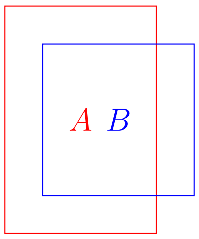
CO316 - Computer Vision (60006)

Lecture 1 - Introduction

Computer vision tries to build a system that can understand the world in a similar way to a human. At a higher level, the pipeline for vision consists of sensing an image or video, processing it, and then understanding it. For a human, the sensor is the eyes, and the processor is done by the primary visual cortex. On the other hand, a sensor can be a camera, or some form of medical imaging device, and the processor is the computer itself (and more importantly, the algorithm).

A **classification** problem has the goal of determining the **label** of what is in the picture. Classification is considered to be successful if one of the labels the algorithm predicts matches the true label. On the other hand, object **detection** attempts to draw a bounding box around an object (where are objects in the picture). We can quantify the success of detection based on the following. Consider the following, where the region in red is drawn by a human, and the region in blue is predicted by the algorithm;



We consider the detection of the intersection over union (IoU) is above 0.5;

$$\text{IoU} = \frac{A \cap B}{A \cup B} > 0.5$$

Another more complex piece of information we can extract is to perform **image segmentation**, allowing us to draw contours for each object.

Applications

Computer vision is used in our lives daily;

- **face detection**

  This can be noticed in most camera applications on modern smartphones, when a small box is drawn around faces. The algorithm first extracts **Haar** features from an image, and then determines (with these features) whether a region is a face or not.

  One example of these features is checking the contrast between the eyes and nose (horizontally); as the eyes tend to be quite dark in comparison. Another contrast is checked, this time between your eyes, as the nose tends to be brighter.

- **automatic number plate recognition**

  Automated barriers in parking lots can read number plates in order to calculate how long a car stays. Similarly, this can also be used to recognise building numbers, which is overlaid onto *Google Maps*, allowing for a large database of street numbers to be built in an automated fashion.

- **autonomous driving**
- **image style transfer**

  Choi et al. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation - used for changing features on inputs. Related to face motion capture (see *Face2Face*). Also see *DeepFake*.

- **Kinect**

  Works by taking a depth image, segmenting it into body parts, locating key points and building a skeleton.
- **design**

  See *OpenAI*'s *DALL-E*, combining NLP and computer vision by generating images based on the concepts of words in a sentence.
- **healthcare**

  Medical image analysis can be used for disease diagnosis. For example, identifying breast cancer lesions from mammograms.

Lecture 2 - Image Formation

An image, in RGB format, can be represented as pixels, each being three numbers. A digital image is formed from a lighting source being reflected into an optics sensor (eyes, cameras, etc).

Light

A **point light source** originates from a single location in space, such as a small light bulb, or the sun. This can be described with three properties; location, intensity, and the spectrum.

On the other hand, an **area light source** is more complex. For example, this could be a ceiling light; a rectangle of point lights.

Reflectance

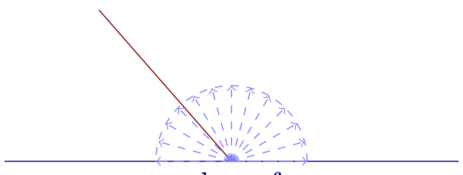
When light emitted from the source hits the surface of an object, it will be reflected. To describe this, we typically use the **bidirectional reflectance distribution function (BDRF)** to model this behaviour (where λ is the wavelength, L_r is the output power, and E_i is the input power);

$$f_r(\underbrace{\theta_i, \varphi_i}_{\text{incident}}, \underbrace{\theta_r, \varphi_r}_{\text{reflected}}, \lambda) = \frac{\mathrm{d}L_r}{\mathrm{d}E_i}$$

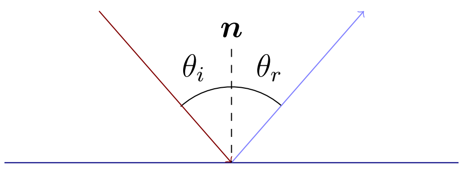
While this is a very general model, it is very complex.

As such, we can use **diffuse reflection**, where light is assumed to be scattered uniformly in all directions. This has a constant BRDF - this says that regardless of the incident or reflected directions, nor the wavelength, the power will be constant;

$$f_r(\theta_i, \varphi_i, \theta_r, \varphi_r, , \lambda) = f_r(\lambda)$$



On the other hand, we can use **specular reflection** which performs reflections in a mirror-like fashion. The reflection and incident directions are symmetric with respect to the surface normal $\boldsymbol{n}$, such that $\theta_r = \theta_i$, with the same amount of power;



While these two are the **ideal** cases, the majority of cases, we see a combination of both of those, as well as **ambient** illumination. Ambient illumination accounts for general illumination which could be complicated to model. For example, these could be repeated reflections between walls (which would be very difficult to calculate), and we instead assume that there is some light that exists in the 3D space representing the room. Another example could be a distance source, such as the sky (which has atmosphere).

Combining these, we can use the **Phong** reflection model. This is an empirical model that describes how a surface reflects light as a combination of ambient, diffuse, and specular components.

'Duality' with Computer Graphics

Using the game engine to produce example images is useful, as we are able to directly obtain the labels of objects from the engine itself, as well as visual output. As such, we can use these images as training for a model, since we also have an associated label map. This synthetic data is complementary to time-consuming manual annotations.

Optics and Sensors

Both our eyes and cameras work in similar ways, with a lens governed by the thin lens equation, where f denotes the focal length of the lens, u denotes the distance from the subject to the lens, and v denotes the distance from the lens to the image;

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

Our eyes work by light rays being focused by the cornea and lens onto the retina, where vision begins with two neural cells. The **cone** cells are responsible for colour vision, and function in bright light. On the other hand, the **rod** cells have little role in colour vision, but function in dim light.

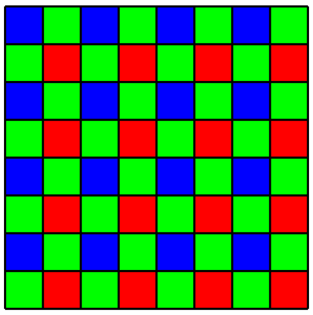
Humans have three types of cone cells (**trichromatic vision**), which have different response curves. The short cone cells respond to short wavelength lights (violet, blue), whereas the medium cone cells respond to medium wavelength lights (green), and long cone cells respond to long wavelength lights (red). Occasionally, there may be two cone cells, or four, which are referred to as **dichromacy** or **tetrachromacy** respectively.

Note that colours are not objective physical properties of light or electromagnetic wave (which have a physical property of wavelength). Colour is a subjective feature, dependent on the visual perception of the observer. Since **rod** cells are more sensitive to light, they are the primary source of visual information at night.

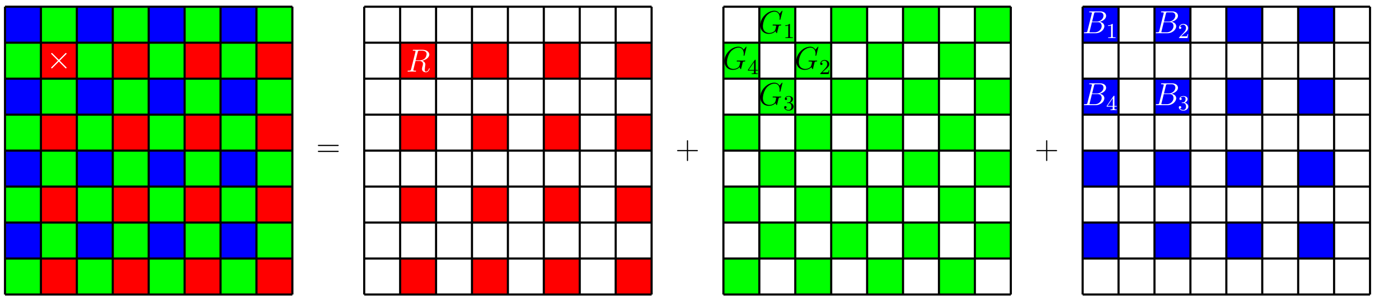
On the other hand, camera sensors have two common types;

- **CCD** (charged-coupled device) — often used in handheld cameras
- **CMOS** (complementary metal-oxide semiconductor) — used by most smartphone cameras

These sensors convert incoming light into electron charges, which are then read. **Bayer** filter arrays are a way to arrange RGB filters on sensors, half of which are green, and the remaining two quarters are red and blue. This mimics the human eyes, which are most sensitive to green light;



CMOS works by having sensors underneath each of these filtered portions, which can report an electrical signal. However, note that only one colour is available at each pixel (therefore the rest must be interpolated from the neighbours, by using bilinear interpolation; which simply averages the 4 neighbours). For example, consider the following pixel (denoted as a white cross);



$R_\times = R$ $\qquad G_\times = \frac{G_1+G_2+G_3+G_4}{4}$ $\qquad B_\times = \frac{B_1+B_2+B_3+B_4}{4}$

Note that the use of different filters, and this interpolation, can lead to slightly different colours between cameras.

Image Representation

The earliest colour space was described in 1931 by CIE, by performing a colour matching experiment. In this experiment, an observer attempts to match different levels of red, green, and blue lights to match a target light. This allows for colours to be represented in 3D space, as (X, Y, Z), corresponding to the different levels. Colours can also be represented on a 2D plane, by normalising brightness;

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \\ z &= \frac{Z}{X+Y+Z} \\ &= 1 - x - y && \text{therefore redundant} \end{aligned}$$

Here X, Y, Z are primary colours (R, G, B), and x, y are chromacity / colour after removing brightness. This is much easier to draw. However, this colour space, also known as the **gamut** of human vision, was invented before computer screens.

The sRGB (standard RGB) space was created by *HP* and *Microsoft* in 1996 for use on monitors, printers, and the internet.

sRGB definition	x	y
red	0.64	0.33
green	0.30	0.60
blue	0.15	0.06

This is represented by a triangle (which is a subset) in the gamut of human vision. As this is a subset, it cannot produce all the colours visible by the human eye.

There are other colour spaces, such as HSV, CMYK, and so on. Note that CMYK is a **subtractive** colour model, starting from white, whereas RGB is an **additive** colour model, where we start from black. There can also be an alpha channel in RGB, which represents transparency. In a greyscale image, the three components are equal, hence only require one number.

Quantisation

Note that this is covered in lecture 3.

Quantisation maps a continuous signal to a discrete signal. The pictures from a camera are a continuous signal, but when it is stored on the camera, it is quantised to a discrete signal. Numerical errors can occur during this process, and the magnitude of the errors depends on the number of bits used (less error with more bits; 16 bits can store from 0 to 65535, compared to 8 bits storing from 0 to 255).

Physically, an analog-to-digital convert (ADC) is used to perform the conversion. The energy of photons are converted into voltage, amplified, and then converted.

Compression

In order to reduce the cost of storage to transmission, compression may be used. Lossy compression loses information after the compression (such as discrete cosine transform (DCT) in JPEG, often used for images or videos). However, lossless compression can also reduce the file size (less efficient compared to lossy), and is preferred for archival purposes or important imaging, where detail needs to be recovered.

Lecture 3 - Image Filtering I

Note that in this course, most of the examples will be done on greyscale images, but can be applied to the channels individually. Some examples of filters include;

- identity filter

  Does nothing to the image.
- low-pass / smoothing (moving average, Gaussian)

  Removes high-frequency signals, and keep low-frequency signals.
- high-pass / sharpening

  Similar to the previous filter (keeps high-frequency signals, and removes low-frequency).
- denoise (median, non-local means, block-matching and 3D filtering)

Moving Average Filter

This is commonly used for 1D signal processing (time series), such as stocks, which can be quite noisy. To smooth out a noisy curve, it moves a window across the signal (and calculates the average value within the window) - the larger the window size, the smoother the result.

In a two dimensional case, we can use a **filter kernel** (for example, with a 3×3 kernel);

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

At each pixel, we apply the kernel centred at the pixel, and take the average of the pixels around it (and itself) to create a new output image. When we blur with a larger kernel, such as 7×7, we end up with a blurrier image. Note that the output image is smaller than the input image. We can pad the image with zeroes (anything outside of the picture is 0), or by mirroring the pixels (copying the boundary pixels).

Consider an image of size $N \times N$, and a kernel size of $K \times K$. At each pixel, we perform K^2 multiplications (by the kernel weights), and then $K^2 - 1$ summations. This has to be done for each pixel, hence N^2 times. Therefore this results in N^2K^2 multiplications and $N^2(K^2 - 1)$ summations; giving a **complexity** of $O(N^2K^2)$. However, we'd like to reduce this, if possible.

If a big filter can be separated as two filters (**separable filter**) applied consecutively, we can perform the first operation, and then the second. An average in a 2D window can be done as an average across rows (horizontal), and then an average across columns (vertical);

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} * \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$$

Doing these two filters, we end up with an equivalent result to the original 2D filter. Note that $*$ is a convolution (see next lecture).

Consider the complexity of separable filtering. The image size remains as $N \times N$, however we have two kernels, of $1 \times K$ and $K \times 1$ respectively. At each pixel we do K multiplications followed by $K - 1$ summations. Again, this is done for N^2 pixels, and twice (once for each filter). Therefore, the total number of multiplications is $2N^2K$ multiplications and $2N^2(K - 1)$ summations. This is better, in contrast to the original complexity, as we have complexity of $O(N^2K)$ - which will make a difference for large K.

A moving average filter removes high frequency signals (noise or sharpness), which results in a smooth but blurry image.

Gaussian Filter

The kernel is a 2D Gaussian distribution;

$$h(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

Here we have $i, j = 0, 0$ as the centre of the kernel. While the support is infinite, small values outside the range $[-k\sigma, k\sigma]$ can be ignored (very small values, such as $k = 3$ or $k = 4$). Note that σ is a manually defined parameter. This is a separable filter, which is equivalent to two 1D Gaussian filters with the same σ, with one along the x-axis and the other along the y-axis;

$$\begin{aligned} h(i, j) &= h_x(i) * h_y(j) \\ h_x(i) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{i^2}{2\sigma^2}} \end{aligned}$$

High-pass Filter

One design is to do the following;

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{identity}} + \underbrace{\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \right)}_{\text{high-frequency}} = \begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{17}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$$

We can add a high frequency signal to the identity, in order to enhance it.

Median Filter

This is a non-linear filter (not performing an average calculation by multiplication). This moves a sliding window, and replaces the centre pixel with the median value in the window - this is not a linear equation.

Lecture 4 - Image Filtering II

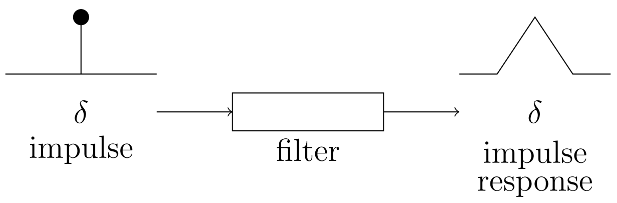
Mathematical Description

Consider a simple filter, of size 3, in 1D. With an input f, and an output g, it can be written as the weighted average in a window;

$$g[n] = \frac{1}{3}f[n-1] + \frac{1}{3}f[n] + \frac{1}{3}f[n+1]$$

In general, filtering takes in an input signal f, processes it and generates an output signal g. A filter is a device (or process) that removes unwanted components or features from a signal (keeps / enhances wanted filters).

In order to mathematically describe a filter, we need the concept of **impulse response**; the output of a filter when the input is an **impulse signal** (only have a signal at a single time point);



For a continuous signal, we treat an impulse as a Dirac delta function $\delta(x)$, whereas for a discrete signal, we treat an impulse as a Kronecker delta function $\delta[i]$;

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1$$

$$\delta[i] = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases}$$

The impulse response h completely characterises a **linear time-invariant** filter. Note that we can consider a filter as time-invariant if, by shifting the input signal some number of time steps k, the output signal will remain the same (shape and values) but shifted by the **same** number of time steps. For example;

- $g[n] = 10 \cdot f[n]$ is time-invariant and amplifies the input by a constant
- $g[n] = n \cdot f[n]$ is **not** time-invariant since the amount it amplifies the input depends on the time step n

As long as we know h, and have an input x, we can calculate the output signal y. Since it uniquely describes a filter, we often denote a filter by its impulse response function h.

Additionally, a filter can be **linear**. If it is a linear system, when two input signals are combined linearly, their outputs will also be combined linearly. Let $f_1[n]$ lead to an output $g_1[n]$, and $f_2[n]$ to $g_2[n]$.

$$\text{output}(\alpha f_1[n] + \beta f_2[n]) = \alpha g_1[n] + \beta g_2[n]$$

Convolution

Most of the filters we've previously covered are linear time-invariant. Since h characterises how the system works (as it is linear time-invariant), it's possible for the output g to be described as the **convolution** between an input f and impulse response h;

$$g[n] = f[n] * h[n]$$

We can describe an input signal $f[n]$ as the following, where each time step is a constant multiplied by a spike;

$$f[n] = f[0]\delta[n] + f[1]\delta[n-1] + f[2]\delta[n-2] + f[3]\delta[n-3] + \dots$$

However, since we know the output of $\delta[n]$ is $h[n]$, we can write the output as;

$$g[n] = f[0]h[n] + f[1]h[n-1] + f[2]h[n-2] + f[3]h[n-3] + \dots$$

The output mathematical operation is defined as a convolution, where a signal f and a filter with impulse response / convolution kernel h is defined as;

$$g[n] = f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[m]h[n-m]$$

The continuous form (previously we have only considered the discrete form);

$$g(t) = f(t) * h(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)\,\mathrm{d}\tau$$

Focusing on the discrete case, we notice the following (when we replace m with $n - m$);

$$\sum_{m=-\infty}^{\infty} f[m]h[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]h[m]$$

This shows that the convolution of f and h is equivalent to the convolution of h and f (commutativity);

$$f[n] * h[n] = h[n] * f[n]$$

By expanding the equations, we can also show that convolution satisfies associativity, such that;

$$f * (g * h) = (f * g) * h$$

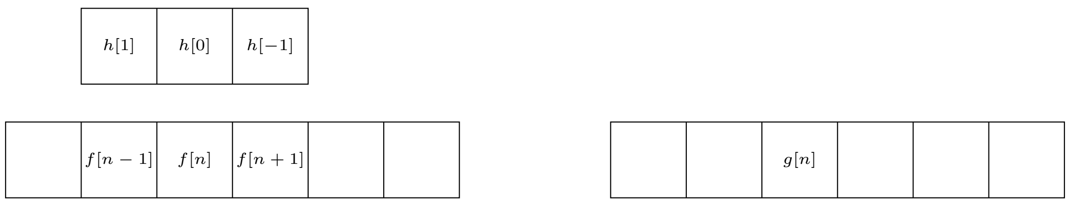
We also have distributivity;

$$f * (g + h) = (f * g) + (f * h)$$

And also differentiation;

$$\frac{\mathrm{d}}{\mathrm{d}x}(f * g) = \frac{\mathrm{d}f}{\mathrm{d}x} * g = f * \frac{\mathrm{d}g}{\mathrm{d}x}$$

With a more concrete example, we can visualise it as follows (this is the moving average of size 3);



In our case, we have the kernel $h[n]$, and the values;

$$\begin{aligned} h[-1] &= \frac{1}{3} \\ h[0] &= \frac{1}{3} \\ h[1] &= \frac{1}{3} \\ h[n] &= \left[\tfrac{1}{3},\ \tfrac{1}{3},\ \tfrac{1}{3}\right] \end{aligned}$$

This can be expanded into the 2D case, which is used for image filtering;

$$g[m,n] = f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j]$$

This can also be written as the following, replacing $m-i, n-j$ by i, j;

$$g[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[m-i,n-j]h[i,j]$$

By using the property of associativity, if a big filter (call it f_b) can be written as the convolution of g and h (smaller filters), we can first convolve f with g, then with h - this is used for separable filtering;

$$f * f_b = f * (g * h) = (f * g) * h$$

For example (note that we have padded zeroes, but in code we do not need that);

$$\underbrace{\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}}_{f_b} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \end{bmatrix}}_{g} * \underbrace{\begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix}}_{h}$$