Department of Electrical and Computer Engineering University of Victoria

ECE 490 REPORT 1

Title: Survey and Review and Computer Vision Theory and Methods

Date Submitted: **2022-05-27**

Name: Matthew Ebert

Supervisor: Dr. David Capson P.Eng

Table of Contents

1. Introduction 6
  1.1 Computer Vision: A Discrete Eyes 6
  1.2 The Challenge 6
2. Images and Morphology 7
  2.1 Digital Images and Mathematical Representation 7
    2.1.1 The Convolution Operator 8
    2.1.2 Histograms and 1D Plots 9
  2.2 Image Operations 9
    2.2.1 Shifting 9
    2.2.2 Thresholding 10
    2.2.3 Erosion and Dilation 10
    2.2.4 Opening and Closing 11
  2.3 Filters 12
    2.3.1 Gaussian Smoothing 12
    2.3.2 The Median Filter 12
    2.3.3 The Bilateral Filter 13
    2.3.4 Sharpening 14
3. Detecting Features 14
  3.1 Edge Detection 14
    3.1.1 Sobel Operator 15
    3.1.2 The Canny Edge Detector 16
    3.1.3 The Laplacian in Edge detection 16
  3.2 Boundary Detection 17
    3.2.1 Hough Transforms 17
    3.2.2 RANSAC 18
4. Object Detection 18
  4.1 Challenges 19
  4.2 Binary Shape Analysis 19
  4.3 SIFT Detector and Blob Detection 20
  4.4 Segmentation 21
    4.4.1 Mean Shift 21
  4.5 Feature Matching 22
  4.6 Convolutional Neural Networks 22

4.6.1 Creating the Weight Vector 24
4.6.2 Training with YOLOV5 24
4.7 Other Object Detection Models 25
5. Object Tracking 25
5.1 Single Target Tracking 25
5.1.1 GOTURN Tracking 25
6. General Discussion and Observations 26
6.1 Limitations of Computer Vision Techniques 26
6.2 Applications in Virtual Servo 27
7. Conclusion 27
References 29
APPENDIX A: LAB 1 31
Objective 31
Apparatus 31
Procedure and Results 31
Image Morphology 31
Feature Detection 35
Object Detection 36
Object Tracking with GOTURN 38
Discussion 41
Conclusion 42

Table of Figures

Figure 1: Lena; a benchmark image in the computer vision field 7
Figure 2: 3 types of digital image [1].. 8
Figure 3: The discrete convolution operator [2] 8
Figure 4: Example of a 'Moving Average' convolution on a gray scale image [1]....................... 8
Figure 5: Image Histogram [1]. 9
Figure 6: Histogram plot of the image of a flower (RGB) [3].. 9
Figure 7: 1D gray scale image plot. (Intensity vs position) [2]. 9
Figure 8: Example of image shifting [1]... 10
Figure 9: Example of image thresholding. Not that some of the object’s structure is lost during binarization [2].The image also shows the original images histogram. The large divide between the two groups of pixel intensities makes it ideal for thresholding. 10
Figure 10: An example of binary image (b) dilation and (c) erosion. (a) is the original image [3]. Note that black is equal to 1 and white to 0 in this image. 11
Figure 11: Dilation operator [4].. 11
Figure 12: Erosion operator [4].. 11
Figure 13: Opening of an image by erosion then dilation [5]... 11
Figure 14: Closing of an image by dilation then erosion [3]. 11
Figure 15: an original image with noise [3].. 12
Figure 16: Effect of gaussian smoothing on an edge (1D, intensity vs position) [2] 12
Figure 17: An example of gaussian smoothing on the original image [3]................................. 12
Figure 18: An example of applying a median filter to the original image [3]. 13
Figure 19: A bilateral filter applied to a step edge [3].. 13
Figure 20: An example of applying a bilateral filter to the original image [3]........................... 13
Figure 21: Example of applying a moving average filter to sharpen an image by subtracting from the original [1].. 14
Figure 22: Example 1D edges (intensity vs position) [2]. 14
Figure 23: 2D differentiation (gradient) [1].. 15
Figure 24: 3x3 image gradient filters [1]. 15
Figure 25: Sobel edge operators [2]... 16
Figure 26: The Sobel operator separated into operations. 16
Figure 27: Hough Transform: line boundary detection graphical voting representation. Note the intersection of the points corresponds to the best fit line [3]... 17
Figure 28: Hough transform for line detection [8].. 18
Figure 29: Hough transform for circle detection [8].. 18
Figure 30: Type of image geometric transformations [10]... 19
Figure 31: Basic set of 2D planar transformations [3].. 19
Figure 32: Labeling object in a binary image [2]. 19
Figure 33: The SIFT detector using DoG to find interest points [12]. 20
Figure 34: SIFT Key point orientation descriptor [3]... 20
Figure 35: Examples of semantic segmentation [3].. 21
Figure 36: Graphs detailing the use of mean shift. Plots are in L*, u*, v* space [3]. 21
Figure 37: Methods of object detection with feature matching [3]. 22
Figure 38: Simplification of CNN in CV [1]... 23
Figure 39: Dot product simplified convolution of an image with a weighted predictor [1] 23
Figure 40: Visualized weight vectors from a 2-layer CNN [14] 24

Figure 41: GoogLeNet CNN for object identification [15]. 24
Figure 42: Architecture of GOTURN [17]. 26
Figure 43: Displaying a local image with OpenCV.. 32
Figure 44: Lena gray scaled with OpenCV.. 33
Figure 45: Lena binarized with OpenCV.. 33
Figure 46: Lena with different Gaussian blurs applied with OpenCV. 34
Figure 47: Figure 38: Lena with different median blurs applied with OpenCV.......................... 34
Figure 48: Lena with a custom filter applied with OpenCV.. 35
Figure 49: SIFT detector used to find key points. 35
Figure 50: A Hough transform implementation to detect circles. Code modified from [19] 36
Figure 51: Implementation of object detection with OpenCV and Yolo. Code modified from [20]. 36
Figure 52: Object to train the custom CNN on. 36
Figure 53: Labeled training data for the yolov5 model. 37
Figure 54: Importing training data into google collaborator. 37
Figure 55: Training the model. 37
Figure 56: Testing the trained yolov5 model. 38

1. Introduction

This report comprises a brief overview of basic computer vision (CV) theory. The objective of this review is to be a primer for further research into computer vision control systems or virtual servo. Consequently, the review focuses on building a foundation for object detection and tracking. Mostly static image analysis will be covered by this report and only a brief overview of object tracking. Video calibration, optical flow, stereo vision, and other video application will be left for later study.

The topics to be covered include definitions and objective of CV, image morphology and operations, feature detection, object detection, CV with artificial intelligence, and single online object tracking. While the list and topics do not begin to cover the full field of CV, our research indicates they provide a basis for further research into applied CV.

This report is meant as a summary. In depth mathematical proofs or examples will be omitted here.

1.1 Computer Vision: A Discrete Eyes

Vision is regarded as the primary sense of humans. From our perspective, the brain seems to almost effortlessly analyses and understand the images capture by our eyes. However, the energy it takes to accomplish these tasks is significant. Although making up, on average, around 2% of a human's mass, it consumes almost 20% of its energy.

Computer systems, when faced with the same problem of using light to understand its surroundings, fall short in one key area: analysis. Sophisticated camera technology can far outperform any human eye, but the ability to near instantaneously identify and object, or track that object through obstacle is still very challenging for a computer.

Computer vision can be defined as a system which allows computers to derive meaningful information from digital images and videos. In the current paradigm, this means processing an array of digital pixel values to establish objects, orientation, and motion.

1.2 The Challenge

The challenge posed by CV is to interpret an image accurately and quickly.

Let us consider the following,

We are a computer system presented with a sample digital image for analyses. We must determine what is in the image. We have a data set of other, labeled, images. From these we need to define, compare, and recognize the object in the sample image.

From this perspective, we can carefully guide ourselves through the processes of analyzing the image to the point when a conclusion can be made.

Figure 1: Lena; a benchmark image in the computer vision field

2. Images and Morphology

The first step to take in completing the challenge is to define what an image is and find useful operations which highlight or extract data in the image.

2.1 Digital Images and Mathematical Representation

A digital image may be defined as a NxMxC matrix with the indices corresponding to the location of each pixel, and the value(s) representing the intensity of light/color of the pixel. The number of columns and row give the height and width of the image in pixels (N and M). The third dimension, C, is 1 for a gray or binary image, and 3 for color images. By convention the value of each pixel takes on 0-1 for binary images, 0-255 for gray scale images, and 0-255 for each of the 3 dimensions for color images (Figure 2).

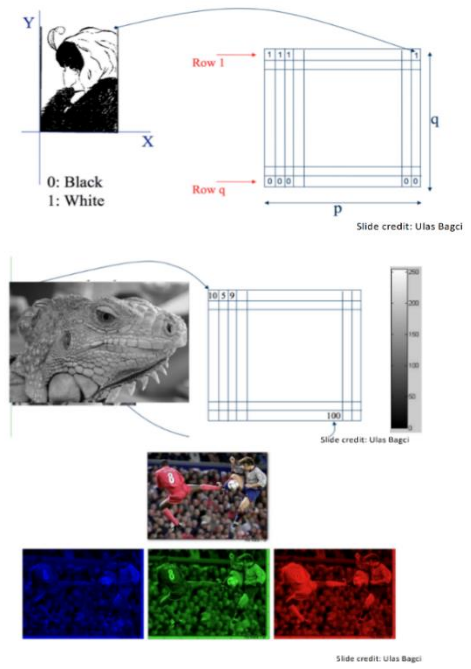


Figure 2: 3 types of digital image [1].

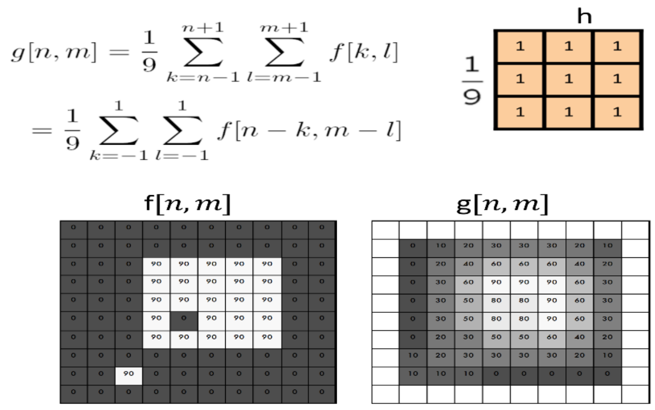
2.1.1 The Convolution Operator

The use of the convolution operator becomes immediately apparent for a digital image. Discrete convolution takes a weighted sum of all values around a point for all points in a signal (Figure 1). In terms of an image, the operator computes a weighted sum of all surrounding pixel and the target pixel and stores the calculated value in a new image at the target pixel's position.

In CV images are convolved with a kernel or mask to obtain a desired effect. An example is shown below with f and the original image, h as the kernel, and g as the transformed image.

$$F(x,y) = f(x,y) \otimes g(x,y) = \sum_i \sum_j f(i,j)g(x-i, y-j)$$

Figure 3: The discrete convolution operator [2]



$$g[n,m] = \frac{1}{9}\sum_{k=n-1}^{n+1}\sum_{l=m-1}^{m+1} f[k,l]$$
$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$\mathrm{f}[n,m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\mathrm{g}[n,m]$

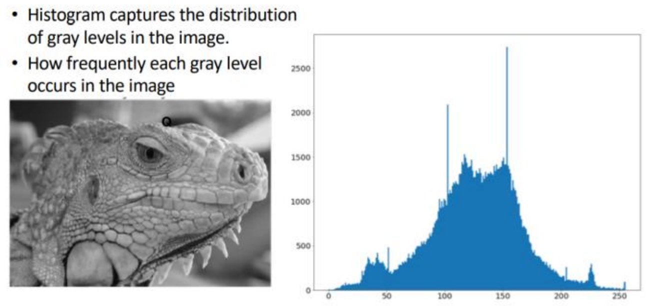
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Figure 4: Example of a 'Moving Average' convolution on a gray scale image [1].

Individual kernels have a large range of effects on an image. The process of changing or altering an image in this way is known as image morphology. Several morphological techniques are described in the following sections.

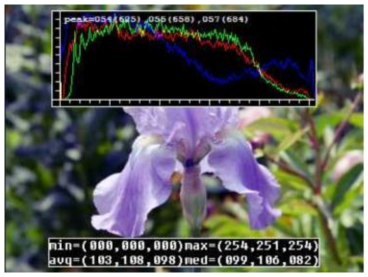
2.1.2 Histograms and 1D Plots

Histograms and 1D plots are used to help identify and differentiate images. The histogram (Figure 5, Figure 6), show the distribution of intensities in the image; the histogram does not contain any positional data. The 1D plot can help demonstrate the effect of filters and kernels on an image. The 1D plot shows intensity vs position for a uniform plane or a 1-pixel width line (Figure 7).



- Histogram captures the distribution of gray levels in the image.
- How frequently each gray level occurs in the image

Figure 5: Image Histogram [1].



```
peak=054(625),056(658),057(684)
min=(000,000,000)max=(254,251,254)
avg=(103,108,098)med=(099,106,082)
```

Figure 6: Histogram plot of the image of a flower (RGB) [3].

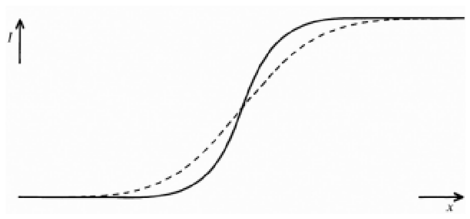


Figure 7: 1D gray scale image plot. (Intensity vs position) [2].

2.2 Image Operations

The following section show several common kernels and algorithms and their effects on images.

2.2.1 Shifting

Shifting an image involves uniformly moving all pixel in a direction (Figure 8). Shifting produces a line of zeros at the beginning of the shift direction and deletes the last line of pixels.

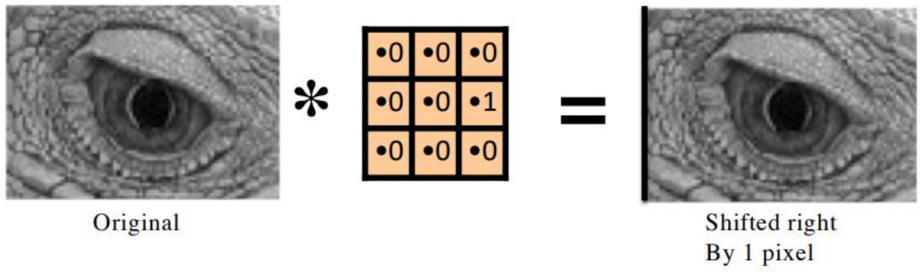


Figure 8: Example of image shifting [1].

2.2.2 Thresholding

Thresholding produces a binary image from a gray scale image. To perform a threshold operation, set the desired threshold level; if the pixel is above or equal to the threshold level, set it 1; otherwise, set it to zero. This operation can be useful to isolate objects or features if a large gap persists between the intensities of the object and the surrounding background. The histogram in Figure 9 shows an ideal image for thresholding. Much information is lost during thresholding, but this simplification has large advantages in terms of calculation speed and grouping of objects.

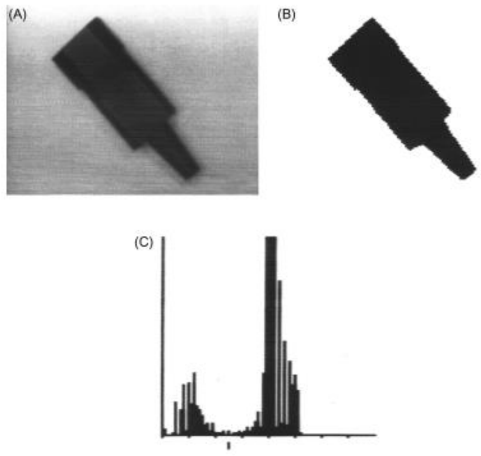


Figure 9: Example of image thresholding. Not that some of the object's structure is lost during binarization [2].The image also shows the original images histogram. The large divide between the two groups of pixel intensities makes it ideal for thresholding.

2.2.3 Erosion and Dilation

Erosion and dilation techniques expand and contract binary images (Figure 10).

Dilation convolves a binary structure element (a kernel matrix) over the image for which,

- if any of the positive (1) structure elements covers a 1, set pixel covered by the target pixel to 1

Erosion uses the same method but

- if the positive (1) values of the structuring elements are not completely covered in the image, set the image pixel covered by the target to zero.

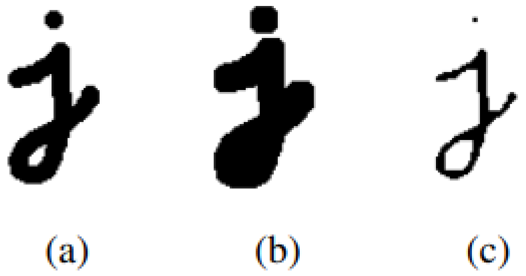


Figure 10: An example of binary image (b) dilation and (c) erosion. (a) is the original image [3]. Note that black is equal to 1 and white to 0 in this image.

Mathematically, dilation expands the image positives based on the structuring image.

$$X \oplus B = \{x + b | x \in X,\ b \in B\} = \cup\{B_x | x \in X\} = \cup\{X_b | b \in B\}$$

Figure 11: Dilation operator [4].

Erosion shrinks the image positives base on the structuring image.

$$X \ominus B = \{z \in P | B_z \subseteq X\}$$

Figure 12: Erosion operator [4].

2.2.4 Opening and Closing

Opening and closing use a combination of erosion and dilation to alter the image. With opening, sharp inlets and corners are contracted; this creates a slightly smaller image with more ‘smooths and straight’ lines. Closing completes the same task but expands the corners, created a slightly larger image (Figure 13, Figure 14).

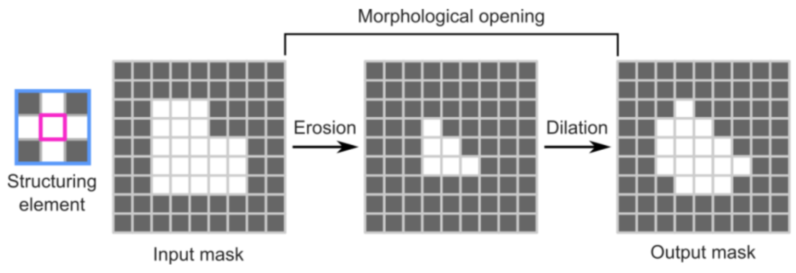


Figure 13: Opening of an image by erosion then dilation [5].

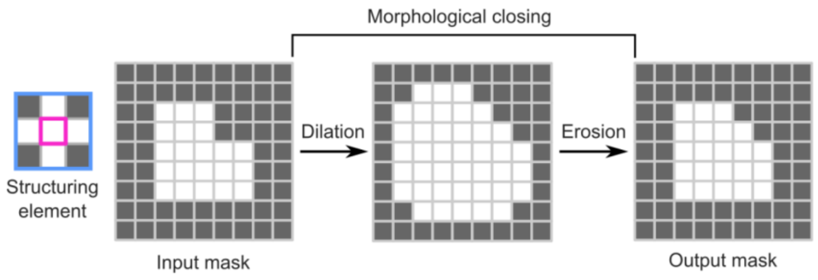
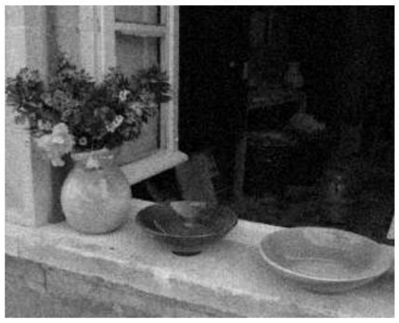


Figure 14: Closing of an image by dilation then erosion [3].

2.3 Filters

Filters apply changes to an entire image. Filters use the convolution operator and specialized kernels to achieve changes such as smoothing, edge enhancement, and noise removal. The image in Figure 15 contains noise and sharp edges. It is used as an example in this section.



(a)

Figure 15: an original image with noise [3].

2.3.1 Gaussian Smoothing

The gaussian kernel, (Figure 16), decreases the gradients of edges through a type of normalization. The gaussian is a linear low pass filter which smooths and blurs an image [2]. This filter also suppresses noise which is evident when comparing Figure 15 and Figure 17.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

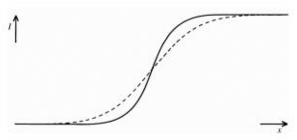
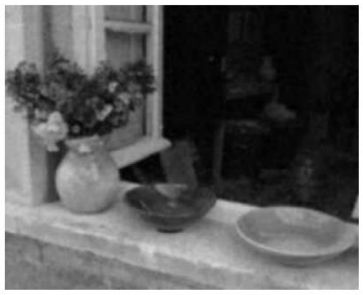


FIGURE 3.2

Blurring of object edges by simple Gaussian convolutions. The simple Gaussian convolution can be regarded as a grayscale neighborhood "mixing" operator, hence explaining why blurring arises.

Figure 16: Effect of gaussian smoothing on an edge (1D, intensity vs position) [2]



(b)

Figure 17: An example of gaussian smoothing on the original image [3].

2.3.2 The Median Filter

The median filter is a non-linear filter which selects the “median value from each pixel’s neighborhood” and assigns it as the target value [3]. The median filter excels at eliminating noise because ‘noise’ pixel values usually lie far from true value. Compared to the gaussian filter, the median filter preserves edges and details; the image appears only slightly softened rather than blurred. However, this operation is more expensive and complex to compute than some other noise reduction filters.

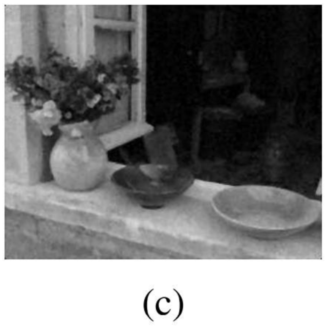


Figure 18: An example of applying a median filter to the original image [3].

2.3.3 The Bilateral Filter

The bilateral filter combines both weighted filtering and outlier rejection [3]. The bilateral filter combines a domain kernel (gaussian) and a range kernel. The combination of these kernels allows for noise filtering while persevering edges and details. Bilateral filters can also be used iteratively and in guided image filtering [3].

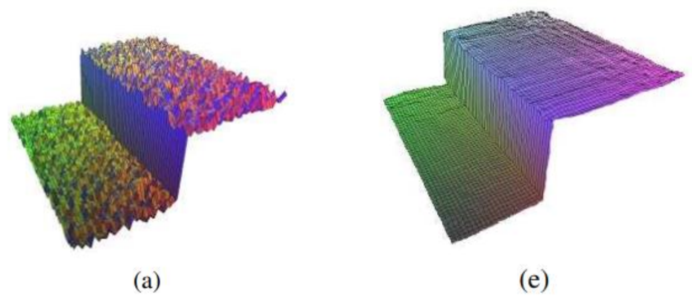


Figure 19: A bilateral filter applied to a step edge [3].

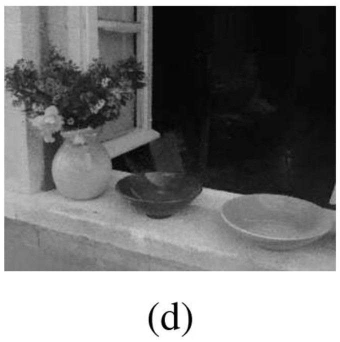


Figure 20: An example of applying a bilateral filter to the original image [3].

2.3.4 Sharpening

A somewhat unintuitive way of sharpening an image involves subtracting a blurred form of the image from the original. The example below shows an image doubled in intensity then subtracted by a moving-average blurred image. This produces an image where the expression of details and edges is enhanced.

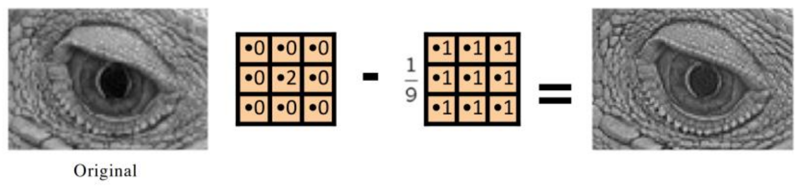


Figure 21: Example of applying a moving average filter to sharpen an image by subtracting from the original [1].

Some examples to demonstrate filtering were created on the ‘lena’ image in Appendix A.

3. Detecting Features

We now know several processes by which to alter an image. Now, we can use these techniques to isolate specific features for identification.

3.1 Edge Detection

In a 2D image, objects are defined by edges. In mathematical terms, an edge, in an image, is a region where the gradient is large. The image below shows several types of edges in 1 dimension.

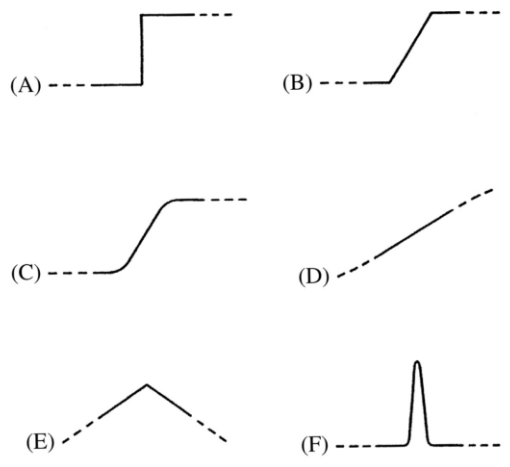


Figure 22: Example 1D edges (intensity vs position) [2].

To detect these sharp changes in the image we examine the image gradient. Since the image is 2D, the gradient may be separated into x and y directions. Consequently, we may define an edge with a slope and direction; [1].

Given function $f(x,y)$

Gradient vector
$$\nabla f(x,y) = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

Gradient magnitude
$$|\nabla f(x,y)| = \sqrt{f_x^2 + f_y^2}$$

Gradient direction
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$$

Figure 23: 2D differentiation (gradient) [1]

To apply the gradient technique to an image, we may create approximate gradient filter (Figure 24). There are several filters which can achieve edge filtering. The one below is perhaps the simplest.

$$\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

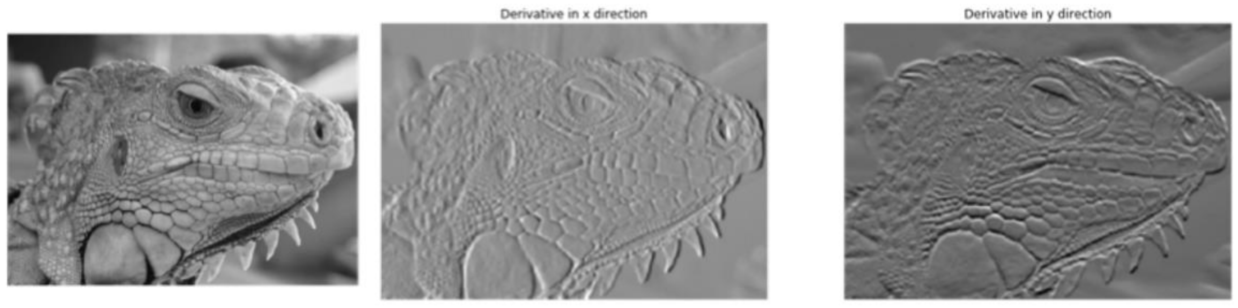


Figure 24: 3x3 image gradient filters [1].

3.1.1 Sobel Operator

Simply applying a gradient filter to image proves less effective when noise is present. The noise creates an erratic derivative, preventing edge isolation.

If a smoothing operator, like the gaussian filter is applied before the gradient operator, this problem can be mitigated. This combined effect can be implements with use of the Sobel edge kernel.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 25: Sobel edge operators [2].

The effect of this operator is shown in the image below.

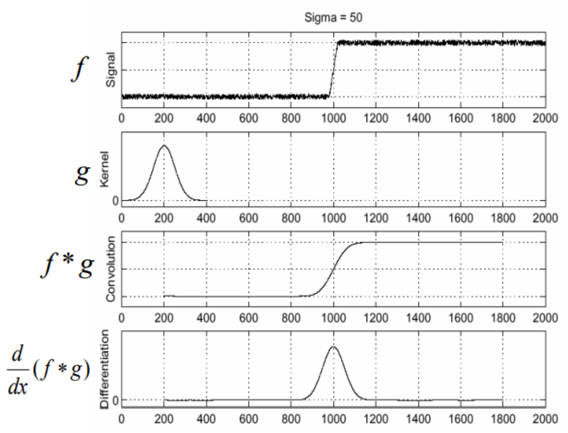


Figure 26: The Sobel operator separated into operations.

3.1.2 The Canny Edge Detector

The Canny operator attempts a different method of edge detection. It is a very popular solution to edge detection [3]. Canny describes his method in detail in a well cited paper [6].

Canny method is summarized as follows.

1. Smooth the image with a gaussian filter
2. Use the Sobel operator to compute the gradient around the edge
3. Find the magnitude and direction of the gradient at each pixel.
4. Compute a 1D Laplacian along the direction of the gradient for each pixel.
5. Find strong ‘zero crossing’ of the Laplacian (i.e. where the Laplacian changes sign).

Step 4 is perhaps the most interesting. By computing a Laplacian in the direction of the gradient of the pixel, the detector is essentially customized to be the best detector in that region [7].

Additionally, since zero crossing (See 3.1.3 for details) are used to detect edges, linking them into contour is “straightforward” [3].

3.1.3 The Laplacian in Edge detection

The gradient of the gradient is known as the Laplacian. Like the gradient, the Laplacian also responds to edges. However, instead of having of reaching a maximum on an edge, the Laplacian crosses the horizontal axis (hence zero crossing). Consider Figure 22 c). The Laplacian measure the change in gradient; therefore, the Laplacian will start from zero and reach a maximum in the first ¼ of the line c). Then as the ‘acceleration’ of the line switches from positive to negative (at the center of c)), the Laplacian will cross zero (hence zero-crossing) and indicate an edge.

Since the Laplacian find the center of an edge, the Laplacian excels at creating boundaries of edges through searching the neighboring pixels Laplacians and systematically joining edged together.

3.2 Boundary Detection

After finding the edges of an image, we must group these images into complete boundaries.

One of the challenges with grouping edges into boundaries is dealing with occlusion and vanishing points. Objects are often partially blocked or shielded by other object which hides their complete shape.

3.2.1 Hough Transforms

The Hough transform is a popular method for boundary detection. The Hough transform uses edge ‘voting’ to find the most probable boundary and line locations.

In the original form, the shape of the object must be known (for example circle radius). Using the known shape, all potential lines or arcs are calculated for each edge point. These ‘potential’ lines are plotted on a parameter plot. The best fit line parameters then corresponds to the point of highest intersection on the parameter plot. Figure 27 shows the fitting of a line using the line equation descriptor

$$r_i(\theta) = x_i \cos(\theta) + y_i \sin(\theta) \qquad [3]$$

and an accumulator array.

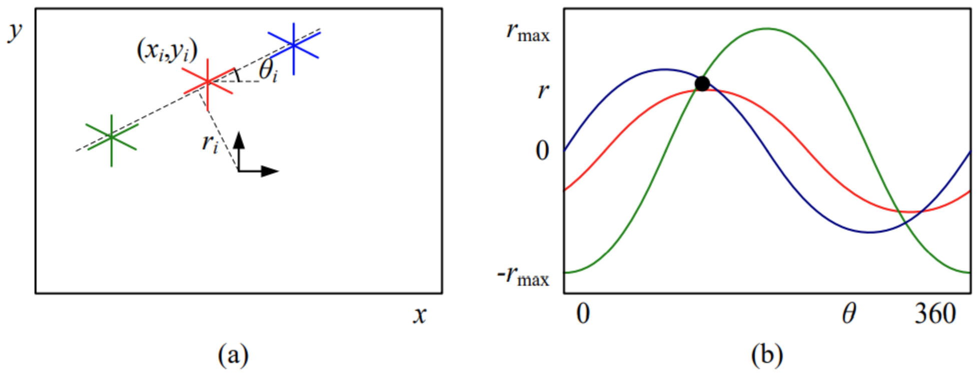


Figure 27: Hough Transform: line boundary detection graphical voting representation. Note the intersection of the points corresponds to the best fit line [3].

This method is expanded to detect all straight lines in an image (Figure 28) and all circles of a certain radius Figure 29.

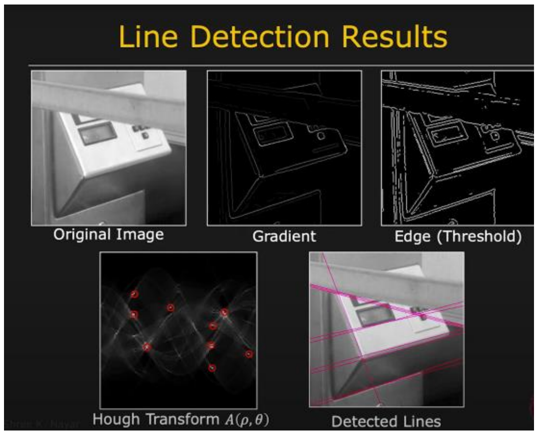


Figure 28: Hough transform for line detection [8]

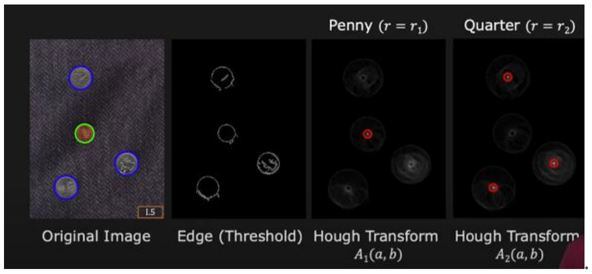


Figure 29: Hough transform for circle detection [8]

A generalized Hough transform was developed by D. H. Ballard [9]. This method uses a chosen shape reference points and gradient directional information. The edges are iteratively selected and plotted which creates intersection points (maxima) that "correspond to the possible instances of the shape S" [9].

3.2.2 RANSAC

RANSAC uses hypothesis to test for lines and boundaries. For lines, it takes 2 point and draws a line between them, then finds the line with the most intersecting edge points.

This is a 'lighter weight' method than the Hough transform but can be less efficient [3].

4. Object Detection

We can now identify specific features in an image. Next, we must match unique feature to objects and identify them.

Following feature detection comes object recognition or detection. Object detection involves isolating and grouping specific pixels which relate to real world objects.

4.1 Challenges

Transformations, occlusion, and lighting are some of the biggest challenges when it comes to object detection. If an object is perfectly oriented on a white background, matching the object with a template becomes trivial. But in reality, objects appear in widely varying ways to the camera eye. It may be distorted by a geometric transformation (Figure 30), partially block, or receiving glare from a lighting source. Consequently, advance techniques are needed to find objects in real world scenarios.

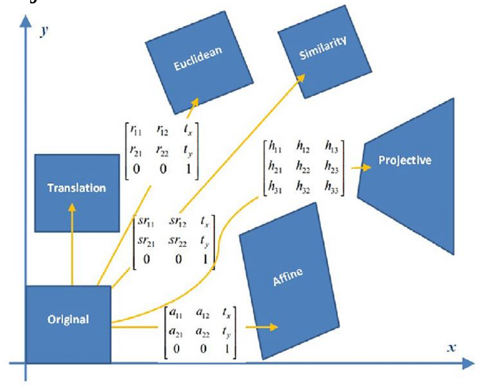


Figure 30: Type of image geometric transformations [10].

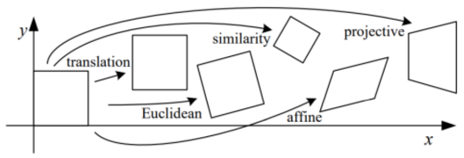
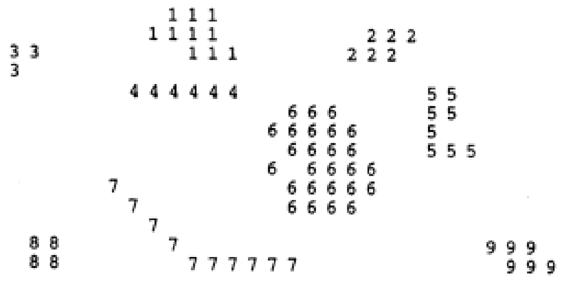


Figure 8.2 *Basic set of 2D planar transformations*

Figure 31: Basic set of 2D planar transformations [3].

4.2 Binary Shape Analysis

Binary shape analysis may be the simplest form of object detection. With a 2-dimension array of 1's and 0's, object detection in this scenario involves grouping 1's together. Algorithms for assigning these grouping are detail by E.R. Davies [2]. Computationally, these methods are inexpensive and precise. However, the cases in which binary images can be used for object detection are limited since many scenes cannot be subjected to thresholding without significant loss of data.



```
                 1 1 1
               1 1 1 1                  2 2 2
3 3                1 1 1              2 2 2
3
             4 4 4 4 4 4                        5 5
                             6 6 6              5 5
                           6 6 6 6 6            5
                             6 6 6 6            5 5 5
                           6   6 6 6 6
           7                 6 6 6 6 6
             7               6 6 6 6
               7
  8 8            7                                 9 9 9
  8 8              7 7 7 7 7 7                       9 9 9
```

Figure 32: Labeling object in a binary image [2].

4.3 SIFT Detector and Blob Detection

The historical methods of object detection rely on interest points (key points). These interest points are pixel combination that are (ideally) unique to the object. An object usually has many interest points.

Many sources have shown that corners and edges are not effective interest points (They do not appear frequently enough). Blob detection seems to have better results.
One popular object detection method which uses interest points is SIFT (scale invariant feature transform). SIFT gained popularity for its resistance to changes in scale and orientation. The SIFT transform has two parts, the detector and descriptor.

The SIFT ([11]) detector uses the difference of gaussian as an approximation of the Laplacian of gaussian. The LoG can detect interest points based the maximum rate of change in the area of interest; however, the process is slow. SIFT speeds up the process by using the DoG approximation. The DoG is calculated for several octaves of scale (
Figure 33). Each DoG is search for local maximums which present a potential interest point. Each scale is then compared to find the best scale of each interest point, leaving a list of the 'best' interest points in the image.

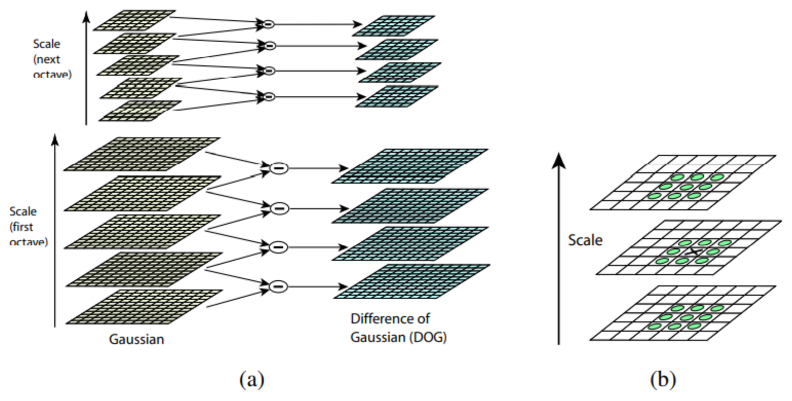


Figure 33: The SIFT detector using DoG to find interest points [12].

The key points are then made orientation invariant by taking a weighted gradient within a window around each key point. The direction of the gradients in the window are used to find the principal directions. This produces a descriptor which is used for key point matching.

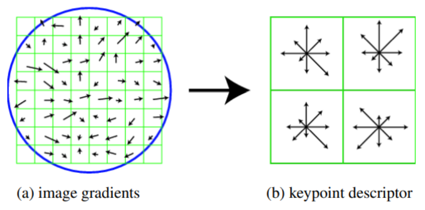


Figure 34: SIFT Key point orientation descriptor [3].

SIFT is defined for 2D [11]; thus, it is less effective at identifying 3D objects especially when the object perspective changes drastically.

4.4 Segmentation

Segmentation is the process of grouping pixel into 2d representation of an object.

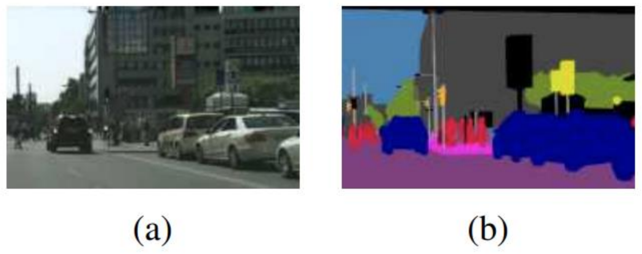


Figure 35: Examples of semantic segmentation [3].

4.4.1 Mean Shift

The mean shift algorithm segments objects based on a clustering technique. Using the plot (e) in Figure 36, the mean shift algorithm can be summarized

For each pixel,

- Let the pixel climb the hill/cluster with the steepest gradient in its region.
- The highest point of each hill is the mode of the cluster
- Assign the pixel to the cluster where it reached the mode.

In other words, assign each pixel to the most prominent (steepest) mode peak in its vicinity.

The pixel climbs the hill by iteratively computing the mean or centroid within a window *W* of the pixel then moving the pixel to that centroid.

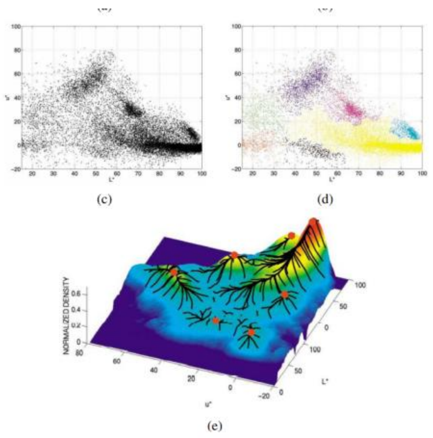


Figure 36: Graphs detailing the use of mean shift. Plots are in L^*, u^*, v^* space [3].

The advantages of this method are,

- The number of segments is not needed

- Simple algorithm
- No initialization
- Handles outliers well

The disadvantages are
- Heavily dependent on window size
- Computationally expensive

Sources: [13], [3]

4.5 Feature Matching

Feature matching is the process of defining an image based on its feature through use of a data base.

In the example of SIFT object detection, key points from the target image are compared with key points from data base images. The match key points are linked to objects then ranked in order of probability.

Convolutional Neural networks preform the same task with a different method. With CNN, features are extracted then labeled and ranked in the same process (See 4.6).

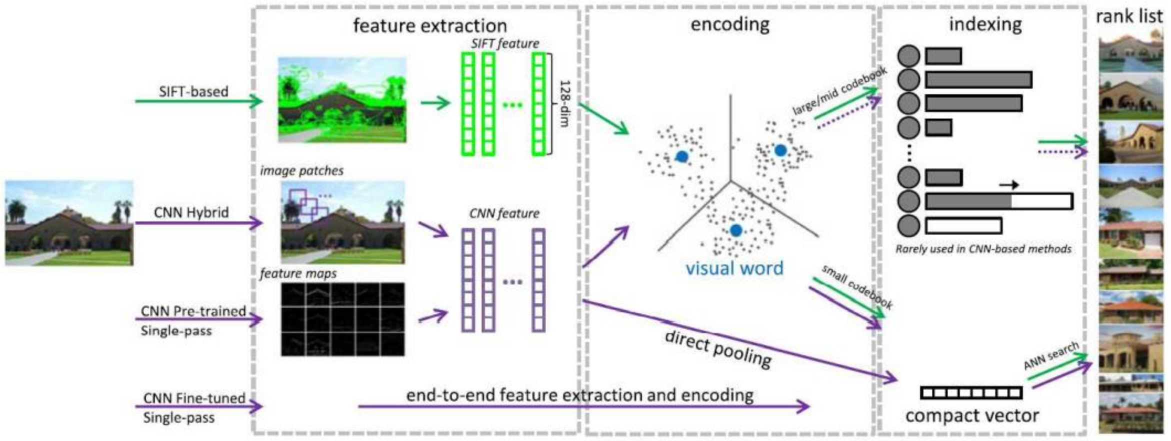


Figure 37: Methods of object detection with feature matching [3].

Types of features vary widely. They can range from image histograms, directional gradients (HoGs), edge filtered images, or the raw image itself. The type of features fed into object detection algorithms and methods depends highly on the method and application.

4.6 Convolutional Neural Networks

Convolutional Neural Networks (CNN) have become the modern object detector today [3]. There are many sub methods of this technique including R-CNN, Fast R-CNN, and GoogLeNet. This type of computing is considered deep learning due to the layers of computation involved.

In simple terms CNN use a network of weighted vectors to classify features then select the most probable classification.

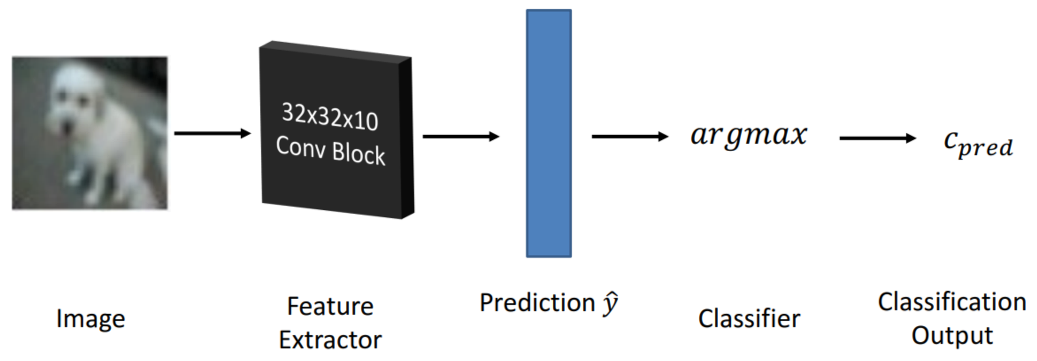


Figure 38: Simplification of CNN in CV [1].

The feature extraction is done through a type of convolution with a trained weighted vector. Several libraries such as YOLO have developed training algorithms and pretrained weight vectors. Every weight vector is convolved with the image to produce a probability of the object being of that type. The highest probability is then selected as the output.
The simplest classification takes the linear form

$$\boldsymbol{f} = \boldsymbol{Wx} + \boldsymbol{b}$$

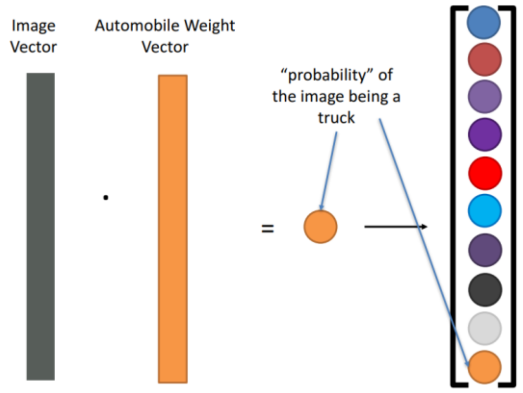


Figure 39: Dot product simplified convolution of an image with a weighted predictor [1] .

with vector x as the feature input and f as the classification. To create a second layer in the network, a second weight vector multiplied with the first probability. Figure 40 shows a graphical depiction of a 2-layer neural network. The 'fuzzy' image below correspond to the classification weight vectors. These essentially depict what a dog, plane, car, ect. look like to a computer.

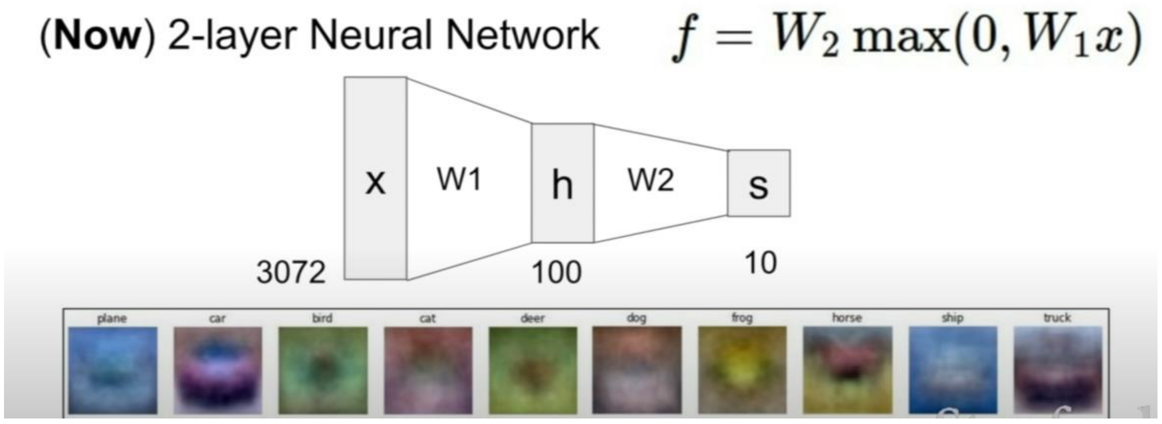


Figure 40: Visualized weight vectors from a 2-layer CNN [14] .

When this idea is expanded into a more advanced CNN, many layers of convolution and prediction form. Each layer represents a classification step towards the end goal of determining the object.

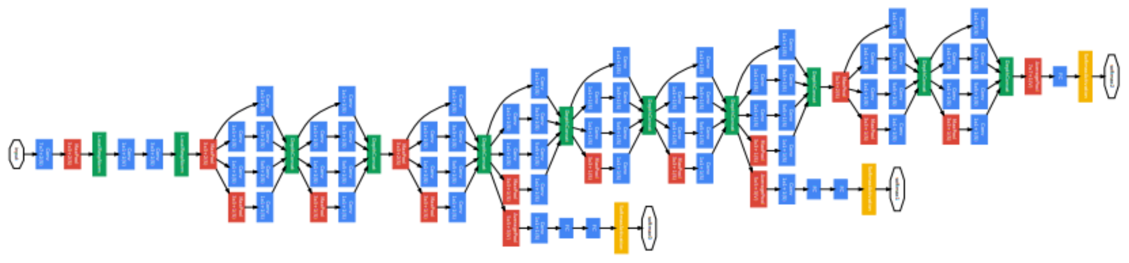


Figure 41: GoogLeNet CNN for object identification [15].

CNN have proved exceptionally fast at object detection and tracking. Additionally, given trained model, they are very easy to implement and use.

4.6.1 Creating the Weight Vector

Creating the weight vectors for a CNN is a standard supervised machine learning process. Labeled training images are given to a minimization algorithm such the gradient descent, which creates a vector which minimizes the error or loss of the model. These models are subsequently tested on validation sets. In more advance models the models are created several times with different parameters based on the validation testing [14].

4.6.2 Training with YOLOV5

While pretrained data sets such as yolo-coco are easy to implement, custom trained data sets can be more effective given a sufficient training data. YOLOv5 was developed in PyTorch and can be trained in online environments such as Googles Collab. This free software allows users to easily train and deploy their own models using YOLO's algorithm. An example of this training process can be found in Appendix A

When YOLOv5 was released, it was benchmarked as the fastest object recognition software on the COCO data set.

4.7 Other Object Detection Models

Most successful object detection models today are AI based. While CNN are the most popular now, alternatives such as RNN and clustering exist [3].

5. Object Tracking

We now can identify object in an image. How can we follow them in a video feed?

This section offers a brief introduction to object tracking. More coverage on object tracking will be provided in future reports.

Object tracking in CV is the practice of following an object (with a bounding box) through the frames of a video. This involve comparing, correlating, and predicting the object's behavior and view in different frames. In the case of AI models, it also can involve tuning the tracking model.

There are two types of tracking: Online and Offline. Online is suitable for real-time application and only uses a small number of current a previous frame to track the object. Offline tracking is suitable for video analysis because it uses batches of frames (including future frames) to track the object [16].

Tracking can also be subdivided into single and multi-target tracking. For this report we will cover single target tracking.

5.1 Single Target Tracking

Single target tracking has three steps:

- Initializing: Selecting the target for the tracker
- Predicting: Estimating where the object will be in the next frame based on previous frames
- Correcting: Detect the object in the next frame and compare that with your prediction to improve the model and continue tracking the object.

5.1.1 GOTURN Tracking

GOTURN is a fast, popular, deep learning-based tracking algorithm. GOTURN is a fixed tracking neural network because it does not change its convolution weights in real-time. While this make the model one of the fastest trackers, it is prone to failure, wandering, or losing a tracker because it does not update its model to finetune the tracker [16].

GOTURN considers two frames when tracking: the current and previous. To predict the next frame, GOTURN creates a search region for both the Current and Previous frames (based on the current frame), then preforms a series of separate and combined convolutions to predict the target in the next frame.

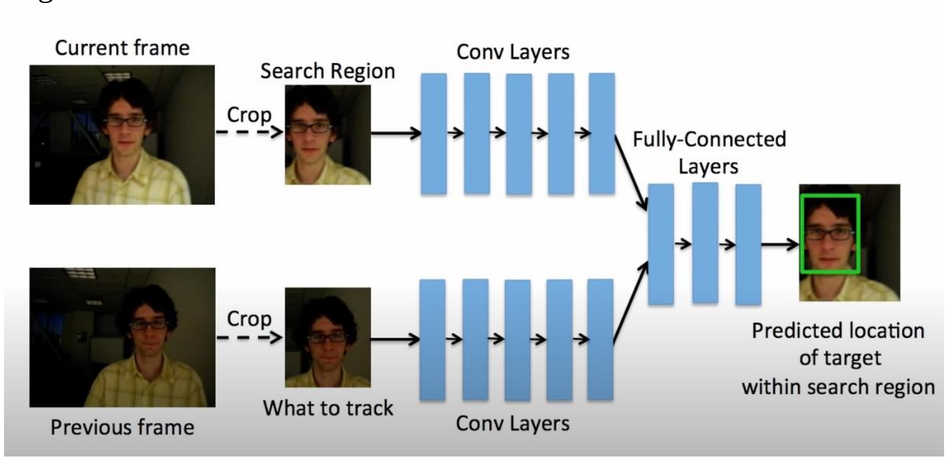


Figure 42: Architecture of GOTURN [17].

GOTURN implemented with Caffe. An example of using GOTURN can be found in Appendix A.

6. General Discussion and Observations

“A picture is worth a thousand word; but what words?” (Unknown). This statement cleanly sums up the major problem of computer vision. A picture contains an immense amount of data, but that data is useless without something, like the human brain, to analyze it.

6.1 Limitations of Computer Vision Techniques

We have discussed many methods of altering an image. These include filters, edge detectors, and noise suppression techniques such as the median filter. All these techniques have one attribute in common: they remove information. In fact, many of the techniques discussed in this report remove information in such a way as to leave only the *useful* information as defined by the algorithm. This leads to the conclusion that not all pixels are equal; some image information is more valuable than other.

We continue to see this procedure of isolating image data throughout feature detection. Edge detectors attenuate planes and homogenous sections of the image to enhance edges. Hough transforms scan images for only certain shapes. SIFT isolates only high interest points for comparison base on the Laplacian of the image. All of these techniques seek to find patterns in simpler set of data.

While these techniques have shown to be effective, they can be limited in their application because they, essentially, are a form of reductionism. They break the images down into fundamental bits to explain the whole. However, reductionism has a fundamental flaw: it operates well in *complicated systems* (i.e. a car engine, where all relationship are static and defined; but fails to fully explain a *complex system* (i.e. such as a biological system where relationship change continuously).

The movement towards AI centered CV may be an effort to advance from this methodology. Models such as CNN's, apply weights to large features, segmented pixels, and images to ask questions and receive probabilistic answers.

Where something like SIFT asks,
"Does the key points in this image match enough key points on this image of a dog?"
CNN's asks successive question,
"Is this an animal? -> Does it have four legs? -> Is this a dog?".

The downside to an AI approach is the massive amount of quality training data needed to make it effective. A data set such as COCO contains over 330K images most of which have been labeled [18]. This places an incredible amount of front-loaded work on AI models.

6.2 Applications in Virtual Servo

As I have yet to cover Virtual Servo methods, this section contains speculation on the potential uses of the techniques discussed so far.

Perhaps the most directly applicable method for Virtual Servoing may be real time object tracking. If the velocity and position of an object can be determined, a control function may be used to respond to the object.
Object detection could also be used in real-time control to indicate static states of a machine. Additionally, object tracking is directly dependent on object detection for its initialization (i.e. finding the correct object to track).

Image morphology and feature detectors could also prove beneficial by increasing the speed at which object tracking and detection are performed. For example, if only edges or circles are needed in the application, using the Sobel edge detector or Hough transform may increase the accuracy and speed of other methods to track or locate said objects by preprocessing the image. Filters may also be useful in reducing noise or lighting issues from the camera setup. If an image could be binarized without loss of essential data, the performance of any CV operation would increase immensely.

7. Conclusion

The objective of this report was to produce a summary of Computer Vision techniques for use in Visual Servo. Although an in-depth mathematical or algorithmic analysis was omitted, the information covered should suffice for use of the techniques in pre-built libraries such as OpenCV.

This report initially goes over the definition and challenges of CV; Image analysis was named as the most difficult problem face by computer vision. Next, the basics of digital images and image operation were discussed including methods of defining and image and the discrete convolution operator. Methods and techniques in image morphology were covered including Gaussian Filter, Median Filter, and Opening and Closing of images. A sample of feature detectors such as the Canny edge detector and Hough transforms were given. The effects of the gradient and laplacian operators were described in the context of edge and boundary formation. Building on these techniques, we examined several object detection methods including the SIFT detector and Mean Shift Segmentation. Further researching object detection, we analyzed the properties and advantages of CNN's and other deep learning computer methods. In addition to this topic, the training and creation of the weight vectors involved in neural networks were discussed. Finally, a brief overview of object tracking and the GOTURN tracker was provided.

In future research, object tracking should be covered in more depth. Additionally, methods in which finite object data such as position and velocity could be extracted from digital images should be studied. The opportunities of stereo vision could also prove valuable.

References

[1] J. C. Niebles and R. Krishna, "CS131 Computer Vision: Foundations and Applications," Fall 2018. [Online]. Available: http://vision.stanford.edu/teaching/cs131_fall1819/syllabus.html. [Accessed 18 May 2022].

[2] E. R. Davies, Computer Vision: Principles, Algorithms, Applications, Learning, 5th ed., 2018.

[3] R. Szeliski, Computer Vision:, 2nd ed., Springer, 2021.

[4] M. N. Favorskaya and L. C. Jain, Computer Vision in Control Systems-1: Mathematical Theory, 2015.

[5] University of Colorado, "Image Analysis," University of Colorado Boulder, [Online]. Available: https://canvas.colorado.edu/courses/61439/assignments/syllabus. [Accessed 18 May 2022].

[6] J. Canny, "A Computational Approach to Edge Detection," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,*, Vols. PAMI-8, no. 6, 1986.

[7] Columbia University, "Canny Edge Detector | Edge Detection," 2 March 2021. [Online]. Available: https://www.youtube.com/watch?v=hUC1uoigH6s&t=146s. [Accessed 18 May 2022].

[8] Columbia University, "Hough Transforms | Boundary Detection," 2 March 2021. [Online]. Available: https://www.youtube.com/watch?v=XRBc_xkZREg. [Accessed 20 May 2022].

[9] D. Ballard, "GENERALIZING THE HOUGH TRANSFORM TO DETECT ARBITRARY SHAPES," *Pattern Recognition,* vol. 13, no. 2, pp. 111-122, 1981.

[10] S. A and M. Soycan, "Perspective correction of building facade images for architectural applications," Engineering Science and Technology, an International Journal., 2018.

[11] G. D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision,* 2004.

[12] OpenCV, "OpenCV," doxygen, 25 May 2-22. [Online]. Available: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html. [Accessed 25 May 2022].

[13] Columbia University, "Mean-Shift Segmentation | Image Segmentation," 25 May 2021. [Online]. Available: https://www.youtube.com/watch?v=PCNz_zttmtA. [Accessed 24 May 2022].

[14] Stanford University School of Engineering, "Lecture 4 | Introduction to Neural Networks," Youtube, 11 Aug 2017. [Online]. Available: https://www.youtube.com/watch?v=d14TUNcbn1k. [Accessed 27 May 2022].

[15] C. Szegedy, L. W and e. al, "Going deeper with convolutions," 17 Sep 2014. [Online]. Available: https://arxiv.org/pdf/1409.4842v1.pdf. [Accessed 25 May 2022].

[16] Dynamic Vision and Learning Group, "CV3DST - Object tracking," Youtube, 17 Jul 2020. [Online]. Available: https://www.youtube.com/watch?v=QtAYgtBnhws. [Accessed 27 May 2022].

[17] R. Fergus, "Computer Vision -CSCI-GA.2271-001," New York University, 2021. [Online]. Available: https://cs.nyu.edu/~fergus/teaching/vision/index.html. [Accessed 18 May 2022].

[18] COCO, "Coco Data Set," [Online]. Available: https://cocodataset.org/#home. [Accessed 27 May 2022].

[19] OpenCV, "OpenCV," 26 May 2022. [Online]. Available: https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html.

[20] OpenCV, "github," 19 Oct 2021. [Online]. Available: https://github.com/opencv/opencv/blob/4.x/samples/dnn/object_detection.cpp. [Accessed 26 May 2022].

[21] J. Bigun, Vision with Direction: A Systematic Introduction to Image Processing and Computer Vision, 2006.

[22] LearnOpenCV, "GOTURN : Deep Learning based Object Tracker," Youtube, 22 Jul 2018. [Online]. Available: https://www.youtube.com/watch?v=SygkiWNSkWk. [Accessed 27 May 2022].

APPENDIX A: LAB 1

Objective

The objective of this laboratory is to implement standard computer vision techniques using the OpenCV library in C++.

The following tasks are implemented

Basic Image Morphology
- Display an image
- Gray scale the image
- Binarize the image
- Filter the image with several kernels (Gaussian, Median …)

Feature Detection
- Edge detection algorithms
- Hough Transforms

Object Detection
- SIFT Transforms
- CNN
- YOLOV5 Model Training

Object Tracking
- GO TURN

Apparatus

The set up for the experiment includes a desktop computer and sample images. The desktop computer uses
- VS Code environment
- CMake
- gcc compiler
- OpenCV

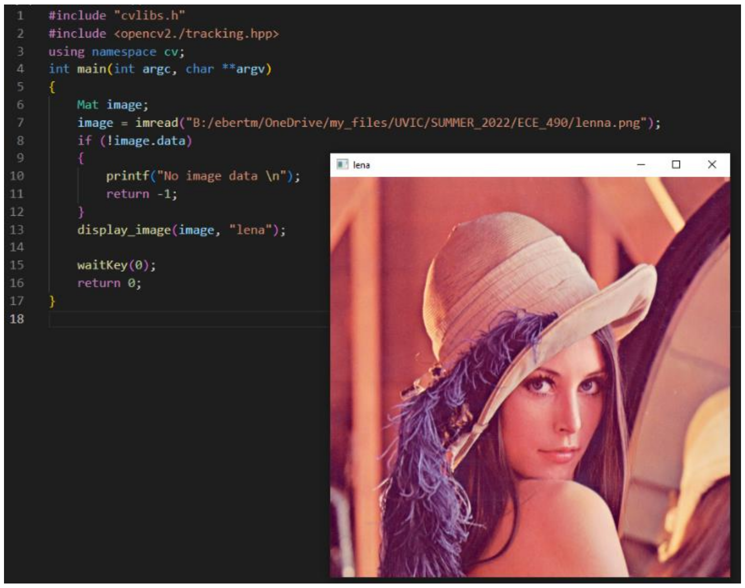
Procedure and Results

The following section outlines both the procedure and the results from this laboratory

Image Morphology

An environment and complier tools were set up within VS Code. The sample image ‘lena.png’ was placed in the main directory for use.

To begin, the OpenCV library was used to import and display an image as a MAT type.



```cpp
#include "cvlibs.h"
#include <opencv2./tracking.hpp>
using namespace cv;
int main(int argc, char **argv)
{
    Mat image;
    image = imread("B:/ebertm/OneDrive/my_files/UVIC/SUMMER_2022/ECE_490/lenna.png");
    if (!image.data)
    {
        printf("No image data \n");
        return -1;
    }
    display_image(image, "lena");

    waitKey(0);
    return 0;
}

```

Figure 43: Displaying a local image with OpenCV

This image when then subjected to a variety of morphological alteration using the OpenCV library functions.

First the image was gray scaled using the *cv::ctvColor()* function.

```
Mat imgray(image.size(), CV_8U);
cvtColor(image, imgray, COLOR_BGR2GRAY);
display_image(imgray, "Gray Scale");
```

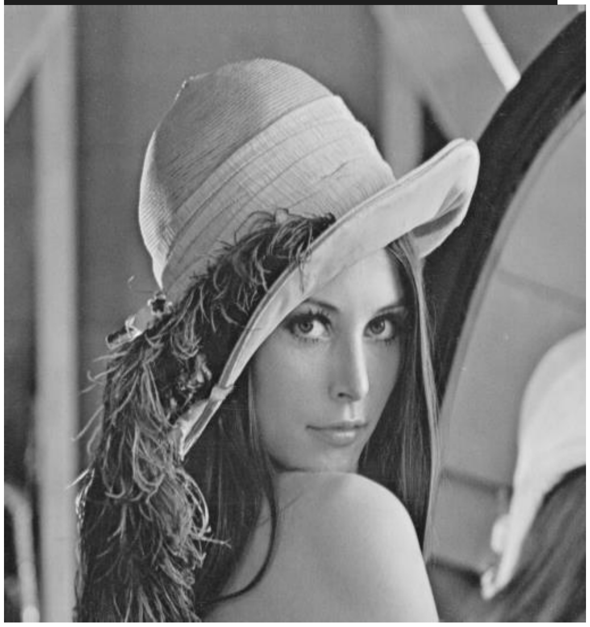
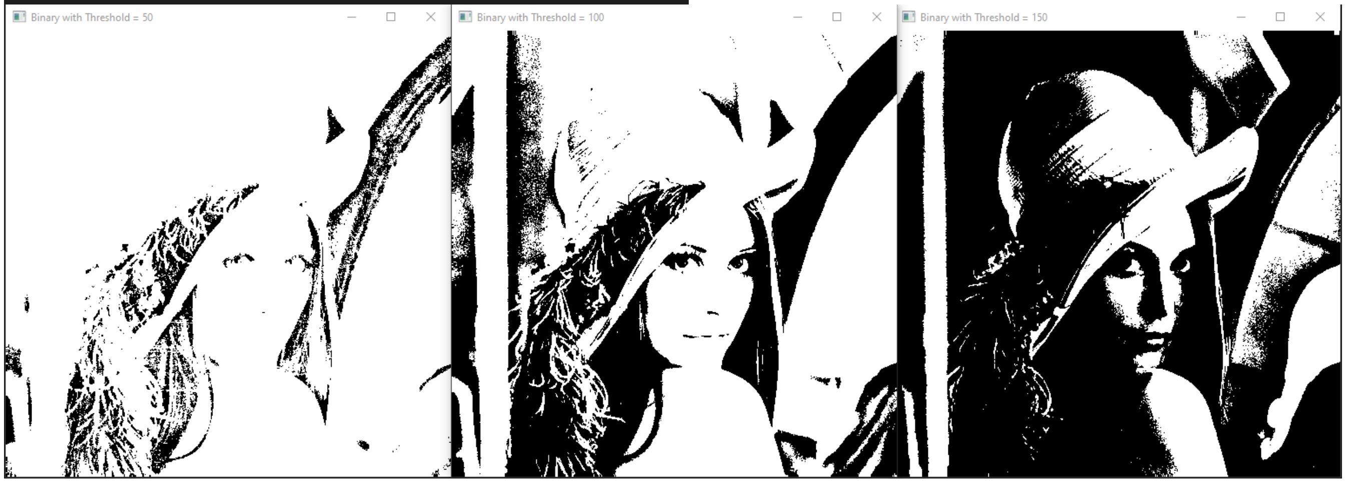


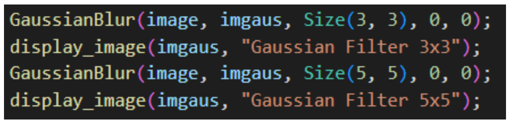
Figure 44: Lena gray scaled with OpenCV

From the gray scale image, several binary images were produces with the cv::*threshold()* function using different threshold values.

```
Mat imbinary(imgray.size(), imgray.type());
threshold(imgray, imbinary, 50, 255, THRESH_BINARY);
display_image(imbinary, "Binary with Threshold = 50");

threshold(imgray, imbinary, 100, 255, THRESH_BINARY);
display_image(imbinary, "Binary with Threshold = 100");

threshold(imgray, imbinary, 150, 255, THRESH_BINARY);
display_image(imbinary, "Binary with Threshold = 150");
```


Binary with Threshold = 50
Binary with Threshold = 100
Binary with Threshold = 150
Figure 45: Lena binarized with OpenCV

A gaussian blur was applied to the original image with a 3x3 and 5x5 kernel.



```
GaussianBlur(image, imgaus, Size(3, 3), 0, 0);
display_image(imgaus, "Gaussian Filter 3x3");
GaussianBlur(image, imgaus, Size(5, 5), 0, 0);
display_image(imgaus, "Gaussian Filter 5x5");
```

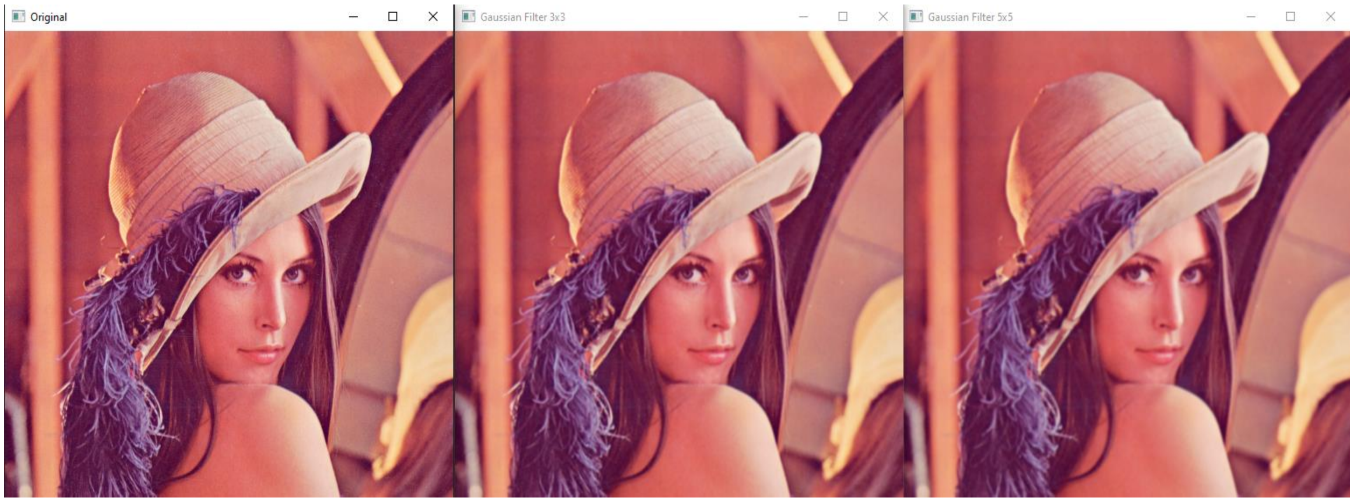
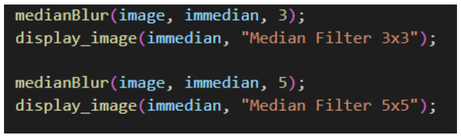


Figure 46: Lena with different Gaussian blurs applied with OpenCV.

The same procedure was done with a median filter using the *medianBlur()* function.



```
medianBlur(image, immedian, 3);
display_image(immedian, "Median Filter 3x3");

medianBlur(image, immedian, 5);
display_image(immedian, "Median Filter 5x5");
```

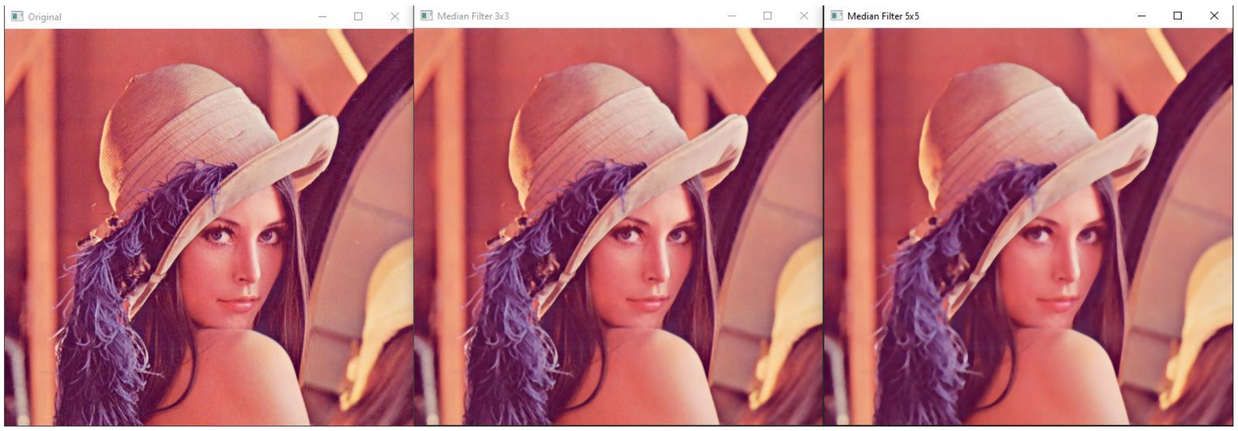
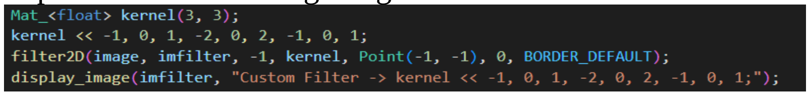


Figure 47: Figure 38: Lena with different median blurs applied with OpenCV.

Next a custom filter was applied to the original image with the kernel

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

to produce the following image



```
Mat_<float> kernel(3, 3);
kernel << -1, 0, 1, -2, 0, 2, -1, 0, 1;
filter2D(image, imfilter, -1, kernel, Point(-1, -1), 0, BORDER_DEFAULT);
display_image(imfilter, "Custom Filter -> kernel << -1, 0, 1, -2, 0, 2, -1, 0, 1;");
```

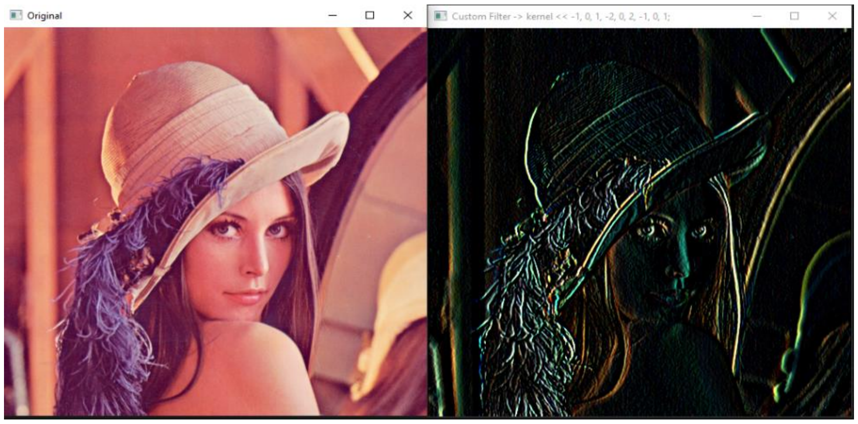


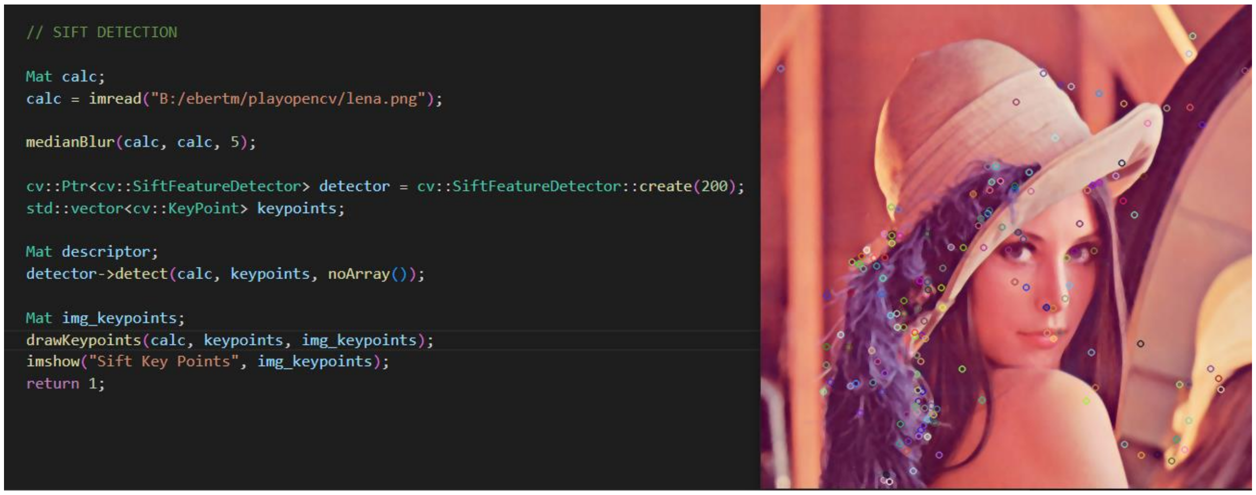
Figure 48: Lena with a custom filter applied with OpenCV

The kernel emphasizes the edges of the image. It is in fact the Sobel Edge detector.

Feature Detection

Using the same code environment, functions were create to detect image features.

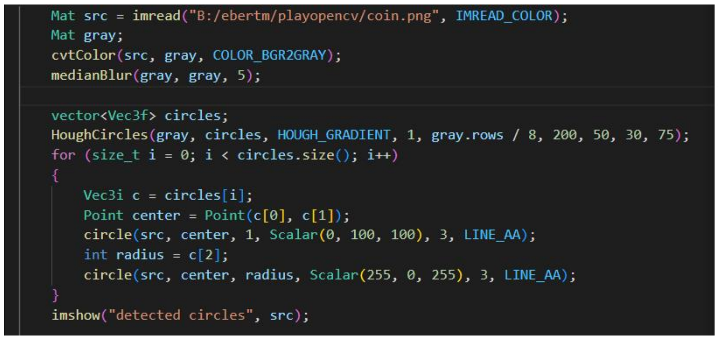
The SIFT detector was employed on the lena.png. The detector was set to detect the 200 highest interest points in the image. The results are shown in the image below. Notice the key points appear around high gradient areas.



```
// SIFT DETECTION

Mat calc;
calc = imread("B:/ebertm/playopencv/lena.png");

medianBlur(calc, calc, 5);

cv::Ptr<cv::SiftFeatureDetector> detector = cv::SiftFeatureDetector::create(200);
std::vector<cv::KeyPoint> keypoints;

Mat descriptor;
detector->detect(calc, keypoints, noArray());

Mat img_keypoints;
drawKeypoints(calc, keypoints, img_keypoints);
imshow("Sift Key Points", img_keypoints);
return 1;
```

Figure 49: SIFT detector used to find key points.

Next, the Hough transform was used to detect circles in an image. The created function makes use of the *HoughCircles()* function. The parameters of the transform were set to detect circles between 30 and 75 pixels in radius with a minimum gap of 50 pixels between circle centers. An image of coins was fed into the function. The results are shown in pink below.



```
    Mat src = imread("B:/ebertm/playopencv/coin.png", IMREAD_COLOR);
    Mat gray;
    cvtColor(src, gray, COLOR_BGR2GRAY);
    medianBlur(gray, gray, 5);

    vector<Vec3f> circles;
    HoughCircles(gray, circles, HOUGH_GRADIENT, 1, gray.rows / 8, 200, 50, 30, 75);
    for (size_t i = 0; i < circles.size(); i++)
    {
        Vec3i c = circles[i];
        Point center = Point(c[0], c[1]);
        circle(src, center, 1, Scalar(0, 100, 100), 3, LINE_AA);
        int radius = c[2];
        circle(src, center, radius, Scalar(255, 0, 255), 3, LINE_AA);
    }
    imshow("detected circles", src);
```

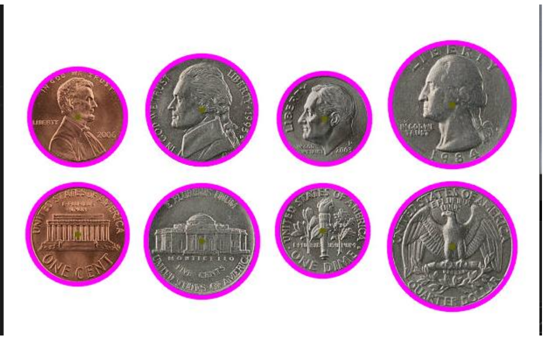
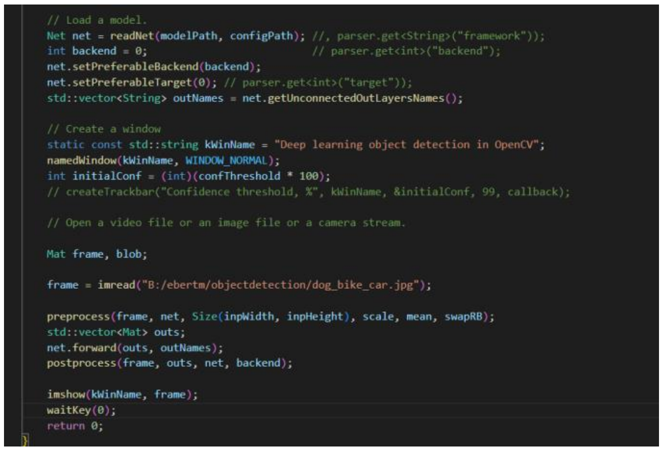


Figure 50: A Hough transform implementation to detect circles. Code modified from [19] .

Object Detection

For the final part of the experiment, a program was run to detect common objects using CNN and the YOLO algorithm. The implemented code was modified from the OpenCV tutorial library [20]. A standard image of a dog bike and car was given as input to the function. The function returned the image below showing the objects detected in green square with the confidence written above.



```
    // Load a model.
    Net net = readNet(modelPath, configPath); //, parser.get<String>("framework"));
    int backend = 0;                          // parser.get<int>("backend");
    net.setPreferableBackend(backend);
    net.setPreferableTarget(0); // parser.get<int>("target"));
    std::vector<String> outNames = net.getUnconnectedOutLayersNames();

    // Create a window
    static const std::string kWinName = "Deep learning object detection in OpenCV";
    namedWindow(kWinName, WINDOW_NORMAL);
    int initialConf = (int)(confThreshold * 100);
    // createTrackbar("Confidence threshold, %", kWinName, &initialConf, 99, callback);

    // Open a video file or an image file or a camera stream.

    Mat frame, blob;

    frame = imread("B:/ebertm/objectdetection/dog_bike_car.jpg");

    preprocess(frame, net, Size(inpWidth, inpHeight), scale, mean, swapRB);
    std::vector<Mat> outs;
    net.forward(outs, outNames);
    postprocess(frame, outs, net, backend);

    imshow(kWinName, frame);
    waitKey(0);
    return 0;
}
```

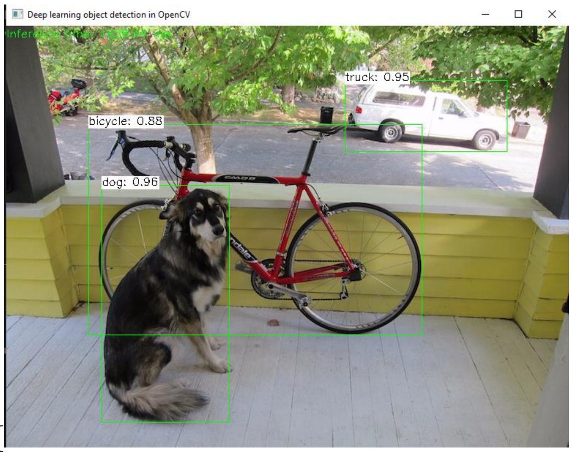


Figure 51: Implementation of object detection with OpenCV and Yolo. Code modified from [20].

Training a YOLOV5 Model.

Using YOLOv5, Google Collaborator, and roboflow, a detection CNN was trained with 12 images. These images contained a water bottle and thermos as seen below.

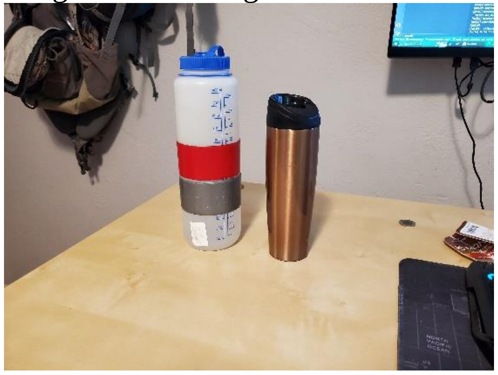


Figure 52: Object to train the custom CNN on.

12 images of varying orientation, lighting, and background we taken and labeled using the online software Roboflow.

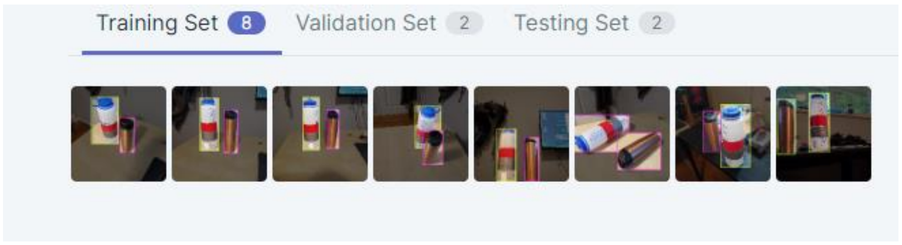
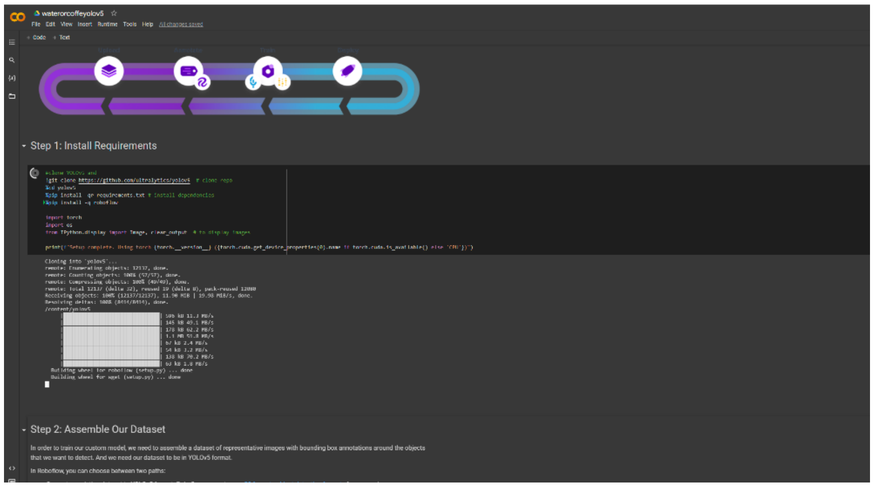


Figure 53: Labeled training data for the yolov5 model.

This data was formatted and imported into Google Collaborator, which gives free access to 3 GPUs which accelerates training speed.



Step 1: Install Requirements

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 12137, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 12137 (delta 32), reused 19 (delta 8), pack-reused 12080
Receiving objects: 100% (12137/12137), 11.90 MiB | 19.98 MiB/s, done.
Resolving deltas: 100% (8414/8414), done.
/content/yolov5
     |████████████████████████████████| 596 kB 11.3 MB/s
     |████████████████████████████████| 145 kB 49.1 MB/s
     |████████████████████████████████| 178 kB 62.2 MB/s
     |████████████████████████████████| 1.1 MB 51.8 MB/s
     |████████████████████████████████| 67 kB 2.4 MB/s
     |████████████████████████████████| 54 kB 3.2 MB/s
     |████████████████████████████████| 138 kB 70.2 MB/s
     |████████████████████████████████| 63 kB 1.8 MB/s
  Building wheel for roboflow (setup.py) ... done
  Building wheel for wget (setup.py) ... done
```

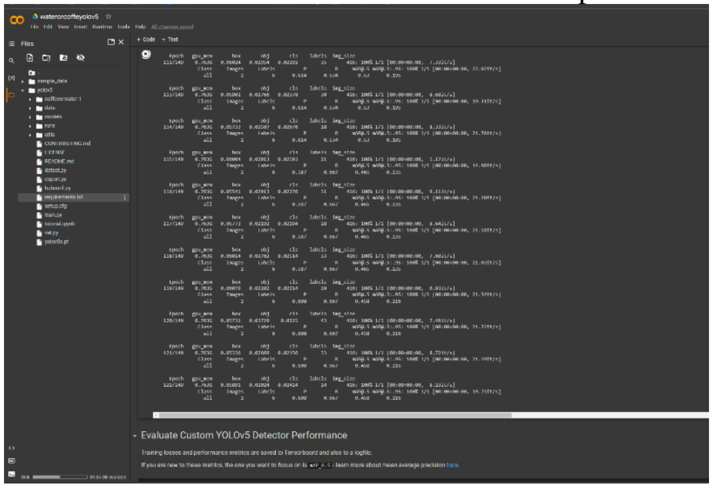
Step 2: Assemble Our Dataset

In order to train our custom model, we need to assemble a dataset of representative images with bounding box annotations around the objects that we want to detect. And we need our dataset to be in YOLOv5 format.

In Roboflow, you can choose between two paths:

Figure 54: Importing training data into google collaborator.

The model was then trained with 8 of the 12 picture and validated with 2.



```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
   112/149    0.763G   0.06024   0.02554   0.02292        35       416: 100% 1/1 [00:00<00:00,  7.33it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 22.02it/s]
                 all          2          6      0.614      0.554       0.53      0.195

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   113/149    0.763G   0.05902   0.02766   0.02378        30       416: 100% 1/1 [00:00<00:00,  8.60it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 19.31it/s]
                 all          2          6      0.614      0.554       0.53      0.195

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   114/149    0.763G   0.05733   0.02507   0.02676        28       416: 100% 1/1 [00:00<00:00,  8.33it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.76it/s]
                 all          2          6      0.614      0.554       0.53      0.195

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   115/149    0.763G   0.06064   0.02013   0.02503        21       416: 100% 1/1 [00:00<00:00,  5.17it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 14.98it/s]
                 all          2          6      0.587      0.667      0.465      0.235

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   116/149    0.763G   0.05541   0.02913   0.02276        31       416: 100% 1/1 [00:00<00:00,  9.11it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.20it/s]
                 all          2          6      0.587      0.667      0.465      0.235

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   117/149    0.763G   0.05773   0.02192   0.02194        28       416: 100% 1/1 [00:00<00:00,  8.64it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.18it/s]
                 all          2          6      0.587      0.667      0.465      0.235

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   118/149    0.763G   0.06014   0.02762   0.02514        33       416: 100% 1/1 [00:00<00:00,  7.60it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.03it/s]
                 all          2          6      0.587      0.667      0.465      0.235

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   119/149    0.763G   0.06079   0.02202   0.02214        29       416: 100% 1/1 [00:00<00:00,  6.93it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.54it/s]
                 all          2          6      0.608      0.667      0.458      0.216

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   120/149    0.763G   0.05731   0.01728    0.0225        43       416: 100% 1/1 [00:00<00:00,  7.46it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.22it/s]
                 all          2          6      0.608      0.667      0.458      0.216

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   121/149    0.763G   0.05336   0.02669   0.02156        33       416: 100% 1/1 [00:00<00:00,  8.72it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 21.19it/s]
                 all          2          6      0.608      0.667      0.458      0.216

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   122/149    0.763G   0.05891   0.02024   0.02424        24       416: 100% 1/1 [00:00<00:00,  8.23it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 19.73it/s]
                 all          2          6      0.608      0.667      0.458      0.216
```

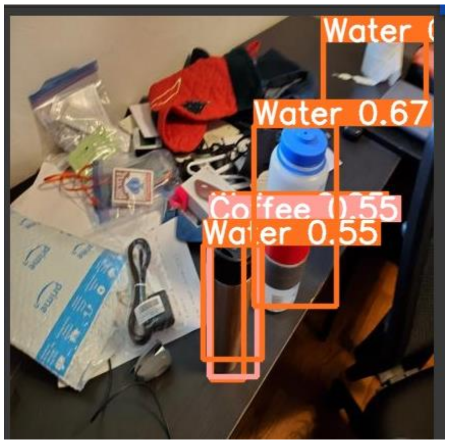
Evaluate Custom YOLOv5 Detector Performance

Training losses and performance metrics are saved to Tensorboard and also to a logfile.

If you are new to these metrics, the one you want to focus on is `mAP_0.5` - learn more about mean average precision here.

Figure 55: Training the model.

The best model was selected and tested on the last 2 photos to produce the following results.



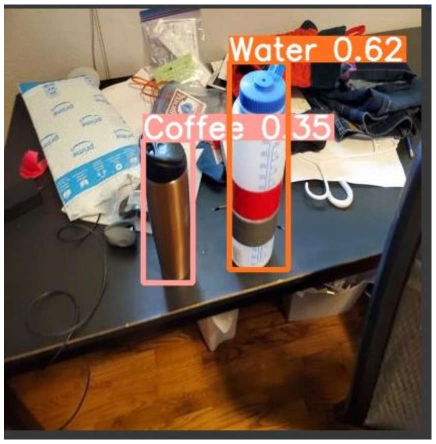


Figure 56: Testing the trained yolov5 model.

For a data set of 12 images, the confidence is remarkably high. However, as can be seen in the first image, several false positives were detected. More training data would drastically improve these results.

Object Tracking with GOTURN

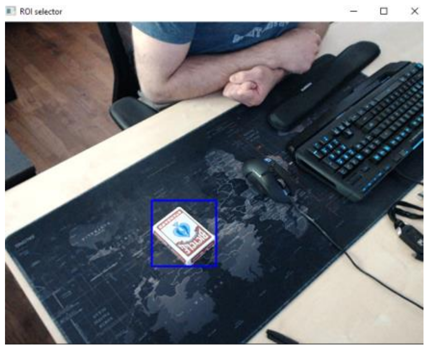
The tracking algorithm GOTURN was implemented in real time on a webcam.

A webcam with a default resolution of 1080p at 30fps was connected to the PC. The environment was then configured to run on a single core of a Ryzen 3900x CPU.

The following code was developed to run the GOTURN deep learning tracker with OpenCV on the local webcam in Realtime.

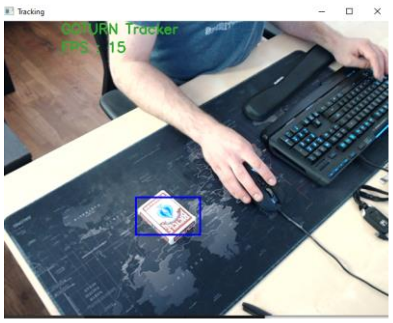
```
// Modified from code by Satya Mallick
// https://learnopencv.com/object-tracking-using-opencv-cpp-python/
//Edited by Matthew Ebert 2022-05-27
#include <opencv2/opencv.hpp>
#include <opencv2/tracking.hpp>
#include <opencv2/core/ocl.hpp>
using namespace cv;
using namespace std;

// Convert to string
#define SSTR(x) static_cast<std::ostringstream &>(           \
                  (std::ostringstream() << std::dec << x)) \
                  .str()

int main(int argc, char **argv)
{

    Ptr<Tracker> tracker;
    tracker = TrackerGOTURN::create();

    VideoCapture video(0);
    //  Exit if video is not opened
    if (!video.isOpened())
    {
        cout << "Could not read video file" << endl;
        return 1;
    }
    // Read first frame
    Mat frame;
    bool ok = video.read(frame);

    Rect bbox = selectROI(frame, false);

    rectangle(frame, bbox, Scalar(255, 0, 0), 2, 1);
    imshow("Tracking", frame);
    tracker->init(frame, bbox);

    while (video.read(frame))
    {
        // Start timer
        double timer = (double)getTickCount();

        // Update the tracking result
        bool ok = tracker->update(frame, bbox);

        // Calculate Frames per second (FPS)
        float fps = getTickFrequency() / ((double)getTickCount() - timer);

        if (ok)
        {
            // Tracking success : Draw the tracked object
            rectangle(frame, bbox, Scalar(255, 0, 0), 2, 1);
        }
        else
        {
            // Tracking failure detected.
            putText(frame, "Tracking failure detected", Point(100, 80), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 255), 2);
        }

        // Display tracker type on frame
        putText(frame, "GOTURN Tracker", Point(100, 20), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(50, 170, 50), 2);

        // Display FPS on frame
        putText(frame, "FPS : " + SSTR(int(fps)), Point(100, 50), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(50, 170, 50), 2);

        // Display frame.
        imshow("Tracking", frame);

        // Exit if ESC pressed.
        int k = waitKey(1);
        if (k == 27)
        {
            break;
        }
    }
}
```

At the beginning of the program, the user is prompted to select the region (object) of interest from a still image taken from the camera.

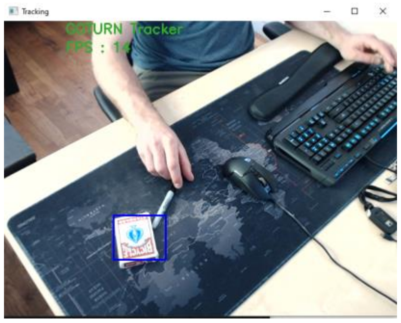


In this case, a deck of cards was used as the object of interest.

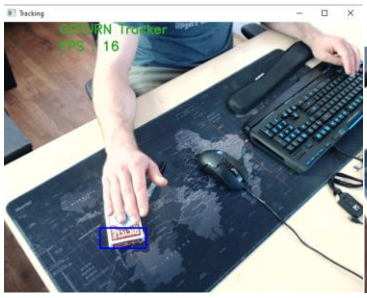
The GOTURN tracker was initiated and correctly held the bounding box around the desired object. The model managed to track as 15 FPS in a still image.

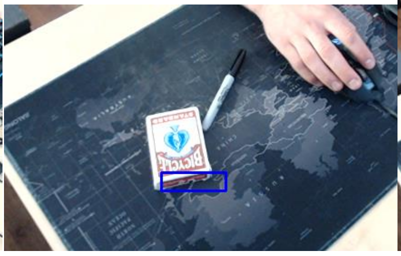


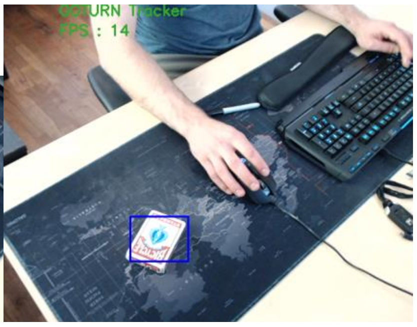
Next, the object was moved slightly back and forth without occlusion. The FPS dropped slightly, but the tracker continued to follow the correct object.



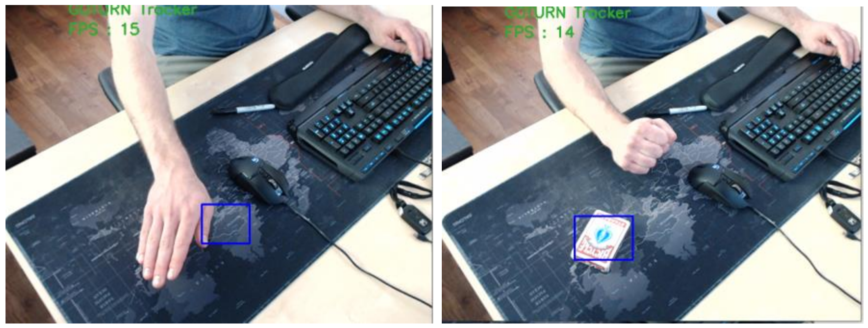
While the object was at rest, with was partially covered. The tracker quickly changed in size to contain only the visually remaining part of the object. When the obstruction was removed, after a brief delay, the tracker regained readjusted to contain the full object once again. The FPS remained the around 15 for the duration of this test.



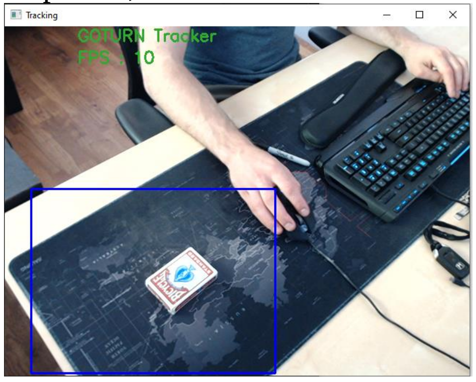




The object was fully obscured for a period of 5 seconds. The tracker wandered slightly away from the objects location in that time. However, when the obstuction was remove, the tracker recovered and found the object once again.



Finally, while fully obstructed, the object was moved from it's original location. During the time the object was ocluded and moving the tracker window began to widen. When the object reapeared, the tracker remained at the large size and failed to converge to the object.



The following performance metrics were taken from several parts of the lab

Function	**Run Time**	**Unit**
Gaussian Filter	0.015	s
Hough Transform	0.02	s
CNN object detection with Yolo	0.52	s
Object Tracking with GOTURN	15	FPS

Discussion

While initially complex to compile and setup in a PC environment, OpenCV proved very effective to use. Function and parameters are well documented by OpenCV and a huge range of function and techniques are available from the open source community. Image morphology was particularly easy to implement. Most effects can be accomplished through a single line of code.

Feature detection proved a bit more complicated. The run time of the Hough transform began to become significant with a run time of t = 0.011392s.

Creating the program for Object detection was many lines of code. The CNN code with YOLO included loading the models, preprocessing the image, applying the model, and post processing the image for display. With a single CPU processor, the run time of object detection was 0.5s. Again, very significant in terms of processing time. However, with a more processing hardware, this may speed up considerably.

In terms of training a custom data set, while the process was lengthy, the tutorials and online environments made to process simple. It also outlined the strength of YOLOv5. With only a few images, some images produced as much as a 60 percent confidence.

Using GOTURN for object tracking achieved quite impressive results. With only a single CPU, the tracker was able to follow a non-occluded object at 15fps. However, any occlusion or extremely fast changes in motion caused the tracker to fail. Other trackers are known to deal with this type of challenge by tuning updating and redetecting the target object.
It was also noticed that immediately after the tracker was initialized, it had a chance of wandering from the object if it was moving. This may be cause by the lack of previous frames when the tracker begins.

Conclusion

The objective of this laboratory was to achieve familiarity with the OpenCV library and CV applications. Image morphology was preformed on several benchmark images. SIFT and Hough feature detection was implemented successfully. Object detection, model training, and object tracking were also preformed successfully. It was noted that run time of function rose significantly with complexity.

Future experiment should seek to improve the performance of the tracking an object detection.