COMP37212 Computer Vision

Izunna Aneke

2020

Contents

Acknowledgements

These notes are based on the COMP37212 course from The University of Manchester.

1 Introduction

1.1 What is Computer Vision?

The goal of Computer Vision is to write computer programs that can interpret images.

Computer Vision **interprets visual information** using a Computer. This involves understanding the scene content, recognition of scene components and extracting key information.

Computer Vision is also used for taking **measurements**. This includes quantitative features such as: size, density, distance and shape, but also involves classification of features in a scene.

We get **input** from a range of sources like TV cameras. We can also use special devices like MR scanners, laser range-finders and satellite images.

1.2 Can computers match human vision?

Yes and no. In general Humans are much better at *hard* things, whilst Computers can be better at *easy* things.

1.3 Is it as good as human vision?

Computer Vision can be **worse** than human vision. The reason being Computer Vision is very constrained in terms of the types of scenes a given algorithm can handle.

There is no ‘general purpose’ Computer Vision algorithm. Application-specific Computer Vision systems are developed and they cannot handle any type of vision task. Computer Vision also suffers from limited knowledge of image context and difficulty adapting to new situations.

Computer Vision can be **better** particularly for **quantitative analysis** (i.e. taking measurements and metric differences). Computer Vision can also handle a variety of different inputs such as: images in different spectral ranges (e.g. IR or UV) and non-optical sensors such as magnetic resonance images (MRI).

2 Introduction to Basic Image Analysis

2.1 Basic Image Analysis

Basic image analysis is limited to simple 2D scenes that can be described as 'background' and 'objects' (or foreground).

TBC

3 Placeholder

4 Placeholder

5 Placeholder

6 Placeholder

7 Segmentation and Clustering

7.1 Segmentaiton and grouping

7.1.1 Grouping in vision

The goal of segmentation is to gather features that belong together. The motivation is that despite having a high resolution camera, we may want to summarise the content in the image. In particular, we obtain an intermediate representation that compactly describes key parts of an image or video.

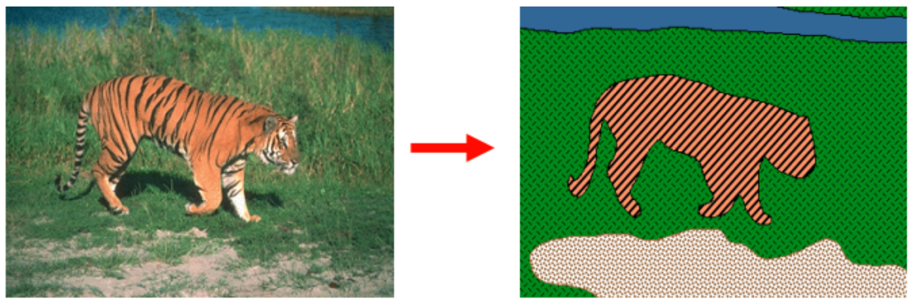


Figure 7.1.1: Example of segmentation

The type of segmentation in Figure 7.1.1 is what we as humans do naturally. We see the Tiger as separate to the grass. Perhaps we may also see the sandy area as seperate from the grass and so on.

7.1.2 Examples of grouping in Computer Vision

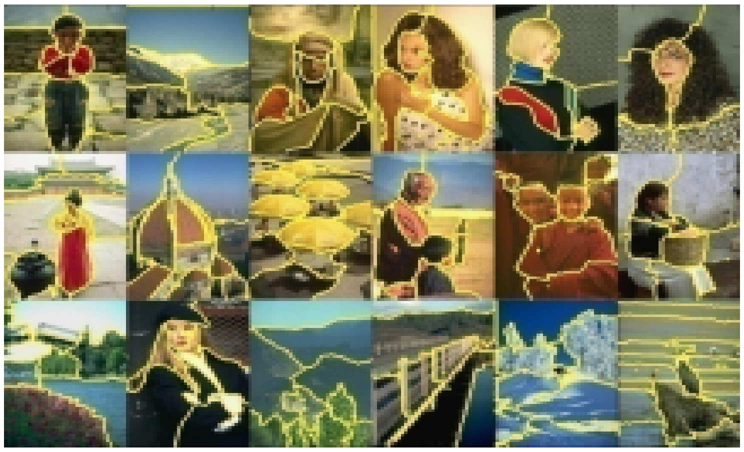


Figure 7.1.2: Determining image regions

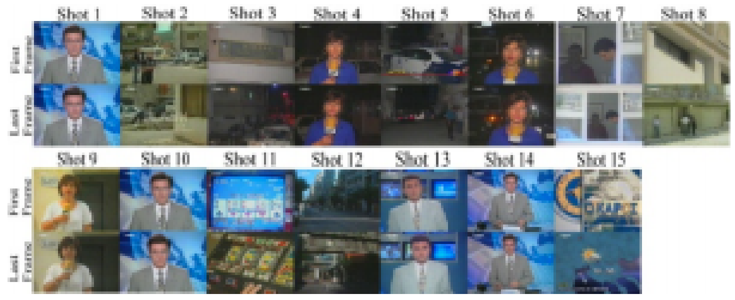


Figure 7.1.3: Grouping video frames into shots

7.1.3 Basics of grouping in human vision

As humans we are constantly grouping features together. In a similar way in computer vision we want to group together pixels that *belong together*. The Gestalt school from psychology offers some insight as to how humans group features together.

7.1.4 The Gestalt School

The Gestalt school informs us that grouping is key to visual perception. Consider the following image.

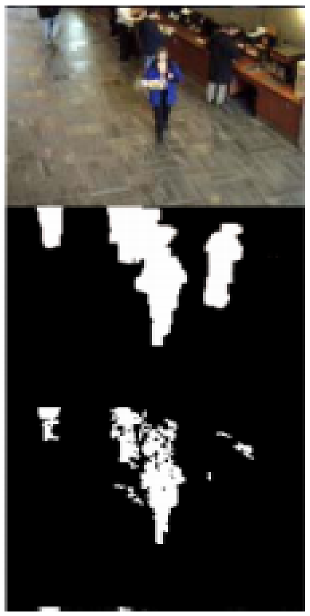


Figure 7.1.4: Background Subtraction

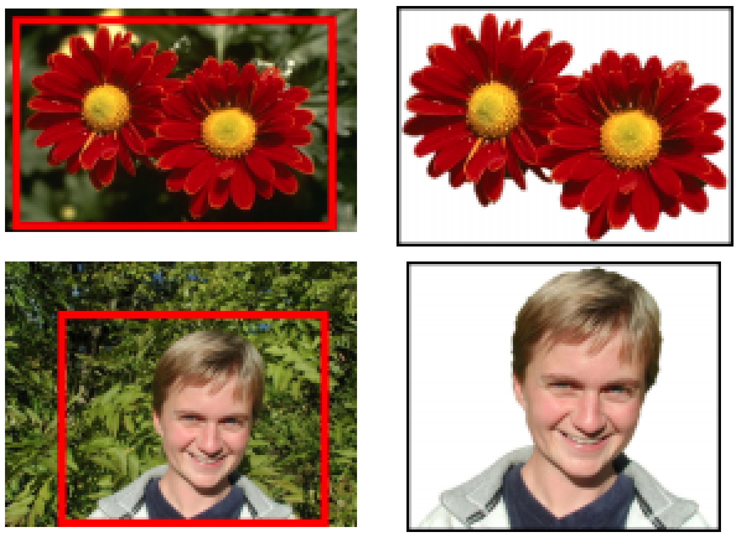


Figure 7.1.5: Image segmentation

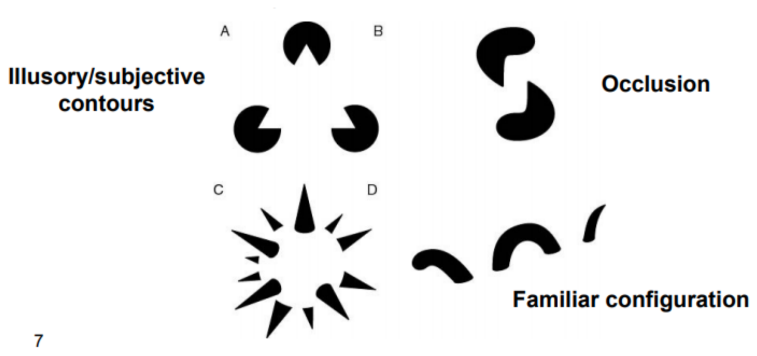


Figure 7.1.6: Gestalt School

Image A is simply three black circular objects. Despite this our brains can't help but segment these and thus perceive a triangle which isn't actually present. Similar C looks like a sphere despite it simply being black cones.

7.1.5 Gestalt Factors

The following are some factors that play into how we group features together.

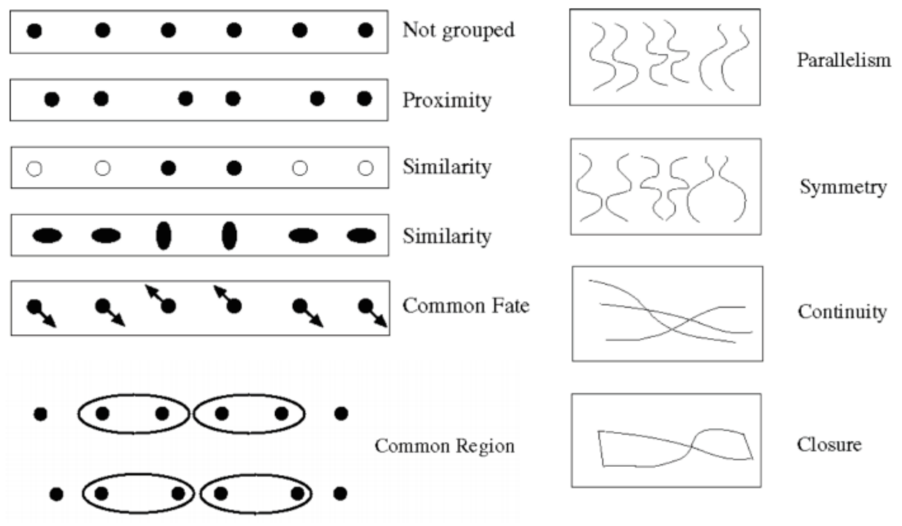


Figure 7.1.7: Gestalt Factors

7.1.6 Back to pixels

Our goal is to group (i.e. segment) pixels. We will do this with the Gestalt factors in mind, however we aren't able to incorporate all of them so we will focus on the simpler factors such as: similarity (pixel values), proximity (pixels close together), etc.

Let us define Segmentation more concretely.

Definition 7.1.1. *Segmentation*
Segmentation is the compact representation of image data in terms of a set of ***components****. Components share common* ***visual properties*** *and these properties can be defined at different levels of abstraction.*

7.1.7 The Goals of Segmentation

We wish to emphasise that the goal of segmentation is to separate an image into coherent objects. To motivate this, let us consider the following examples of human segmentation.

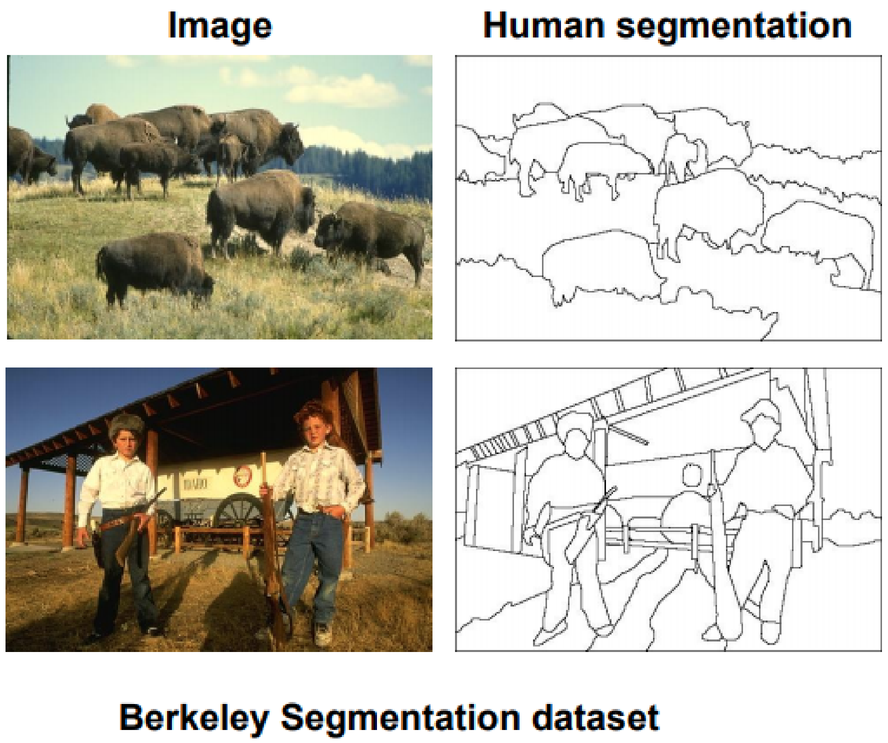


Figure 7.1.8: Human segmentation

Figure 7.1.9 shows us that humans don't even agree on how to segment an image.

Given this knowledge how can we expect a computer to do properly segment

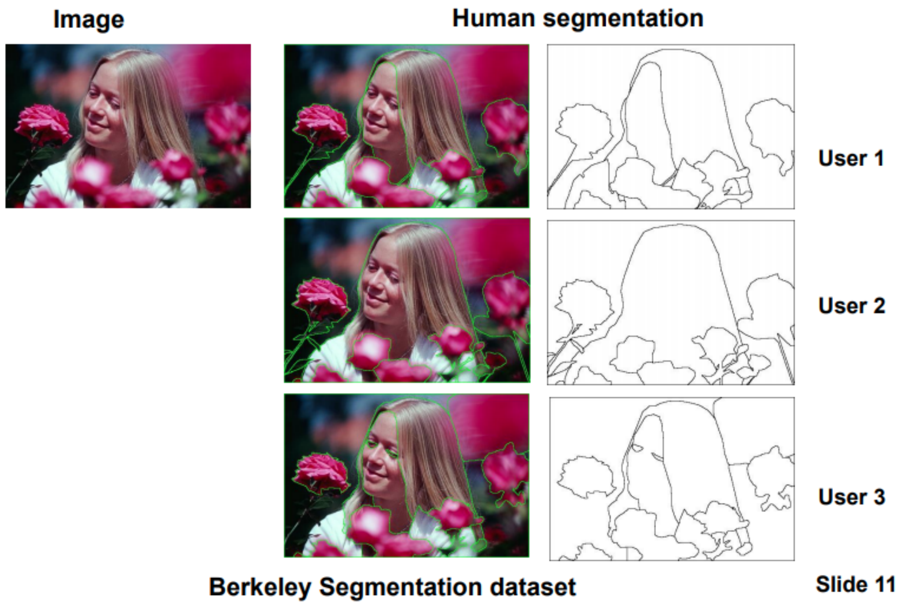
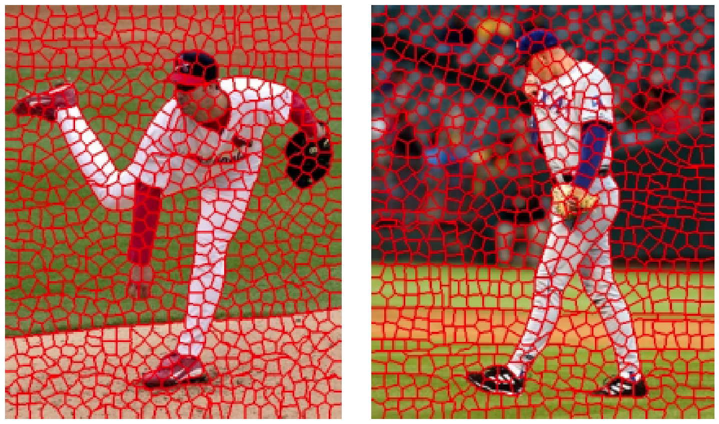


Figure 7.1.9: Human segmentation

an image? It turns out that we don't need a computer to do a perfect human segmentation. The type of segmentation depends on the particular task at hand.



X. Ren and J. Malik. Learning a classification model for segmentation. ICCV 2003.

Figure 7.1.10: Image segmentation based on similar pixels

Figure 7.1.10 shows segmentation using super pixels. This helped improve efficiency as rather than considering each and every pixel we can consider regions of super pixels.

7.2 Segmentation as clustering

Segmentation can be viewed as a clustering problem as we will see in this section.

7.2.1 Image Segmentation: Toy Example

Suppose we have the following input image with no noise (we can generate such an image using a computer).

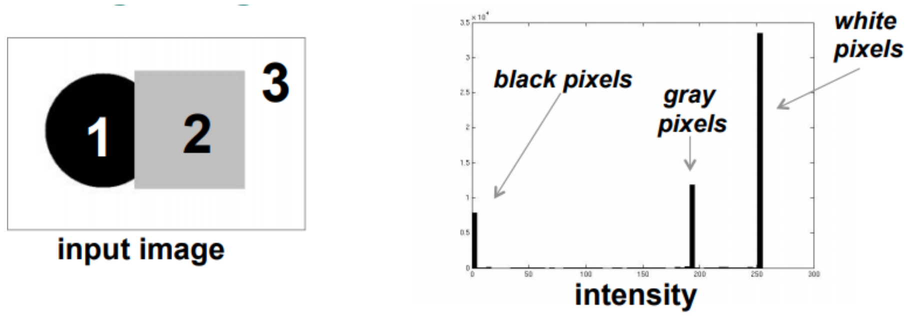


Figure 7.2.1: Input image without noise

Figure 7.2.1 is trivial to segment. We simply create a histogram of the pixel values and label the peaks. The three discrete peaks in the histogram forms our three groups and so we can label the black circle, the grey square and the white background as required.

However this situation is too artificial. We are more likely to have the following variation of the input image with noise.

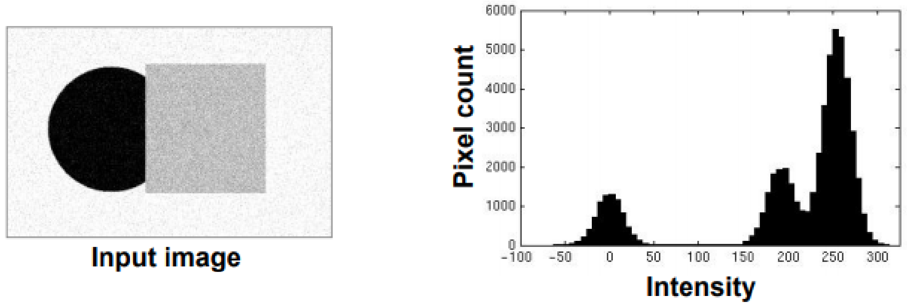


Figure 7.2.2: Input image with noise

As you can see the histogram isn't as discretised as before. For example the black pixels are now across a certain intensity range, i.e. they vary in terms of how **black** the pixels are. The same can be said about the white background and the grey square. As a result, in order to segment this new input image we need to **cluster**.

In order to appreciate clustering let us consider the histogram from Figure 7.2.2 in 1 dimension as follows.

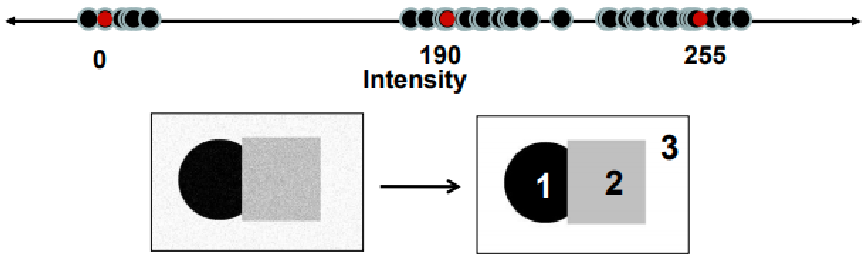


Figure 7.2.3: Clustering in 1D

Our goal here is to choose three **centres** from the intensity values to consider as centroids of our clusters.

We then associate pixels to a given cluster based on which centre it is closest to according to the following sum of squared distances (SSD) equation. The **best cluster centre** is therefore the ones that minimise the SSD between all pixel points, p, and a cluster centre, c_i.

$$\Sigma_p ||p - c_i||^2 \tag{1}$$

This allows us to split our points into three groups as shown in Figure 7.2.3.

7.2.2 Clustering

At this stage we have clusters but we may not have good representatives for each cluster. For example there may be a pixel that is exactly on the boundary between all the other clusters. Since we now know all the pixels that belong to each cluster, we recalculate a new centre for each cluster thanks to the calculations SSD computations we have just carried out.

We repeat this process of recalculating a new cluster centre and associating representatives based on Equation 1 until we found that the centres stop moving. Thus we arrive at the following clustering solution.

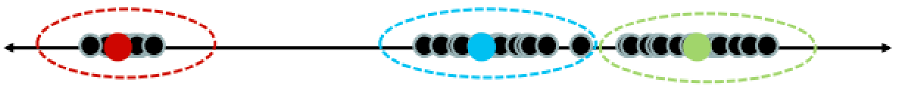


Figure 7.2.4: Finished clusters

We have so far essentially described the K-Means clustering method. We will now summarise it.

7.2.3 K-Means Clustering

Given pixel points that we wish to cluster, we start by arbitrarily initialising k cluster centres. Repeat the process of associating pixels to centres using Equation 1 and recalculating new cluster centres. We stop when the cluster centres stop changing. Thus the algorithm proceeds as follows.

K-Means Clustering: Algorithm

1. Arbitrarily initialise the cluster centres: $c_1, c_2, ..., c_k$.
2. For each pixel point p, associate p to a cluster c_i by calculating the sum of squared distances (SSD) (i.e. Equation 1) and associating p to the c_i that yields the minimum SSD. That is, put p in the i-th cluster that is closest to it.
3. Recalculate the centres of each cluster, c_i, by finding the mean of pixel points in c_i.
4. If c_i has changed, then go to step 2.

K-Means Clustering: Properties

- K-Means will always converge to *some* solution.
- The solution may be a local minimum i.e. not the optimal solution for the objective function given by Equation 1.

7.2.4 Feature Space: 1D

The type of feature space we use dictates the precise way we will group pixels together. Here we are using the familiar 1D feature space that we have been considering so far. In this 1D case we group pixels based on **colour similarity**. That is we cluster pixels together that are *close* in terms of their brightness or pixel value intensity.

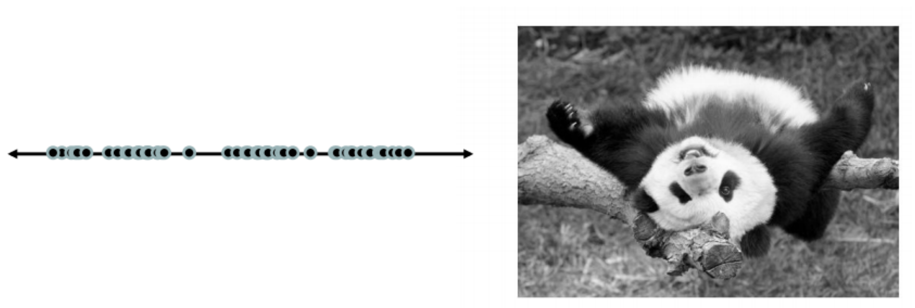


Figure 7.2.5: 1D Feature space clustering

The following shows the results of running K-Means on the image of the panda in Figure 7.2.5 with $k = 2$ and $k = 3$ clusters.

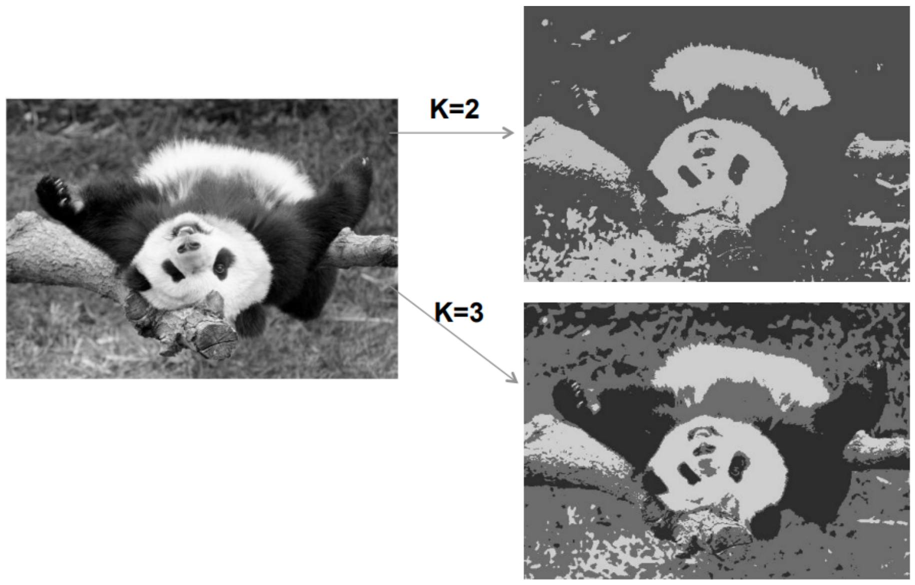


Figure 7.2.6: 1D Feature space with different clusters.

When $k = 2$ we essentially have a binary segmentation since we only have two clusters. We essentially obtain a binary black or white image. However when $k = 3$ when we are able to perhaps segment the image more appropriately.

7.2.5 Feature Space: 3D

Suppose we are now trying to run K-Means on an RGB image, our feature space is now actually 3D since we have three colour channels to consider.

This time we will group pixels based on **colour similarity**. For example, pixels that are *blueish* will be grouped together. The following is a demonstration of this in action.

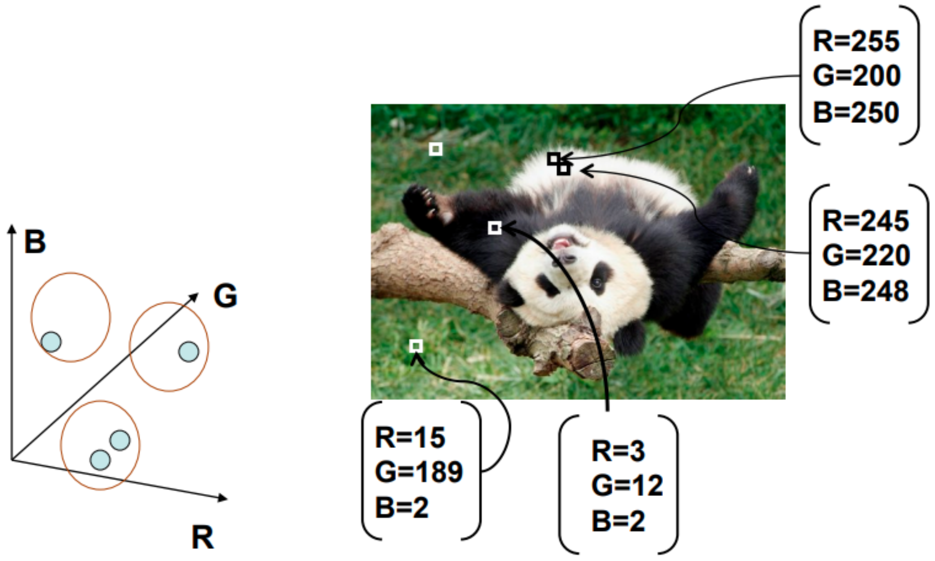


Figure 7.2.7: 3D Feature space

Notice that the limitation with working with only the 3D (r, g, b) feature space is that we cannot segment specific details. For example if we were only interested in the Panda's eyes then using only (r, g, b) would not allow us to adequately cluster in order to only obtain the eyes. To see this, consider the following figure.

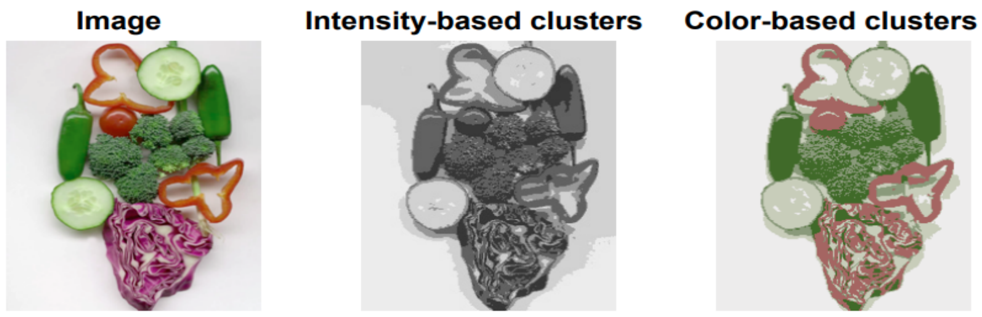


Figure 7.2.8: Intensity-based clustering vs Colour-based clustering

In Figure 7.2.8, the 1D intensity-based feature space and the 3D (r, g, b) colour-based feature space.

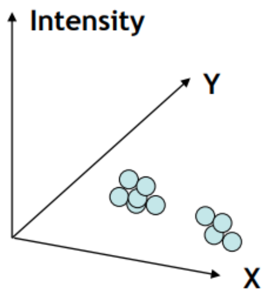
From this we can observe some problems with the using (r, g, b) feature space. Namely, the clusters that form are **not spatially coherent**. There are red pappers towards the middle-right, but there is also a red pepper and a red tomato towards the top.

For some applications, clustering based on (r, g, b) feature space alone may be desireable however we aren't able to for example pick out the two peppers separately. This is due to the lack of spatial coherence.

We can cluster using the 5D (r, g, b, x, y) feature space to enforce spatial coherence. Here (x,y) denotes the position of a pixel point.

7.2.6 Feature Space: Another 3D feature space

Let us now consider clustering the Panda image using the 3D (intensity, x, y). We group pixels based on **intensity and position** similarity.



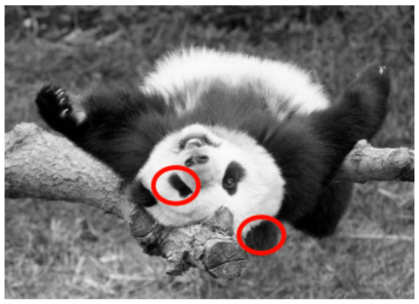


Figure 7.2.9: 3D (intensity, x, y) feature space

Using this feature space, we are able to cluster and pick out specific features such as the eye or ear. Hence we have encoded both similarity and proximity with this feature space.

7.2.7 K-Means Clustering: Summary

Pros

- K-Means Clustering is **simple and fast** to compute. It is a good starting point for clustering. K-Means is often used as a starting point for a more refined segmentation.
- K-Means is also guaranteed to converge, albeit only to a local minimum.

Cons

- It often isn't obvious how we should choose the number of clusters, k. Although there are algorithms that can help us with this choice.

- A big problem is that K-Means is very **sensitive to outliers** as shown in Figure 7.2.10 below which shows, with (A), K-Means undesirable clusters and, with (B), the ideal clusters we were looking for.

- **Converges to a local minimum** so the solution is starting point dependent. As a result we may have to run K-Means with different starting points and then choose the best solution that we get from all the different starting points.

- K-Means **detects only spherical clusters**. So if the dataset is not linear, as in Figure 7.2.11 (B), we get undesirable clusters. This is because of the distance metric that K-Means is using (Equation 1) which is the Euclidean distance.

  From the study of Metric Spaces, we know that the Euclidean metric yields circular neighbourhoods (i.e. *balls*) in 2D and spherical neighbourhoods in 3D.

- K-Means gives a binary YES/NO answer as to whether a point belongs to a cluster. Often in Computer Vision, we want to work with probabilities (i.e. 67% chance for class 1, 33% chance for class 2) so that we can evaluate the uncertainty later on.

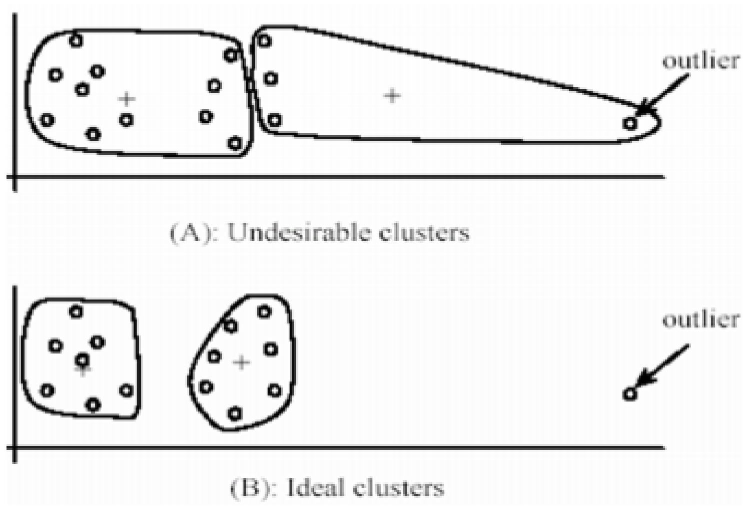


Figure 7.2.10: 3D (intensity, x, y) feature space

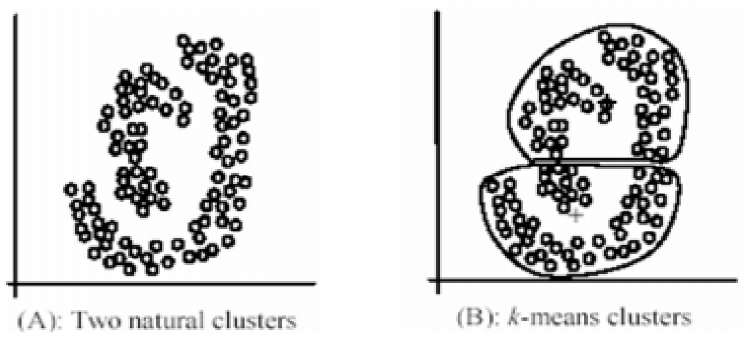


Figure 7.2.11: 3D (intensity, x, y) feature space

7.3 Probabilistic clustering

7.3.1 Motivation

Probabilistic clustering involves associating a notion of *likelihood* to how we associate a pixel point to a given cluster. As we have seen in the previous section, K-Means completely neglects this notion.

Probabilistic clustering addresses the folllowing questions which K-Means does not:

- What is the probability that a point p is in cluster m?
- What is the shape of each cluster?

The idea is to look at the data that we wish to cluster and consider whether the points fit a known continuous function such as the gaussian distribution.

This function is called our **generative model** since we assume that points in our dataset are generated by sampling this continuous function. The generative model is defined by a **vector of parameters**, θ, which encodes values such as position and mean.

7.3.2 Mixture of Gaussians

We will now consider the Mixture of Gaussians (MoG) method for probabilistic clustering. However, we will first need the following definition.

Definition 7.3.1. *Covariance*
Let x and y be vectors of length n.

Then the covariance of x and y is given by, cov(x, y) $= \Sigma_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n}$

Remark. *Covariance provides a measure of the strength of the correlation between two or more sets of random variates.*

This leads nicely to the following definition.

Definition 7.3.2. *Covariance Matrix*
Let $x = (x_1, ..., x_n)$ be a vector of length n.

Then the covariance matrix, denoted by Σ, for x is an $N \times N$ matrix that gives the covariance between each pair of elements of x. Σ is given by,

$$\Sigma = \begin{bmatrix} cov(x_1, x_1) & \cdots & cov(x_1, x_n) \\ \vdots & \ddots & \vdots \\ cov(x_n, x_1) & \cdots & cov(x_n, x_n) \end{bmatrix}$$

Remark. *The diagonal entries in the covariance matrix is the variance.*

The Mixture of Gaussians (MoG) is a generative model for clustering. We can use MoG in the following situaiton.

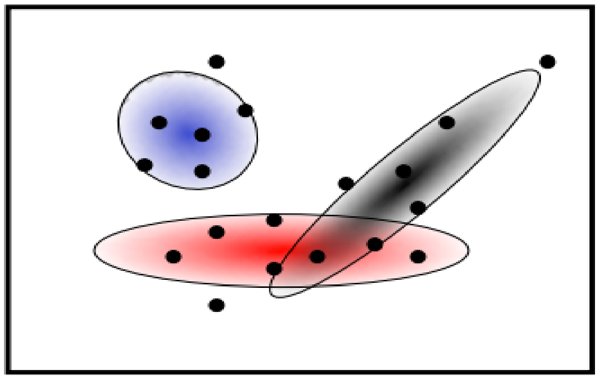


Figure 7.3.1: Mixture of Gaussians

Notice that the black points more or less follow a gaussian distribution. Hence we use the gaussian distribution as a model for generating each cluster. This results in us forming the three clusters as seen in Figure 7.3.1.

Computing the Gaussian Mixture

Let $x = (x_1, ..., x_d)$ be a vector of length d. We associate a blob, b, to a cluster k based on the following multivariate gaussian probability distribution:

$$P(x|\mu_b, \Sigma_b) = \frac{1}{\sqrt{(2\pi)^d det(\Sigma)}} e^{-\frac{1}{2}(x-\mu_b)^\top \Sigma^{-1}(x-\mu_b)} \tag{1}$$

Here μ_b denotes the mean of points in the blob b and Σ_b denotes the covariance matrix for the points in b.

The **likelihood** of observing the point x is given by a weighted mixture of gaussians as follows,

$$P(x|\theta) = \Sigma_{b=1}^{k} \alpha_b P(x|\theta_b), \text{where}\, \theta = [\mu_1, ..., \mu_d, \Sigma_1, ..., \Sigma_b] \tag{2}$$

7.3.3 Expectation Maximisation (EM)