Computational Vision
Revision Notes

James Brown

April 13, 2017

Contents

1 Introduction

These are notes I have written in preparation of the 2017 Computation Vision exam. This year the module was run by Hamid Deghani (H.Dehghani@cs.bham.ac.uk).

Computational vision is the acquisition of knowledge about objects and events in the environment through information processing of light emitted or reflected from objects. In short - we want to make a computer know what is where, by looking through information. We can also use computational vision to do automatic inference of properties of the world from images.

2 Human Vision

As humans we have evolved eyes which perceive the visible section of the electromagnetic spectrum, which falls between the wavelengths of 380nm - 760nm. Red light lies at the longer end (760nm) of visible light, and purple at the shorter end (380nm). Visible light is strongly absorbed in the human eye because it can cause an electron to jump to a higher energy levels - yet it does not have enough energy to ionize cells. The evolutionary process of evolving eyes began more than 3 billion years ago with the formation of photopigments. These are molecules where light incident upon them will trigger a physical or chemical change. Photopigments capture photons which lead to the release of energy in the photopigment. This is may be used for photosynthesis or a behavioural reaction (a nerve reaction). A single photocell contains multiple layers to catch light, not just one. This increases the chance of catching any one individual photon - if it's not caught by the first layer it's much more likely to be caught by the second and so on.

2.1 Image Formation

Photocells contain a light sensitive patch of photopigments. Using a single cell we can capture light in 1 dimension, but we can't really 'see'. All we can do is tell if the light is on, or off. With multiple cells we can have better direction resolution and with multiple cells we have a very wide aperture - we can't tell exactly where the image is. The image formed will be very bright but extremely fuzzy. This became curved over time and this which really helped with direction resolution as light incident on the left side of the curve must have come from the right. Images formed this way are still very blurry and will result in multiple projections of the same image. Over time eyes evolved to become pinhole cameras which form sharp yet dim images by allowing light to come from a single source (the pinhole) - effectively throwing away loads of potential information about the image. The solution to form sharp and bright images was to use a lens at the front of the eye. The lens focuses all incoming light to a single point and from there we can use our simple pinhole camera for forming images. It should be noted that the image formed is upside down to what exists in reality.

2.2 Retina Processing

The human retina contains two kinds of photoreceptors which respond to incident light - **rods** (around 120 million) and **cones** (around 6 million). Rods are extremely sensitive photosensors and respond to a single photon of light. Multiple rod cells converge to the same ganglion cell and therefore neuron within the which results in poor spatial resolution. Rods are responsible simply for detecting the presence of light, and as a result make up the entirety of our night-vision. On the other hand, cones are active at much higher light levels and responsible for the detection of different colours of light. Cones have a much higher resolution as they are processed by several different neurons. Within the eye we have a receptive field, which is the area on which light must fall for a neuron to be stimulated. It should be noted that receptive field in the center of the eye is much smaller than it is for the periphery of the eye.

2.3 The Visual Pathway

Vision is generated by photoreceptors in the retina. All the information captured leaves the eye by way of the optic nerve. There is a partial crossing of axons at the optic chiasm. This is only partial as information from both eyes is sent to both sides of the brain - this allows us to process depth. After this chiasm, the axons are called the optic tract. The optic tract wraps around the midbrain to get to the lateral geniculate nucleus (LGN). The LGN axons fan out through the deep white matter of the brain and ultimately to the visual cortex.

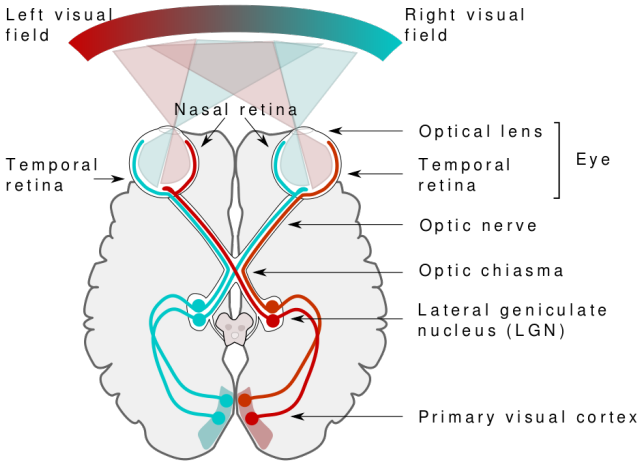

Figure 1: The visual pathway in the human brain

2.4 Colour Processing

Objects in the world selectively absorb some wavelengths (a colour) and reflect other wavelengths. Human retinas contain three different types of cones to respond to different colours. This gives us the ability to distinguish different forms of the same object - for example: unripe, ripe and off fruit. Many theories of colour vision have been proposed and some have been hard to disprove until recently.

In 1802 Young proposed that the eye has three different types of receptors, each of which are sensitive to a single hue. From this he proposed that any colour can be produced by appropriate mixing of the three primary colours. This is known as the trichromatic theory.

Hering suggested that colour may be represented in a visual system as 'opponent colours' in the 1930's.

3 Edge Detection

3.1 Intensity Images

An intensity image is a data matrix whose values represent intensities within some range. Each element of the matrix corresponds to one image pixel. An indexed image consists of a data matrix, X, and a colour map matrix, *map*. *map* is a m-by-3 array of double containing floating point values in the range $[0, 1]$. Each row in the map specifies the red, green an blue components of a single color. Each cell in the indexed image then specifies the corresponding colour from the colour map. In an intensity image can be thought of as a function $f(x, y)$ mapping coordinates to intensity. From this we can calculate an intensity gradient.

$$\overrightarrow{G}[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix}$$

This is a vector which we can think of as having an x and y component. We can calculate both the magnitude and direction for this gradient of intensity.

$$M(\overrightarrow{G}) = \sqrt{G_x^2 + G_y^2} \qquad \alpha(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

When calculating the magnitude, figuring out a square root can be very computationally expensive. It's possible to replace this with an approximation: $M(\overrightarrow{G}) = |G_x| + |G_y|$

3.2 Approximating the Intensity Gradient

In order to approximate the gradient we may use a variety of different masks over the image, such as the Roberts and Sobel detectors. These masks use the idea of convolution in order to calculate the rate of change of intensity at each pixel. Convolution is the computation of weighted sums of image pixels. For each pixel in the image, the new value is calculated by translating the mask to the pixel and taking the weighted sum of all the pixels in the neighbourhood. Once we have a value for the rate of change of the intensity gradient we can threshold our image to produce the locations of edges.

4 Noise Filtering

When taking images we also gather a lot of noise which results in fake edges once we apply our edge detectors. We would like to remove this noise and we many filters which can be implemented by the idea of convolution. The most widely used of these filters is the Gaussian filter, although there are other simpler filters such as the mean filter.

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

Figure 2: An example mean filter

$$\begin{bmatrix} 0 & .01 & .02 & .01 & 0 \\ .01 & .06 & .11 & .06 & .01 \\ .02 & .11 & .16 & .11 & .02 \\ .01 & .06 & .11 & .06 & .01 \\ 0 & .01 & .02 & .01 & 0 \end{bmatrix}$$

Figure 3: An example Gaussian filter

The mean filter averages a pixels value with all of the surrounding intensities for a new value. A Gaussian filter follows the same principle as this but uses a weighting, valuing pixels closer to the original more than pixels farther away. The Gaussian filter has an extra bonus of being able to be applied as two individual 1D Gaussian filters in sequence rather than one large 2D filter as shown below.

$$\begin{bmatrix} 0.0545 & 0.2442 & 0.4026 & 0.2442 & 0.0545 \end{bmatrix} \qquad \begin{bmatrix} 0.0545 \\ 0.2442 \\ 0.4026 \\ 0.2442 \\ 0.0545 \end{bmatrix}$$

Figure 4: A Gaussian filter deconstructed to two 1D filters which can be applied in sequence

5 Advanced Edge Detection

Intensity changes can be caused by geometric events such as surface orientation discontinuities, depth discontinuities, color discontinuities and texture discontinuities. It can also be caused by

non-geometric events such as illumination changes, specularities, shadows and inter-reflections. All of these events can be used to try to find edges in the image. When we try to detect edges, we are trying to produce a line 'drawing' of a scene from an image of that scene. We use this to extract important features of an image, such as corners and curves. These features are then used by higher-level computer vision algorithms.

5.1 Main steps in Edge Detection

Regardless of methods, there a few major steps to edge detection:

1. **Smoothing**: suppress as much noise as possible, without destroying any of the true edges.
2. **Enhancement**: apply differentiation to enhance the quality of edges (i.e. sharpening).
3. **Thresholding**: determine which edge pixels should be discarded as noise and which should be retained (i.e. threshold edge magnitude).
4. **Localization**: determine the exact edge location.

Most of the time, points that lie on an edge are detected by

1. Detecting the local *maxima* or *minima* of the first derivative.
2. Detecting the *zero-crossings* of the second derivative

There are a few practical issues that come with this. When smoothing an image, the smoothing effect achieved depends on the mask size (for example, with a Gaussian filter it depends on σ). A larger mask will reduce noise more, but it also worsens localization and adds uncertainty to the location of the edge.

Based on this we can draw up some criteria for an optimal method of edge detection:

1. **Good detection**: Minimize the probability of false positives and false negatives (spurious edges and missing real egdes).
2. **Good localization**: Detected edges must be as close as possible to the true edges.
3. **Single response**: Minimise the number of local maxima around the true edge.

Canny showed that the first derivative of Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization.

5.2 Hysteresis

Hysteresis thresholding uses two thresholds - a low threshold t_l and a high threshold t_h (usually this is $2t_l$). This makes the assumption that important edges should be along continuous curves in the image. It allows us to follow a faint section of a given line and to discard a few noisy pixels that do not constitute a line but have produced large gradients. We begin by applying the high threshold - this marks edges that we can be fairly sure that are genuine. Starting from these points, and using the directional information derived earlier, edges can be traced throughout the whole image. While tracing an edge, we apply the lower threshold, allowing us to trace faint sections of edges as long as we find a starting point.

6 Hough Transform

So far we haven't actually found edge segments, just lots and lots of edge points. The Hough Transform is a common approach to finding parameterised line segments. Every straight line in an image can be described by an equation. Each edge point in an image, if considered in isolation, could lie on an infinite number of straight lines. In Hough Transform each edge point votes for every line it could be on, and the lines with the most votes win.

7 Scale Invariant Feature Transform

We can now detect features in images. The most basic feature is edges, but through the use of Hough transform we can also detect lines and even geometric shapes such as circles. We want to detect features for various reasons, such as object recognition, wide baseline matching and tracking. In order to do this we need to make sure our features are scale invariant. Good features should be robust to all sorts of nastiness that can occur between two different images. There are many different causes of invariance:

- Illumination
- Scale
- Rotation
- Affine transformation (scaling and stretching)
- Full perspective

7.1 Dealing with Illumination Invariance

The easiest way to achieve illumination invariance is to normalise the image. This is done by creating a histogram of all intensity values within the image. We can then stretch the histogram to match others of the same image in different illuminations (other images with different illumination should still have the same shape of histogram, even in not the exact same histogram). This stretch can be achieved through simple linear normalization.

7.2 Dealing with Scale Invariance

One way to ensure our image is scale invariant is by trying to scale it up and down, and see if a feature exists regardless of how much the image is scaled. There are two main methods to achieve this: Pyramids and Scale Space (DOG method).

In the pyramids method, we divide the width and height of the image by 2. We take the average of 4 pixels for each new pixel and repeat this process until the image is very small. We can then run our filters (edge detectors) over the image to check for the existence of certain features. If the feature still exists all the way down, we can say that the feature is scale invariant. On the other hand, the DOG (Difference of Gaussian) method is very similar to pyramids. Instead will fill the gaps with blurred images. This is like having a nice linear scaling without any expense. We take the features from the differences of the images, and if the feature is repeatably present in between Difference of Gaussians it is scale invariant and we should keep it.

7.3 Dealing with Rotational Invariance

We can deal with rotational invariance in much the same way as we deal with illumination invariance. We want to rotate all features to go in the same way in a determined manner. To do this, we take a histogram of all edge directions in the image and then rotate the image to the most dominant rotation (or possibly even the second most dominant if it's good enough).

8 Face Recognition

This section is about face recognition, not detection - we will not be taking an image and finding the face within.

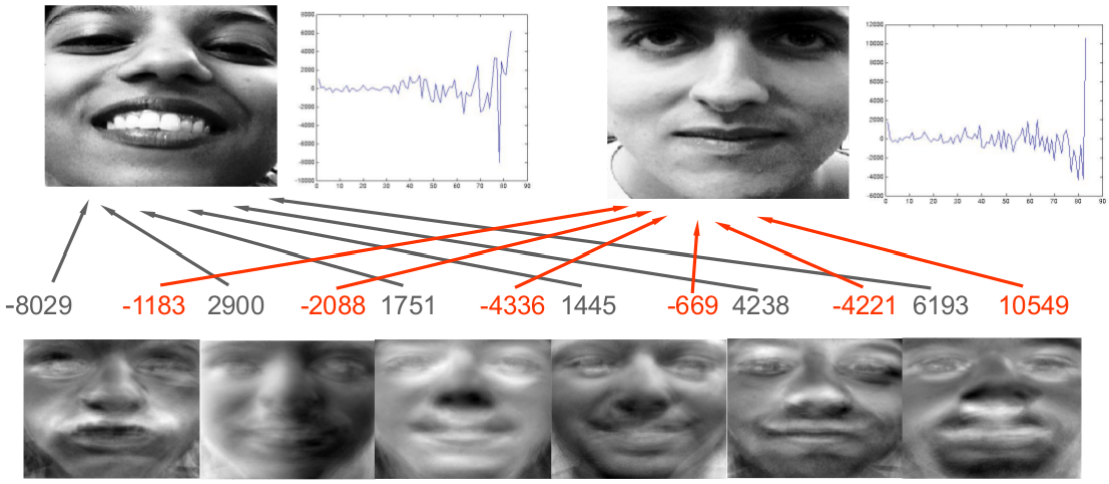

Figure 5: A figure showing how to build any face from a set of basis faces

8.1 Eigenfaces

We can have a face, which is a weighted combination of some parts or basis faces. The basis faces are called eigenfaces. The basis faces can be differently weighted to represent any face, so we can use different vectors of weights to represent different faces.

Obviously we must pick the set of basis faces, and to do this we need a set of real training faces. From these we can then learn a set of basis faces which best represent the differences between each of the training faces. We'll use a statistical criterion for measuring the notion of 'best representation of the differences between training faces'. Then we can store each face as a set of weights from those basis faces.

8.2 Finding Eigenfaces

There are two possible ways we can use eigenfaces - firstly we can store and then reconstruct a face from a set of weights and secondly we can recognise a new picture of a familiar face.

Initially, we have to figure out how to learn eigenfaces. A method called principle components analysis (PCA). In order to use this we have to understand what an eigenvector is and also what covariance is.

To begin, we imagine that our face is simply a (high dimensional) vector of pixels. For humans it's easier to think about 2 dimensions, so we'll do just that and have our data in just two dimensions. But when we have data in two dimensions, we only really need one dimension to represent it. To do this we can take a line through space, and then take the projection of each point onto that line - doing this means we have represented our data in 'one' dimension. The choice of line is important, some lines represent the data in this way well, and some badly. The projection onto some lines separates the data well, and the projection onto some lines separates it badly. However, rather than using a line we can perfom roughly the same trick using a vector v. When using a vector, we instead have to scale the vector to obtain any point on the line and this vector is called an **eigenvector**. The value μ, which we use to project any point on the vector, is our **eigenvalue**.

An eigenvector is a vector v that obeys the rule $\mathbf{A}v = \mu v$ where $\mathbf{A}$ is a matrix, μ is a eigenvalue. A few rules about eigenvectors:

- We think of matrices as performing transformations on vectors
- We think of eigenvectors of a matrix as being special vectors that are scaled by that matrix
- Different matrices have different eigenvectors
- Only square matrices can have eigenvectors, and not all square matrices even have them

- An n by n matrix has at most n different eigenvectors
- All the distinct eigenvectors of a matrix are orthogonal (perpendicular).

With our data, we need to find the vector which can be used to separate the points in our data as much as possible. This vector turns out to be the vector which expresses the direction of the correlation. The covariances then can be expressed as a matrix. The covariance of two variables is

$$cov(x_1, x_2) = \frac{\sum_{i=1}^{n}(x_1^i - \bar{x_1})(x_2^i - \bar{x_2})}{n-1}$$

This covariance matrix has eigenvectors and eigenvalues. The largest eigenvalue shows you the most dominant vector (corresponding to the direction in which the data varies more). This is principle components analysis.

8.3 Using Eigenfaces

All we are going to do is treat the face as a point in a high dimensional space, and then treat the training set of face pictures as out set of points. In order to train, we calculate the covariance matrix of the faces and then find the eigenvectors of that covariance matrix. The found eigenvectors are the eigenfaces or basis faces. Eigenfaces with bigger values will explain more of the variation in the set of faces (they are more distinguishing).

When we see an image of a face we can transform it to the face space with $\mathbf{W}_k = \mathbf{X^i}.v_k$. There are $k = 1...n$ eigenfaces v_k, and the i^{th} face in the image space is a vector x^i with corresponding weight w_k. We calculate the corresponding weight for every eigenface. Recognition is now simple, we can find the euclidean distance d between our face and all the other stored faces in the face space with

$$d(w^1, w^2) = \sqrt{\sum_{i=1}^{n}(w_i^1 - w_i^2)^2}$$

The closest face in the face space is the chosen match.

9 Motion

We aim to detect which changes have occurred in two different frames, and we can use this to deduce what motions have induced those changes. There are four possible possibilities for a dynamic camera and world:

- Stationary camera, stationary object
- Stationary camera, moving objects
- Moving camera, stationary object
- Moving camera, moving objects

But how do we detect changes? Firstly, detect features in two different frames. Then simply look at the difference between features at time t and time $t+1$. Any differences in the two features, indicate motion. More formally, we can say $F(x, y, i)$ is the intensity of the image at time i, at point x, y. It follows that the difference picture, DP, is defined as

$$DP_{12}(x, y) = \begin{cases} 1 \text{ if } |F(x, y, 1) - F(x, y, 2)| > \tau \\ 0 \text{ otherwise} \end{cases}$$

This approach is not without issues. If we have random noise in the image, the difference image will include all the noise points in both images even if no motion has occurred at all. To solve this, we may want to filter out all pixels that are not part of some larger structure. We use the idea of connectedness to achieve this, in particular 4 or 8 connectedness. Two pixels can be said

to be 4-neighbours is they share a common boundary. Two pixels are said to be 8-neighbours if they share at least one common corner. Two pixels are also 8-neighbours if we can create a path of 8-neighbours from one to the other. To remove noise, pixels that are not in a connected cluster of a certain size are removed from the difference image.

When determining motion, there is not enough information in the local intensity changes to give us the information that we want. This can be demonstrated via the aperture problem. When we only have a small area in which to observe, we cannot tell exactly what direction the object is moving (for example, something appearing to move down and right may only be moving to the right). To get around this we have to pick the particularly interesting areas of the image. This explains why edges are very bad for determining motion. Corners are a good example of particularly interesting areas. Once we have found these particularly interesting areas, we would like to match them between different images over time.

The Moravec operator is a corner detector. It defines interest points as points where there is a large intensity variation in every direction, which is the case at corners. One side-effect is that it is very sensitive to detecting isolated pixels as corners.

Once we have points of interest, we try to match the points in one image with the points in another. Using this we can estimate the motion that has occured. Motion correspondance is guided by three principles:

1. **Discreteness**: a measure of the distinctiveness of individual points
2. **Similarity**: a measure of how closely two points resemble one another
3. **Consistency**: a measure of how well a match conforms with nearby matches

To calculate the degree of similarity between two patches take the sum of the squared differences in the pixel values (L1-norm - absolute difference).

$$w_{ij} = \frac{1}{1 + \alpha s_{ij}}$$

When we want to try to match two patches, we calculate the weights for all the possible matches for patch i(t). How to calculate the weight is shown in the above equation. We assume some value for α, such as 1. Once we have all the weights, we normalise to produce probabilities for each of the patches to calculate the best matches between patches.

10 ROC Analysis

Receiver operating characteristic analysis provides us tools to select possibly optimal models and to discard suboptimal ones. In order to carry out ROC analysis we need to count the different kind of errors that are possible. When classifying if a pixel is an edge or not, there are 4 possible situations - **true positive** (predicted it was an edge and it was), **false positive** (predicted it was an edge and it wasn't), **false negative** (predicted it wasn't an edge and it actually was) and **true negative** (predicted it wasn't an edge and it wasn't).

Once classifying each pixel, we then count the total number of each label (TP, FP, TN, FN) over the large dataset. In ROC analysis we use two statistics:

$$\text{Sensitivity} = \frac{TP}{TP+FN} \qquad \text{Specificity} = \frac{TN}{TN+FP}$$

Sensitivity is the likelihood of spotting a positive case when presented with one or the proportion of edges we find. Specificity is the likelihood of spotting a negative case when presented with one or the proportion of non edges that we find. To define a ROC space we only need the true positive rate (TPR) and the false positive rate (FPR). The TPR is simply the sensitivity and the FPR is $1 -$ specificity. We use the FPR and TPR as x and y axes respectively of the ROC space.

All of the optimal detectors lie on the convex hull - the best possible position is for a detector to lie in the top left corner. It should be noted that if the edge detector lies below the dotted line then we can instantly make the detector better by simply just flipping the output of the algorithm!

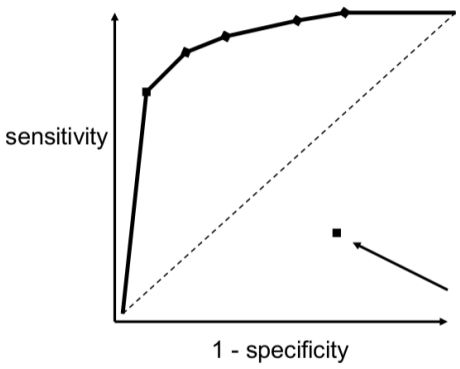

Figure 6: A sample ROC space

11 Object Recognition

12 Model Based Object Recognition

List of Figures

Index