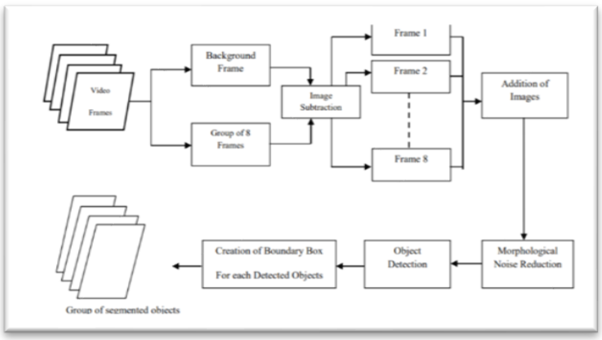

Figure 22 Steps involved in Moving object detection using Frame differencing technique.

Figure 23 Image difference technique with different values of Threshold (A) threshold = 15 (B) threshold = 30 (C) threshold = 25

Figure 24 Optimum results at threshold value =20

Analysis:

- When the value of threshold is kept at 15 clear demarcation of object is seen, however inside the boundaries pixelation can be seen. Moreover, we can see the noise because of surroundings this is indicative of the fact that the results are not as good as expected.
- On empirically examining different values of threshold for frame differencing best results were observed at a threshold value of 20. In this case the background was properly removed whereas the moving cars were also demarcated clearly.
- It is also evident that the results produced for fast moving objects were quite better than those for slow moving object hence it can be concluded that the technique should be applied for fast moving objects for better results.
- From the above results it can be concluded that frame differencing is a very simple algorithm, easy to implement and doesn't even require high computational resources.
- Moreover, there are few other factors that can produce an impact such as lighting conditions and types of background for example bimodal background will not produce good results.

Part 2: Gaussian mixture approach for moving object detection.

Classical frame differencing model is not so good in detecting the moving object especially at low speeds and with light mutations. An improvement over this is to use gaussian mixture model. The working of a gaussian mixture model is shown by figure 25.

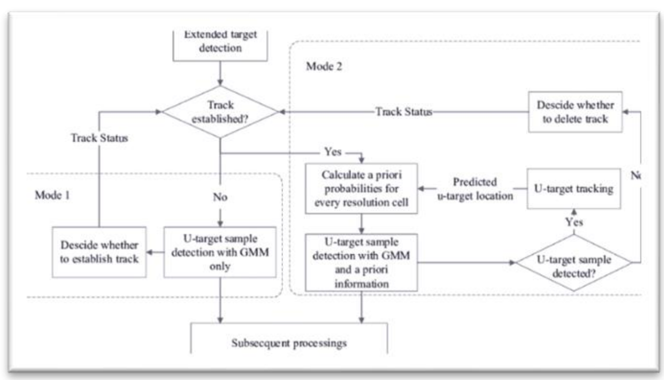

Figure 25 Flowchart of Gaussian Mixture model

Figure 26 Gaussian model with n-frames=50 , n-gaussian = 6 , Initial Variance =20

Figure 27 Gaussian model with n-frames =60 and n-gaussian=120 , initial Variance=20

Figure 28 Gaussian model with n-frames =60 , n-gaussian =200 , initial Variance = 5

Analysis

Usually for Gaussian filter the chosen filter size is 3 times the value of standard deviation and total filter size is taken to be 6 times the standard deviation value.

As evident in results Gaussian filter produces better results as compared to Frame difference method for moving object detection.

Task 4: Treasure Hunting

Part 1: Results Obtained

The results obtained for 3 different tasks with varying difficulty level are shown below. All the tasks were successfully completed.

Easy:

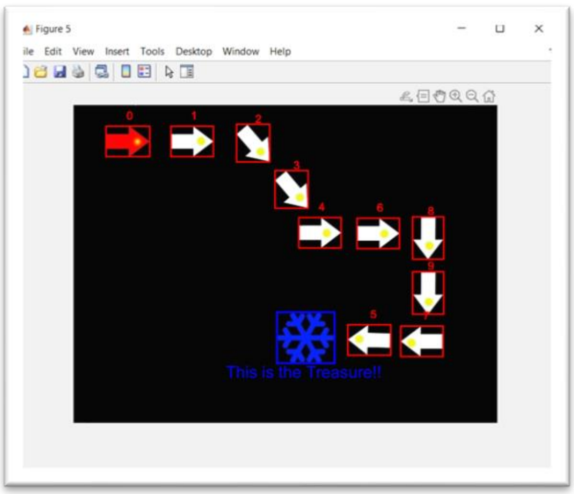

Figure 29 Result obtained for Treasure hunt with difficulty level Easy

Medium:

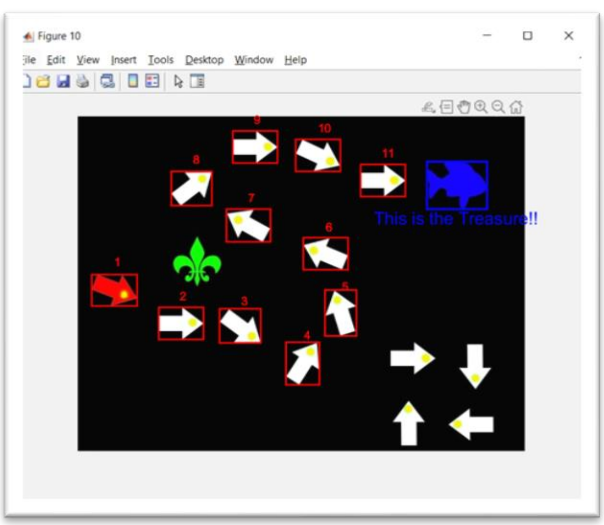

Figure 30 Result obtained for Treasure Hunt with difficulty level Medium

Difficult:

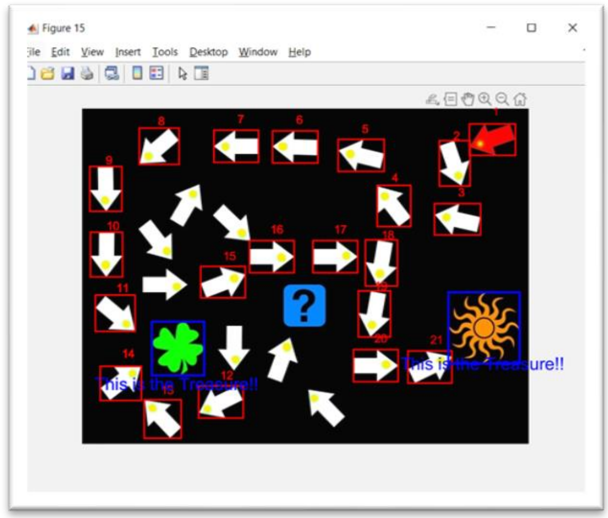

Figure 31 Result obtained for Treasure hunt with difficulty level Hard

Part 2: Results of binarization of the images and the value of threshold found.

To make the image analysis easier Binarization is carried out on RGB images to obtain black and white image for further processing. The **threshold** value used for binarization of all the three images is **0.05**

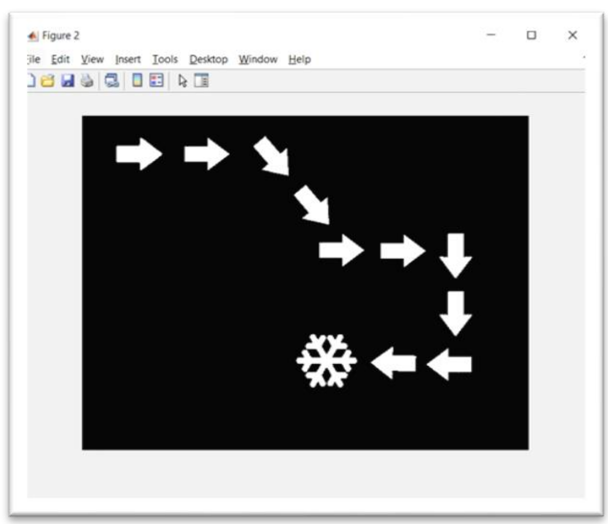

Figure 32 Binary Image obtained for Easy Task

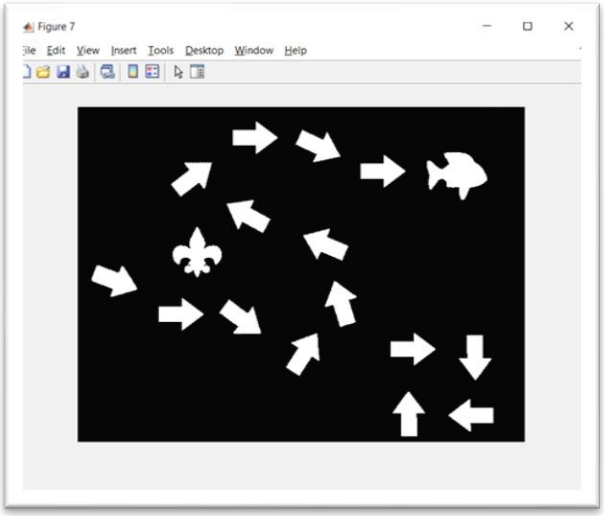

Figure 33 Binary Image obtained for Medium Task

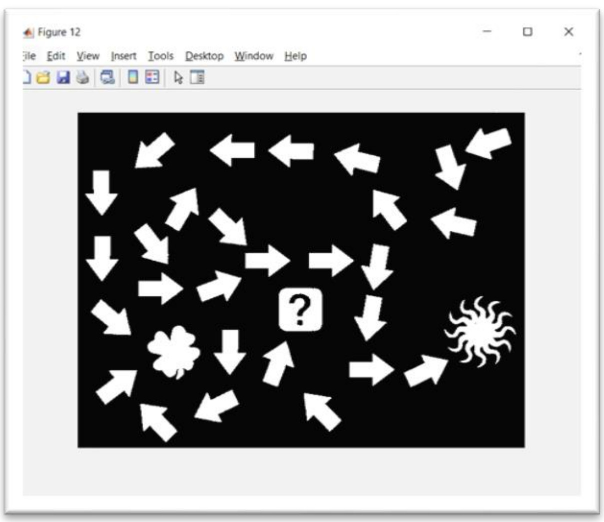

Figure 34 Binary Image obtained for Difficult Task

Part 3: Explanation of Solution

Note: Code has been self-Written by taking reference from the provided code

The steps involved in solution are discussed below. The code is similar for all the three tasks with minute changes made as per the problem statement.

1.Loading Image

```
%% Reading image
im = imread('Treasure_hard.jpg'); % change name to process other images
figure.
imshow(im);

```

2. Binarization of Image

```
%% Binarisation
bin_threshold = 0.05; % parameter to vary
bin_im = im2bw(im, bin_threshold);
figure;
imshow(bin_im);

```

3. Extracting the connected Components

```
%% Extracting connected components
con_com = bwlabel(bin_im);
figure;
imshow(label2rgb(con_com));

```

In this block a function **bwlabel** is used this function labels the connected objects in the image. These components are then connected to RGB image using **label2rgb** function.

4. Extracting properties of labelled components.

```
%% Computing objects properties
props = regionprops(con_com);

props=arrow_finder(im,props);

```

The properties of labelled components are obtained in an object called props.

Fields	Area	Centroid	BoundingBox
1	1530	[34.4732,11...	[11.5000,83.500...
2	1528	[35.5903,20...	[12.5000,177.50...
3	1582	[49.9532,29...	[19.5000,267.50...
4	1582	[57.5550,39...	[26.5000,369.50...
5	1567	[108.4167,1...	[81.5000,158.50...
6	1558	[109.1438,5...	[82.5000,28.500...
7	1494	[117.9813,2...	[87.5000,229.50...
8	1574	[113.9682,4...	[89.5000,417.50...

Figure 35 Properties of identified objects obtained in object props

These properties are then passed to a function called arrow finder which finds arrows and adds their properties to props object.

Code for arrow_finder function:

```
function [modified_props] = arrow_finder(im,props)
len=length(props);
arrow_number=0;
for i=1:len
    sect=imcrop(im,props(i).BoundingBox);
    % extract RGB channels separately
    red_channel = sect(:, :, 1);
    green_channel = sect(:, :, 2);
    blue_channel = sect(:, :, 3);
    % label pixels of yellow colour
    yellow_map = green_channel > 220 & red_channel > 220 & blue_channel < 30;
    parameter=regionprops(yellow_map);
    if isempty(parameter)== 0
        props(i).arrow_id=arrow_number;
        arrow_number=arrow_number+1;
    else
        props(i).arrow_id='Not_arrow'
    end
end
modified_props=props;
end
```

This function takes properties of connected objects along with original image as input parameters and classify the arrows as separate object from treasure by identifying presence of yellow dots between arrows. And provides Id to arrow on basis of presence of yellow dots and labels the objects without yellow dots as "Not Arrow".

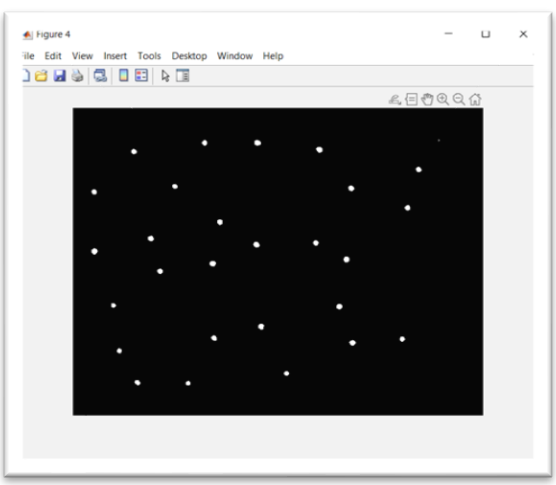

Figure 35 Identified Yellow dots with the help of arrow_finder function

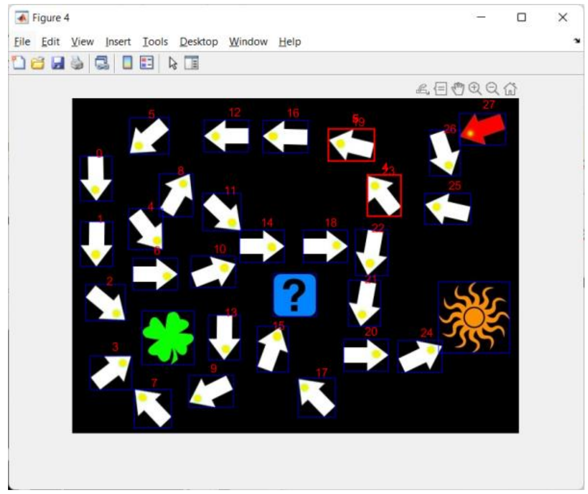

Figure 36 Arrows provided with Id Based on presence of yellow dots

Fields	Area	Centroid	BoundingBox	arrow_id
1	1530	[34.4732,11...	[11.5000,83.500...	0
2	1528	[35.5903,20...	[12.5000,177.50...	1
3	1582	[49.9532,29...	[19.5000,267.50...	2
4	1582	[57.5550,39...	[26.5000,369.50...	3
5	1567	[108.4167,1...	[81.5000,158.50...	4
6	1558	[109.1438,5...	[82.5000,28.500...	5
7	1494	[117.9813,2...	[87.5000,229.50...	6
8	1574	[113.9682,4...	[89.5000,417.50...	7
9	3540	[138.6184,3...	[100.5000,305.5...	'Not_arrow'
10	1597	[151.1002,1...	[125.5000,108.5...	8

Figure 37 Field arrow_id added to props the arrows are provided with unique id and object not identified as arrows are labelled as "Not_arrow" as highlighted by red box

5.Computing the properties of yellow regions

```
%% computing the centroid of yellow regions
red_channel = im(:, :, 1);
green_channel =im(:, :, 2);
blue_channel = im(:, :, 3);
yellow_map = green_channel > 220 & red_channel > 220 & blue_channel < 30;
yellow_dots = bwlabel(yellow_map);
figure;imshow(yellow_dots);
yellow_cent = regionprops(yellow_dots);
```

In this section of code properties of yellow dots are computed.

6.Adding the properties of yellow dots to corresponding arrows in props object.

```
%% Adding the centroid of arrows with their other properties

for loop1=1:28
    dist=[];
    disp(dist)
    for loop2=1:31

dist(loop2)=eucledian_dist(yellow_cent(loop1).Centroid,props(loop2).Centroid);
    end
    [M,I]=min(dist);

props(I).Cetroid_of_yellow_dot=yellow_cent(loop1).Centroid;

props(I).BoundingBox_of_yellow_dot=yellow_cent(loop1).BoundingBox;
 end

```

This block of code helps in getting all the properties of arrows and corresponding yellow dots in a single object called props.

Fields	Area	Centroid	BoundingBox	arrow_id	Cetroid_of_yellow_dot	BoundingBox_of_yellow_d
1	1530	[34.4732,11...	[11.5000,83.500...	0	[33.6792,131.5849]	[29.5000,127.5000,8,8]
2	1528	[35.5903,20...	[12.5000,177.50...	1	[34.1143,224.3714]	[29.5000,219.5000,10,10]
3	1582	[49.9532,29...	[19.5000,267.50...	2	[63.7000,308.7750]	[60.5000,305.5000,7,7]
4	1582	[57.5550,39...	[26.5000,369.50...	3	[72.8696,379.7174]	[68.5000,375.5000,8,8]
5	1567	[108.4167,1...	[81.5000,158.50...	4	[121.9167,204.3500]	[117.5000,199.5000,9,9]
6	1558	[109.1438,5...	[82.5000,28.500...	5	[95.6226,68.6226]	[91.5000,64.5000,9,8]
7	1494	[117.9813,2...	[87.5000,229.50...	6	[136.3519,255.4444]	[131.5000,251.5000,9,8]
8	1574	[113.9682,4...	[89.5000,417.50...	7	[101.1132,429.2453]	[96.5000,425.5000,9,8]
9	3540	[138.6184,3...	[100.5000,305.5...	'Not_arrow'	[]	[]

Figure 38 All the properties added in a single object.

Part 7: Drawing bounding boxes over arrows and visualizing their labels

```
%% Drawing bounding boxes
n_objects = numel(props);
figure;
imshow(im);
hold on;
for object_id = 1 : n_objects
    rectangle('Position',
props(object_id).BoundingBox,
'EdgeColor', 'b');
    if props(object_id).arrow_id ~=
'Not_arrow'
text(props(object_id).Centroid(1),props(object_id).Centroid(2)-35,sprintf("%d",props(object_id).arrow_id),Color='r');
    end
end
hold off;
```

The image depicted in Figure 36 is obtained after this step:

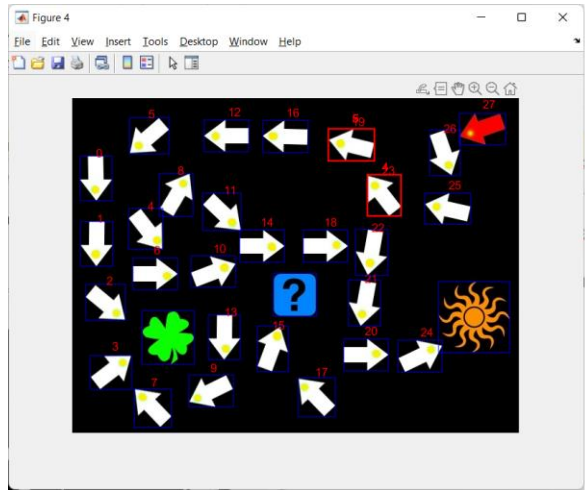

Figure 39 Visualization of bounding boxes and arrows obtained after step 7

8.Findig Arrow:

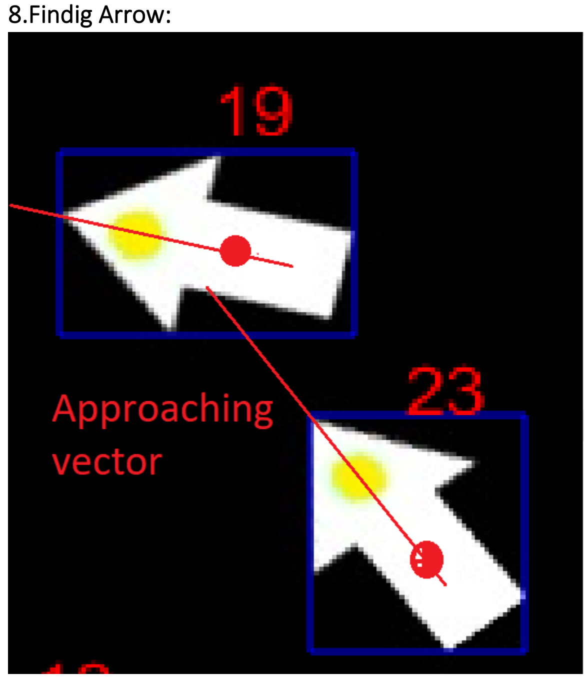

Figure 40 Arrow detection mechanism

The approaching vector the directional directive obtained by subtracting the co-ordinates of centroid of yellow dot by centroid of arrow. We increment this directional directive with the distance between centroid of yellow dot and centroid of arrow when the acquired coordinates enter the boundary of a bounding box the control transfers to that arrow and the same procedure is repeated till the treasure is reached.

```
current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
```

The mechanism for detecting the co-ordinates reaching inside the bounding box is given by the following line of code.

```
current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
      rectangle('Position', props(current_object).BoundingBox, 'EdgeColor', 'r',LineWidth=1.5);
      if props(current_object).arrow_id ~= 'Not_arrow'
            text(props(current_object).Centroid(1),props(current_object).Centroid(2)-40,sprintf("%d",arrow_count),Color='r');
      end
co_ordinates=co_ordinates+(current_object_direction*0.5);

```

This is how the code works the final part of the code that helps in searching treasure is given below:

```
%% Finding the object
props(9).treasure_id=1;
current_object=31;
co_ordinates=props(current_object).Cetroid_of_yellow_dot;
go=true;
figure();
imshow(im);
hold on;
% input("Press Enter to start")
arrow_count=1;
while go
    current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
      rectangle('Position', props(current_object).BoundingBox, 'EdgeColor', 'r',LineWidth=1.5);
      if props(current_object).arrow_id ~= 'Not_arrow'
            text(props(current_object).Centroid(1),props(current_object).Centroid(2)-40,sprintf("%d",arrow_count),Color='r');
      end
    co_ordinates=co_ordinates+(current_object_direction*0.5);
     for loop3=1:31
         if co_ordinates(1) > props(loop3).BoundingBox(1) & co_ordinates(2) > props(loop3).BoundingBox(2) & co_ordinates(1) < props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3) & co_ordinates(2) < props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4) & loop3~=current_object
             disp(loop3)
             disp(co_ordinates)
             disp(props(loop3).BoundingBox(1))
             disp(props(loop3).BoundingBox(2))
             disp(props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3))
             disp(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
             pause(0.5)

             if props(loop3).arrow_id == 'Not_arrow'

                 rectangle('Position', props(loop3).BoundingBox, 'EdgeColor', 'b','LineWidth',2);
                 text(props(loop3).Centroid(1)-120,props(loop3).Centroid(2)+50,sprintf("This is the Treasure!!"),Color='b',FontSize=15);
                 if props(loop3).treasure_id==1
                     if current_object_direction(2)>0
                         while co_ordinates(2)<(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
                             co_ordinates=co_ordinates+current_object_direction;
                         end
                     else
                         while co_ordinates(2)>(props(loop3).BoundingBox(2))
                             co_ordinates=co_ordinates+current_object_direction;
                         end
                     end
                 else
                     go = false;
                 end
             else
                 current_object=loop3;
                 disp(current_object)
                 disp(loop3)
                 arrow_count=arrow_count+1;
                 co_ordinates=props(current_object).Cetroid_of_yellow_dot;
             end

         end
     end
end

```

Result: The code worked as expected and was successful in Identifying the treasure.

Task 5: Study and compare Capsule CNN, Siamese CNN, and Yolo CNN with respect to their architecture, principle of operation, advantages and disadvantages and applications with respect to task such as detection, Classification and Segmentation.

Capsule CNN:

CNN have been most important development in the field of image classification and object detection however they have certain limitations. Such as non-localization and focus on object existence. Low 3D viewpoint variations and difficulty in segmentation of overlapping objects to overcome these limitations a research group lead Geoffry Hinton came up with Capsule Nets.

Instead of forwarding individual neuron activation between consecutive layers capsule Networks make use of nested neural networks to output whole vectors. These vectors pass on the probability of detected features. The direction of vector indicates the state of identified features.

Capsule Networks displays activity variance a term coined by Geoffry Hinton, which means as the features are detected in an image the length of vector representing those features remains same however the direction may change.

Instead of blindly forwarding output vector to each capsule in consecutive layer, capsule Nets estimate the output of next layer and passes the vector to those capsules which will output the largest vector.

The state of every individual capsule is estimated as weighted sum of matrix product with estimations of the capsule of preceding layer's coupling coefficient (cij) lying between capsules state and lower layer's capsules.

Architecture:

As explained earlier CapNets uses nesting of one layer into another layer. The inputs after being fed to convolution layers is rearranged and squashed to evolve into 32 capsules of 6x6x8 capsules each. After the squashing is completed to form multiple capsules margin loss is estimated in higher layers to give class probability.

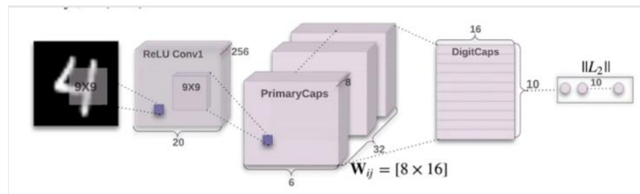

Figure 41 Architecture of Capsule networks.

Principle of operation:

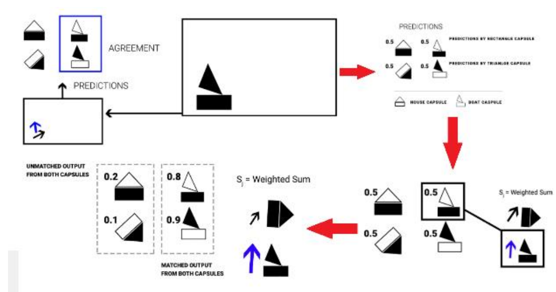

Figure 42 Figure explaining the working principle of Capsule Networks

Advantages of CapsuleNets:

- These networks can learn more robust features even when there is variation of features.
- Shows improved prediction rate and accuracy due to routing between neurons.
- Can even recognize overlapping images in any orientation.

Disadvantages of CapsuleNets:

- Unreliable performance over complex datasets as they are still under development.
- Takes longer time to train due to routing by agreement and are computationally more expensive.

Applications:

- CapsuleNets shine in application areas such as identification of overlapping images and digit recognition
- Used for pose Estimation.

Siamese Networks:

Neural networks can be trained to perform any task but require large datasets for training purpose, however in some practical scenarios availability of data is a big concern to overcome this a new type of network architecture is implemented known as Siamese networks.

The ability to train over limited dataset makes Siamese Network go to choice for applications such as face detection and signature verification.

Architecture:

Siamese Neural Networks deploy two different networks operating parallelly with similar configuration. Both the networks have same parameters and weights. Similar input is provided to both the networks simultaneously and the extracted feature vector from both the networks is compared.

Usually, neural networks one trained on a dataset works perfectly well under similar conditions but if we need to a class to the network the whole network needs to be retrained again on entire dataset. Moreover, Deep Neural Networks require large amount of data to train. On the contrary Siamese Neural Networks (SNNs) extract a similarity function and can very easily be applied on newer classes.

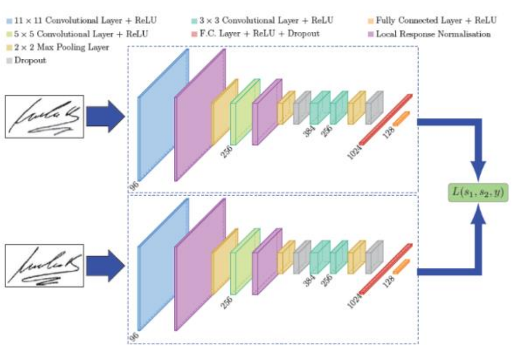

Figure 43 Architecture of Siamese Neural Networks used in Signet

Advantages of Siamese Neural Networks:

- Robust to class imbalance: Siamese Networks deploys one shot learning and can train with very less data.
- Can be easily merged with any state-of-the-art classifier. As the working mechanism of these networks is different from classification, they can easily be merged with any classifier by averaging the result of classifier with a Siamese Network.
- Siamese Networks can learn Semantic Similarity.

Disadvantages of Siamese Networks:

- Training time is high as compared to another networks
- Involves pairwise learning hence does not output probabilities of prediction.

Applications:

- Face detection and Signature Verification
- Used in Robotics example deployed for grasping application in complicated environment
- Used in Human activity recognition.

Yolo Convolution Neural Networks:

Instead of calling yolo a proper CNN it should be treated as an algorithm that uses Neural Networks for real time object detection. The reason behind the popularity of yolo algorithm is its speed and accuracy. It is mainly applied in areas such as traffic signal detection, or detecting people , animals or parking meters.

YOLO has been abbreviated from the term 'You only look once'.It is basically a classification algorithm that detects objects from images in real time and provides probability of detected objects.

It not only predicts class probability but simultaneously bound boxes to those predictions. There has been several enhancement in this algorithm and now we have YOLO and YOLOv3 available for usage.

Architecture:

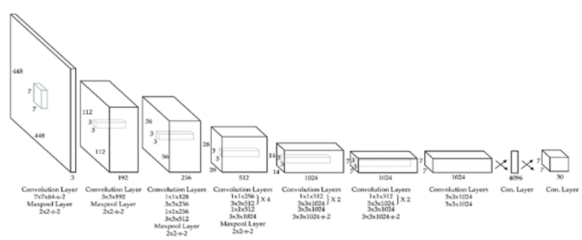

Figure 44 Architecture of Yolo Algorithm.

Working Principle:

Yolo algorithm works based on following three techniques:

- Intersection Over union
- Bounding Box regression
- Residual Blocks.

Intersection Over Union:

Yolo deploys Intersection over union to give an output box that encases the object completely.

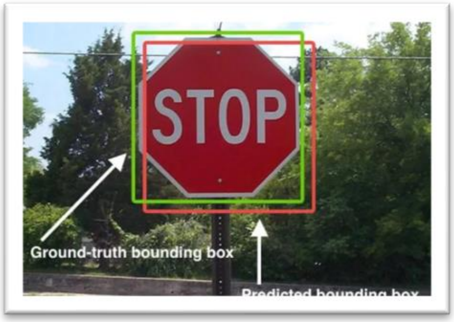

Figure 45Image showing yolo IOU

Bounding box regression:

In bounding box regression, a box is generated with parameters such as width, height and class of object which highlights an object in an image.

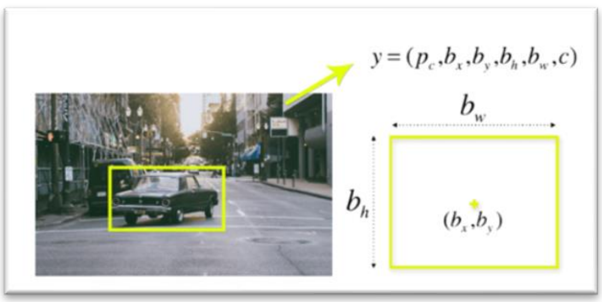

Figure 46 Bounding Box regression

Residual Blocks:

Image is divided into many smaller sections that resembles a grid. Regression is applied over the objects present in this grid to apply bounding block on objects.

Advantages of Yolo:

- Yolo algorithm can detect object at great speed
- Yolo can give very accurate results with minimum errors.
- It possesses excellent learning capabilities.
- It is open source

Disadvantages of Yolo:

- Difficulty in identifying objects significantly smaller than the size of the grid
- Sometimes it makes errors in plotting bounding boxes.
- Difficulty identifying new objects if object has unusual aspect ratio.

Applications:

- Used in Autonomous systems such as driverless cars and UAVs
- Is also used for automatic audio description, helping blind people visualize the scenes

Conclusion

This report has been subdivided into 5 different tasks. The first task involved basics understanding of Computer vision. Study of histograms of a grayscale image, RGB image and deriving interpretations form those histograms to enhance the understanding of basic concepts. It also involved applying different types of filters over images subjected to different types of noises. This helped in learning basic image processing task such as noise removal, edge detection using Sobel , canny and Prewitt filters . The second task was related to optical flow algorithm and where the learning outcome was to learn corner detection and use of Lucas-Kanada algorithm to detect motion in an object. The third task involved study of moving object detection and application of frame differencing approach and gaussian mixture approach. The fourth task was the most interesting and challenging one which involved the application of all the acquired knowledge and coding skills in order to detect treasure from an image. This task was with increasing difficulty level which helped in developing skill progressively. The fifth task involved research and presentation of various state of the art computer vision techniques such as Capsule CNNs, YOLO CNN, and Siamese CNN Overall, it can be concluded that this report was a great learning exercise to start in the field of Machine Vision.

1. intel. (2018, 06 20). *Understand Capsule Network Architecture*. Retrieved from intel.com: https://www.intel.com/content/www/us/en/developer/articles/technical/understanding-capsule-network-architecture.html
2. **J, S. B. (2020, September 2). *a-friendly-introduction-to-siamese-networks-*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942**
3. **karimi, G. (n.d.). *introduction-to-yolo-algorithm-for-object-detection*. Retrieved from section.io: https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/**

Appendix:

Code for Task 2

Determine the optical flow for Gingerbread Man

```
img1=imread("GingerBreadMan_first.jpg");
img2=imread("GingerBreadMan_second.jpg");

img1_gray=rgb2gray(img1);
img2_gray=rgb2gray(img2);

opticFlow = opticalFlowLK('NoiseThreshold',0.009);

flow=estimateFlow(opticFlow,img1_gray);
flow=estimateFlow(opticFlow,img2_gray);

figure,imshow(img1)
title("Ginger Breadman First");
hold on
plot(flow,'DecimationFactor',[5 5],'ScaleFactor',10);
hold off

figure,imshow(img2)
title("Ginger Breadman Second");
hold on
plot(flow,'DecimationFactor',[5 5],'ScaleFactor',10);
hold off

```

Visualize the track of one point of red box in video

```
clear
clc

%% Creating video reader object
video=VideoReader('red_square_video.mp4');

gt=load("red_square_gt.mat");
gt=struct2array(gt);

%% creating an optical flow object
opflow=opticalFlowLK("NoiseThreshold",0.009);

predicted_corner_cordinates=[];
frame=1;
figure("Name",'Path of box');
%% loop for reading all the frames
while hasFrame(video)
    frameRGB=readFrame(video);
    frameGray=rgb2gray(frameRGB);
    flow=estimateFlow(opflow,frameGray)
    % cordinates of top left point of the box
    c=corner(frameGray);
    left=min(c(:,1));
    top=min(c(:,2));
    imshow(frameRGB);
    hold on;
    plot(flow,'DecimationFactor',[5,5],'ScaleFactor',10);
```

```
    plot(left,top,'ko');


    % estimateFlow to determine the optical flow object
    flow=estimateFlow(opflow,frameGray);

    % adding the obtained co-ordinates of rectangel to the list
    left_new=left+flow.Vx(round(top),round(left));
    top_new=top+flow.Vy(round(top),round(left));
    predicted_corner_cordinates(end+1,:)=[left,top];
    pause(10^-3)

end

%% plotting the path of box , predicted and actual
figure("Name","Plot showing the path of corner point");
for i=1:video.NumFrames
    hold on;
    set(gca, 'Ydir', 'reverse')
    plot(predicted_corner_cordinates(i,1),predicted_corner_cordinates(i,2),'r<');
    plot(gt(i,1),gt(i,2),'k>');
    legend('Predicted Loacation', 'Ground truth track')
    hold off;
end

%% Plotting the squared error
errors=[]
sum=0
for i=1:(video.NumFrames)
    error=sqrt((gt(i,2)-predicted_corner_cordinates(i,2))^2)+((gt(i,1)-predicted_corner_cordinates(i,1))^2);
    sum=sum+error;
    errors(end+1)=error
    hold off
end
figure('Name','Error at each location');
plot(1:video.NumFrames,errors(1:video.NumFrames));
RMSE = sqrt(sum/(video.NumFrames))

```

Code for Task 4: Treasure Hunt

Easy Task

```
clear
clc

%% Reading image
im = imread(['Treasure_easy' ...
    '.jpg']); % change name to process other images
figure;
imshow(im);

%% Binarisation
bin_threshold = 0.05; % parameter to vary
bin_im = im2bw(im, bin_threshold);
figure;
imshow(bin_im);


%% Extracting connected components
```

```
con_com = bwlabel(bin_im);
figure;
imshow(label2rgb(con_com));


%% Computing objects properties
props = regionprops(con_com);

props=arrow_finder(im,props);

%% computing the centroid of yellow regions
red_channel = im(:, :, 1);
green_channel =im(:, :, 2);
blue_channel = im(:, :, 3);
yellow_map = green_channel > 220 & red_channel > 220 & blue_channel < 30;
yellow_dots = bwlabel(yellow_map);
yellow_cent = regionprops(yellow_dots);



%% Adding the centroid of arrows with their other properties

for loop1=1:10
    dist=[];
    disp(dist)
    for loop2=1:11
        dist(loop2)=eucledian_dist(yellow_cent(loop1).Centroid,props(loop2).Centroid);
    end
    [M,I]=min(dist);
    props(I).Cetroid_of_yellow_dot=yellow_cent(loop1).Centroid;
    props(I).BoundingBox_of_yellow_dot=yellow_cent(loop1).BoundingBox;
 end

%% Drawing bounding boxes
n_objects = numel(props);
figure;
imshow(im);
hold on;
for object_id = 1 : n_objects
    rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'b');
    if props(object_id).arrow_id ~= 'Not_arrow'
    text(props(object_id).Centroid(1),props(object_id).Centroid(2)-35,sprintf("%d",props(object_id).arrow_id),Color='r');
    end
end
hold off;

%% Finding the object
current_object=1;
co_ordinates=props(current_object).Cetroid_of_yellow_dot;
go=true;
figure();
imshow(im);
hold on;
% input("Press Enter to start")
while go
current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
      rectangle('Position', props(current_object).BoundingBox, 'EdgeColor', 'r',LineWidth=1.5);
      if props(current_object).arrow_id ~= 'Not_arrow'
         text(props(current_object).Centroid(1),props(current_object).Centroid(2)-40,sprintf("%d",props(current_object).arrow_id),Color='r');
      end
```

```
        co_ordinates=co_ordinates+(current_object_direction*0.5);
        for loop3=1:11
            if co_ordinates(1) > props(loop3).BoundingBox(1) & co_ordinates(2) > props(loop3).BoundingBox(2) & co_ordinates(1) < props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3) & co_ordinates(2) < props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4) & loop3~=current_object
                disp(loop3)
                disp(co_ordinates)
                disp(props(loop3).BoundingBox(1))
                disp(props(loop3).BoundingBox(2))
                disp(props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3))
                disp(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
                pause(2)
                if props(loop3).arrow_id == 'Not_arrow'
                    rectangle('Position', props(loop3).BoundingBox, 'EdgeColor', 'b','LineWidth',2);
                    text(props(loop3).Centroid(1)-120,props(loop3).Centroid(2)+50,sprintf("This is the Treasure!!"),Color='b',FontSize=15);
                    go=false
                else
                    current_object=loop3;
                    disp(current_object)
                    disp(loop3)
                    co_ordinates=props(current_object).Cetroid_of_yellow_dot;
                end

            end
        end
    end

```

Medium Difficulty

```
clear
clc

%% Reading image
im = imread('Treasure_medium.jpg'); % change name to process other images
figure;
imshow(im);

%% Binarisation
bin_threshold = 0.05; % parameter to vary
bin_im = im2bw(im, bin_threshold);
figure;
imshow(bin_im);


%% Extracting connected components
con_com = bwlabel(bin_im);
figure;
imshow(label2rgb(con_com));


%% Computing objects properties
props = regionprops(con_com);

props=arrow_finder(im,props);

%% computing the centroid of yellow regions
red_channel = im(:, :, 1);
```

```
green_channel =im(:, :, 2);
blue_channel = im(:, :, 3);
yellow_map = green_channel > 220 & red_channel > 220 & blue_channel < 30;
yellow_dots = bwlabel(yellow_map);
yellow_cent = regionprops(yellow_dots);



%% Adding the centroid of arrows with their other properties

for loop1=1:15
    dist=[];
    disp(dist)
    for loop2=1:17
        dist(loop2)=eucledian_dist(yellow_cent(loop1).Centroid,props(loop2).Centroid);
    end
    [M,I]=min(dist);
    props(I).Cetroid_of_yellow_dot=yellow_cent(loop1).Centroid;
    props(I).BoundingBox_of_yellow_dot=yellow_cent(loop1).BoundingBox;
 end

%% Drawing bounding boxes
n_objects = numel(props);
figure;
imshow(im);
hold on;
for object_id = 1 : n_objects
    rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'b');
    if props(object_id).arrow_id ~= 'Not_arrow'
    text(props(object_id).Centroid(1),props(object_id).Centroid(2)-35,sprintf("%d",props(object_id).arrow_id),Color='r');
    end
end
hold off;


%% Finding the object
current_object=1;
co_ordinates=props(current_object).Cetroid_of_yellow_dot;
go=true;
figure();
imshow(im);
hold on;
arrow_count=1;
while go
current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
      rectangle('Position', props(current_object).BoundingBox, 'EdgeColor', 'r',LineWidth=1.5);
      if props(current_object).arrow_id ~= 'Not_arrow'
         text(props(current_object).Centroid(1),props(current_object).Centroid(2)-40,sprintf("%d",arrow_count),Color='r');
      end
     co_ordinates=co_ordinates+(current_object_direction*0.5);
     for loop3=1:17
         if co_ordinates(1) > props(loop3).BoundingBox(1) & co_ordinates(2) > props(loop3).BoundingBox(2) & co_ordinates(1) < props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3) & co_ordinates(2) < props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4) & loop3~=current_object
             disp(loop3)
             disp(co_ordinates)
             disp(props(loop3).BoundingBox(1))
             disp(props(loop3).BoundingBox(2))
             disp(props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3))
```

```
              disp(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
              pause(2)
              if props(loop3).arrow_id == 'Not_arrow'
                  rectangle('Position', props(loop3).BoundingBox, 'EdgeColor', 'b','LineWidth',2);
                  text(props(loop3).Centroid(1)-120,props(loop3).Centroid(2)+50,sprintf("This is the Treasure!!"),Color='b',FontSize=15);
                  go=false
              else
                  current_object=loop3;
                  disp(current_object)
                  disp(loop3)
                  arrow_count=arrow_count+1;
                  co_ordinates=props(current_object).Cetroid_of_yellow_dot;
              end

          end
      end
  end

```

Difficult Task

```
clear
clc

%% Reading image
im = imread('Treasure_hard.jpg'); % change name to process other images
figure;
imshow(im);

%% Binarisation
bin_threshold = 0.05; % parameter to vary
bin_im = im2bw(im, bin_threshold);
figure;
imshow(bin_im);


%% Extracting connected components
con_com = bwlabel(bin_im);
figure;
imshow(label2rgb(con_com));


%% Computing objects properties
props = regionprops(con_com);

props=arrow_finder(im,props);

%% computing the centroid of yellow regions
red_channel = im(:, :, 1);
green_channel =im(:, :, 2);
blue_channel = im(:, :, 3);
yellow_map = green_channel > 220 & red_channel > 220 & blue_channel < 30;
yellow_dots = bwlabel(yellow_map);
figure;imshow(yellow_dots);
yellow_cent = regionprops(yellow_dots);



%% Adding the centroid of arrows with their other properties

for loop1=1:28
```

```
    dist=[];
    disp(dist)
    for loop2=1:31
        dist(loop2)=eucledian_dist(yellow_cent(loop1).Centroid,props(loop2).Centroid);
    end
    [M,I]=min(dist);
    props(I).Cetroid_of_yellow_dot=yellow_cent(loop1).Centroid;
    props(I).BoundingBox_of_yellow_dot=yellow_cent(loop1).BoundingBox;
 end

%% Drawing bounding boxes
n_objects = numel(props);
figure;
imshow(im);
hold on;
for object_id = 1 : n_objects
    rectangle('Position', props(object_id).BoundingBox, 'EdgeColor', 'b');
    if props(object_id).arrow_id ~= 'Not_arrow'
    text(props(object_id).Centroid(1),props(object_id).Centroid(2)-35,sprintf("%d",props(object_id).arrow_id),Color='r');
    end
end
hold off;

%% Finding the object
props(9).treasure_id=1;
current_object=31;
co_ordinates=props(current_object).Cetroid_of_yellow_dot;
go=true;
figure();
imshow(im);
hold on;
% input("Press Enter to start")
arrow_count=1;
while go
      current_object_direction=direction(props(current_object).Centroid,props(current_object).Cetroid_of_yellow_dot);
      rectangle('Position', props(current_object).BoundingBox, 'EdgeColor', 'r',LineWidth=1.5);
      if props(current_object).arrow_id ~= 'Not_arrow'
         text(props(current_object).Centroid(1),props(current_object).Centroid(2)-40,sprintf("%d",arrow_count),Color='r');
      end
     co_ordinates=co_ordinates+(current_object_direction*0.5);
     for loop3=1:31
         if co_ordinates(1) > props(loop3).BoundingBox(1) & co_ordinates(2) > props(loop3).BoundingBox(2) & co_ordinates(1) < props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3) & co_ordinates(2) < props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4) & loop3~=current_object
             disp(loop3)
             disp(co_ordinates)
             disp(props(loop3).BoundingBox(1))
             disp(props(loop3).BoundingBox(2))
             disp(props(loop3).BoundingBox(1)+props(loop3).BoundingBox(3))
             disp(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
             pause(0.5)

             if props(loop3).arrow_id == 'Not_arrow'

                 rectangle('Position', props(loop3).BoundingBox, 'EdgeColor', 'b','LineWidth',2);
                 text(props(loop3).Centroid(1)-120,props(loop3).Centroid(2)+50,sprintf("This is the Treasure!!"),Color='b',FontSize=15);
                 if props(loop3).treasure_id==1
```

```
                   if current_object_direction(2)>0
                       while co_ordinates(2)<(props(loop3).BoundingBox(2)+props(loop3).BoundingBox(4))
                       co_ordinates=co_ordinates+current_object_direction;
                       end
                     else
                          while co_ordinates(2)>(props(loop3).BoundingBox(2))
                         co_ordinates=co_ordinates+current_object_direction;
                          end
                     end
                 else
                     go = false;
                 end
             else
                 current_object=loop3;
                 disp(current_object)
                 disp(loop3)
                 arrow_count=arrow_count+1;
                 co_ordinates=props(current_object).Cetroid_of_yellow_dot;
             end

         end
     end
end

```