Network for Traffic Monitoring in a Smart City

Submitted June 2020, in partial fulfillment of
the conditions for the award of the degree **Bachelor of Computer Engineering at The University of Newcastle, Australia.**

Thomas Courtney

3175353

Supervised by Assoc. Prof. Kaushik Mahata

School of Engineering

University of Newcastle

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____ / _____ / _____

Contents

List of Tables

List of Figures

Chapter 1

Theoretical Background

This section of the report expands upon the theory behind the design of the system. Digital image processing and computer vision are broad fields and so only concepts relevant to the system are explored.

1.1 Digital Images

Digital images consist of discrete valued data points called pixels arranged in a grid where it is the value of these pixels that determine the appearance of each cell. An image may be coloured, grayscale or binary depending on the format of the pixels that it's comprised of. A coloured pixel is represented by a three dimensional vector containing values for red, green and blue light.

$$Y = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1.1}$$

The value range of the a coloured pixel's, Y, elements R, G and B is set by the image's encoding, the JPEG format for example, has 8-bit encoding thus the value range for each element is $[0, 255]$ and so 16 million unique colours can be expressed. A grayscale image

is comprised of single dimensioned pixels and is thus monochromatic being able to only represent the amount of light in an image but not its colour. A grayscale pixel's value is referred to as its *intensity*. For an 8-bit encoded grayscale image its highest value is 255 and expresses white and its lowest value is 0 expressing black, in between these values are 254 shades of gray (see Figure 1.1). A binary image is represented by only black and white pixels whose values are either 0 or 1 respectively [1]. In image processing the format is selected based on the information that's important, in many situations only grayscale information is required because colour doesn't add any important information to the image. It is important to recognize which image format to use because computations are more expensive for a three dimensional coloured image than a single dimensioned grayscale or binary image.

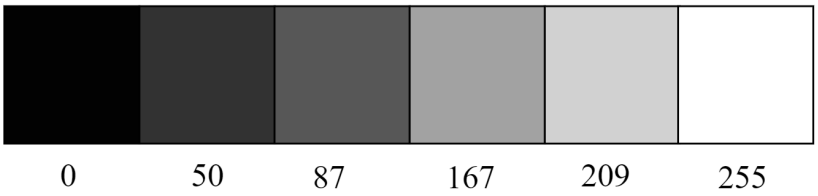
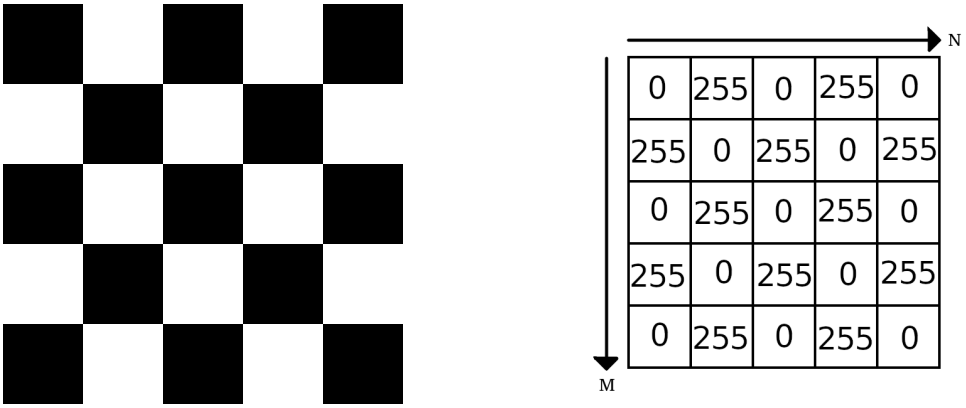


Figure 1.1: Grayscale value and colour range. Img: Learning Processing - Daniel Shiffman

In software digital images are represented as an array of values where each cell corresponds pixel in the image and size of the array matches the resolution of the image. A colour image has a three dimensional array because it has two dimension for resolution (image size) and another dimension for size of each pixels RGB vector, i.e. $M \times N \times 3$. Figure 1.2 shows a simple digital image and its array representation.



(a) (b)

Figure 1.2: (a) A simple 8-bit grayscale image (b) Array representation of a grayscale image.

2D array representation allows pixels to be referenced by their relative position in the M and N directions. Furthermore, M and N may be substituted for the x and y axes of the Cartesian plane. In fact, a digital image is just a sampled version of a two dimensional function where each pixel is described by a coordinate (x,y) and an intensity (z) at that point.

$$z = f(x, y) \tag{1.2}$$

This representation is extremely useful as it means that two dimensional operations may be applied to a images. Rates of change, derivatives, of a digital images are important in computer vision because they help to identify features like lines and object edges. Figure 1.3 is a 3D plot of a 2D image where you can see that there is a rapid change in value between different coloured squares, shown as steep cliff edges.

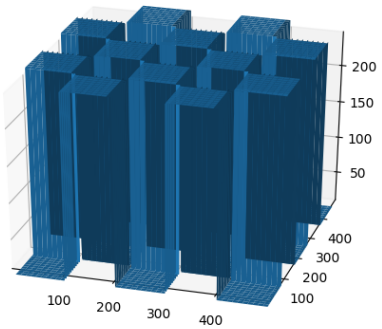


Figure 1.3: Surface Plot of Figure 1.2a

1.2 The Gaussian

The Gaussian provides a consistent model for normal distributions, aka the bell curve, as in Figure 1.4 & 1.5. Normal distributions follow the *central limit theorem* where in if a histogram is taken of a sufficiently large number of independent random variables they will distribute with a central most probable value and symmetrically fall away either side of this value. Many datasets follow this trend closely enough that the Gaussian can be used to approximate a probability distribution for them.

The Gaussian function is defined by three parameters, its mean μ, standard deviation σ and amplitude A.

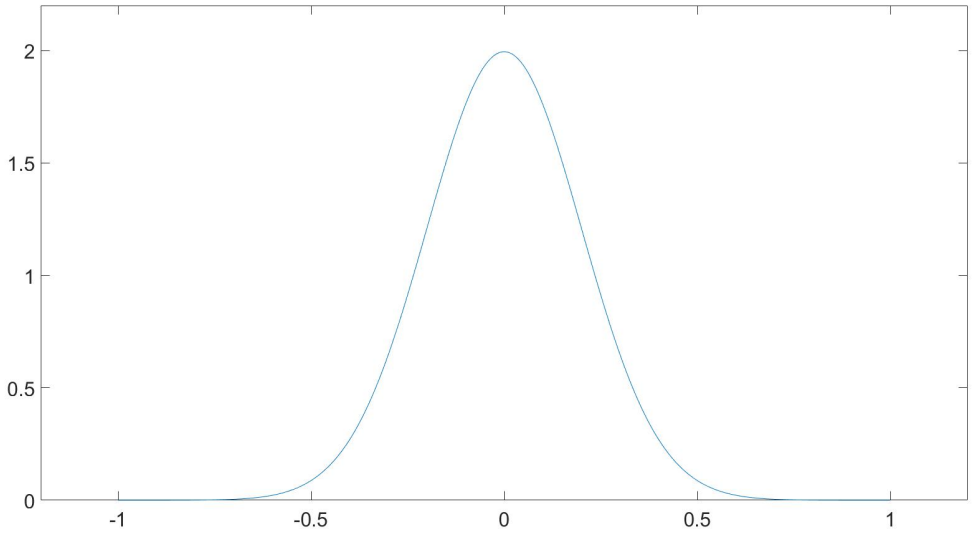


Figure 1.4: The Gaussian Function in 2D with μ=0, σ=0.2 and $A = 2$.

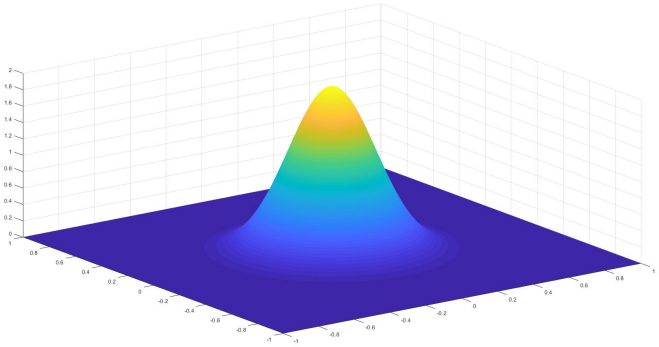


Figure 1.5: The Gaussian Function in 1D with μ=0, σ=0.2 and $A = 2$.

The general one dimensional Gaussian is described:

$$f(x) = \frac{A}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{1.3}$$

The mean value, the central limit, of the sample distribution is the value that has the greatest likelihood[1] in the distribution. The standard deviation is how much the distribution is spread out, the greater the standard deviation the fatter the distribution. The amplitude is the likelihood at the mean of the distribution. In Figure 1.4, for example, the likelihood of a value being 0 is 2. Generally, the curve is used to determine a sample's probability density within a value range. This is the area under the distribution given an interval and is calculated using integration, as shown in 1.4.

$$P(x) = \int_a^b \frac{A}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx \tag{1.4}$$

Data sets can have many dimensions and distributions need to be able to express all of them. The notation for a multi-variate Gaussian distribution, for a k-dimensional random vector $\boldsymbol{X} = (X_1, \ldots, X_k)^T$, is

[1] Likelihood is the value of the distribution given a fixed sample. Probability is the value of a sample given a fixed distribution.

$$\boldsymbol{X} \sim \mathcal{N}_k(\mu, \sigma^2) \tag{1.5}$$

1.3 Image Processing

Image processing is the modification of digital image via the application of one or many techniques in order to change its appearance or extract some information from it, such techniques can be used to sharpen edges, remove noise, apply blur, rotate an image and perform image compression. These techniques are performed by software and can be described mathematically and algorithmically. The following sections describe some image processing fundamentals and image processing techniques relevant to the system that is the subject of this report.

1.3.1 Thresholding

Thresholding is an important image processing technique where a grayscale image (1.1) is converted to a binary image by setting a pixel intensity threshold that determines sorts the original images pixels into either black or white pixels. This is useful when you wish to sort the pixels of an image into two sets, for example you may wish to label all pixels as either foreground objects or background objects. Equation 1.6 gives a mathematical description of thresholding a grayscale image, $F(x, y) = z$, where T is the threshold intensity [2]. Figure 1.6 shows an example of thresholded image whereby setting the threshold intensity to approximately higher than the intensity of the background the white horse is isolated, however the black horse has a light intensity lower than the threshold and so it merges into the background. If the threshold was set very low then the black horse would be isolated.

$$BinaryOutput = \begin{cases} 1 & \text{if } z >= T \\ 0 & \text{if } z < T \end{cases} \tag{1.6}$$

(a) Grayscale image. Img: Joe Amon. (b) Binary image result of thresholding.

Figure 1.6: Performing thresholding on a grayscale image to isolate a white horse.

1.3.2 Linear Filtering

Linear filtering of an image is an interaction between a filter and a digital image. The output of this operation is the result of the weighted average of the image pixels in a neighborhood specified by the location and size of the filter. The mathematical operation that describes the weighted average produced by the interaction of a linear filter with an image is called convolution (see sections 1.3.2 & 1.3.2). A filter is characterised by its *kernel*, K, which is comprised of coefficients, or weights, that affect the output of the filter. Convolution multiplies each of the filter's coefficients with its corresponding pixel, sums the results of each multiplication and stores it at the location matching the filter's origin in the output image. Figure 1.8 visualizes a single kernel application where the filter placed over a neighborhood of pixels in a digital image. The reference pixel is the location for which the result is stored in the output image, to perform this operation on the entire image the process is repeated with the reference pixel centered over every pixel in the image [2]. Figure 1.7 shows the general form of a linear filter kernel and that the divisor that determines the value of the result is the sum of all coefficients in the filter kernel.

$$K = \frac{1}{\sum_{i=0}^{M}\sum_{j=0}^{N} c_{i,j}} \begin{bmatrix} c_{0,0} & c_{0,1} & \dots & c_{0,n} \\ c_{1,0} & c_{1,1} & \dots & c_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,0} & c_{m,1} & \dots & c_{m,n} \end{bmatrix}$$

Figure 1.7: General form of a linear filter.

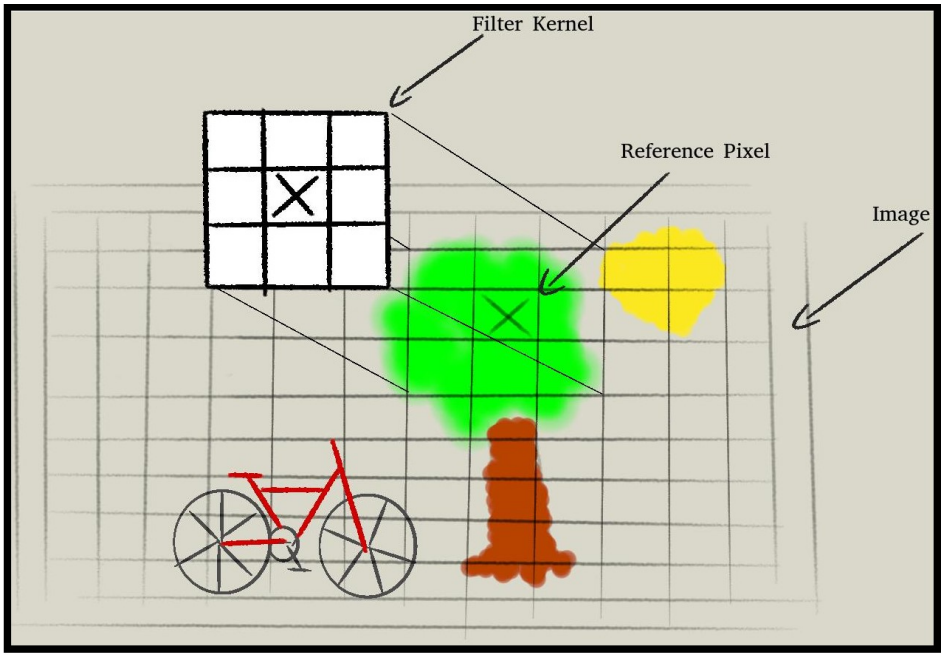


Figure 1.8: Visualization of a filter kernel application.

Correlation

The correlation of a linear filter $h(u, v)$ to an image $f(i, j)$ with the output $g(i, j)$ may be described mathematically as in Equation 1.7 but denoted more succinctly as in Equation 1.8

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-l}^{l} f(i+u, j+v)h(u,v) \tag{1.7}$$

$$g = f \otimes h \tag{1.8}$$

Correlation measures the *similarity* between two signals and is used in image processing to measure the similarity between digital images and linear filters. Correlation produces output spikes not just where the image value is greatest because digital images and filters can contain negative values. For example a byte encoded grayscale image could be represented by [-127,128]. The significance of this is that when negative intensities are multiplied with other negative intensities they produce positive values and when negative values are multiplied with positive values they produce a negative, thus wherever a filter has the best match with the

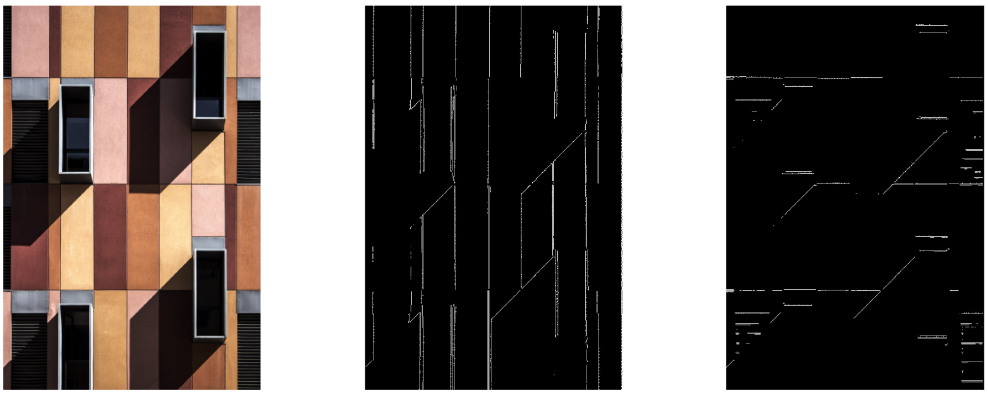
image a spike in positive values will occur and where the signals are dissimilar the output will be diminished [3][4].

This simple method can be used to isolate features in an image by correlating an image with a filter that matches the desired feature. A feature could be as simple as a straight line as in a Sobel filter (Figure 1.9) or as specific as the outline of a car wheel. This method of feature detection is called *template matching* [5]. In Figure 1.10 vertical and horizontal Sobel filters are correlated with an image a building wall and produce images that emphasize lines that match the orientation of the respective filter. The result is a binary image the output has been thresholded (see section 1.3.1) to isolate high intensity regions where similarity is greatest.

$$k = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \qquad k = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

(a) Horizontal Sobel-Feldman Filter Mask (b) Vertical Sobel-Feldman Filter Mask

Figure 1.9: Sobel Filters



(a) Img: Simone Hutsch (b) Vertical similarity. (c) Horizontal similarity.

Figure 1.10: Application of Sobel filters to exaggerate lines.

In performing correlation at the edge of an image the portions of the filter can be outside the boundaries of the the image. To deal with this scenario an image can be prepared with *padding* which adds additional pixels around the border of the image. There are many methods of

padding an image such as mirroring which reflects pixels at the images edge into the new border and constant padding which just add a single colored border around the image [6].

Correlation is a *linear shift-invariant* (LSI) operator which means that it doesn't matter where a filter is correlated with an image or in what order it is applied to it's pixels, the result will be the same [7]. More formally an LSI operator satisfies the principals of super-positioning and shift invariance. Given a filter h and signals f_1 and f_2 it doesn't matter whether the signals are added before they're filtered or the filter is applied individually and added the super-positioning principle states that the result is the same (equation 1.9). The symbol $\circ$ represents the linear shift invariant operator).

$$h \circ (f_1 + f_2) = h \circ f_1 + h \circ f_2 \tag{1.9}$$

The principal of shift invariance states that if the signal $g(i, j)$ is equal to the shifted signal $f(i+k, j+l)$ then these signals are still equivalent after the application of filter h to both of them (equation 1.10).

$$g(i, j) = f(i + k, j + l) \Leftrightarrow \ (h \circ g)(i, j) = (h \circ f)(i + k, j + l) \tag{1.10}$$

Figure 1.11 demonstrates a side effect of correlation which is to flip both horizontally and vertically the output signal relative to the the original image [4]. This occurs because as a filter scans over an image the output of the neighborhood is stored at the filter's origin. This effect can be negated by flipping the filter in both directions before applying it to the signal and is the only difference between correlation and convolution. This difference means however that in convolution the similarity between signals is not been measure but the effect one signal has on another. The next section covers convolution in greater detail.

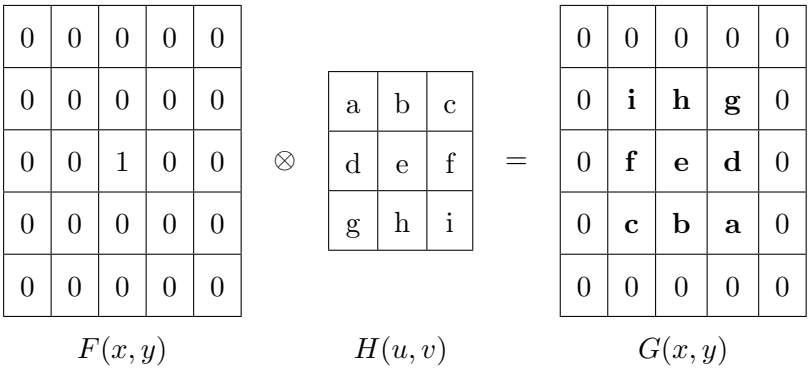


Figure 1.11: Correlation of a filter and an image.

Convolution

Convolution is also a linear operation that is shift invariant. It is very similar to correlation except that where correlation measure similarity between signals convolution measures the effect of one signal on another. It is described mathematically by the expression,

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-l}^{l} f(u,v)h(i-u, j-v)$$

Notice that the filter $h(i,j)$ is flipped 180 degrees in both dimensions as compared to equation 1.7, this causes the output to match the orientation of the original image. Convolution may be denoted as follows,

$$g = f * h$$

Convolution is essentially the same as correlation except that it doesn't flip the output relative to the original image as can be seen in Figure 1.12 and it is the operation that is performed when a linear filter is applied to a digital image.

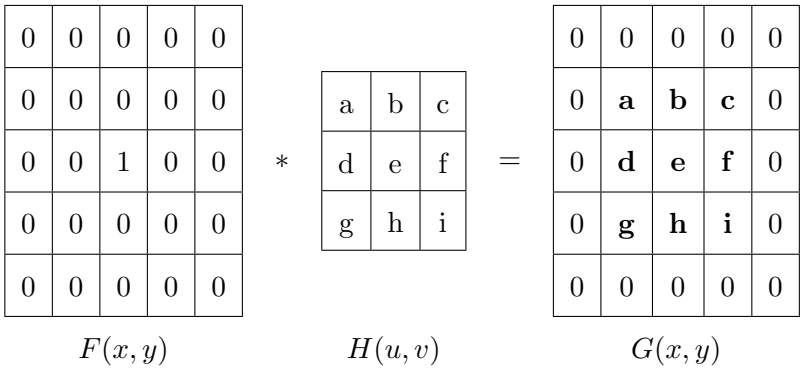


Figure 1.12: Convolution of a filter and an image.

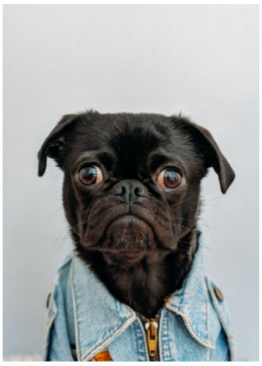
Some Important Kernels

Kernels are just the weightings that define the characteristics of a filter, also known as a filter's mask. A filter kernel is nearly always square so as to have a center cell which sits atop a reference pixel. The result of the filter's application at that reference pixel will be stored in the output image at the location of the reference pixel. Notice in Figure 1.8 how the mask sits over the reference pixel.

Box Filter

A box filter also known as a moving average filter simply outputs the average of its inputs because the filter weights are evenly distributed (Figure 1.14). By passing this filter over an image its sharpness is reduced because the filter flattens out difference in values generating a smooth effect which can be observed in Figure 1.13. This filter's 'flattening' effect is useful for removing noise which stands out from other pixel values.

(a) Img: Charles Deluvio



(b) Blurred pug

Figure 1.13: Application of a 17x17 Box filter to blur image.

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

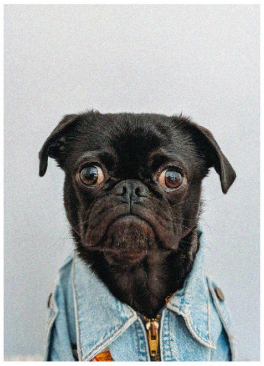
Figure 1.14: 3x3 Box Filter Kernel

Gaussian Kernel

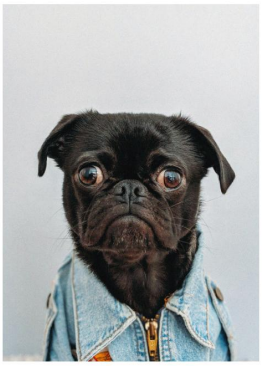
The Gaussian is a significant kernel in image processing as it is good at filtering out noise. This kernel models the Gaussian function (see Section 1.2) or normal distribution. It works in a similar fashion to the moving average filter but is superior at filtering noise as it addresses two properties about images that are generally true,

1. The 'real' value of a pixel is probably the same or similar to its neighbors.
2. Each pixel of noise in an image is added independently.

The Gaussian addresses both of these qualities by having high value coefficients at the filter's center and lower values tapering out to the edges of the filter (Figure 1.16). This means that values closer together are more strongly correlated than those further away from the reference pixel as can be observed in the 7x7 Gaussian filter kernel in Figure 1.15.



(a) Noisy pug



(b) Denoised pug

Figure 1.15: Application of a 7x7 Gaussian filter to denoise image.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0002 & 0 & 0 & 0 \\ 0 & 0 & 0.0113 & 0.0837 & 0.0113 & 0 & 0 \\ 0 & 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 & 0 \\ 0 & 0 & 0.0113 & 0.0837 & 0.0113 & 0 & 0 \\ 0 & 0 & 0 & 0.0002 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

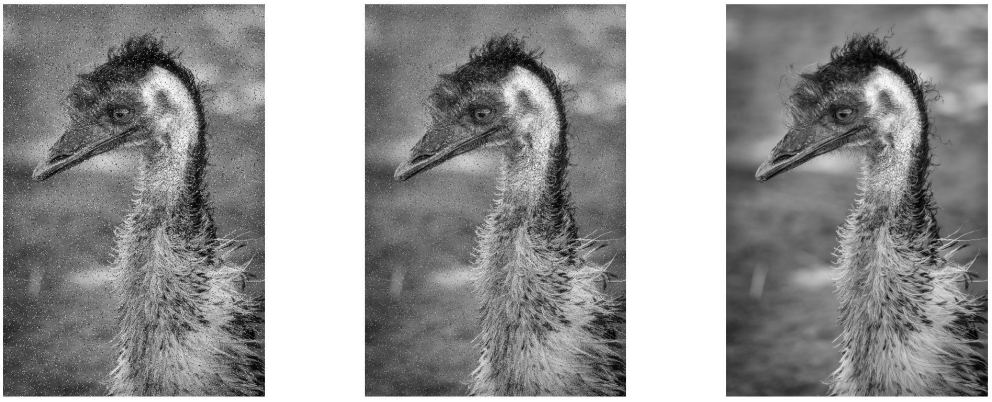
Figure 1.16: 7x7 Gaussian Filter Kernel, Sigma = 0.5

1.3.3 Non-Linear Filtering

Non-linear filtering is set of operations that is performed on a neighborhood of pixels in a digital image. In linear filtering the sum of an image's responses to two or more filters is the same as the sum of those filters applied to the image once. This is not the case in non-linear filtering, however there are scenarios where a non-linear filter can achieve superior results. A non-linear filter's application to an image is not simply it's convolution with that image.

The Median Filter

The median filter is useful for denoising an image, much like the linear box and Gaussian filters (section **??**). A median filter's applications returns the median valued pixel found in the neighbourhood under its mask. This is a more complex operation than to convolve one signal with another however it can be computed in linear time [8]. It is superior to the Gaussian filter at denoising images, particularly where intensities vary largely, which would skew the weighted average of a linear filter. In Figure 1.17 the median filter demonstrates a superior ability to remove salt and pepper noise as compared to the Gaussian filter.



(a) Salt and Pepper Noise (b) 7x7 Gaussian (c) 7x7 Median Filter

Figure 1.17: Filtering out salt and pepper noise with Gaussian and median filter. Original image by Grant Durr.

Morphology

Morphological operations are yet another set of neighborhood operators however unlike filtering they are designed to modify the shape of objects in binary images specifically. Producing a binary image from a coloured one involves first converting that image to grayscale and then thresholding (Equation 1.6) its intensity such that pixels are converted to a 1 or 0 depending on whether they lay above or below the threshold (Figure 1.18). These are useful operations when you wish to modify an object's shape in order for it to fit some criteria, for example its total area or whether or not it's an open or closed contour. All morphological operators work by convolving a binary *structuring element* with an image and thresholding the output pixel.

There are two fundamental morphological operations, *dilation* and *erosion*, that comprise all others.

(a) RGB Image (b) Grayscale Conversion (c) Binary Thresholding

Figure 1.18: Conversion of an RGB image to a binary image. Original image by Adam Birkett

Dilation

Dilation fills out or enlarges an object in a binary image in a way that is characterized by the specific structuring element used for the operation. A structuring element itself is just a binary image, generally a lot smaller than the target image itself, of a similar nature to a filter kernel, having an origin or reference pixel at their center. In Figure 1.19b the dilation causes objects to fill in gaps and even small spurious objects are inflated. This is because the structuring element 1.19f is simply a 9x9 pixel square where each convolution of the filter with the image outputs a 1 if the element overlaps at least one pixel of a binary object. This is a useful operation for filling in gaps in objects or thickening lines.

Erosion

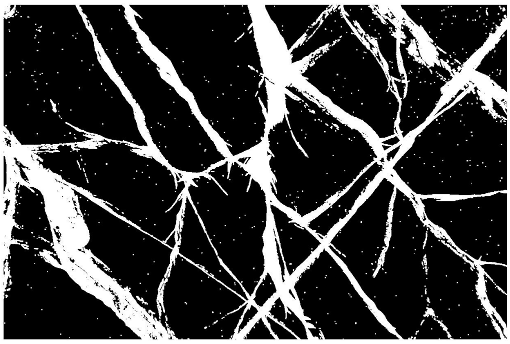
Erosion performs the opposite function to dilation causing objects to contract rather than grow. Again, the structuring element is convolved with a binary image but and the output is placed at the structuring element's origin in the resulting image. Intuitively, wherever the structuring element is not completely inside an object the output is stored as 0 (black). In Figure 1.19 the structuring element we see that many smaller entities are erased from

the image and many salient features are removed from large objects. This has the effect of 'cleaning up' the binary image of all speckley and spiky features, the nature of the cleanup depends on the size and shape of the structuring element, in this case the structuring element (1.19f) is a square of size 9, so it acts like an eraser wiping anything that's approximately the same size or smaller.

Opening and Closing

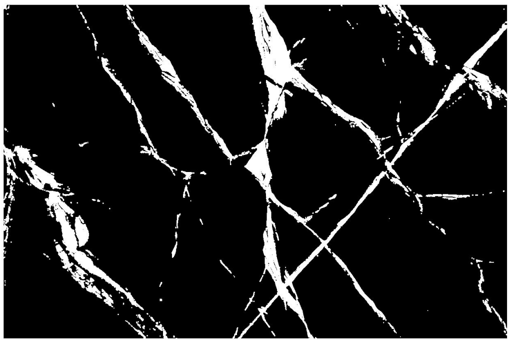
Opening and closing are composite operations comprised of an erosion and dilation, each of which may use the same or different structuring element. An opening is an erosion followed by a dilation which is useful for isolating and cleaning up objects that might be just 'kissing' by first removing entities that are probably too small to be of interest, refining of object edges and breaking of thin connections and then smoothing of the remaining objects. In Figure 1.19d it can be seen that many spiky entities are removed and some thin connection broken creating discrete objects that are smooth in somewhat thicker. Closing is the opposite ordered operation to opening performing first a dilation then an erosion. This is useful is you want to join objects together and preserve objects that might have been wiped out by an initial erosion. In Figure 1.19e it can be seen to close many small gaps between objects and fill them in but still result in smooth large objects as a consequence of the erosion. Therefore, an opening can be thought of as useful for separating objects from one another and closing is good at joining them together.

(a) Binarized image



(b) Dilation

(c) Erosion



(d) Opening

(e) Closing

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

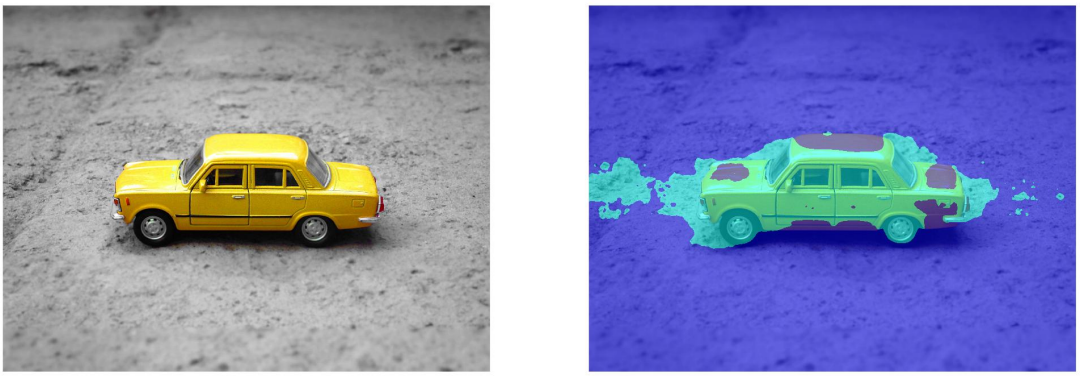
(f) Structuring element

Figure 1.19: Effect of morphological operations on a binary image using a 9x9 square structuring element. Original image by

1.4 Computer Vision

1.4.1 Clustering

Clustering is a method of segmenting an image into disjoint sets known as classes. This is useful for separating types of features or objects in an image. For example in Figure 1.20 the toy car has been segmented from the background.



(a) Image by Gustavo, Upsplash. (b) Segmentation of car from background.

Figure 1.20: Segmentation of toy cars using K-Means Clustering.

In an image every pixel can be individually classified or groups of pixels known as superpixels may be treated as single entities. Entities are sorted based on their similarity. For example, a pixel's intensity is a feature that could be used to cluster it with other pixels of similar intensity. As seen in (1.11) an entity's comparable traits may be represented by a feature vector. An entity may have any number of features which as also regarded as dimensions, d.

$$\vec{v} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \tag{1.11}$$

K-Means

There are many methods by which to cluster, K-Means is a popular method due to its speed and simplicity. Data points are sorted into one of K clusters where the average value of the

cluster is stored in a central data point known as the cluster centroid. Centroids may initially be randomly generated or selected from the given data set.

Data points or samples are placed in the class of the centroid that is closest to them. The distance between them is the Euclidean Distance between their feature vectors as in 1.12.

$$D(\vec{p}, \vec{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{1.12}$$

K-Mean's algorithm proceeds as follows:

1. Place the initial K centroids.
2. Assign all data points to their nearest centroid.
3. Update the centroids' values to be the mean of all data points in their cluster.
4. Repeat steps 2 and 3 until a criteria is met, for example until cluster centroids no longer move (converge).

K-means benefits greatly from its low computational cost such that it is oten performed a number of times with different initial centroids and the instance best result is used. The best result is defined as having the smallest intracluster variance. K-Means is disadvantaged by the implicit trait that it formulates clusters of similar sizes. This happens because the algorithm seeks to minimize variance (spread) in each cluster hence the 'ideal' centroid placement will form distributions spherically about centroids. This method cannot disentangle overlapping samples that belong in different classes. In Figure 1.21 clear edges between clusters and sphereical distributions can be observed. The algorithm can only implement *hard assignments*, as opposed to a *soft assignment* that consider the probability of a sample's class membership.

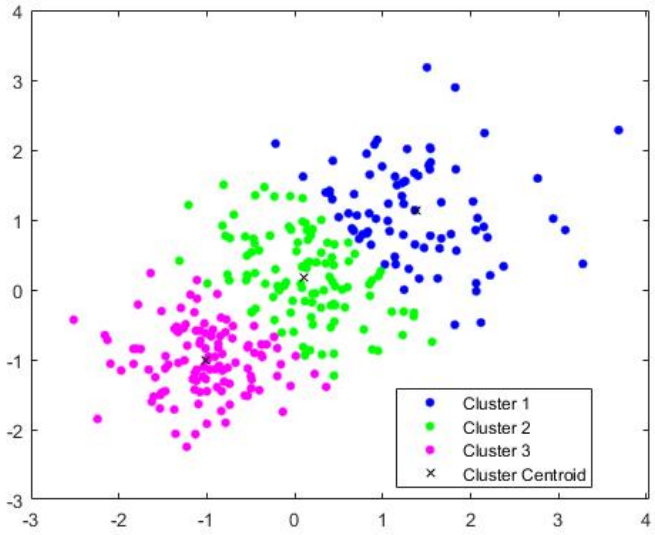
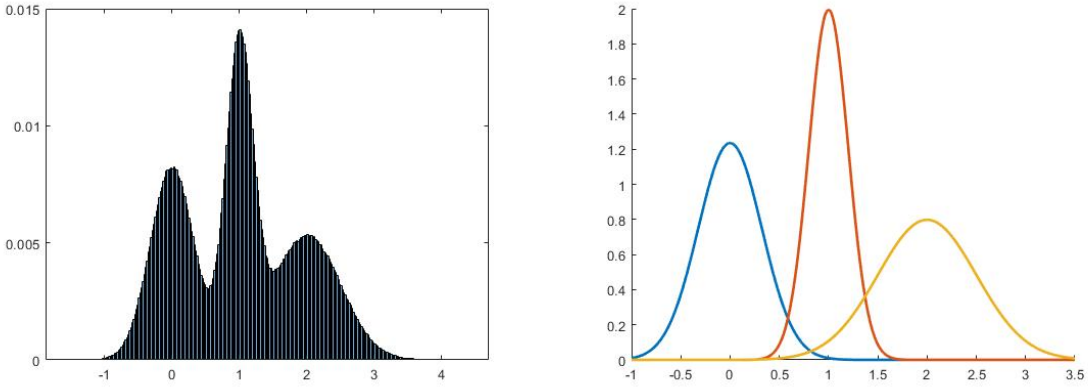


Figure 1.21: K-Means clustering performed on random data.

Mixtures of Gaussians

Mixtures of Gaussian or the Gaussian Mixture Model (GMM) is a method of clustering data that's able to disentangle ambiguous samples by considering a sample's probability of belonging to a class, known as a soft assignment.

Initially, K Gaussian functions are randomly generated corresponding to K clusters. If a dataset has low dimensionality, by taking a histogram of its values the Gasussians' initial conditions can be approximated, as in Figure 1.22a. By superpositioning all of the Gaussians a sample's complete probabilistic model is created, i.e. a model for *all* clusters, see Figure 1.22b.



(a) Normalized Histogram of 1D samples. (b) Gaussians derived from 1D histogram.

Figure 1.22: Formulation of Mixture of Gaussians.

With each iteration of the GMM algorithm the parameters of the model's component Gaussians are tuned according to the covariance between samples in each cluster. In 1.13 X and Y are the variables being compared, $\overline{X}$ and $\overline{Y}$ are variable means and n is the number of samples. For a multidimensional dataset a covariance matrix $\boldsymbol{\Sigma_k}$ will be generated and each Gaussian will require a mean vector $\boldsymbol{\mu_k}$ as opposed to a scalar.

$$Cov(X,Y) = \frac{\Sigma(X_i - \overline{X})(Y_j - \overline{Y})}{n} \tag{1.13}$$

The algorithm seeks the highest covariance possible in each of its clusters. It is by considering the covariance of samples that the GMM is able to best classify ambiguous samples. The higher the covariance (aka correlation) between sample dimensions the more likely it is they belong in the same cluster. This can observed in Figure 1.23, the data being clustered is the same as in Figure 1.21 but notice the elliptical shape of the Gaussian cluster distributions as opposed to the spherical shape of K-Means clustering.

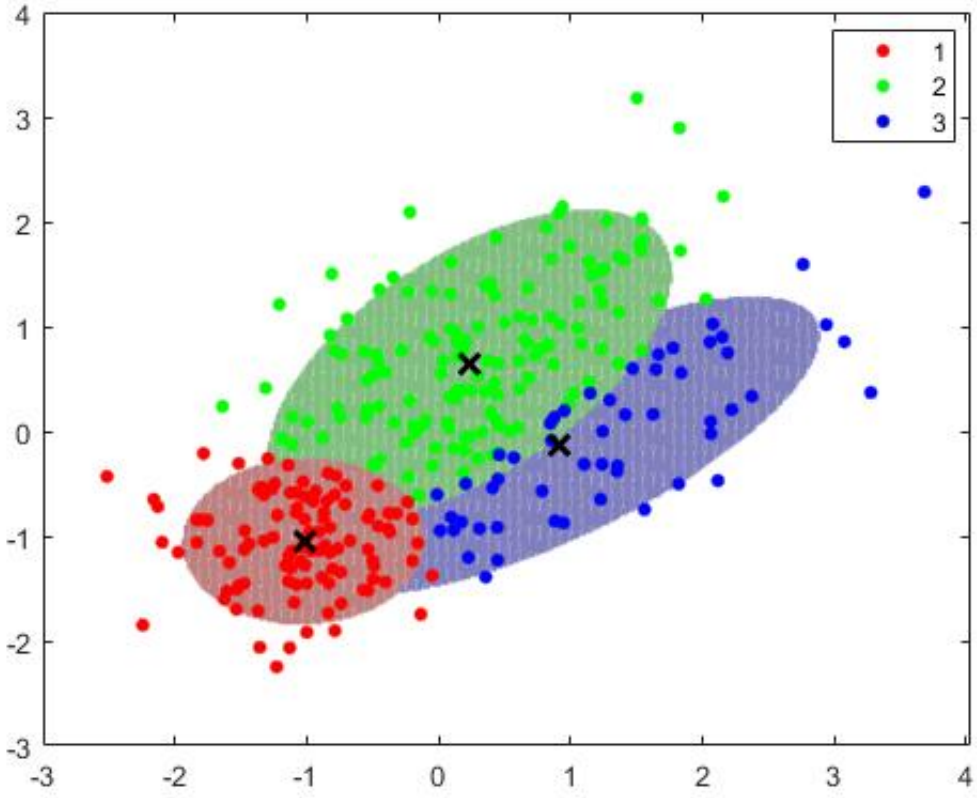


Figure 1.23: Clustering using GMM. X marks cluster mean.

In Figure 1.24 the Gaussians from Figure 1.22b have been scaled to have the amplitudes $\pi_1 = 0.3$, $\pi_1 = 0.5$ and $\pi_1 = 0.2$. These are weightings that give the mixing ratio of the GMM and represent each cluster's proportion of the total samples. The more samples a cluster contains the more likely it is a sample belongs to it. The sum of ratios must add to one because the probability a sample belongs to at least one cluster is one, as defined in 1.14 and 1.15.

$$0 \leq \pi_k \leq 1 \tag{1.14}$$

$$\sum_{k=1}^{K} \pi_k = 1 \tag{1.15}$$

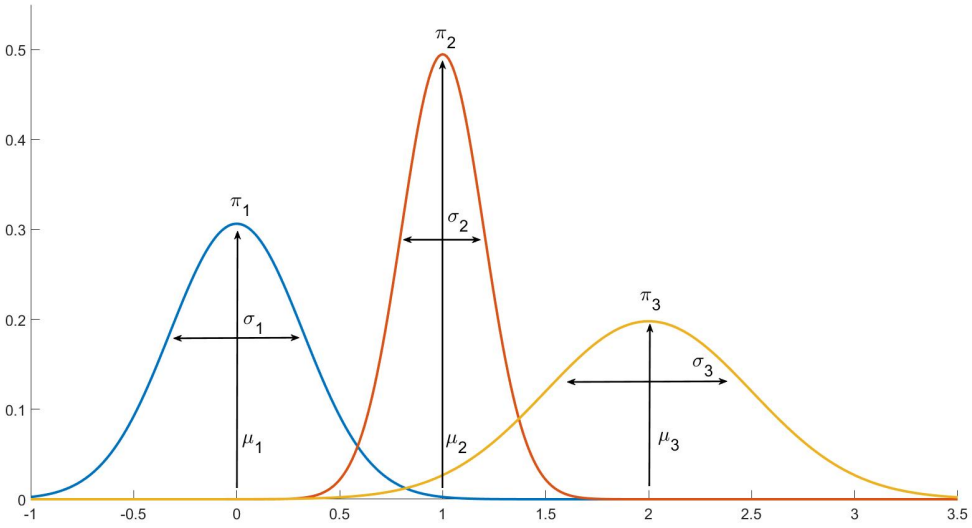


Figure 1.24: Individual Gaussians with scaled weightings.

Each Gaussian's three parameters, mean $\boldsymbol{\mu}_k$, ampltidue π_k and covariance $\boldsymbol{\Sigma}_k$ are updated following each iteration of the algorithm according to the 'Expectation Maximization' algorithm.

Expectation Maximization

Expectation Maximization (EM) is a method of updating the parameters of cluster Gaussians that seeks to maximize a sample's likelihood of belonging to said Gaussians (1.16).

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.16}$$

A binary indicator z_i exists for every sample x_i, it is 1 if the sample belongs in the cluster k and 0 otherwise such that $z_{ik} \in \{0, 1\}$ and $\sum_k z_{ik} = 1$. If the data is multidimensional then the sample and inidicator are both vectors, $\boldsymbol{x}$ and $\boldsymbol{z}$. The marginal probability $p(\boldsymbol{z}_k = 1)$ is the probability that a sample $\boldsymbol{x}$ is in cluster k. This quantity is completely specified by the mixture weight π_k for each Gaussian because the area under a Gaussian component is equal to its mixing ratio, i.e.

$$p(\boldsymbol{z}_k = 1) = \pi_k \tag{1.17}$$

If we know that a sample $\boldsymbol{x}$ is from cluster k the *likelihood* of seeing it in the associated Gaussian is the value of the Gaussian at that point,

$$p(\boldsymbol{x} \mid \boldsymbol{z}_k = 1) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \tag{1.18}$$

The conditional probability of $\boldsymbol{z}_{ik}$ given the value of sample, $\boldsymbol{x}_i$ is denoted as $\gamma(\boldsymbol{z}_{ik})$. This is the probability that a sample belongs to cluster k given its value $\boldsymbol{x}_i$ and is the quantity of interest when trying to classify a sample. Using Bayes Theorem [REF APPENDIX] quantity can be found

$$\gamma(\boldsymbol{z}_{ik}) \equiv p(\boldsymbol{z}_{ik} = 1|\boldsymbol{x}_i) = \frac{p(\boldsymbol{z}_{ik} = 1)p(\boldsymbol{x}_i \mid \boldsymbol{z}_{ik} = 1)}{\sum_{j=1}^{K} p(\boldsymbol{z}_{jk} = 1)p(\boldsymbol{x}_j \mid \boldsymbol{z}_{jk} = 1)} \tag{1.19}$$

$$= \frac{\pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{j=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{1.20}$$

To maximize the likelihood of each sample being in each Gaussian the distribution parameters are modified. This is achieved by taking derivatives of the log of the likelihood function (1.16) with respect to each Gaussian parameter and setting the result to 0 to find local maxima. To take the log of the likelihood function assume all samples in are in an $N \times D$ matrix $\boldsymbol{X}$ and the corresponding indicators are in an $N \times K$ matrix $\boldsymbol{Z}$.

$$ln\, p(X|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} ln\left\{ \sum_{k=1}^{K} \pi_k(N)(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{1.21}$$

The EM algorithm is comprised of [X] steps

1. Generate the initial K Gaussian parameters mean $\boldsymbol{\mu}_k$, covariance $\boldsymbol{\Sigma}_k$ and mixing ratios

π_k either randomly or informed by a histogram.

2. Estimate the likelihood sample n was generated by cluster k for all samples.

$$\gamma(\boldsymbol{z}_{ik}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{j=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{1.22}$$

3. Maximize the Gaussian parameter using derivatives of log likelihood for each parameter.

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n \tag{1.23}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})^T \tag{1.24}$$

$$\pi_k = \frac{N_k}{N} \tag{1.25}$$

where

$$N_k = \sum_i z_{ik}$$

4. Repeat steps 2 and 3 until convergence of log likelihood (1.16) or parameters.

The Gaussian Mixture Model method of clustering is computationally complex compared to K-Means however it's ability to differentiate ambiguous samples by considering covariance is superior.

Bibliography

[1] Daniel Shiffman. *Learning Processing.* Elsevier Inc., 2008.

[2] Richard Szeliski. *Computer Vision - Algorithms and Applications.* Springer, 2011.

[3] Sabyasachi Sahoo. Deciding optimal kernel size for cnn.

[4] Aaron Bobick. Introduction to computer vision. Udacity, 2018.

[5] Jan Erik Solem. *Computer Vision with Python.* O'Reilly, 2012.

[6] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering.* CRC Press, 2004.

[7] Richard E. Woods Rafael C. Gonzalez and Steven L. Eddins. *Digital Image Processing using MATLAB.* Gatesmark, 2009.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press Ltd, 2009.

Appendix A

Hardware Specifications

A.0.1 Raspberry Pi 4

A.0.2 Pi Cam

Appendix B

Mathematical Theorems and Proofs