Computer Vision

Matteo Galiazzo

May 27, 2025

Contents

1 Introduction

The difference between computer vision and image processing is the fact that computer vision is the process of extracting information from images, while image processing aims at improving the quality of images. Quite often image processing helps computer vision. The informations we want to extract from the images could be counting, object orientation, object classification, measurements... Computer vision is challenging since we lose depth from the images (3D information becomes 2D), scale and illumination varies, and there's object occlusion (object hiding other objects).

Computer vision started with hand-crafted decision rules that only required few images as example, and evolved to machine learning where the algorithm learns a decision rule, but the training requires hundreds to thousands of images. The big paradigm shift happened with deep learning (e2e learning) which learned both the image representation and the decision rules, but required thousands to millions of words. Deep learning has been enabled by better networks, better hardware and more data.

2 Fundamentals of Image Processing and Computer Vision

2.1 Images

An imaging device gathers the light reflected by 3D objects to create a 2D representation of the scene.

2.1.1 Pinhole camera model

The "pinhole camera" is the simplest camera model we can define. Light goes through the very small pinhole (to not have saturation) and hits the image plane. Geometrically, the image is achieved by drawing straight rays from scene points through the hold up to the image plane.

This simple geometrical model turns out to be a good approximation of the geometry of image formation. However, useful images can hardly be captured by means of a pinhole camera.

The **geometric model of image formation** in a pinhole camera is known as **perspective projection**.

M : scene point
m : corresponding image point
I : image plane
C : optical centre (pin hole)
Optical axis: line through C and orthogonal to I
c : intersection between optical axis and image plane (image centre or piercing point)
f : focal length
F : focal plane

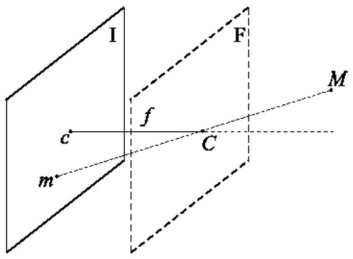


Figure 1: perspective projection

Which, by writing the points as vectors and the plane's coordinates, becomes the geometric model in the image 2

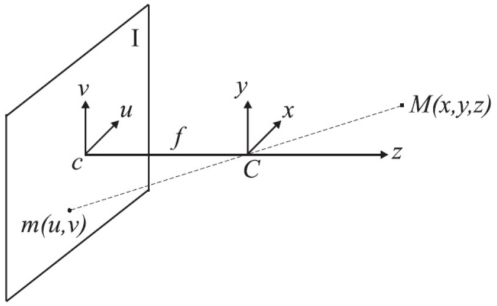


Figure 2: geometric model of image formation

Given the reference frame in the image 2

- u is the horizontal axis in the image plane.
- v is the vertical axis in the image plane.
- x and y are the respective axis in the 3D reference system. It's called the **camera reference system** because it is "attached" to the camera.

For the perspective model these axis must be parallel

The equations to map scene points into their corresponding image points are defined as:

$$\frac{u}{x} = -\frac{f}{z} \rightarrow u = -x\frac{f}{z} \qquad \frac{v}{y} = -\frac{f}{z} \rightarrow v = -y\frac{f}{z}$$

The minus sign means the axis gets inverted (as we can see in the visualization, and it's what happens in the brain). We can get rid of the sign, since the image plane can be though of as lying in front rather than behind the optical centre.

Image coordinates are a scaled version of scene coordinates (function of depth). When z increases, since it's at the denominator in both the equations, the terms gets smaller (object gets smaller in the image). When f increases, since it's at the numerator in both the equations the term gets bigger (object gets bigger in the image)

As we previously said, the image formation process deals with mapping a 3D space onto a 2D space, and so to the loss of depth information. A given scene point is mapped into an image point, but an image point is mapped onto a 3D line. For an image point we can only state that its corresponding scene point lays on a line, but cannot disambiguate a specific 3D point along the line.

2.1.2 Stereo images

We use multiple images to create stereo vision. Given correspondences, 3D information can be recovered easily by triangulation. We can use two cameras, or two cameras and an infrared sensor to project guides to align the images.

For standard stereo geometry there are some assumptions we have to make:

- The cameras have parallel (x, y, z) axes.
- The image planes of both cameras are coplanar and aligned.
- Both cameras have identical focal lengths.

Based on this, the transformation between the two reference frames is just a translation, usually horizontal. For stereo vision is also really important to **sense two images at the same moment**.

The cameras are displaced at a given quantity b called baseline. The **disparity** is the difference between the horizontal coordinates in the left and right images.

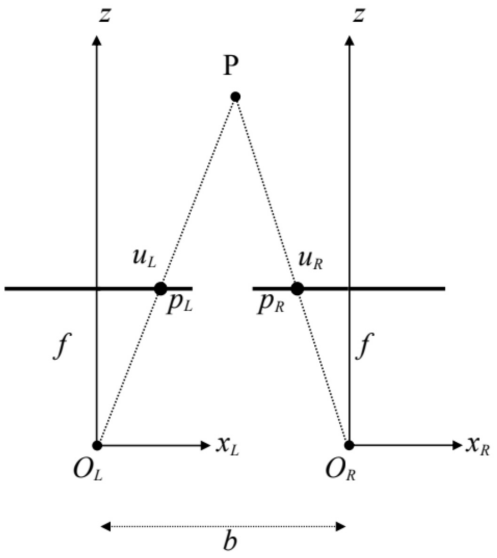


Figure 3: standard stereo geometry

In standard stereo geometry since we are given just two 2D images there is no info about the correspondence between two points in the two images. We can recall that the camera have parallel axes, and so we know that we can search for the correspondence along the horizontal lines. This task is called stereo matching.

2.1.3 Stereo correspondence

In stereo correspondence, given a point in one image, we have to find it in the other image which is the projection of the same 3D point. Such image points are called corresponding points. **Points farther away have a smaller disparity, while close points have a larger disparity**.

Figure 4: Corresponding points look similar in the two images

The image of a 3D line segment of length L lying in a plane parallel to the image plane at distance z from the optical centre will exhibit a length given by:

$$l = L\frac{f}{z}$$

This relationship is more complicated for an arbitrarily oriented 3D segment, as its position and orientation need to be accounted for as well. For a **given position and orientation, length always shrinks alongside distance**.

Perspective projection maps 3D lines into image lines. **Parallelism between 3D lines is not preserved** (except for lines parallel to the image plane). This is the reason why if we look at a really long road into the distance we have the perception that the road becomes thinner, and the lines of the road intersect in the distance. The images of parallel 3D lines intersect at a point, called **vanishing point**, which isn't necessarily within the image.

If the lines are parallel to the image plane they meet at infinity.

2.1.4 Epipolar geometry

What if the two cameras are no longer aligned? Do we need to search through the whole image? We can project the line related to point P_L in the right plane and search across that line. The issue is that this projection can be computed only if the transformation between the two cameras is known (the relative mapping between the two cameras).

It is almost impossible to build a stereo rig which is perfectly aligned horizontally. Searching through oblique epipolar lines is awkward, and computationally is less efficient. What people do in practice is to convert epipolar geometry to standard geometry with rectification/warping. We warp the images as if they were acquired through a standard geometry, then we can compute and apply to both images a transformation known as rectification.

2.1.5 Depth of Field (DOF)

A scene point is on focus when all its light rays, gathered by the camera, hit the image plane at the same point. In a pinhole device this happens to all scene points because of the very small size of the hole, so that the camera features an infinite Depth of Field (DOF).

The drawback is that such a small aperture allows gathering a very limited amount of light. The image is really sharp, but has a really low light. If a point is projected onto a circle instead of a point (bigger pinhole) the image is not sharp (not on focus). If we cannot gather enough light through the aperture we have to integrate through time, by using a longer exposure time.

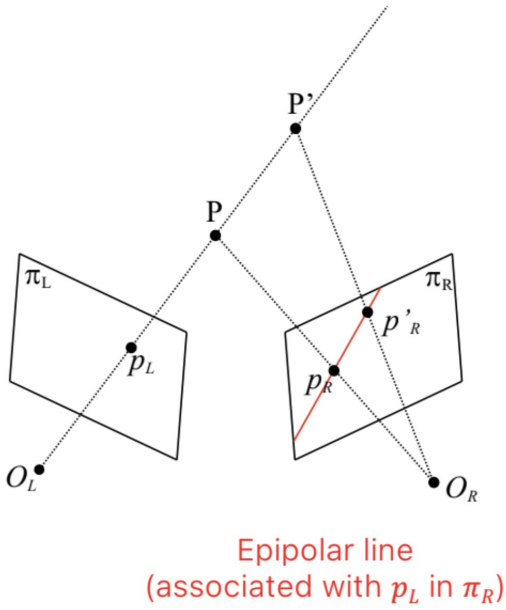


Figure 5: Epipolar line

2.1.6 Lenses

Lenses concentrate light, so we use them to gather more light from a scene point and focus it on a single image point. This enables much smaller exposure times. This way Depth Of Field is no longer infinite, and only a limited range of points can be simultaneously on focus in a given image.

We will consider the approximate model known as thin lens equation, which is useful to graphically determine the position of a focused image point:

- Rays parallel to the optical axis are deflected to pass through F.
- Rays through C are undeflected.

If the image is on focus, the image formation process obeys to the perspective projection model:

- The center of the lens is the optical center.
- The distance v acts as the effective focal length of the projection.

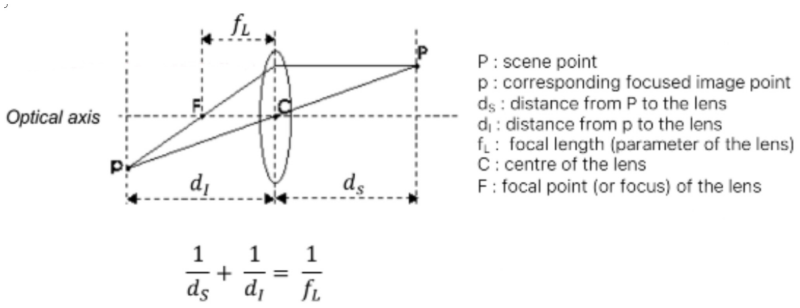


Figure 6: Scheme of a lens

Choosing the distance of the image plane determines the distance at which scene points appear on focus in the image. Scene points in front and behind the focusing plane will result out-of-focus, thereby appearing in the image as **blur circles**, rather than points.

The advantage of lenses is to have a small exposure time for capturing moving objects but we pay in terms of Depth Of Field.

As we can see in 2.1.6 having a small aperture (pinhole camera model) results in everything being on focus, but we need lots of light, or the image will be dark, since few light can enter in the sensor. We could also increase our exposure time to let more light in, but in that case we need the objects in our image to be still. Having a bigger aperture results in more light coming in in a small time fraction, but the lens distortion causes the light to be focused only at certain distances, and creates blur cirlces in other zones, so we get a lower depth of field, as we can see in 7.

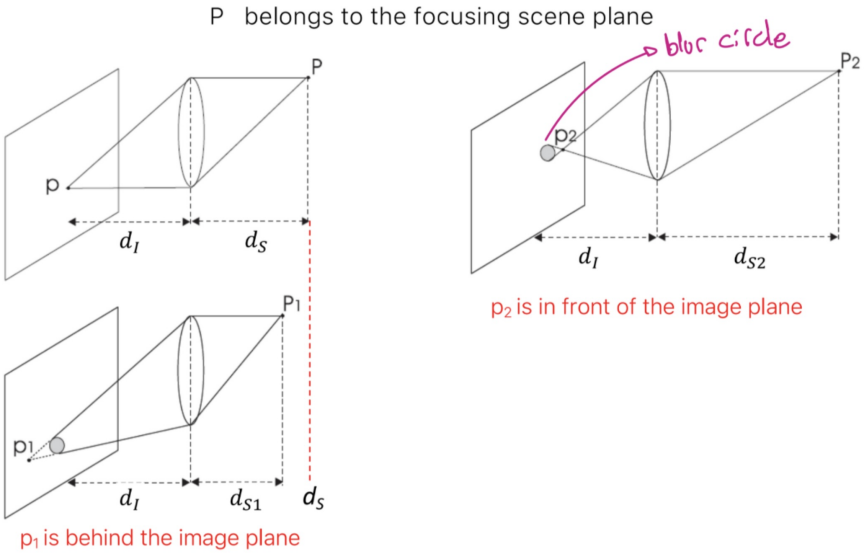
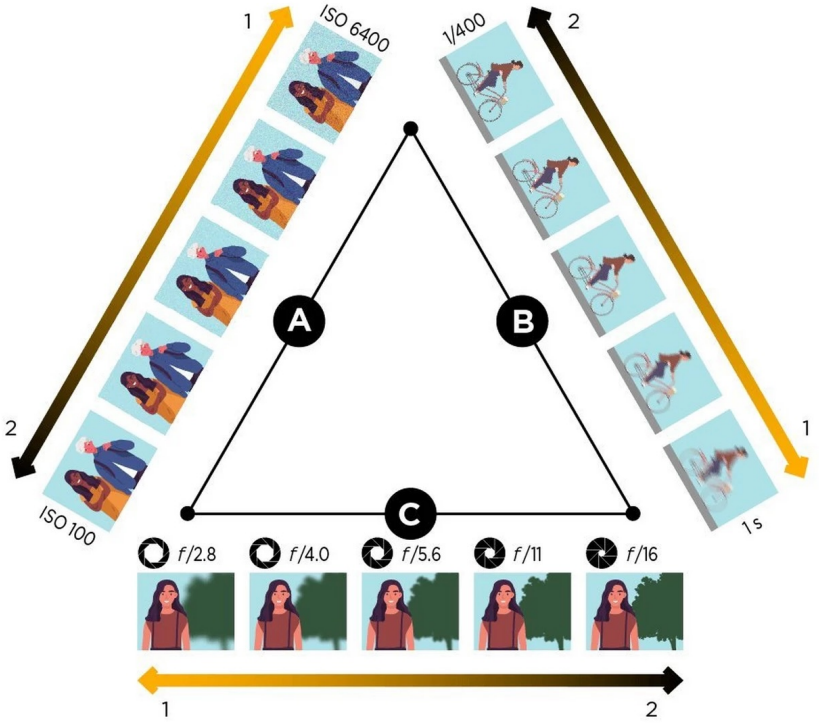


Figure 7: Lens focusing at various distances



2.1.7 Diaphragm

In theory, when imaging a scene through a thin lens, only the points at a certain distance can be on focus, all the others appear blurred into circles. However, as long as the circles are smaller than the size of the photosensing elements (a single pixel), the image will still look on-focus.

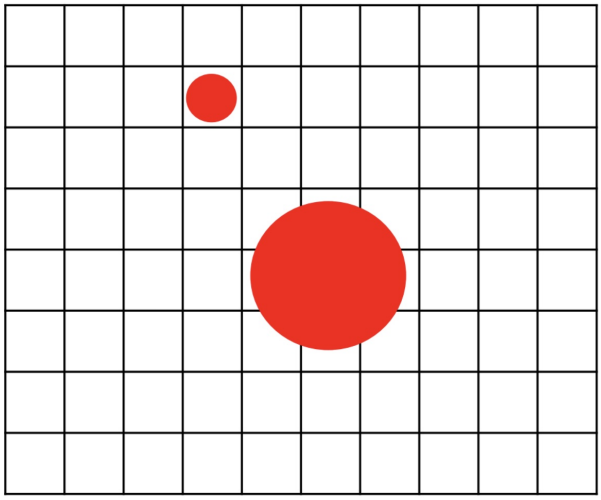


Figure 8: Blurring at pixel level

The range of distances across which the image appears on focus, due to blur circles being small enough, determines the (Depth Of Field) of the imaging apparatus. Cameras often deploy an adjustable diaphragm (iris) to control the amount of light gathered through the effective aperture of the lens.

- Reduce aperture → less light → smaller blur circle
- More aperture → more light → larger blur circle.
- Close the diaphragm → increase depth of field → not enough light → increase exposure time → moving object → motion blur.

2.1.8 Focusing mechanism (manually changing depth of field)

To focus on objects at diverse distances we need a mechanism that allows the lens (or the lens subsystem) to translate along the optical axis with respect to the fixed position of the image plane.

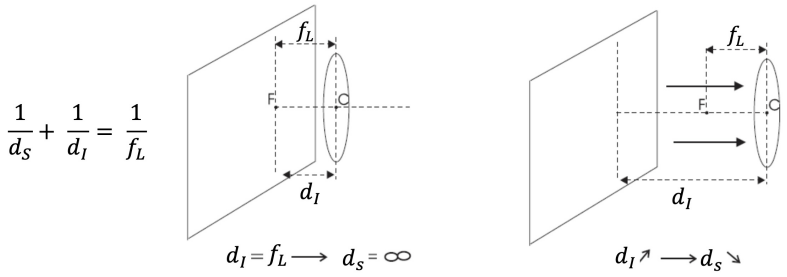


Figure 9: Focusing mechanism

At one end position ($d_I = f_L$) the camera is focused at infinity (objects at inifity are on sharp focus). The focusing mechanism allows the lens to be translated farther away from the image plane up to a certain maximum value, which determines the minimum focusing distance.

2.1.9 Image digitization

How do we convert a continuous image to a discrete one which can be represented on a computer? The process can be divided in two steps: sampling and quanitzation.

- **Sampling**: the planar continuous image is sampled along both the horizontal and vertical directions to pick up a matrix of $N \times M$ samples (pixels).

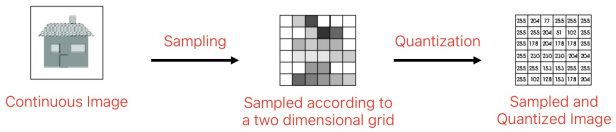


Figure 10: The two steps of the image digitization process

- **Quantization**: the continuous range of values associated with pixels is quantized into $l = 2^m$ discrete levels known as gray-levels. m is the number of bits used to represent a pixel, with the memory occupancy (in bits) of a gray-scale image given by $B = N \times M \times m$. Coloured digital images are typically represented within computers using 3 bytes per pixels. Both more pixels and more bits per pixel result in a higher quality image.

The more bits we spend for its representation, the higher the quality of the digital image (it becomes a closer approximation to the ideal continuous image). This applies both to sampling and quantization.

2.1.10 Camera sensors

The sensor is a matrix of photodetectors. During exposure time, each detector converts the incident light into a proportional electic charge. The companion circuitry reads-out the charge to generate the output signal, which can be either digital or analog. For digital cameras the sensor includes the necessary ADC circuitry.

Today, the two main sensor technologies are:

- **Charge Coupled Devices (CCD)**, where the sensor and circuit are separated.
- **Complementary Metal Oxide Semiconductor (CMOS)**, where everything is on the same circuit.

CCD/CMOS sensors can't sense colors, so we place an array of optical filters in front of the photodetectors, to render each pixel sensitive to a specific range of wavelengths.

2.1.11 SNR

The intensity measured at a pixel under perfectly static conditions varies due to the presence of random noise. The main noise sources are:

- **Photon Shot Noise**: the number of photons collected during exposure time is not constant.
- **Electronic Circuitry Noise**: generated by the electronics.
- **Quantization Noise**: related to the ADC conversion.
- **Thermal Noise (Dark Current Noise)**: random charge observed due to thermal excitement.

SNR can be expressed both in decibels and bits.

2.1.12 Dynamic Range (DR)

If the sensed amount of light is too small, the "true" signal cannot be distinguished from noise. Given $E_{\min}$: the minimum detectable irradiation, and $E_{\max}$, the saturation irradiation. The Dynamic Range (DR) of a sensor is defined as $DR = \frac{E_{\max}}{E_{\min}}$, and like the SNR, it is often specified in decibels or bits.

Like SNR, the higher the DR the better it is. A **higher DR** corresponds to the ability of the sensor to simultaneously **capture in one image both the dark and bright** structures of the scene.

2.2 Image Filtering

2.2.1 Noise and image filters

In computer vision we have to deal with noise. Noise is always different, and more noticeable in uniform regions of the image. The simplest thing to reduce noise is to output the average of the pixel color over time, to get **almost** the ideal noiseless value.

$$O(p) = \frac{1}{T}\sum_{t=1}^{T} I_k(p) = \frac{1}{T}\sum_{t=1}^{T}(\tilde{I}(p) + n_t(p))$$

This technique works well on still images.

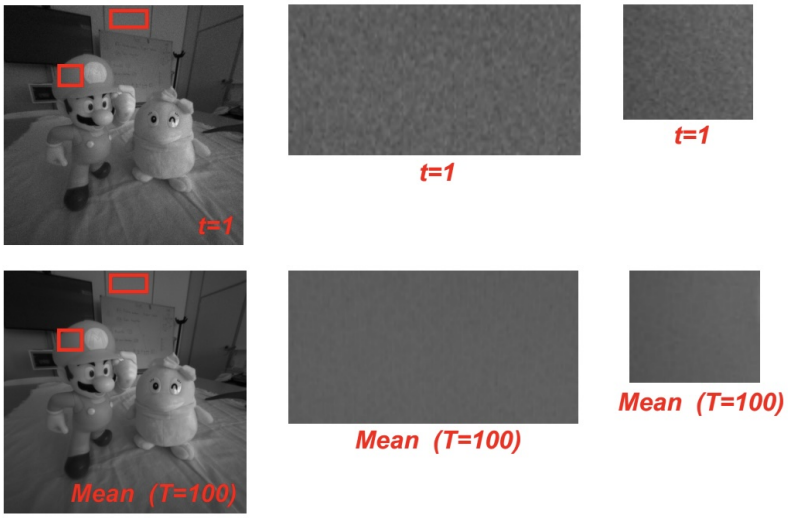


Figure 11: Simple denoising

If we are given a simple image, we may compute a mean across neighbouring pixels, like a spatial rather than temporal mean. The size of the square of the neighbouring pixels is a tradeoff.

This is a very basic denoising filter.

Image filters are image processing operators that compute the new intensity (colour) of a pixel, p, based on the intensities (colours) of those belonging to a neighbourhood (support) of p.

2.2.2 Convolution

An important sub-class of filters is given by **Linear** and **Translation-Equivariant (LTE)** operators.

The **application of filters** in image processing consists in a **2D convolution** between the input image and the impulse response function of the LTE operator.

LTE operators are used as feature extractors in Convolutional Neural Networks (CNNs).

Given an input 2D signal $i(x, y)$, a 2D operator $Ti(x, y)$ is said to be linear if and only if

$$T\{\alpha i_1(x, y) + \beta i_2(x, y)\} = \alpha o_1(x, y) + \beta o_2(x, y)$$

with $o_1 = Ti_1$ and $o_2 = Ti_2$ and α, β are two constants. The operator is said to be **translation-equivariant** if and only if: $T\{i(x - x_0, y - y_0)\} = o(x - x_0, y - y_0)$

If the operator is LTE, the output signal is given by the **convolution** between the input signal and the impulse response (point spread function) $h(x, y) = T\delta(x, y)$.

$$o(x, y) = i(x, y) * h(x, y) = T\{i(x, y)\}$$

$$i(x, y) * h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} i(\alpha, \beta) h(x - \alpha, y - \beta)\, d\alpha d\beta$$

Convolutions have some useful properties:

- **Associative property**: $f * (g * h) = (f * g) * h$ (useful because we can decompose kernels and obtain faster operations).
- **Commutative property**: $f * g = g * f$.
- **Distributive property**: $f * (g + h) = f * g + f * h$.
- **Convolution commutes with differentiation**: $(f * g)' = f' * g = f * g'$.

The correlation of signal $i(x, y)$ with respect to signal $h(x, y)$ is defined as:

$$i(x, y) \circ h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} i(\alpha, \beta) h(x - \alpha, y - \beta)\, d\alpha d\beta$$

Correlation is not commutative, unlike the convolution.

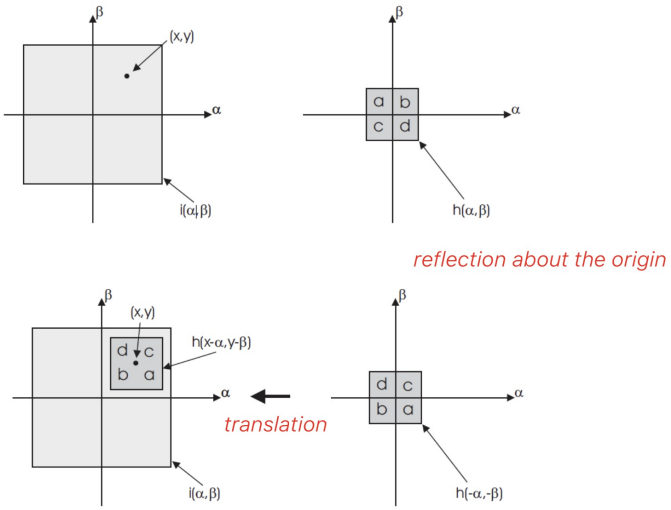


Figure 12: A graphical view of convolution

2.2.3 Discrete convolution

Normal convolution is useful for signal theory, but we want to have a discrete convolution, where we use summations instead of integrals. The four convolution properties highlighted for the convolution hold for the discrete one too.

2.2.4 Practical implementation

CNNs learn flipped kernels. In image processing both the the input image and the kernel are stored into matrices of given finite sizes, with the image being much larger than the kernel. Conceptually, to obtain the output image we need to slide the kernel across the whole input image and compute the convolution at each pixel.

We have an issue on borders, since the kernel "goes out" of the matrix of the image. We have two main options to solve this issue:

- `CROP`: common in image processing.
- `PAD`: preferred in CNNs. We can do zero-padding, replicate the first pixel many times, reflect the first k pixels (half of the kernel)...

Without padding convolutions shrink the images.

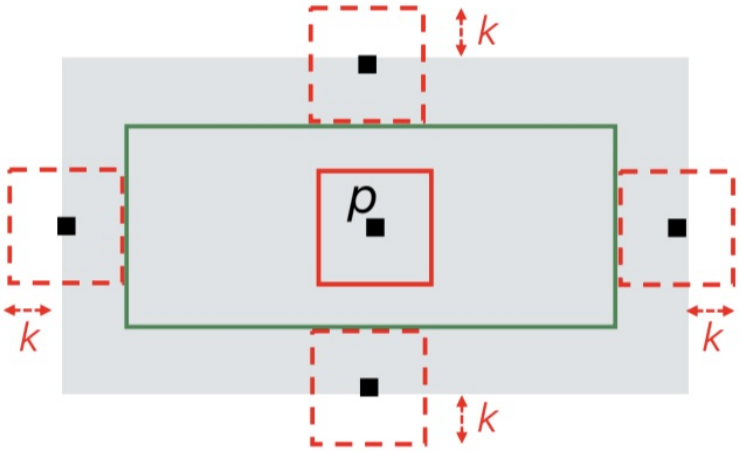


Figure 13: Issues with convolutions

2.2.5 Mean filter

Mean filtering is the **simplest and fastest way to denoise an image**. It consists in replacing each pixel intensity by the average intensity overa chosen neighbourhood. According to signal

processing theory, **the Mean Filter carries out a low-pass filtering operation**, which in image processing is **also referred to as image smoothing**. Smoothing is often aimed at image denoising, but sometimes is used to cancel out small-size unwanted details that might hinder the image analysis task. Linear filtering **reduces noise but blurs the image**, so we lose sharpness.

2.2.6 Gaussian filter

The Gaussian filter is an LTE operator whose impulse response is a 2D Gaussian function.

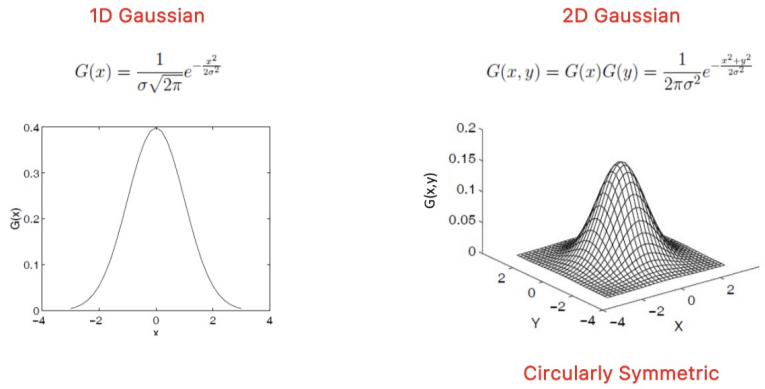


Figure 14: 1D and 2D Gaussian plot

The larger the size of the gaussian kernel, the more accurate the approximation will be, but the computational cost grows with the filter size. We should then use larger sizes for filters with higher σ, smaller sizes whenever σ is smaller. A typical rule is to choose the size of the filter by capturing the interval $[-3\sigma, +3\sigma]$ since it captures 99% of the energy of the Gaussian impulse. To speedup the filtering we can apply the 2D gaussian by doing 2 1D gaussian filterings.

We also observe that the higher the σ, the higher is the smoothing caused by the filter. We can use this filter to remove details from the image.

2.2.7 Median filter

There is noise that Gaussian filters can't handle well, that is the salt and pepper noise. It's usually caused by image corruption (or broken pixels in the sensor). Linear filtering is ineffective and just blurs the image.

We can use a **non-linear filter**, where each pixel intensity is replaced by the **median over a given neighbourhood**. Median filtering counteracts impulse noise effectively, since **outliers tend to fall at either the top or the bottom end of the sorted intensities**. Median filtering tends to keep sharper edges than linear filters such as the Mean or the Gaussian.

Figure 15: Example of the power of the median filter

The median filter can effectively denoise the image without introducing significant blur, yet, Gaussian-like noise, such as sensor noise, cannot be dealt with by the median, as this would require computing new noiseless intensities.

2.2.8 Bilateral filter

The bilateral filter preserves edges while filtering out noise. It's an advanced non-linear filter to accomplish **denoising of Gaussian-like noise without blurring the image**. It's also called edge preserving smoothing.

$$O(p) = \sum_{q \in S} H(p,q) \cdot I_q \qquad H(p,q) = \frac{1}{W(p)} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(p,q))$$

- **spatial distance**: $d_s(p,q) = ||p-q||_2 = \sqrt{(u_p - u_q)^2 + (v_p - v_q)^2}$.
- **range (intensity) distance**: $d_r(I_p, I_q) = |I_p - I_q|$.
- **normalization factor**: $W(p) = \sum_{q \in S} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(p,q))$

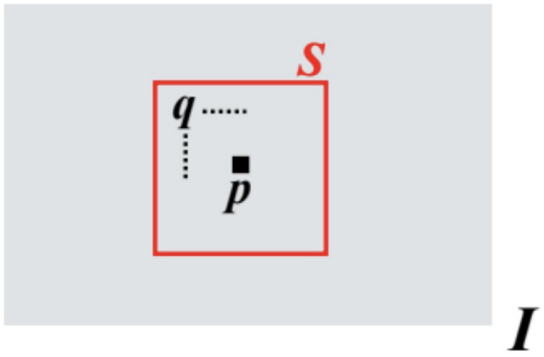


Figure 16: Application of the bilateral filter

Given the supporting neighbourhood, neighbouring pixels take a larger weight as they are both closer and more similar to the central pixel. At a pixel nearby an edge, the neighbourhood falling on the oter side of the edge looks quite different and thus cannot contribute significantly to the output value due to their weights being small. **The kernel has to be recomputed for each pixel**.

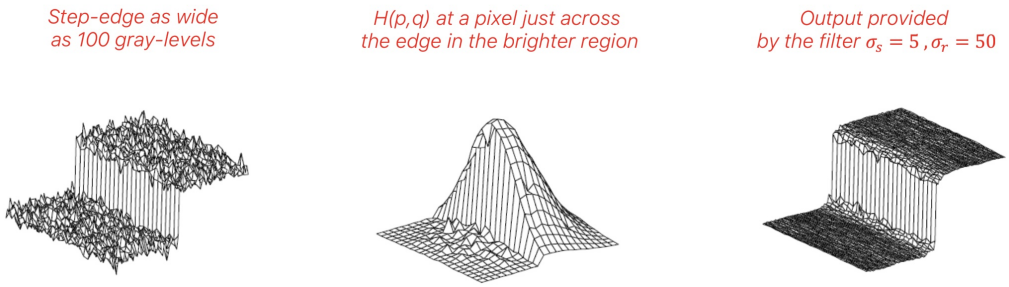


Figure 17: Bilateral filter differences

2.2.9 Non-local means filter

It's another non-linear edge preserving smoothing filter. The key idea is that the **similarity among patches spread over the image** can be deployed to achieve denoising. It's even more expensive than the bilateral filter, since it has to look at more of the image.

2.3 Edge detection

Edge points are local features of the image that capture important information related to its semantic content. Edges are pixels that lie exactly in between image regions of different intensities.

1D step-edge We can detect an edge with a sharp change of a 1D signal (1D step-edge). The simplest edge-detection operator relies on thresholding the absolute value of the derivative of the signal.

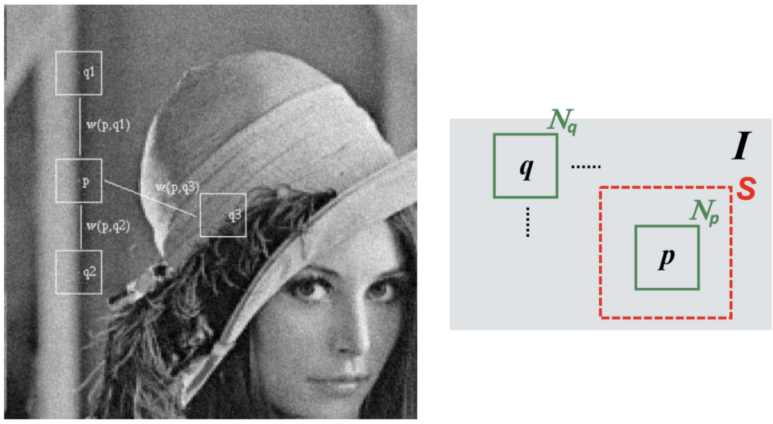


Figure 18: Non-local means filter

Figure 19: Difference between Gaussian filter and non-local means filter

2D step-edge A 2D step-edge is characterized not only by its strength but also by its direction (diagonal edge). The operator which allows us to sense the edge in any direction is the gradient:

$$\nabla I(x,y) = \frac{\partial I(x,y)}{\partial x} i + \frac{\partial I(x,y)}{\partial y} j$$

The **gradient tells the direction along which the function exhibits its maximum variation**. We can use differences to approximate the gradient (since we are in the discrete world and we talk about pixels, not functions). We can estimate the magnitude of the gradient by using different approximations. We choose $|\nabla I|_{\max} = \max(|I_x|, |I_y|)$, since it's fast and invariant with respect to the edge direction.

Edges and noise In real images an edge will not look as smooth as we have seen due to noise. Taking derivatives of noisy signals is an ill posed problem (the solution is not robust with respect to the input variation) since derivatives amplify noise. Noise robustness is usually achieved by smoothing the signal before computing the derivatives required to highlight edges. Smoothing has the side effect of blurring true edges, making it more difficult to detect and localize them, since pixel of different colors will look similar after filtering because of the blending.

Prewitt and Sobel Prewitt and Sobel address the challenge of detecting edges in noisy environments by combining smoothing and differentiation into a single step. The Prewitt and Sobel operators perform spatial filtering by using 3×3 convolution kernels. Each kernel smooths the image in one direction (reducing noise) and computes the derivative in the perpendicular region (highlighting edges). Both operators use two kernels: one detects vertical edges, and one horizontal ones. After convolving the image with both kernels, the gradient magnitude and direction are computed.

$$\text{Magnitude} = \sqrt{G_x^2 + G_y^2} \qquad \text{Direction} = \arctan\left(\frac{G_y}{G_x}\right)$$

Prewitt is better for precise edge localization in cleaner images. Sobel is preferable in noisy images but may slightly blur edges.

2.3.1 Non-Maxima Suppression (NMS)

Detecting edges by gradient thresholding is inherently inaccurate, since **it's difficult to choose the right threshold** a priori. A better approach to detect edges may consist in finding the local maxima of the absolute value of the derivative of the signal.

When dealing with 2D images, we should look for:

- Maxima of the absolute value of the derivative (gradient magnitude).
- Along the gradient direction (orthogonal to the edge direction).

We don't know in advance the correct direction to carry out Non-Maxima Suppression (NMS). The direction has to be estimated locally based on the gradient's direction. The magnitude of the gradient has to be estimated at points which do not belong to the discrete pixel grid. Such values can be estimated by linear interpolation of those computed at the closest point belonging to the grid.

A final thresholding step on the magnitude of the gradient at the points selected by the NMS process typically helps pruning out unwanted edges due to either noise or less important details.

2.3.2 Canny's edge detector

Canny proposed to set forth quantitative criteria to measure the performance of an edge detector and then to find the optimal filter with respect to such criteria. The three criteria he proposed were:

- **Good detection**: correctly extract edges in noisy images.
- **Good localization**: distance between found and true edge should be minimum.
- **One response to one edge**: detect one single edge pixel at each true edge.

A straightforward Canny edge detector can be achieved by:

- Gaussian smoothing.
- Gradient computation.
- NMS along the gradient direction.

2D convolution by a Gaussian can be slow, so we can leverage on separability of the Gaussian function to speedup the calculation: $G(x, y) = G(x)G(y)$.

Non Maxima-Suppression (NMS) is often followed by thresholding of gradient magnitude to help distinguish between true "semantic" edges and unwanted ones. Edge **streaking** may occur when magnitude varies along object contours.

Canny proposed a "hysteresis" thresholding approach relying on a higher T_h and a lower T_l threshold. A pixel is taken as an edge if either the gradient magnitude is higher than T_h or higher than T_l and the pixel is a neighbor of an already detected edge.

The hysteresis thresholding is usually carried out by tracking edge pixels along contours. First the edge candidates are provided by NMS, then all strong edges are picked, and then for each strong edge we track the weak edges along contours.

Zero-crossing Instead of maximum values we can look for zero-crossing of the second derivative of the signal to locate edges (instead of the peaks of the first derivative). This requires significant computational effort since we are calculating second derivatives.

2.3.3 Second derivative along the gradient & Laplacian

The second derivative along the gradient's direction can be obtained as $n^T H n$.

- **Unit vector along the gradient's direction** $n = \frac{\nabla I(x,y)}{||\nabla I(x,y)||}$
- **Hessian matrix** $H = \begin{bmatrix} \frac{\partial^2 I(x,y)}{\partial x^2} & \frac{\partial^2 I(x,y)}{\partial x \partial y} \\ \frac{\partial^2 I(x,y)}{\partial y \partial x} & \frac{\partial^2 I(x,y)}{\partial x^2} \end{bmatrix}$

Computing the second derivative along the gradient turns out very expensive.

Discrete Laplacian We can use the forward and backward differences to approximate first and second order derivatives. It can be shown that the zero-crossing of the Laplacian typically lay close to those of the second derivative along the gradient. Yet, the former differential operator is much faster to compute.

2.3.4 Laplacian of Gaussian (LoG)

A robust edge detector should include a smoothing step to filter out noise. Edge detection by using the Laplacian of Gaussian (LoG) can be summarized as follows:

- **Gaussian smoothing**: $\tilde{I}(x,y) = I(x,y) * G(x,y)$.
- **Second order differentiation** by the Laplacian.
- Extraction of the **zero-crossing** of $\nabla^2 \tilde{I}(x,y)$

Practical implementations of the LoG may deploy the properties of convolutions to speed-up the computation.

We can find sign changes from minus to plus (and viceversa) between two consecutive pixels. Once a sign change is found, the actual edge may be localized:

- At the pixel where the LoG is positive (darker side of the edge)
- At the pixel where the LoG is negative (brighter side of the edge)
- At the pixel where the **absolute** value of the LoG is smaller (it's the best choice, since the edge turns out closer to the true zero-crossing).

Using a **higher σ yields less details**, so usually we use it for noisy images.

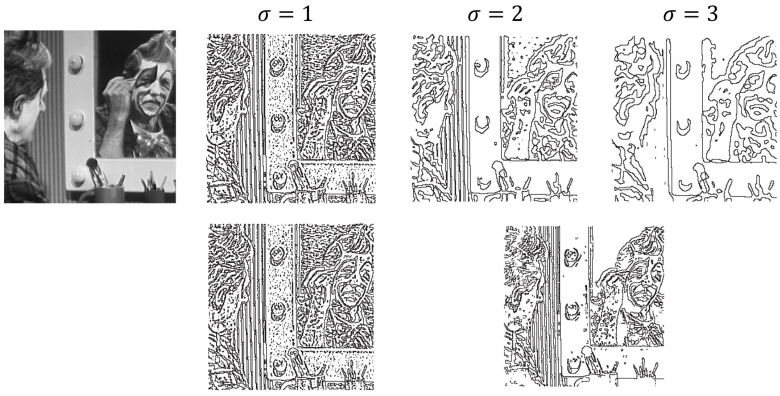


Figure 20: Examples of LoG application

2.4 Feature detection and matching

Feature detection is useful for object detection, since we can look for features in the source image and try to find them again in the target image. Several computer vision tasks deal with finding "Corresponding Points" between two (or more) images of a scene. Correspondences are image points which are the projection of the same 3D point in different views of the scene. Establishing correspondences may be difficult, as the points may look different in different views.

2.4.1 Local invariant features paradigm

The task of establishing correspondences is split into 3 successive steps:

- **Detection** of salient points.
- Computation of a **suitable descriptor** based on pixels in the keypoint neighbourhood.
- **Matching** descriptors between images.

Descriptors should be **invariant** to as many transformations as possible. Ideally we want scale/rotation/illumination invariance.

2.4.2 Properties of good detectors/descriptors

- Detector:
  - Repeatability: it should find the same keypoints in different views of the scene despite the transformations.
  - Saliency: it should find keypoints surrounded by informative patterns.
- Descriptor:
  - Distinctiveness vs. robustness trade-off: the algorithm should capture the salient information around a keypoint.
  - Compactness: the description should be as concise as possible.

Speed is desirable for both, and in particular for detectors, which need to be run on the whole image (while descriptors are computed at keypoints only).

Edge pixels can be hardly told apart as they look very similar along the direction perpendicular to the gradient. Edges are locally ambiguous, since there are many other points that look just the same.

2.4.3 Moravec Interest Point Detector

We can define the **cornerness** at a point p, which is given by the minimum squared difference between the patch centered at p and those centered at its 8 neighbours:

$$C(p) = \min_{q \in n_8(p)} ||N(p) - N(q)||^2$$

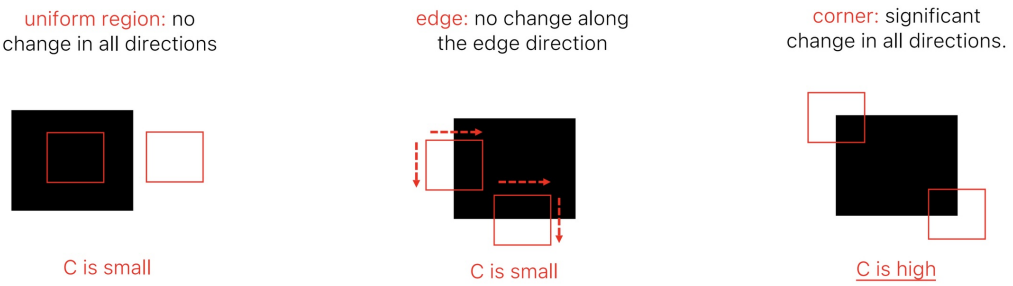


Figure 21: Cornerness of an image segment

After computing the cornerness we threshold and do Non-Maxima Suppression.

2.4.4 Harris Corner Detector

Harris & Stephens proposed to rely on a continuous formulation of the Moravec's "error" function. Assume to shift the image with a generic infinitesimal shift $(\Delta x, \Delta y)$: $w(x, y)$ is a window set to 1 around the pixel under evaluation and 0 in all the image. In the end we consider only the pixel around the (x, y) position.

Due to the shift being infinitesimal, we can deploy Taylor's expansion of the intensity function at (x, y) : $f(x + \Delta x) = f(x) + f'(x)\Delta x$

M encodes the local image structure around the considered pixel. We hypothesize that M is a diagonal matrix $M = \begin{bmatrix} \sum_{x,y} w(x,y) I_x(x,y)^2 & \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{x,y} w(x,y) I_y(x,y) I_x(x,y) & \sum_{x,y} w(x,y) I_y(x,y)^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$

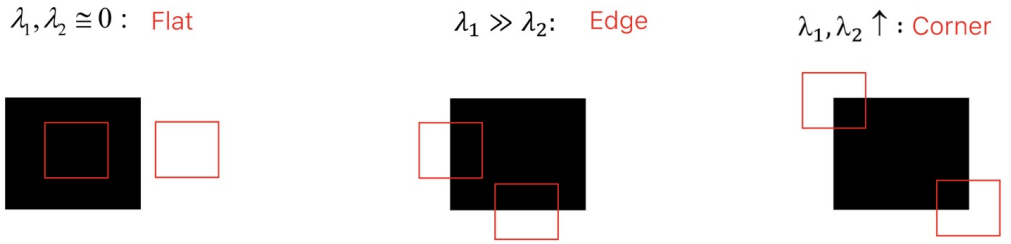


Figure 22: Harris corner detector and lambda values

The previous considerations have general validity as M is real and symmetric, and thus can always be diagonalized by a rotation of the image coordinate system.

$$M = R \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R^T$$

The columns of R are the orthogonal unit eigenvectors of M. λ_i are the corresponding eigenvalues. R^T is the rotation matrix that aligns the image axes to the eigenvectors of M.

Computing eigenvalues at each pixel is costly, so we can compute a more efficient "cornerness" function that gives a result C that is positive on a corner, negative on an edge an about 0 on a flat surface.

The **Harris corner detection algorithm** can thus be summarized as follows:

1. Compute C at each pixel.
2. Select all pixels where C is higher than a chosen positive threshold T.
3. Within the previous set, detect as corners only those pixels that are local maxima of C (NMS).

Eigenvalues of M are invariant to a rotation of the image axes, and thus so is Harris cornerness function. It also invariant to additive intensity changes. However, it's not invariant to scale changes, as corner structures may degrade or transform into edges under significant scaling, nor to multiplicative intensity variations which scale gradients and alter the corner response magnitude unless thresholds are adaptively adjusted.

2.4.5 Scale-Space

Scale invariance is the main issue adressed by second generation local invariant features. The key idea is to apply a fixed-size detection tools on increasingly down-sampled and smoothed versions of the input image.

Figure 23: We increase the blur as we reduce the image size

As we move along scales, small details should continuously disappear and no structure should be introduced.

A Scale-Space is a **one-parameter family of images** created from the original one so that the structures at smaller scales are successively suppressed by smoothing operations.

A Scale-Space must be realized by Gaussian Smoothing: $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$. A Scale-Space is created by repeatedly smoothing the original image with larger and larger Gaussian kernels.

Feature Detection & Scale Selection As features exist across a range of scales how do we establish at which scale a feature turns out maximally interesting and should therefore be described?

As we filter more by using a higher sigma, derivatives tends to become weaker. To compensate, Lindeberg proposed to multiply/normalize derivatives by sigma (doing scale-normalization).

Scale-Normalized LoG The scale-normalized Laplacian Of Gaussian (LoG) is:

$$F(x, y, \sigma) = \sigma^2 \nabla^2 L(x, y, \sigma) = \sigma^2 (\nabla^2 G(x, y, \sigma) * I(x, y))$$

where σ^2 is the normalization factor.

The **scale-normalized LoG detects blobs** (regions with intensity variations) by combining Gaussian Smoothing (which reduces noise and suppresses fine details) and Laplacian Operator (which highlights regions of rapid intensity change).

Scale normalization is introduced since the raw LoG response diminishes as the scale (σ) increases, biasing detection toward smaller blobs.

Difference of Gaussian (DoG) Lowe proposed to detect keypoints by seeking for the extrema of the DoG (Difference of Gaussian) function across the (x, y, σ) domain.

$$DoG(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

This approach provides a computationally efficient approximation of Lindeberg's scale-normalized LoG.

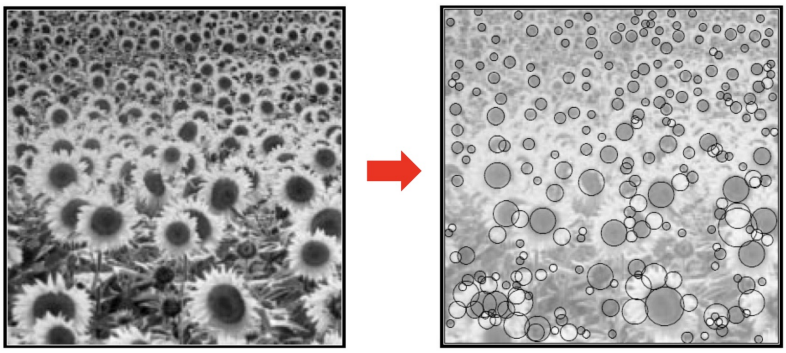


Figure 24: LoG filter locates blobs

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G(x, y, \sigma) * I(x, y)$$

Both detectors are rotation invariant and find blob-like features.

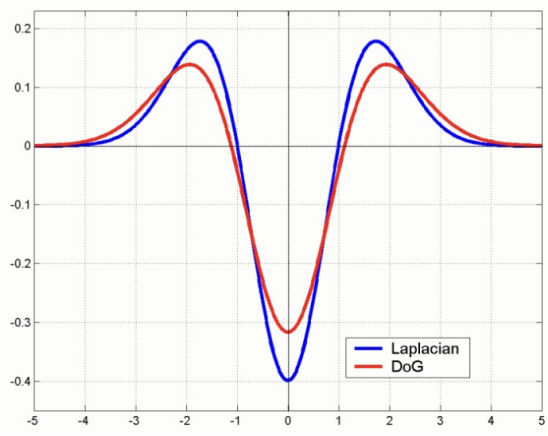


Figure 25: Lowe proves that this is a scaled version of Lindeberg

In the DoG we compute several Gaussian smoothed functions within an octave. For each pair we take the differences, then we seek for extrema in the DoG.

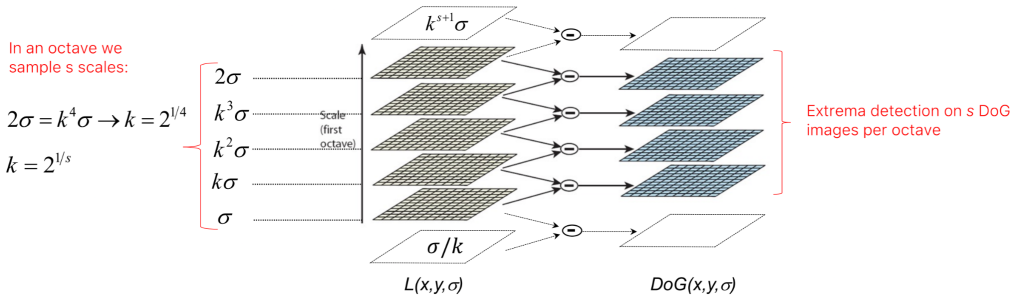


Figure 26: Difference of Gaussian

Extrema detection A point (x, y, σ) is detected as a keypoint if and only if its DoG is higher (or lower) than that of the 26 neighbour (8 at the same scale and $18 = 9 + 9$ at the two nearby scales) in the (x, y, σ) space.

According to the original article the best number of sclaes within an octave is $s = 3$, initial σ for each octave should be $\sigma = 1.6$ and the input image is enlarged by a factor of 2 in both dimensions.

After detecting points as local extrema, the next step is to prune weak and unstable responses to ensure robustness. Keypoints with low contrast, corresponding to weak DoG responses, are often sensitive to noise and less repeatable across different images.

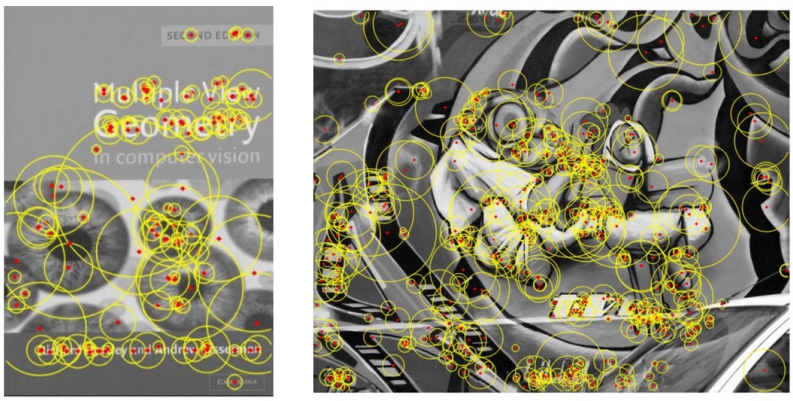


Figure 27: The size of the circle is proportional to the scale of the feature (σ)

2.4.6 Scale and Rotation Invariance Description

DoG are **rotation invariant because of circular simmetry**. Once each keypoint has been extracted, a surrounding patch is considered to compute its descriptor (scale and rotation invariant).

We need to identify a **prominent direction** inherent to the patch (the canonical orientation) and, based on such direction, define a local reference frame. A reasonable choice consists in identifying the **direction along which most of the gradient is found**.

To compute the canoncial orientation we need to define a scale and rotation invariant description. Once each keypoint has been extracted, a surrounding patch is considered to compute its descriptor (scale and rotation invariant).

- Scale invariance: the patch is taken from the stack of images that correspond to the characteristic scale.
- Rotation invariance: a canonical patch orientation is computed, so that the descriptor can then be computed on a canonically-oriented patch (orientation with respect to a new reference system, not the one of the image).

Lowe proposed to compute the **canoncial orientation of DoG keypoints** as follows: given the keypoint, the magnitude and orientation of the gradient are computed at each pixel of the associated Gaussian-smoothed image, L:

- $m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$
- $\theta(x,y) = \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$

We can then compute an **orientation histogram** by accumulating the contributions of the pixels belongigng to a neighborhood of the keypoint location (a bin size of 10 deg). The characteristic orientation of the keypoint is given by the highest peak of the orientation histogram. Other peaks higher than 80% of the main one would be kept as well. A keypoint may have multiple canonical orientations and, in turn, multiple descriptors sharing the same location/scale with diverse orientations (this happen rarely, about on 15% of the keypoints).

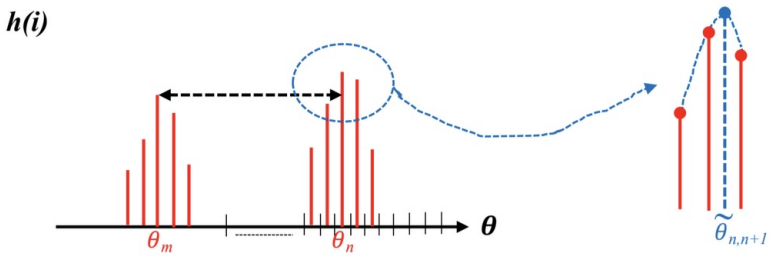


Figure 28: This is an ambiguous keypoint, we have two prominent decisions

2.4.7 SIFT Descriptor

The **SIFT (Scale Invariant Feature Transform)** descriptor is computed as follows:

- 16×16 oriented pixel grid around each keypoint is considered.
- This is further divided into 4×4 regions (each of size 4×4 pixels).

- A gradient orientation histogram is created for each region.
- Each histogram has 8 bins.
- Each pixel in the region contributes to its designated bin according to gradient magnitude and gaussian weighting function centered at the keypoint (with σ equal to half the grid size).

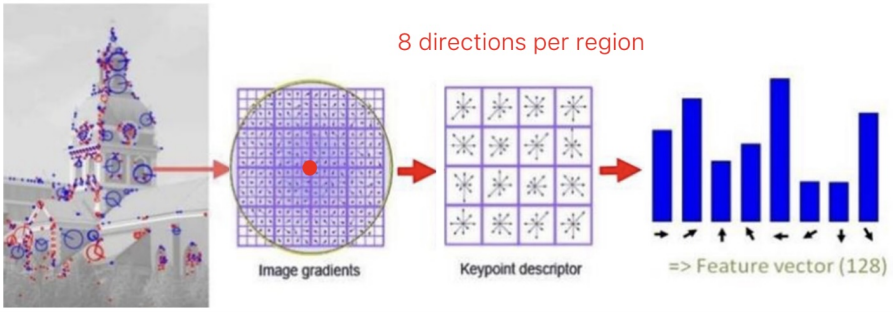


Figure 29: The descriptor size is given by the number of regions times the number of histogram bins per region

Matching process The descriptor is normalized to unit length to gain invariance with respect to affine intensity changes. Descriptors like SIFT are compared across diverse views of a scene to find corresponding keypoints. This is a classical Nearest Neighbour (NN) seach problem. Given a set S of points, p_i, in a matric space M and a query point $q \in M$, find the p, closest to q. We wish to match the local features computed from an image under analysis (target image T) to those already computed from a reference image (R) or a set of reference images.

For each feature in T we look for the most similar one in R:

- The features in T represent the query points, q.
- The features in R provide set S.
- When matching SIFT descriptors the distance typically used is the Euclidean distance.

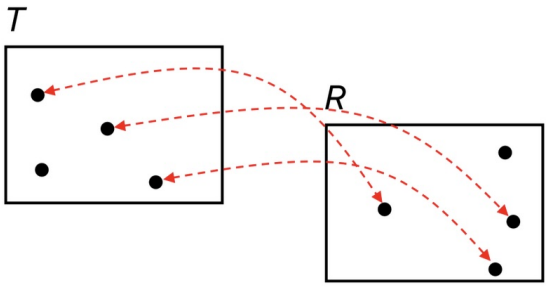


Figure 30: Image matching process with SIFT

The found Nearest Neighbour (NN) does not necessarily provide a valid correspondence as some features in T may not have a corresponding feature in R. We enforce a criteria to accept/reject a match found by the NN search process. The simplest thing to do is to use a threshold. Lowe showed that $T = 0.8$ may allow for rejecting 90% of the wrong matches while missing only 5% of those correct.

Exhaustively searching for the NN of the query feature, q, has linear complexity in the size of S. This is slow, so efficient indexing techniques are exploited to speed-up the NN-search process. The main indexing technique exploited for feature matching is known as k-d tree.

2.5 Camera Calibration

Camera calibration is the process of determining the parameters of a camera model that describes how a 3D scene is projected onto a 2D image. This is crucial for extracting quantitative measurements from images, such as object sizes or distances. We primarily focus on the **perspective projection** model.

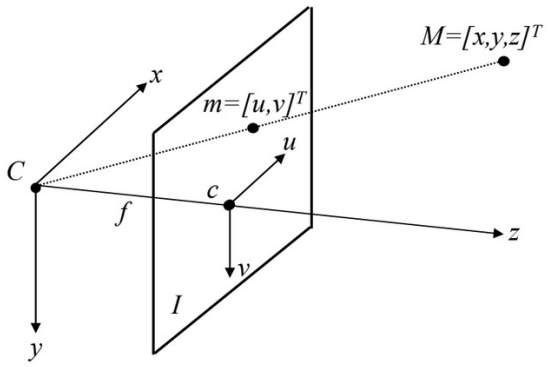


Figure 31: Perspective Projection Model (PPM). A 3D point M in the Camera Reference Frame (CRF) projects to a 2D point m on the image plane I.

2.5.1 From Physical Space to Projective Space

The world we perceive is often modeled as a 3D **Euclidean Space** ($\mathbb{R}^3$). Points are represented by 3D vectors $[x, y, z]^T$ in a chosen reference frame. However, Euclidean geometry struggles with concepts like points at infinity and the projective nature of image formation (e.g., parallel lines appearing to converge).

To handle the geometry of perspective projection more elegantly, we use **Projective Space** (P^3).

- **Homogeneous Coordinates:** A point with Euclidean coordinates (x, y, z) is represented in homogeneous coordinates by an equivalence class of 4D vectors $[\lambda x, \lambda y, \lambda z, \lambda]^T$ for any non-zero scalar λ. A common choice is to set $\lambda = 1$, giving $[x, y, z, 1]^T$.
- **Mapping Back:** To convert from homogeneous coordinates $[X, Y, Z, W]^T$ (where $W \neq 0$) back to Euclidean coordinates, we divide by the fourth component: $[X/W, Y/W, Z/W]^T$.
- **Points at Infinity:** Points at infinity correspond to directions in 3D space. In homogeneous coordinates, they are represented by vectors of the form $[X, Y, Z, 0]^T$, where $[X, Y, Z]^T$ defines the direction. These points cannot be mapped back to the finite Euclidean space.
- **The Origin:** The origin of the Euclidean space $(0, 0, 0)$ is represented by $[0, 0, 0, k]^T$ with $k \neq 0$ in homogeneous coordinates. The vector $[0, 0, 0, 0]^T$ is undefined in projective space.

The main advantage of using homogeneous coordinates is that the non-linear perspective projection equations become **linear transformations** represented by matrix multiplication.

2.5.2 Basic Perspective Projection Matrix (Pinhole Camera)

Let $M = [x, y, z]^T$ be a 3D point expressed in the **Camera Reference Frame (CRF)**, with the origin at the camera's optical center and the z-axis along the optical axis. Let its projection onto the image plane be $m = [u, v]^T$. Assuming the image plane is at $z = f$ (where f is the focal length), the basic perspective projection is: $u = f\frac{x}{z}$ and $v = f\frac{y}{z}$.

Using homogeneous coordinates, we represent the 3D point as $\tilde{M}_{CRF} = [x, y, z, 1]^T$ and the projected 2D point as $\tilde{m} = [u', v', w']^T$, where the final image coordinates are $u = u'/w'$ and $v = v'/w'$. The projection can be written as a linear mapping:

$$w' \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

This can be written compactly as $\tilde{m} \propto P_0 \tilde{M}_{CRF}$, where $\propto$ denotes equality up to a non-zero scale factor, and P_0 is the basic 3×4 perspective projection matrix for a pinhole camera centered at the origin, looking along the z-axis. The scale factor here is $w' = z$.

2.5.3 A More Comprehensive Camera Model

The basic model is often insufficient. We need to account for:

1. **Image Coordinate System:** Image coordinates (u, v) are typically measured in pixels, with the origin often at the top-left corner, not the principal point (where the optical axis pierces the image plane).
2. **Pixel Shape:** Pixels might not be square. We need parameters for pixel size/density $(\Delta u, \Delta v)$ or equivalently, focal lengths measured in pixels ($f_u = f/\Delta u$, $f_v = f/\Delta v$).
3. **World Reference Frame (WRF):** 3D points are usually defined in a convenient WRF, not the CRF. We need to relate the WRF to the CRF.

These factors lead to a more complex projection matrix $\tilde{P}$, which can be decomposed into intrinsic and extrinsic parameters.

Intrinsic Parameter Matrix (A) This 3×3 matrix maps projected points from the camera's normalized image plane (where $z = 1$) to pixel coordinates. It encodes the camera's internal geometry:

- f_u, f_v: Focal lengths expressed in units of horizontal and vertical pixel dimensions.
- (u_0, v_0): Coordinates of the principal point (the image center) in pixels. Sometimes denoted (c_x, c_y).

Assuming zero skew, the intrinsic matrix is:

$$A = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

These **intrinsic parameters** (at least 4: f_u, f_v, u_0, v_0) are specific to the camera and lens, independent of the camera's position or orientation in the world.

The projection from the CRF using only intrinsic parameters can be written by combining A with a standard projection into the normalized image plane:

$$\tilde{P}_{int} = A[I|\mathbf{0}] = \begin{bmatrix} f_u & 0 & u_0 & 0 \\ 0 & f_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This matrix $\tilde{P}_{int}$ maps a 3D point in homogeneous coordinates *relative to the CRF* to its 2D homogeneous pixel coordinates.

Vanishing Points Example Consider a point at infinity $[a, b, c, 0]^T$ representing a direction relative to the CRF. Its projection using $\tilde{P}_{int}$ is:

$$\tilde{m}_\infty \propto \tilde{P}_{int} \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix} = \begin{bmatrix} f_u a + u_0 c \\ f_v b + v_0 c \\ c \end{bmatrix}$$

Assuming $c \neq 0$ (the direction is not parallel to the image plane), we can convert to Euclidean pixel coordinates by dividing by the third component:

$$m_\infty = \begin{bmatrix} f_u \frac{a}{c} + u_0 \\ f_v \frac{b}{c} + v_0 \end{bmatrix}$$

This projected point m_∞ is a **vanishing point** in the image – the point where parallel lines in 3D space with direction $[a, b, c]^T$ appear to converge. If $c = 0$, the direction is parallel to the image plane, and the point projects to infinity in the image plane coordinates (no finite vanishing point).

Extrinsic Parameter Matrix (G) To use points defined in a World Reference Frame (WRF), we need to describe the camera's pose (position and orientation) relative to the WRF. This is done using a rigid body transformation:

- R: A 3×3 rotation matrix describing the orientation of the CRF relative to the WRF.
- T: A 3×1 translation vector describing the position of the CRF origin relative to the WRF origin (expressed in WRF coordinates). Or sometimes, the position of the WRF origin relative to the CRF origin (expressed in CRF coordinates) - convention matters! Let's assume T specifies the CRF origin in WRF for the G matrix below which transforms points *from* WRF *to* CRF.

The transformation from WRF coordinates $\tilde{M}_W = [X, Y, Z, 1]^T$ to CRF coordinates $\tilde{M}_{CRF} = [x, y, z, 1]^T$ is given by:

$$\tilde{M}_{CRF} = G\tilde{M}_W$$

where G is the 4×4 **Extrinsic Parameter Matrix**. A common form relates WRF coordinates to CRF coordinates:

$$G = \begin{bmatrix} R & T \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Here, R rotates the WRF axes to align with the CRF axes, and T translates the origin. This transformation has **6 extrinsic parameters**: 3 for rotation (e.g., Euler angles, axis-angle) and 3 for translation.

The transformation from WRF to CRF coordinates (non-homogeneous) is $M_{crf} = RM_w + T'$, where T' is the translation vector. In homogeneous coordinates, this is often expressed using a 4×4 matrix that incorporates the inverse transformation or directly within the projection matrix.

2.5.4 The Complete Perspective Projection Matrix ($\tilde{\mathbf{P}}$)

Combining the intrinsic and extrinsic parameters, the full projection from a 3D point $\tilde{M}_W$ in WRF homogeneous coordinates to a 2D point $\tilde{m}$ in image homogeneous pixel coordinates is:

$$\tilde{m} \propto A[R|T]\tilde{M}_W$$

Here, the 3×4 matrix $\tilde{P} = A[R|T]$ is the complete **Perspective Projection Matrix (PPM)**.

- R is the 3×3 rotation matrix specifying the camera's orientation.
- T is the 3×1 translation vector specifying the camera's position (specifically, $T = -RC_W$, where C_W is the position of the camera center in world coordinates).

The matrix $[R|T]$ effectively transforms the point from WRF coordinates directly into the CRF, ready for projection and intrinsic transformation by A.

The PPM $\tilde{P}$ encodes all the geometric information of the camera setup:

- Intrinsic parameters (A): Camera's internal geometry (focal length, principal point, pixel properties). (4+ parameters)
- Extrinsic parameters (R, T): Camera's pose (position and orientation) in the world. (6 parameters)

Camera calibration aims to find the numerical values for the parameters in A and, often simultaneously or subsequently, the extrinsic parameters R and T relative to a known calibration target.

2.5.5 P as a Homography

We just need the relation between the camera and the plane, then we can measure everything on that plane.

If the camera is imaging a planar scene, we can assume the z-axis of the World Reference Frame to be perpendicular to the plane such that all 3D points will have their z-coordinate equal to 0. The Perspective Projection Matrix then becomes a simpler transformation defined by a $3x3$ matrix.

$$k\tilde{m} = \tilde{P}\tilde{w} \begin{bmatrix} p_{1,1} & p_{1,2} & \cancel{p_{1,3}} & p_{1,4} \\ p_{2,1} & p_{2,2} & \cancel{p_{2,3}} & p_{2,4} \\ p_{3,1} & p_{3,2} & \cancel{p_{3,3}} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H\tilde{M}$$

Such a transformation, denoted here as H, is known as **homography** and represents a general linear transformation between projective planes. A homography is then a projective transformation between two planes or a mapping between two planar projection of an image.

2.5.6 Lens distortion

The Perspective Projection Matrix is based on the pinhole camera model, however, real lenses introduce distortions with respect to the pure pinhole model. We also have lens distortion, which is modelled through additional parameters that don't alter the form of the Perspective Projection Matrix.

We have two types of lens distortion: barrel distortion which bends straight lines outwards, and pincushion distortion, which causes straight lines to curve inward.

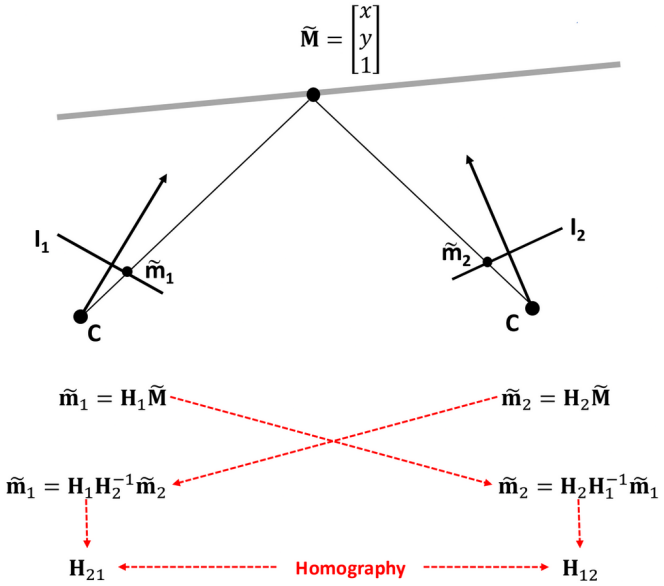


Figure 32: Any two images of a planar scene are related by a homography

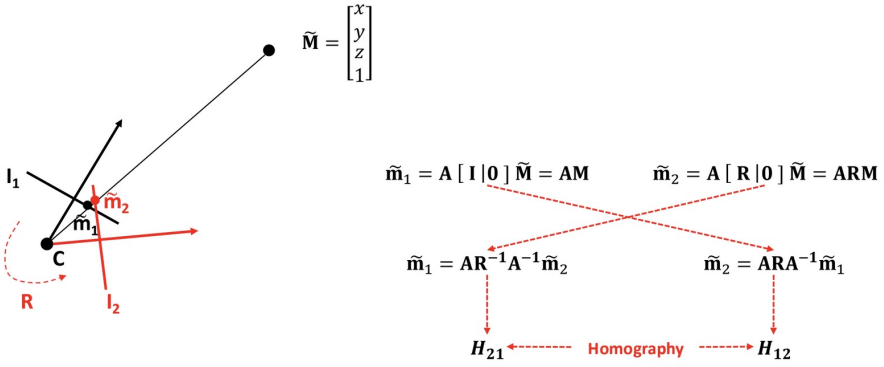


Figure 33: Any two images taken by a camera rotating about the optical center are related by a homography

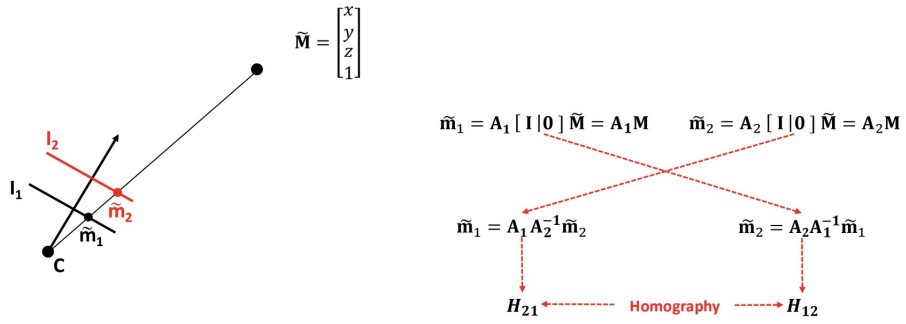


Figure 34: Any two images taken by different cameras in fexed pose are related bu a homography

2.5.7 Calibration

We now have the Perspective Projection Matrix camera model, which can be decomposed in:

- Intrinsic parameter matrix A.
- Rotation matrix R.
- Translation vector T.

Camera calibration is the process where **all parameters defining the camera model are estimated** for a specific camera device. Depending on the application, **either the Perspective Projection Matrix (PMM) only, or also its independent components** (A,R,T) need to be estimated.

There are many camera calibration algorithms, but the basic process always relies on setting up a linear system of equations given a set of known 3D-2D correspondences. To obtain the correspondences we use calibration targets, which have easily detectable features.

The approaches can be split into:

- Those relying on a single image containing a known pattern.
- Those relying on several different images of one given planar pattern.

Since in every frame we change the relative position of the camera and the pattern, we will have a different set of extrinsic parameter for each image. We will have just one set of intrinsic parameter since the camera is always the same across the different pictures.

2.5.8 Zhang's method for Camera Calibration

We use a chessboard pattern of which we know the number of internal corners and the size of the squares. Internal corners can be easily detected by standard algorithms, like the Harris corner detector. Typically the camera is fixed and you move the calibration target in front of the camera. The $[R\,T]$ are estimated with respect to the reference system attached to the target, but it changes alongside with the pattern. In each image the 3D world reference frame is taken at the top-left corner of the pattern. Each image requires its own estimate of the extrinstic parameters, as they are different from one to the other. Due to the choice of the world reference frame associated with calibration images, in each of them we consider only 3D points with $z = 0$. The Perspective Projection Matrix boils down to a simpler transformation defined by an homography 3×3 matrix, like with P as a homography.

$$k\tilde{m} = \tilde{P}\tilde{w} \begin{bmatrix} p_{1,1} & p_{1,2} & \cancel{p_{1,3}} & p_{1,4} \\ p_{2,1} & p_{2,2} & \cancel{p_{2,3}} & p_{2,4} \\ p_{3,1} & p_{3,2} & \cancel{p_{3,3}} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ \cancel{\emptyset} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H\tilde{M}$$

Estimating H_i (DLT algorithm) Given a pattern with m corners, we can write m systems of 3 linear equations, where:

- Both 3D as well as 2D coordinates are known due to the corners having been detected in the i-th image and the unknowns are thus the 9 elements in H_i.
- H_i (and P_i alike) is known up to an arbitrary scale factor, the independent elements in H_i are 8.

There are plenty of methods for the estimation

The Zhang's method can be summarized as:

1. Acquire n images of a planar pattern with m internal corners.
2. For each image compute an initial guess for homography H_i.
3. Refine eaech H_i by minimizing the reprojection error.
4. Get an initial guess for A given the homographies H_i.
5. Given A and H_i, get an initial guess for R_i and T_i.
6. Compute an initial guess for lens distortion parameters k.
7. Refine all parameters A, R_i, T_i, k by minimizing the reprojection error.

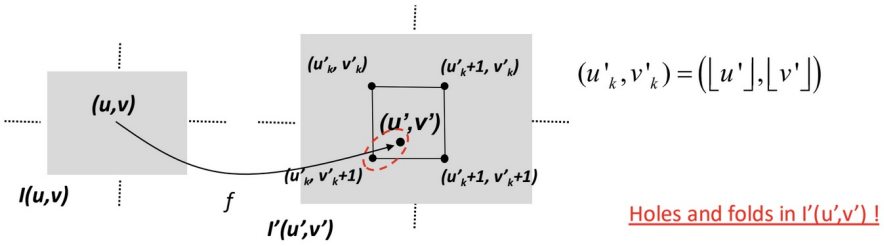


Figure 35: A better choice consists in mapping to the closest point into the destination image

2.5.9 Image warping

After applying the warping function you get real coordinates not discrete ones. During the mapping, some pixels of the destination image may be not rounded perfectly. What if more pixels go to the same position?

Since the coordinates are always real values the different mapping strategies can be to map the closest point or to interpolate between the 4 closest points.

3 Advanced Topics in Deep Learning for Computer Vision

3.1 Recall on CNNs

In representation learning we try to find good ways of representing data, often by learning a useful transformation of the raw data. Deep learning is a subset of representation learning.

3.1.1 Gradient descent

Neural networks are usually trained using iterative, gradient-based optimizers that drive the cost function to a very low value. The convergence point of gradient descent depends on the initial values of the parameters. For feedforward neural networks it's important to initialize all weights to small random values and to initialize biases to zero or to small positive values. To apply gradient based learning we must choose a cost function and how to represent the output of the model. The derivative $f'(x)$ gives the slope of $f(x)$ at the point x. The derivative is useful for minimizing a function because it tells us how to change x in order to make a small improvement in y. A point that obtains the absolute lowest value of $f(x)$ is a global minimum.

To reach the minimum we use the formula $w' = w - \alpha\frac{\partial J(w,b)}{\partial w}$ $b' = b - \alpha\frac{\partial J(w,b)}{\partial b}$, where α is the learning rate, which controls how big are the steps that we take during the gradient descent. If the slope is positive, we subtract the quantity in order to reach the minimum, or else we do the opposite thing.

As the training set size grows to billions of examples, the time to take a single gradient step becomes prohibitively long. Stochastic/Minibatch gradient descent are extensions of the gradient descent algorithm. We can sample a minibatch of examples drawn uniformly from the training set (1 for stochastic). It's like training every time on a different (smaller) training set.

Optimizers: momentum Momentum was designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients. The size of the step depends on how large and how aligned a sequence of gradients are. The step size is largest when many successive gradients point in the same direction. Momentum smooths the step of the gradient descent in its path to the minimum. We compute: $W^{[l+1]} = W^{[l]} - \alpha v_{W^{[l+1]}}$, where $v_{W^{[l+1]}} = \beta v_{W^{[l]}} + (1-\beta)\frac{\partial J(...)}{\partial W^{[l]}}$. Root Mean Square Propagation (RMSprop) uses an exponentially decaying average to discard history from the extreme past so that it doesn't impede the convergence.

Optimizers: Adam The name derives from "adaptive moments", and it can be seen as the combination of RMSprop and momentum. It's fairly robust to the choice of hyperparameters, though the learning rate sometimes needs to be changed from the suggested default. There is a first order and a seond order Adam. The first order Adam uses the gradient and its moving averages (first moments) to update the parameters. The second order Adam incorporates information about the curvature of the loss surface, using approximations of the Hessian or second order derivatives, which leads to more informed and potentially more efficient updates but requires more computational resources.

3.1.2 Convolutions and filters

An image is characterized by height, width and number of channels.

We can also train more than one filer, where each filter outputs a feature map. By stacking activation maps we get a new volume.

Since convolutions shrink the images we can use padding to preserve the edges. We have always considered a stride of 1 (moving the convolutional filter by one pixel), but we can also use different stride sizes to shrink the image.

Pooling A pooling layer is used to reduce the size of the representation in order to speedup computation. Pooling comes usually after each conv layer or after a block (or set) of conv layers. It's applied to each activation map independently.

Receptive fields The input pixels affecting a hidden unit are called its receptive field. We can encode the information in all the pixels to a single point. We go from spatial to semantic information (we transform spatial information to semantic information).

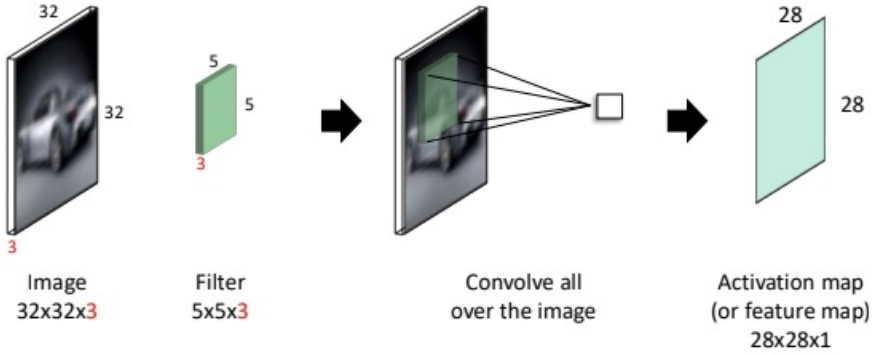


Figure 36: A convolutional filter has the **same depth** of the input volume, so the output dimension is 1.

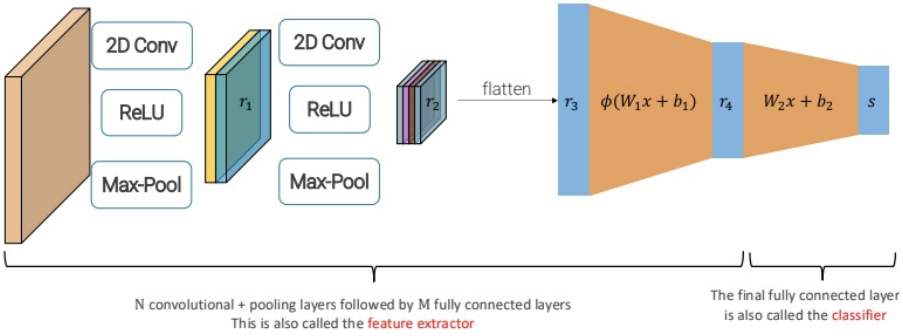


Figure 37: By stacking together convolutional filters, pooling layers and fully connected layers we can design a Deep CNN.

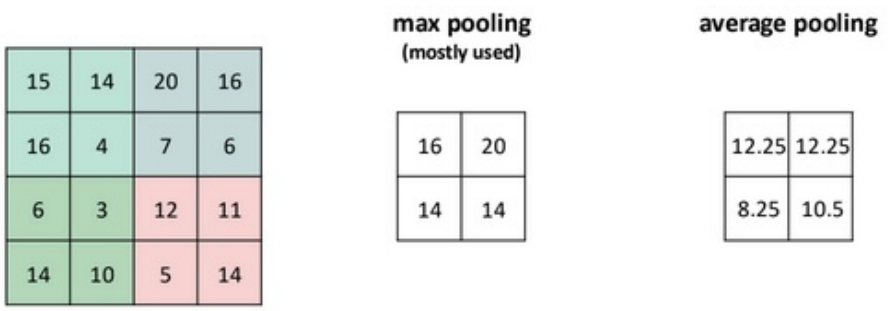


Figure 38: Example: pooling of dimension 2 and stride 2.

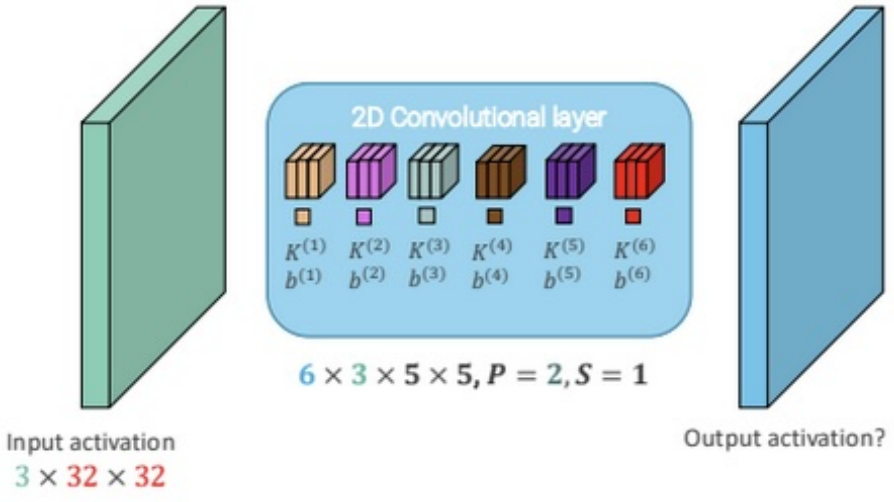


Figure 39:

Convolution parameters and flops As we can see in 39, the number of learnable parametes for the convolutional layer is $6 \times (3 \times 5 \times 5 + 1) = 6 \times 76 = 456$ (6 convolutional blocks, 3 channels per convolutional block, 5×5 convolution,). The size of the output is $H_{\text{out}} = W_{\text{out}} = 32 - 5 + 2 * 2 + 1 = 32$. Hence, there are $6 \times 32 \times 32 = 6144$ values in the output activation ($\approxeq$ 24KB).

Since each of them is obtained as the dot product between the weights and the input, which requires to perform n multiplications and n summations for inputs of size n, i.e. $2n$ flops.

3.1.3 Batch Normalization (BatchNorm)

BatchNorm is a technique designed to stabilize and accelerate the training of deep neural networks by addressing internal covariate shift: the change in the distribution of layer activations caused by updates to preceding layers during training. By normalizing activations at each layer, BatchNorm ensures that gradients are propagated more effectively, enabling coordinated updates across deep networks.

Let $Z^{[l]}$ be a minibatch of activations of the l-th layer to be normalized. To normalize $Z^{[l]}$, we replace it with: $Z^{[l]}_{norm} = \frac{Z^{[l]} - \mu}{\sigma}$, where μ is a vector containing the mean of each unit and σ is a vector containing the mean of each unit and σ is a vector containing the standard deviation of each unit. μ and σ are running averages of the values seen during training. BatchNorm helps with speeding up the training, and has a slight regularization effect (but we don't use it for this reason).

We use LayerNorm for fully connected layers because it normalizes per sample, making it batch-size independent. We use InstanceNorm for convolutional layers because it normalizes per sample and per channel.

3.1.4 Dropout regularization

Dropout randomly deactivates a fraction of neurons during training, effectively training an ensemble of smaller sub-networks. For each layer, you set a dropout probability, determining the chance a neuron is ignored in a forward pass. This helps to prevent overfitting because we don't associate a certain pattern to a certain output of the network.

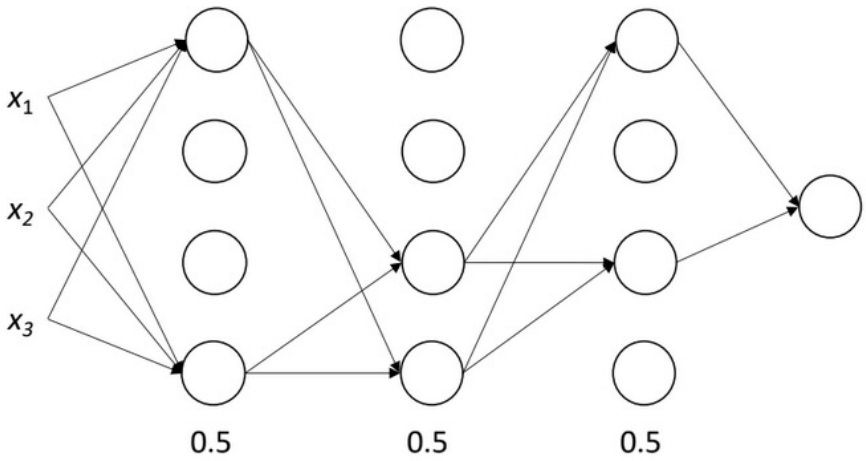


Figure 40: Each layer has a 50% dropout probability

Dropout and normalization are used only at training time. When we test the network we switch off dropout and regularization.

Another powerful regularization technique is just having more data.

3.1.5 Data augmentation

We can create more data by modifying the images we already have. We must only make transformations which keep the labels valid. We can, for example, sample random crops/scales of the data or do color augmentation.

Cutout is the process where we remove a random square region of the input image. This forces the network to use a more diverse set of features, helping generalization. It's gray because we subtract the mean from this pictures, and so the number in the cutout becomes 0.

3.2 CNNs

3.2.1 AlexNet & ZFnet

In 2012 Alex Krizhevsky proposed AlexNet, which had almost a 9% of improvement on SOTA.

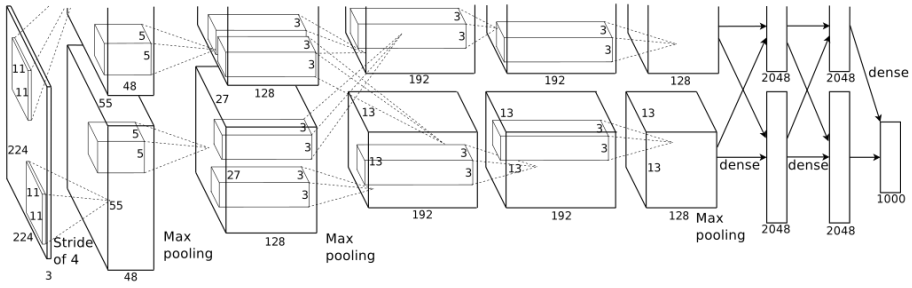


Figure 41: AlexNet architecture

In the image 41, we see only one half of the network, since the architecture is replicated identically 2 times to be used on a 2 GPU setup. The network is composed by a **stem layer** at the beginning, which is a convolution layer that performs a fast reduction in the spatial size of the activations, mainly to reduce memory and computational cost. The first layer has two 11×11 kernels, which extracts corners/edges/blobs.

To counteract these problems ZFnet used 7×7 convolutions with stride 2 in the first layer and 5×5 convolutions with stride 2 also in the second convolutional layer.

3.2.2 VGG

To avoid the vanishing gradient problem VGG used 3×3 convolutions only, and didn't use the stem layer. The network was shallower, and used only convolutions and pooling. At the end it used the same flattening and classification head.

Pre initialization with weights from shallower architectures was crucial to let the training progress, since batch normalization wasn't yet invented.

VGG-16 VGG-19

ConvNet Configuration				
A	B	C	D	E
11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)				
conv3-64	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool				
conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool				
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool				
FC-4096				
FC-4096				
FC-1000				
soft-max				

Figure 42: VGG architecture

VGG introduces the idea of designing a network as repetition of stages, i.e. a fixed combination of layers that process activations at the same spatial resolution. In VGG we have 3 types of blocks:

- `conv-conv-pool`.
- `conv-conv-conv-pool`.
- `conv-conv-conv-conv-pool` (we can get a more complex convolution by combining 3×3 convolutions).

One stage has the same receptive field of larger convolutions but requires less parameters and computation, and introduces more non-linearities.

The disadvantage is that the memory for activation doubles.

convolutional layer	params	flops	ReLUs	number of activations
$C \times C \times 5 \times 5$, $S = 1$, $P = 2$	$25C^2 + C$	$50C^2 W_{in} H_{in}$	1	$C \times W_{in} \times H_{in}$
2 stacked $C \times C \times 3 \times 3$, $S = 1$, $P = 1$	$18C^2 + 2C$	$10C^2 W_{in} H_{in}$	2	$2 \times C \times W_{in} \times H_{in}$

VGG-16 is composed by 138M parameters ($2.3x$ AlexNet), mostly in fully connected layers. The computational cost is $\approx$ 4 Tflops, mainly due to convolutions, and the network uses about $\approx$ 16.5 GB of memory.

3.2.3 Inception v1 (GoogLeNet)

The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was achieved by a carefully crafted design that allows for increasing the depth and width of the network while keeping the computational budget constant.

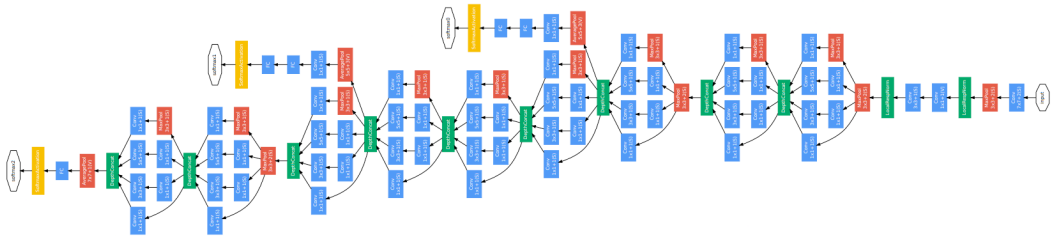
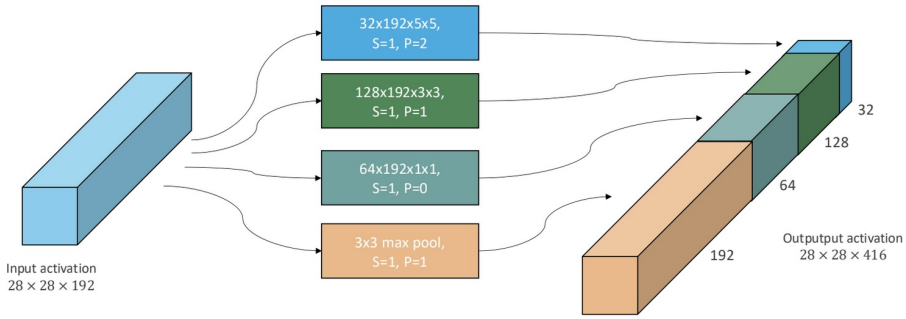


Figure 43: GoogLeNet architecture

The network is composed by stem layers which shrink the image size, a stack of inception modules where a combination of different operations are combined, and a final classifier. There are 22 trainable layers and about 100 modules (the blue and red blocks).

Stem layers Stem layers aggressively downsamples inputs: from 224 to 28 width/height in 5 layers. It brings it down a bit more gently than AlexNet. To reach 28×28, VGG uses 10 layers.

Naïve inception module The main idea of the inception module is that it consists of multiple pooling and convolution operations with different sizes (3×3, 5×5) in parallel, instead of using just one filter of a single size. However, there are two main problems:



- Due to max-pool, the number of channels grows very fast when inception modules are stacked on top of each other.
- 5×5 and 3×3 convolutions on many channels become prohibitively expensive if we stack a lot of them.

1×1 convolutions and the inception module To overcome the problems, we use 1×1 convolutions before feeding the data into 3×3 or 5×5 convolutions. 1×1 convolutions don't reason locally, but along the channel on a single dimension. They allow us to change the depth of the activations while preserving the spatial size. This is effectively a dimensionality reduction.

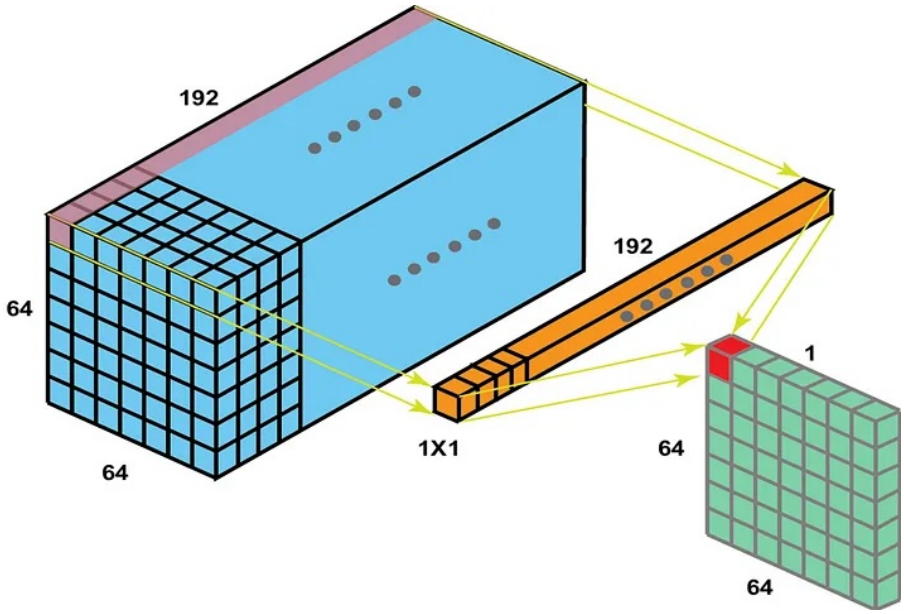


Figure 44: 1×1 convolution

By adding 1×1 convolutions before larger convs and after max pool we can:

- **Control the time complexity** of the larger convolutions by reducing the channel dimension.
- **Control the number of output channels** by reducing the depth of the max pool output.

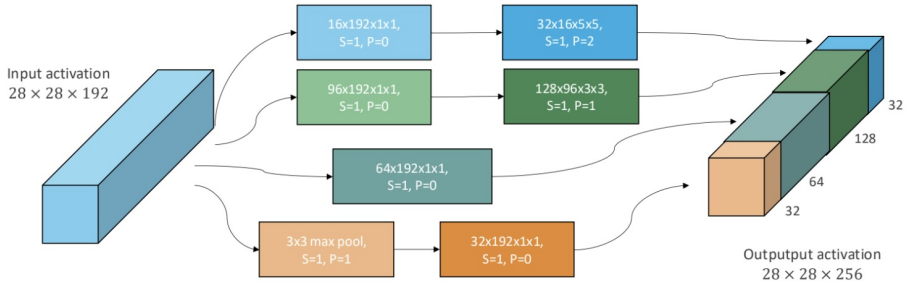


Figure 45: Inception module in GoogLeNet

In 45 we can see how Google exploited the properties of 1×1 convolutions to make an incpetion module without making the complexity explode. The idea is that we take our input, we shrink it by using 1×1 convolutions, and then we apply the 5×5 and 3×3 spatial convolutions. We use 1×1 convolutions **after** pooling to shrink the output (we don't need to compress before the operation but after it).

Fully-connected classifier vs global average pooling As we have seen the last 3 fully connected layers were the heaviest, since we had very high dimensional spatial information. To reduce the number of parameters needed between convolutional features and fully connected layers we get rid of spatial dimensions by averaging them out, because the activations should contain high level information.

3.2.4 Inception v3

Inception v3 leverages convolution factorizations to increase the computational efficiency and to reduce the number of parameters, the two main methods are:

- 3×3 followed by 3×3 instead of a 5×5 (VGG idea).

- 3×3 can be factorized in a 3×1 and 1×3.

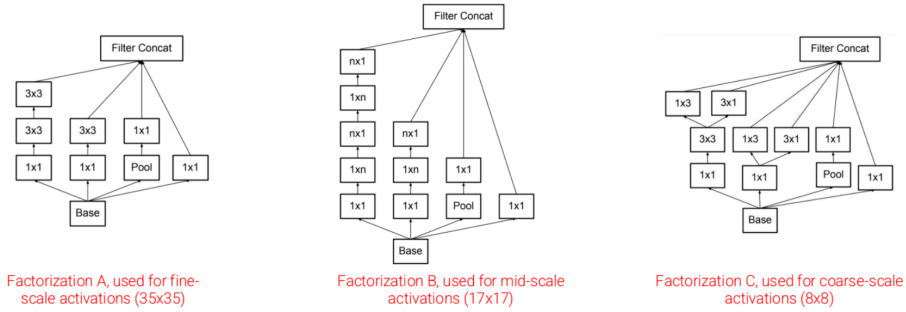


Figure 46: Instead of always using the same inception layers, we use 3 different inception layers

3.2.5 Residual Networks (ResNet)

ResNet was invented to address the problem of vanishing gradients and degradation in very deep neural networks. The core innovation in ResNet is the residual block, which introduces skip connections. In traditional networks, each layer feeds directly into the next layer. In ResNet, the input to a layer is also added to its output, by creating what's called a skip connection (like an identity shortcut). This means that instead of learning a direct mapping $H(x)$, the network learns the residual function $F(x) = H(x) - x$.

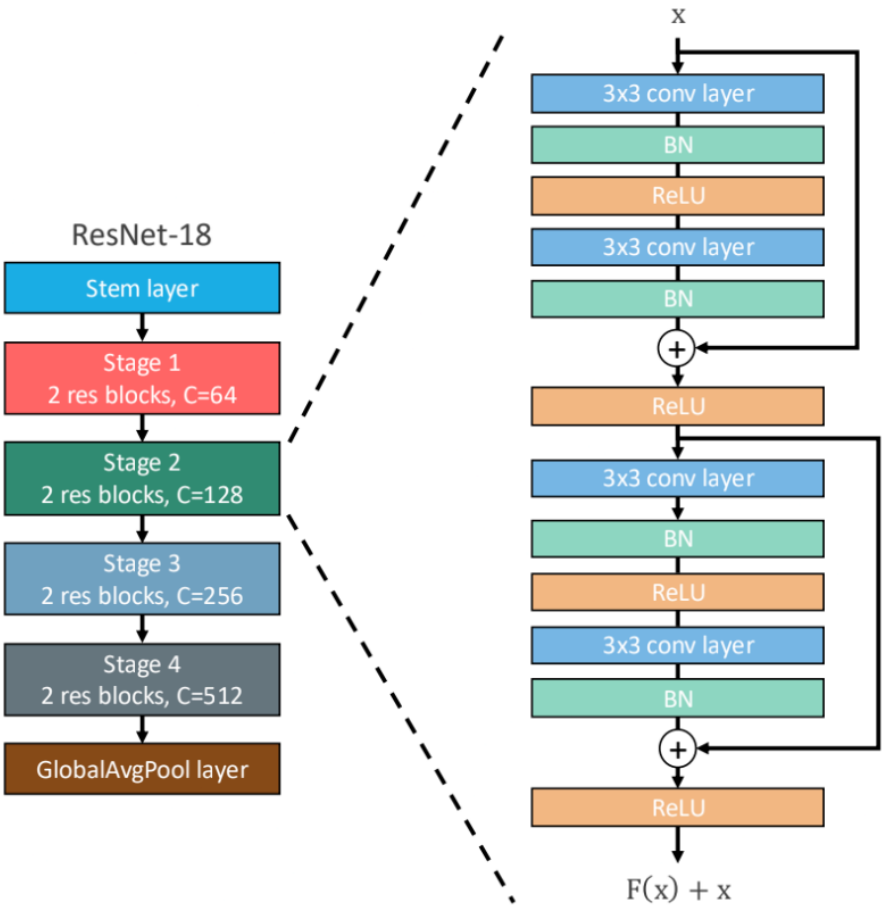
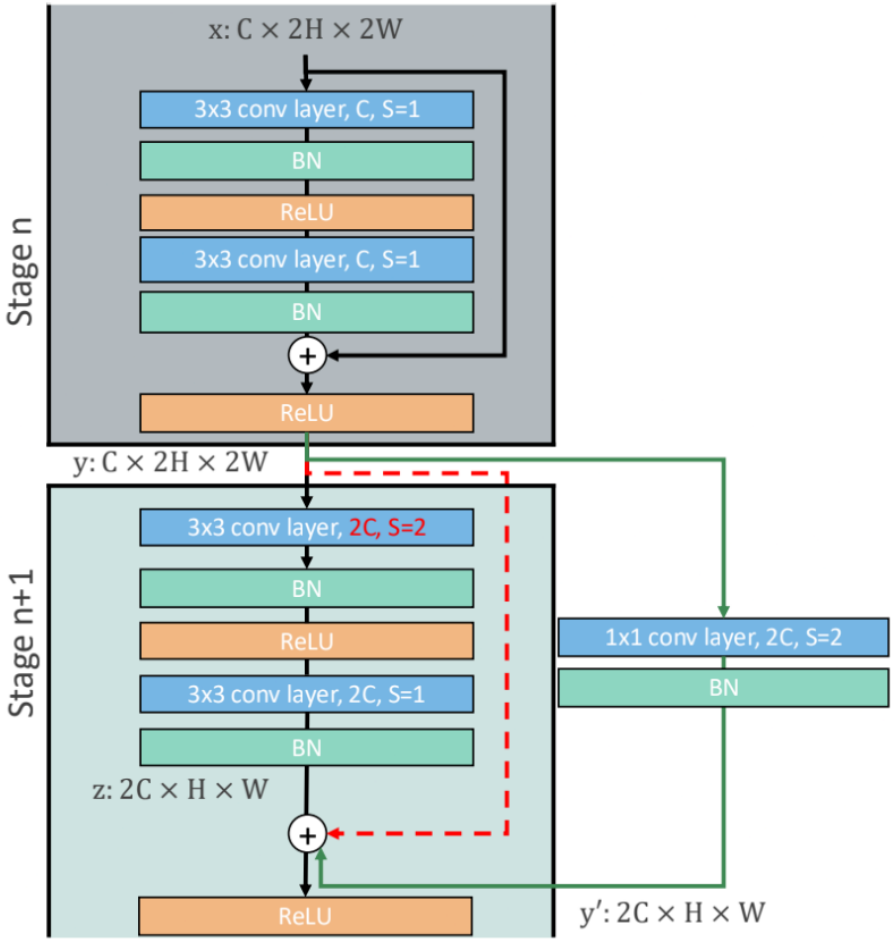


Figure 47: The ResNet architecture

The network is a stack of stages with fixed design rules (inspired by VGG):
- Stages are a stack of residual blocks.
- Each residual block is a stack of two 3×3 convolutions with batch-norm.
- The first block of each stage halves the spatial resolution (with stride 2 convolutions) and doubles the number of channels.

- It uses stem layer and global average pooling as GoogLeNet.



ResNet is still the standard baseline for most tasks today.

The residual blocks described so far **cannot be used as the first block of a new stage**, because the number of channels and the spatial dimensions do not match along the residual connection. The authors then decided to use a 1×1 convolution with stride 2 and $2C$ output channels to expand the dimension of the channels.

Bottleneck residual block A bottleneck residual block is a design used in very deep ResNets (e.g. ResNet-50, 101, 152) to enable training with more layers while maintaining computational efficiency.

3.2.6 ResNeXt

Inception modules are effective multi-branch architectures, which can be thought of as following a split-transform-merge paradigm. ResNeXt is a simpler way to realize multiple pathways, since it decomposes each bottleneck residual block of ResNets into G parallel branches. The number of branches (cardinality) is a new hyperparameter. Once G is chosen, it is possible to obtain a $G \times d$ ResNeXt block with complexity similar to the original ResNet block. d must be proportional to C.

ResNeXt block as grouped convolutions Grouped convolutions split input channels into G independent groups. Each group processes a subset of channels, reducing computational cost and parameters while maintaining feature diversity.

An input with C channels is divided into G groups, each with $\frac{C}{G}$ channels. Each group has its own filters, producing $\frac{K}{G}$ output channels (for K total filters). The outputs are concatenated, resulting in K channels.

This is useful because it's more efficient and each group learns distinct features, enhancing model capacity.

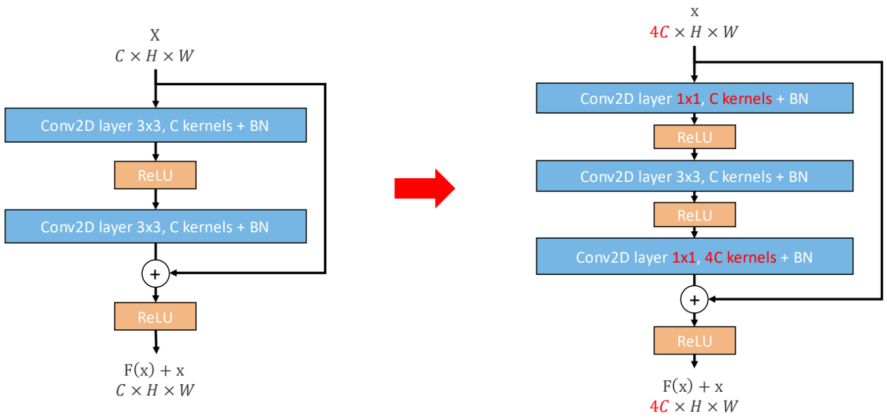


Figure 48: Instead of working with 3×3 convolutions we go from $4C$ down to C and then expand again

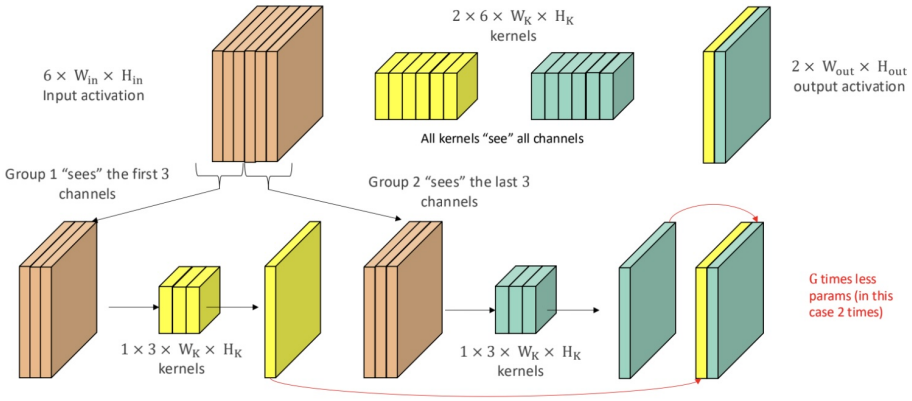


Figure 49: Grouped convolution with $G = 2$

ResNeXt introduces cardinality (the number of groups of G) as a new dimension. It uses grouped convolutions to create parallel pathways within a residual block, improving feature learning without increasing depth/width.

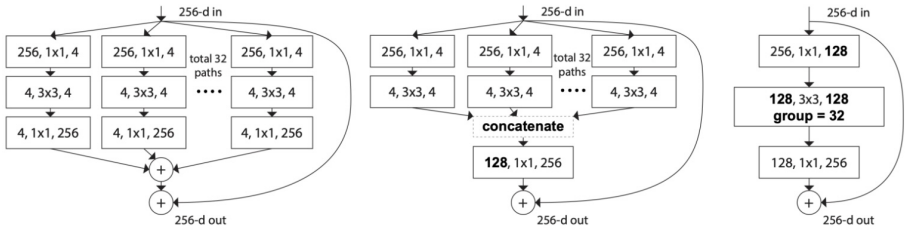


Figure 50: The ResNeXt block as grouped convolutions, on the left we have a normal convolutional block, in the middle we split to reduce computation and on the right we join again.

The structure of the ResNeXt block is:

- 1×1 convolution to compress the input channels.
- 3×3 grouped convoltuion to process compressed features in G parallel groups (cardinality). Each group applies transformations independently.
- 1×1 convolutions expands channels back.
- skip connection adds the original input to the output (residual learning).

ResNeXt is more efficient because there are fewer parameters than widening/deepening the network, and by increasing G we can improve the performance without drastically increasing the compute.

3.2.7 Squeeze-and-Excitation Networks (SENet)

It's the last paper which won CVPR. It proposed the squeeze-and-excitation module to capture global context and to use it to reweight channels in each block. Given the output U of a block with shape $C \times H \times W$ we do:

- Global average pooling to squeeze.
- General bottleneck formed by two fully connected layers with reduction ratio 16.

3.2.8 Depthwise Separable convolutions

F. Chollet one year after the SENet paper proposed the Depthwise Separable convolutions. The standard convolutions filter features based on the convolutional kernels and combine features in order to produce new representations. This is very expensive. Depthwise separable convolutions separates filtering and combination, and this aggressively limits the computational cost.

3.2.9 Inverted residual blocks

MobileNet-v2 introduced the bottleneck residual block to scale up the model depth by increasing the number of layers per block while keeping the computation and number of parameters roughly constant. To this end, it uses a pair of 1×1 convolutions, where the first compresses the number of channels, while the second one expands them. Hence, the 3×3 convoltuion operates in the compressed domain.

As we know, compression usually results in information loss, so in MobileNet-v2 inverted residual blocks were proposed. In this blocks, the first 1×1 convolution expands the channels, while the second compresses them back, according to an expansion ratio t. To limit the increase in computation, the inner 3×3 convolution is realized as a depthwise convolution (single filter per input channel instead of applying the filters across all the input channels).

3.2.10 MobileNet-v2

MobileNet-v2 is a stack of inverted residual blocks with ReLUs in between. The number of channels grows slowly compared to previous architectures, as we can see in the purple rectangle in figure **??**

Whenever spatial dimensions or number of channels do not match between input and output, there are no residual connections.

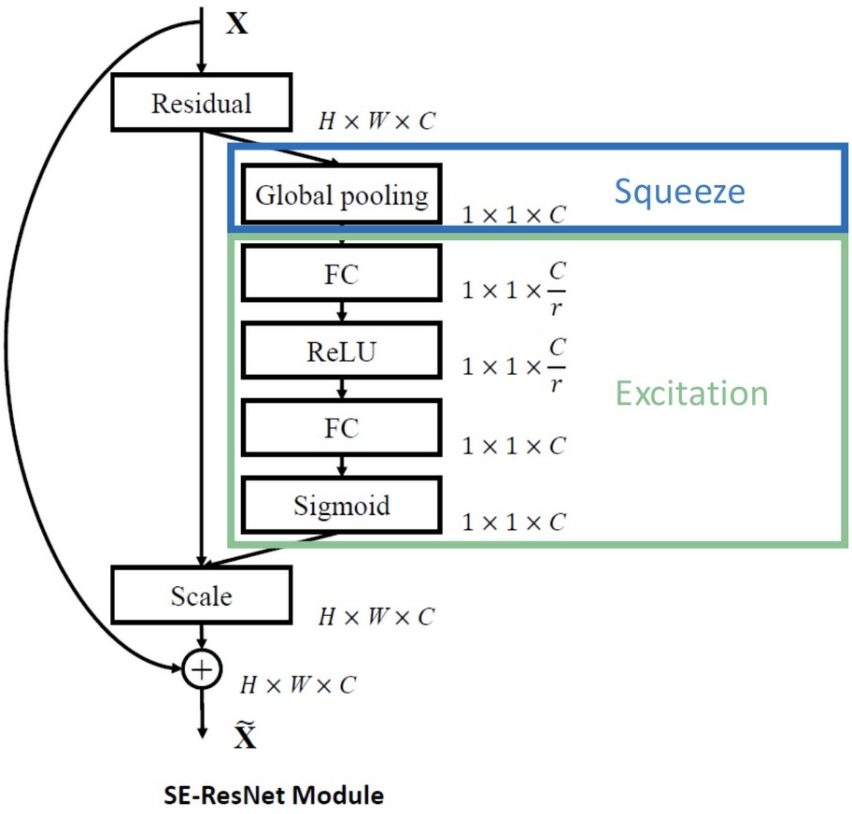
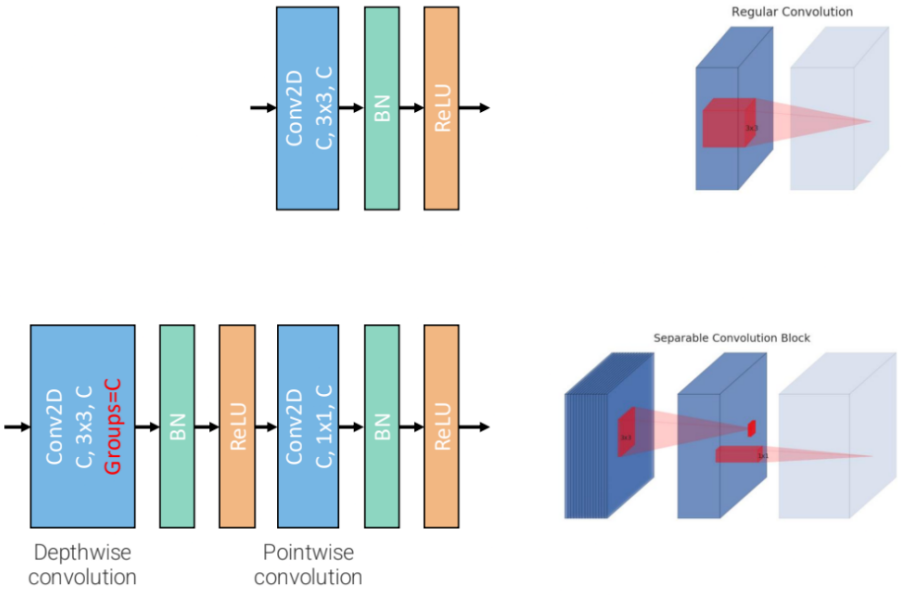
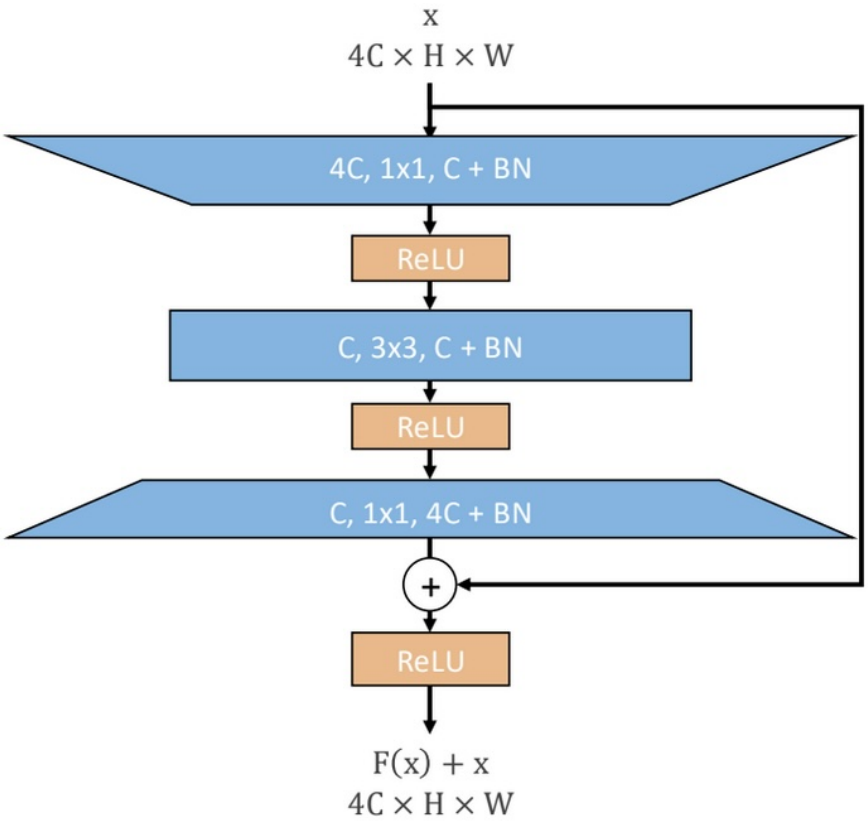
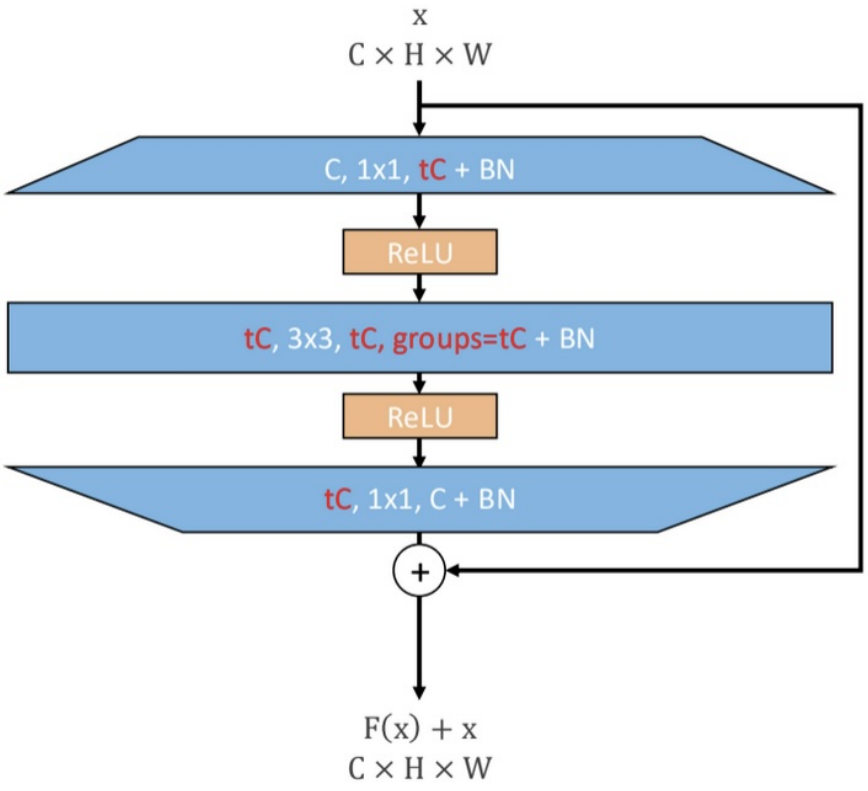


Figure 51: SENet block diagram




x
4C × H × W
4C, 1x1, C + BN
ReLU
C, 3x3, C + BN
ReLU
C, 1x1, 4C + BN
+
ReLU
F(x) + x
4C × H × W


x
C × H × W
C, 1x1, tC + BN
ReLU
tC, 3x3, tC, groups=tC + BN
ReLU
tC, 1x1, C + BN
+
F(x) + x
C × H × W

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

3.2.11 EfficientNet

EfficientNet proposed to design the network based on resources. As we know there are 3 dimensions for scaling a neural network: width, depth and resolution.

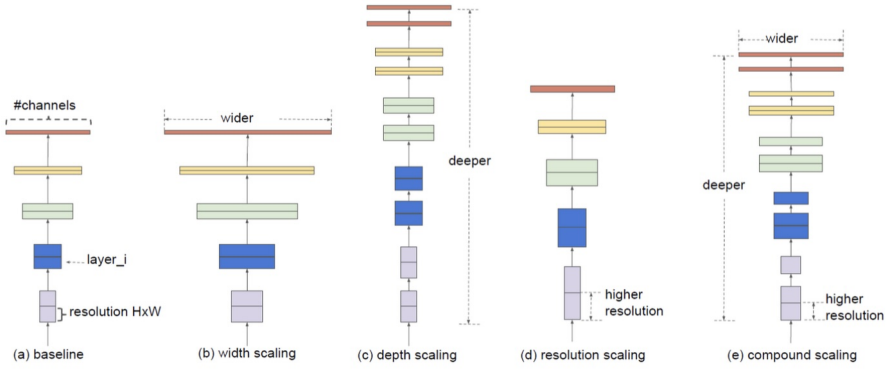


Figure 52: Different types of NN scaling.

Bigger networks with larger width, depth or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturates after reaching 80%, which is a limitation of single dimenison scaling. **Scaling dimensions are not independent**. Intuitively, for higher resolution images, we should increase network depth, such that the larger receptive fields can help to capture similar features that include more pixels in bigger images. Correspondingly, we should also increase netowrk width when the network resolution is higher, in order to capture more fine-grained patterns with more pixels in high resolution images.

EfficientNet uses a compound coefficient ϕ to scale all dimensions in a principled way.

3.3 RNNs & Transformers

In RNNs, which were used to avoid the problem of fixed size input, the gradient computetion involves performing a forward propagation pass moving from left to right through the unrolled graph of the network, followed by a backward propagation pass moving right to left through the graph. The basic problem is that by propagating the gradient over many stages it tends to vanish or explode, so learning long-term dependencies can take a lot of time.

Encoder-Decoder architecture In this type of architecture the encoder processes the input sequence. It emits the context C, usually as a function of its final hidden state. The decoder is

conditioned on that fixed-length vector to generate the output sequence. If the context C is a vector, then the encoder RNN is simply a sequence-to-vector RNN, and the decoder is a vector-to-sequence RNN. The **bottleneck problem** arises because the context C outputted by the encoder RNN has a dimension that is too small to properly summarize a long sequence.

Attention Existing model for neural machine translation encode a source sentence into a fixed-length vector from which a decoder generates a translation. Attention provides a solution to the bottleneck problem. We provide to the decoder the learned token start, then we do the dot product between the start token and the hidden states. All the other hidden states give us a probability for each token.

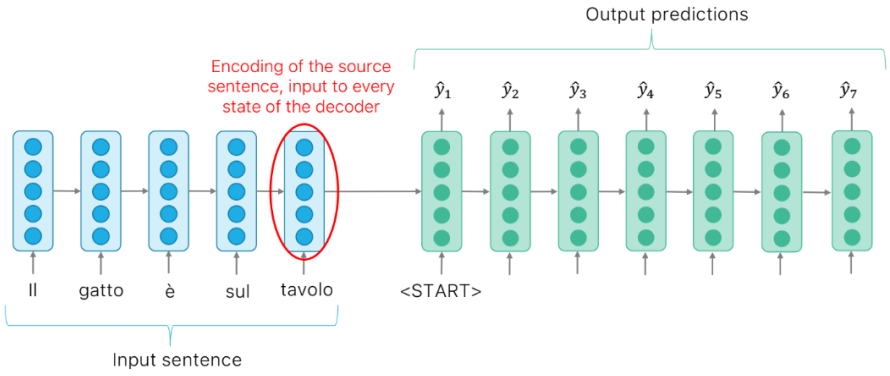


Figure 53: In the classic RNN, the last hidden state is used for all the translation.

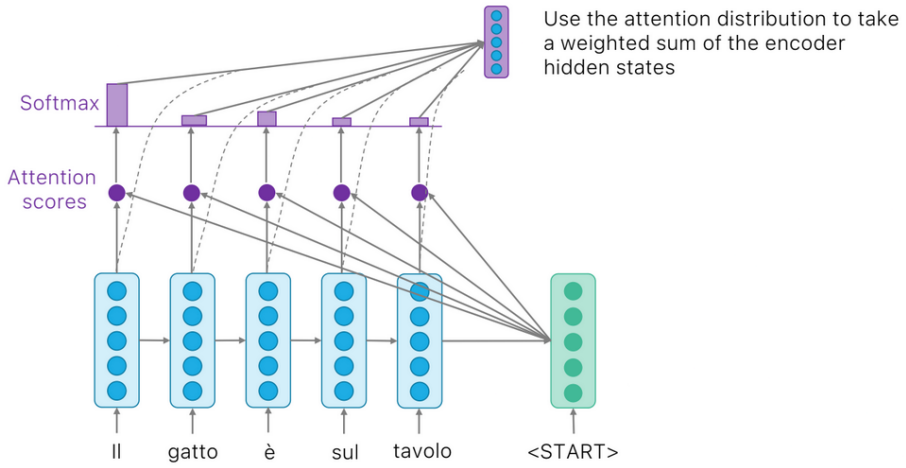


Figure 54: In the attention mechanism, at each step, the current hidden state is compared with all the previous hidden states by computing an attention score.

3.3.1 Transformer architecture

The inherently sequential nature of RNN precludes parallelization within training examples, which becomes critical with longer sequence lengths. The **transformer** is the first model relying entirely on self-attention (and cross-attention) to compute representations of its input and output without using RNNs or convolutions. It's still an encoder-decoder architecture, where the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is **auto-regressive**, consuming the previously generated symbols as additional input when generating the next.

Transformer encoder In the encoder we turn each input word into a vector by using an embedding layer. Each word is embedded into a vector of size d_{model}. If the encoders are stacked, the embedding only happens in the bottom-most encoder. Since we process the words in parallel we

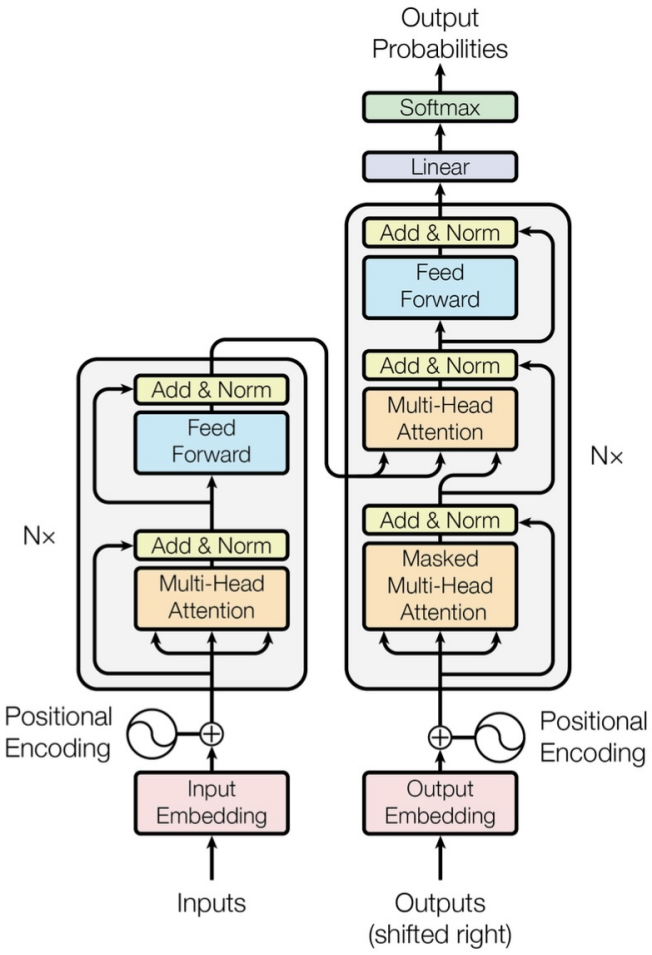


Figure 55: The transformer architecture from the paper "Attention is all you need".

lose positional information, so by using positional encoding we keep this information. The positional encoding has the same dimension d_{model} as the embeddings, so that the two can be summed. There are learned and fixed positional encodings; the authors used sine and cosine functions of different frequencies.

Each layer has two sub-layers:

- A multi-head self attention mechanism.
- A fully connected feed-forward network. The exact same feedforward network is independently applied to each position (shared parameters).

There is also a residual connection around each of the two sub-layers, followed by layer normalization. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model}.

Transfromer decoder The decoder is also composed of a stack of N identical layers. In addition of the two sub-layers of the encoder, the decoder has a third sub-layer, which performs multi-head attention over the output of the encoder stack (cross-attention). As in the encoder tehre are residual connections and layer normalization.

The self-attention sub-layer in the decoder stack is modified so as to **prevent positions from attending to subsequent positions**. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Self-Attention The first setp in calculating self-attention is to create three vectors from each of the embedded words:

- a Query vector.
- a Key vector.
- a Value vector.

These vectors are created by multiplying the embeddings by three matrices learned during the training process. Their dimensionality is $d_k = \frac{d_{\text{model}}}{h}$, while the embedding and encoder input/output vectors have dimensionality of d_{model}.

The matrices are used to compute the self-attention score.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

As we can see there is a little normalization done with $\sqrt{d_k}$, to try to avoid having the results of the softmax squeezed too much into a single value (one hot). The scores of the self-attention determine how much focus to place on other parts of the input sentence as we encode a word at a certain position.

MultiHead Attention The purpose of multi-head attention is to linearly project the queries, keys and values h times with different, learned linear projections. On each of these projected versions of queries, keys, and values we perform the attention function in parallel. The outputs of different heads are then concatenated and once again projected. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

Multi Head Self-Attention computation

Cross-Attention in the cross attention layers, the queries come from the previous decoder layer (masked self-attention), and the keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. It's a very powerful mechanism used in modern generative models to introduce variable lengths conditions.

3.3.2 Vision Transformer (ViT)

We split an image into patches and provide the sequence of linear embeddings of these patches as an input to a Transformer. The image patches are treated the same way as tokens in an NLP application.

As an alternative to raw image patches, the input sequence can be formed from feature maps of a CNN. In this hybrid model, the patch embedding projection is applied to patches extracted from a CNN feature map.

To handle $2D$ images, we reshape the image $x \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened $2D$ patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$.

- (H, W) is the resolution of the original image.
- C is the number of channels.
- (P, P) is the resolution of each image patch.
- $N = \frac{HW}{P^2}$ is the resulting number of patches, which also serves as the effective input sequence length for the transformer.

The transformer uses constant latent vector size d_{model} through all of its layers, so we flatten the patches and map to d_{model} dimensions with a trainable linear projection. We also prepend a learnable embedding to the sequence of embedded patches, whose state at the output of the transfromer encoder serves as the image representation. The classification head is implemented by a MLP with one hidden layer. To retain positional information the ViT uses learnable 1D position embeddings.

At the start, the ViT worked very bad. This is because transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data. The convolution in combination with max pooling/striding makes CNNs approximately invariant to translation. When a module is translation invariant, it means that if we apply translation transformation on the input image the output of the module won't change. To solve the inductive bias they gave the network so much data that the classes appear anywhere.

3.4 Object Detection

Previously, we have done object detection with SIFT, but we were just able to say if the object was in the image, not where it was. Now we want to do detection by putting a bounding box around the object. We do this by returning a set of quadruples: $[x, y, h, w, o]_{i=0}^{K}$, where we have the box position, width, height, and class of the output.

The main challenges are the different lengths of the outputs (since we can detect from 0 to K images), the fact that the outputs have both categorical and spatial information, and the fact that the images are usually processed at higher resolution than neural networks for image classification.

3.4.1 Viola-Jones Object Detector

It's a general purpose object detection framework, but it has been mainly applied to faces. We will now see the three main innovations of the algorithm: the AdaBoost algorithm, cascade, and the use of integral images.

AdaBoost algorithm A Weak Learner (WL) is a simple classifier whose error is slightly better than random guessing. Boosting is a way to train and build an ensemble of M weak classifiers to obtain a Strong Learner SL. After training a weak learner, the examples are re-weighted in order to emphasize those which were incorrectly classifier by the previous weak classifier.

The **AdaBoost** algorithm steps are:

1. Given N training samples $(x^(i), y^(i))$, assign equal weight to each training example $w^(i) = \frac{1}{N}$.
2. Iterate for $j = 1, ..., M$ weak learners:
   (a) Fit the best classifier WL_j to the training data by using the current weights.
   (b) Compute the weighted error rate $\epsilon_j = \sum_{i:\, x^{(i)} \text{ is missclassified}} w^{(i)}$.
   (c) Compute $\beta_j = \frac{1-\epsilon_j}{\epsilon_j}$.
   (d) Updates weights $w^(i) = w^(i)\beta_j$ for wrongly classified examples.
   (e) Re-normalize $w^(i)$ to sum to 1.
3. The Strong Learner is given by a weighted majority vote $SL(x) = \sum_j \ln \beta_j WL_j(x) > 0$, con $\ln \beta_j = \alpha_j$.

The idea is that missclassified examples gain weight, forcing subsequent learners to focus on harder cases.

Haar-like features Weak classifiers used to detect faces are simple rectangular filters, composed of 2 to 4 rectangles applied at a fixed position within a 24×24 patch. Even with this simple definition there are over 160 k possible filters in a 24×24 patch. AdaBoost is used to select a small subset of the most effective filters.

Dataset The dataset consisted of 4916 hand labeled faces, scaled and aligned to a base resolution of 24×24 pixels. The non-face windows were collected by selecting random sub-windows from a set of 9500 images which did not contain faces.

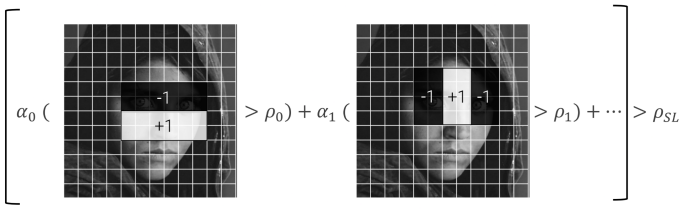


Figure 56: The 2 most effective features selected by AdaBoost on the training set.

Integral images and fast feature computation To speed up the computation of rectangular features, the authors proposed the use of so-called integral images II, where $II(i,j) = \sum_{i' \leq i, j' \leq j} I(i', j')$.

We can compute the value of $II(i,j)$ just by looking at the 3 neighbors: $II(i,j) = II(i, j-1) + II(i-1, j) - II(i-1, j-1) + I(i,j)$, where II are the values in the integral image, and I is the value in the original image.

With integral images we could have an overflow problem if numbers get too big.

Multi-scale sliding window detector At test time, the strong classifier is applied to all spatial locations in the image. Multi-scale detection is necessary, since faces are not necessarily 24×24. To achieve good performance, about 200 features are used to classify each patch. Even if each feature can be computed very fast (thanks to integral images) there are still too many windows in an image to achieve real-time performance.

Faces are far less frequent in an image than background regions. Most of the time is wasted computing a lot of features for background patches. The key idea is then to reject most of the easy background patches with a simpler classifier which can be ran very fast.

Box overlap There will be several overlapping detections. To check if two boxes overlap we measure the Intersection over Union (IoU) score: $IoU(BB_i, BB_j) = \frac{|BB_i \cap BB_j|}{|BB_i| + |BB_j| - |BB_i \cap BB_j|}$. 0.75 corresponds to a good overlap, 0.90 to a perfect one. To obtain a single detection out of a set of overlapping boxes we perform Non Maxima Suppression of boxes. Non Maxima Suppression considers the highest scoring Bounding Box, eliminates all the boxes with overlap greater than a threshold, and repeats this until all the boxes have been tested.

A detection is considered a True Positive if its overlap with the ground truth BB^{GT} is greater than ρ_{IoU}.

3.4.2 Transfer Learning

If we can assume that only one object is present in the image, object detection simplifies to object localization. Transfer learning is a technique where a model trained on one task is reused (or "transferred") for a different but related task. Instead of training a model from scratch, you start with a pre-trained model and fine-tune it for your specific problem. To solve object detection we can reuse any architecture used in image classification, by adding a regression head for predicting the bounding box.

We can apply a classification CNN as a sliding window detector. We have to add a background class to discard background patches. The problem is that there are too many boxes to try, since we have to try lots of positions with different scales and aspect ratios. The solution is to use region proposals.

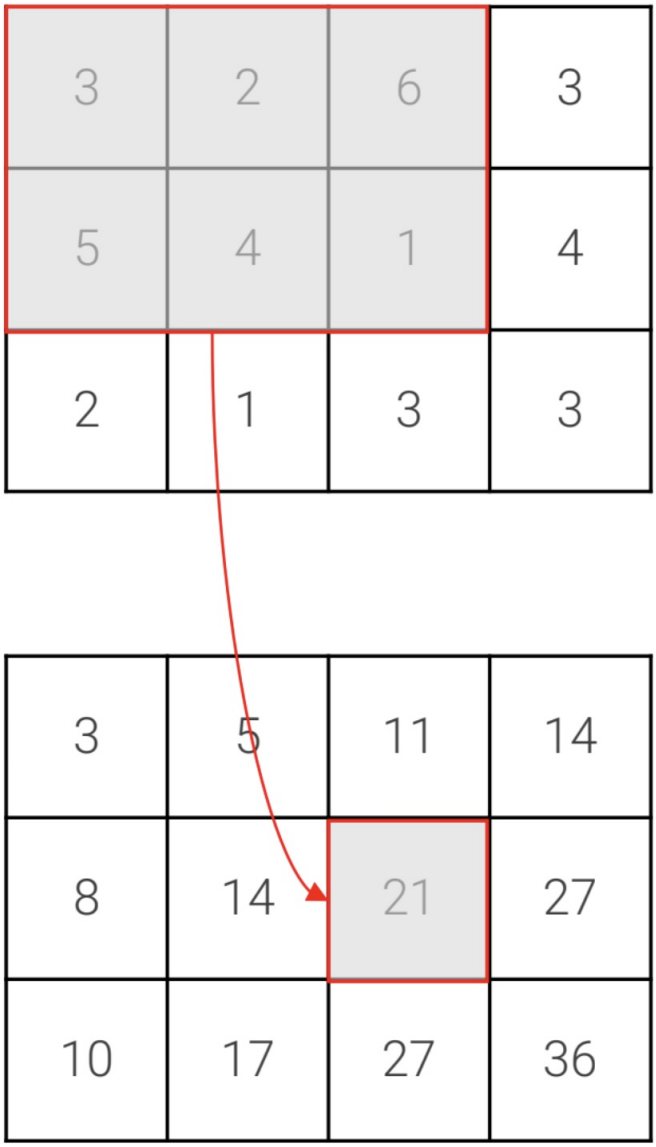


Figure 57: $II(1,2) = II(1,1) + II(0,2) - II(0,1) + I(1,2) = 14 + 11 - 5 + 1$

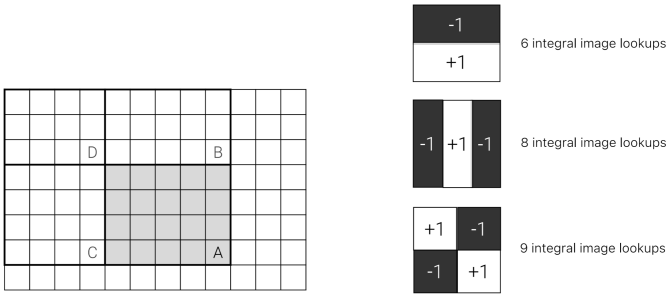


Figure 58: Rectangular filters can be computed in constant time with integral images

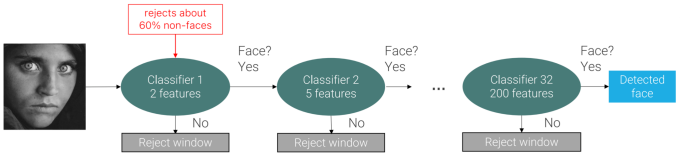


Figure 59: We use more and more features for face detection classifier after classifier

3.4.3 Region proposals

Region proposal algorithms inspect the image and attempt to find regions of an image that likely contain an object. The algorithm as a first step oversegments the image into highly uniform regions, and then, based on similarity scores of color, texture and size it iteratively aggregates them.

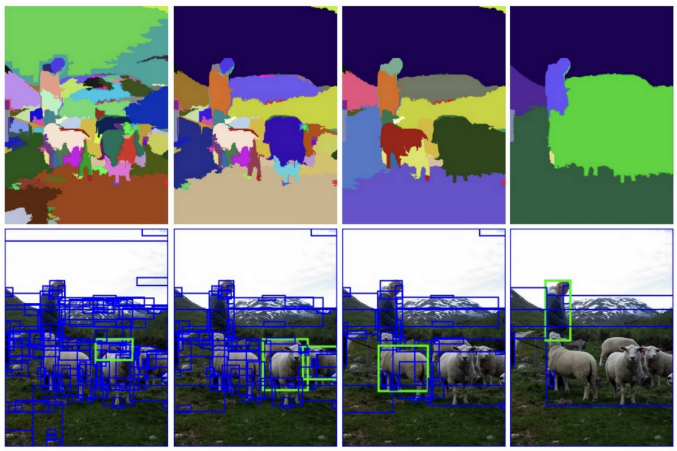


Figure 60: The 2 most similar regions are grouped together and then new similarities are calculated between the resulting region and its neighbors

3.4.4 R-CNN: Region-based CNN

Region-based CNN is the first object detector network. As we can see in 61 we run selective search to get about 2000 proposals, then we anisotropically (not uniformly across all directions) warp these proposals and add some pixels of context into a fixed size, to be able to input them in AlexNet, and for each proposal we get a class and a bounding box correction.

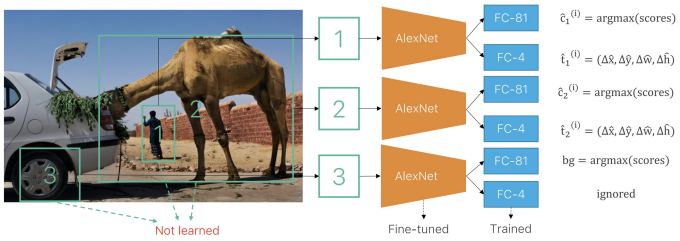


Figure 61: R-CNN network architecture

The problem is that since we have 2 different algorithms (one for the region proposal and one for AlexNet) we can't backpropagate through both of them.

3.4.5 Fast R-CNN

In the Fast R-CNN architecture showed in 62 we can see that the model has been modified to run the first few layers of AlexNet on the original image, apply RoI pooling to crop and warp convolutional features according to proposal, and then run a small per-region network on each region to get the output class and bounding box correction. This results in a faster networks since we don't have to apply 2000 CNN forward passes per image, but we apply most of the CNN before warping.

Fast R-CNN uses the same bounding box correction of R-CNN, but with a smooth L1 loss (equal to L2 near the origin, but smoother elsewhere).

RoI pooling The RoIPool layer converts activations inside Region of Interest, corresponding to rescaled Selective Search regions, into activations with fixed spatial dimensions, which are the ones required by the remaining layers of the network. To do RoI pooling, given a region:

1. Snap the region to the grid.

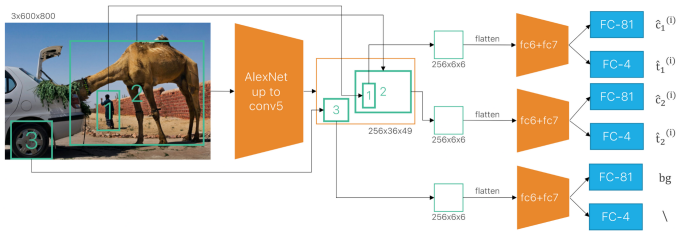


Figure 62: Fast R-CNN network architecture

2. Apply max pooling kernels with approximate size $[H_r/H_o] \times [W_r/W_o]$ and approximate stride $s = [H_r/H_o] \times [W_r/W_o]$.

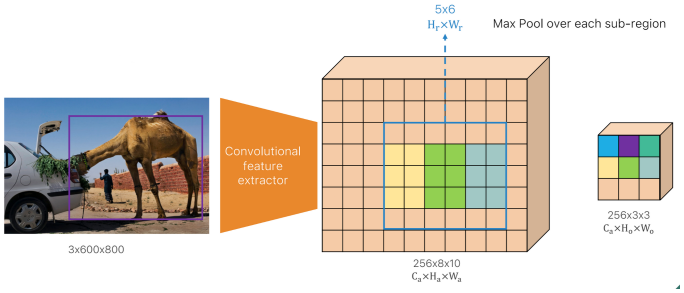


Figure 63: RoI pooling.

3.4.6 Faster R-CNN

In Fast R-CNN proposals are not learned, so the selective search is slow. In Faster R-CNN they introduced a RPN (Region Proposal Network), which learns to predict the proposal box and the objectness score.

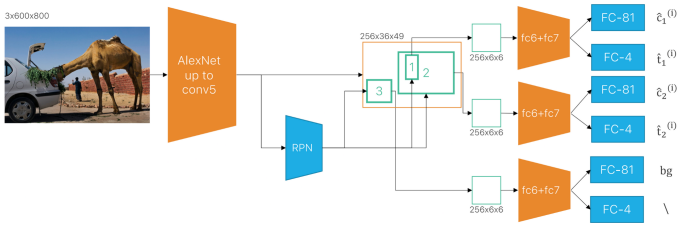


Figure 64: Faster R-CNN network architecture.

Region Proposal Network Region Proposal Networks are applied to a small 3×3 window, which is approximately an object sized window in the original image and doesn't correspond to what we see in 65, predicts the objectness and proposal bounding box.

When the objectness score is low, the localization head produces its output, but it's meaningless and it will be ignored.

Region Proposal Network with anchor We can simplify the creation of proposals. An easier task may be to correct an input proposal, as done in R-CNN.

Objects have different scales and asepct ratios, so we use several anchors with different scale and aspect ratios. The RPN predicts k objectness scores and k corrections.

When training RPNs, given an image and a ground truth bounding box if the anchor has a low IoU with all ground-truth boxes in an image, it's considered as background.

In standard Faster-RCNN, the RPN processes only the last activation of the feature extractor, which is a semantically rich signal since it includes higher level features than previous activations, but very coarse in spatial resolution. Even if small scale anchors are provided, it may miss objects smaller than the grid size.

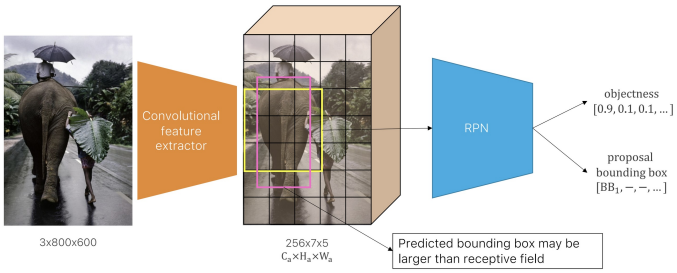


Figure 65: Region Proposal Network architecture.

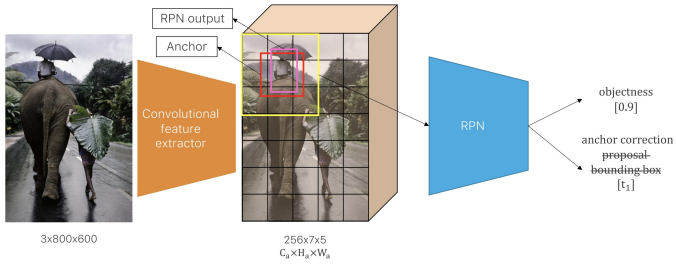


Figure 66: Our proposal here is a fixed scale and aspect-ratio box, known as anchor.

3.4.7 Feature Pyramid Network (FPN)

To do object detection at multiple scales, the naive approach is to obtain feature pyramids with CNNs, by running a CNN at each scale of the image, and then perform detection on each activation. This is very bad for inference and training times.

A better approach is to give as input to the Region Proposal Network different stages of a CNN. This is done because CNNs produce a pyramid of features, with different semantic qualities at different depths. But different stages of a CNN have different input sizes (more channels as we go deeper in the network).

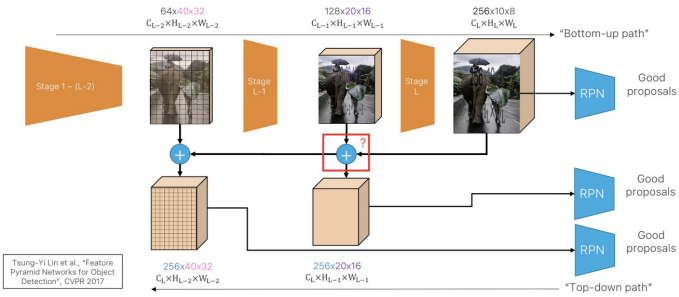


Figure 67: Feature Pyramid Network architecture.

Feature Pyramid Networks overcome this problem by mergoing coarser but higher level features with less effective but more spatially localized features. This has a limited computational overhead.

In 68 we can see how the FPN sums together different size pictures. To fix the resolution we use Nearest Neighbor upsampling. To fix the number of channels we use 1×1 convolutions.

3.4.8 Faster R-CNN with FPN

We can use the FPN for the detection stage. We use the FPN as a feature extractor which provides a pyramid of activations. Then we can do the usual processing with the RPN and the per-region MLP.

Introducing the FPN enables better model accuracy but gives no speedup, since we didn't change anything with the broader architecture.

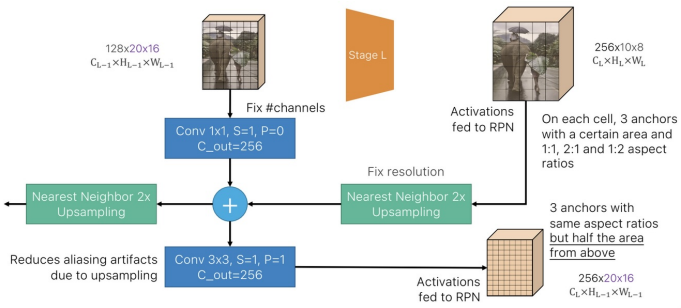


Figure 68: FPN top down path and lateral connections.

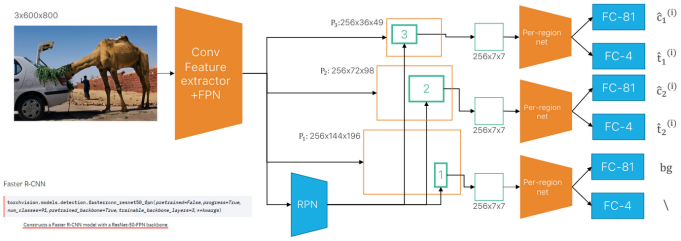


Figure 69: Faster R-CNN with FPN architecture; as we can see we use the FPN together with the convolutional feature extractor.

3.4.9 One Stage detectors

As wee can see from the image 69, we have two main stages in the network architecture. The first stage runs the expensive backbone feature extractor with the FPN on the full image, and the RPN generates the proposals for the bounding boxes. This is done one time per each image. The second stage, which is ran once per proposal, has a RoI pool, and a per-region classification and correction.

Instead of a RPN we could try to do all the job by using anchors, since the only things they are missing is learning the anchor dimensions and the class prediction.

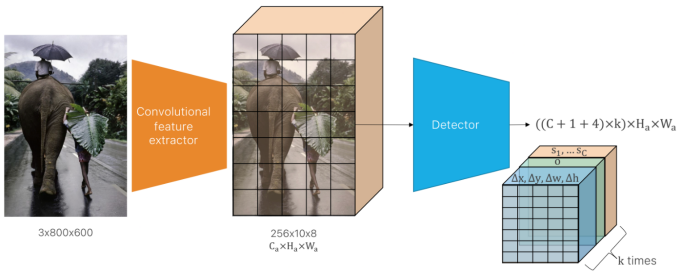


Figure 70: in one-stage detectors we can stack the outputs in a combined single tensor with the appropriate number of channels.

3.4.10 SSD: Single Shot MultiBox Detector

Multiple detectors, which are 3×3 convolutions with $k_s \times (C + 1 + 4)$ output channels where k_s is the number of anchors at that scale, are applied to several activations in the backbone (VGG in this example) and to other activations, created by extending the number of convolutional layers.

3.4.11 YOLOv3

YOLOv3 uses a custom backbone (DarkNet-53), optimized to have a good trade-off between classification accuracy and speed. It uses the idea of multi-scale detections on features with different spatial resolutions, as in FPN but it concatenates (C) activations from different stages instead of summing them.

Anchors size and aspect ratios are learned by running k-means clustering on ground-truth boxes on a specific dataset. We use anchor because we want to start as close as possible to a good

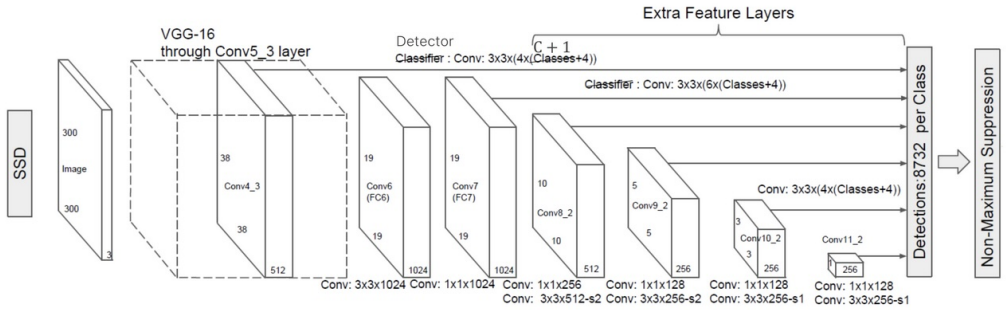


Figure 71: Single Shot MultiBox Detector architecture

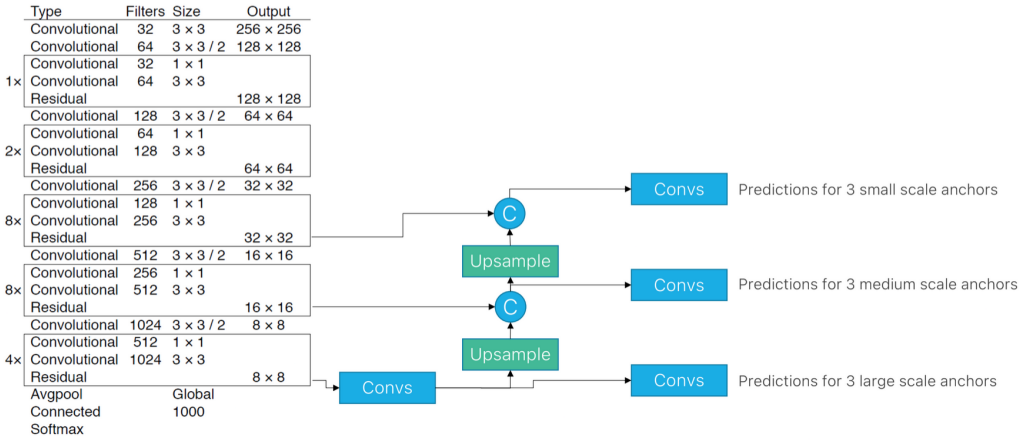


Figure 72: YOLOv3 architecture

solution. The produced anchors are similar on different datasets.

3.4.12 Retina Net

Retina Net is a plain one-stage detector built on top of standard ResNets with FPN. Classification and regression heads are stacks of 3×3 convolutions and do not share parameters (which is the main difference with RPN) as they are solving tasks which require different representations.

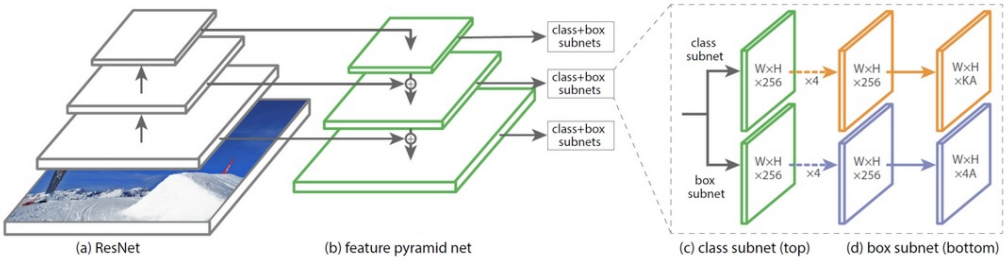


Figure 73: Retina Net architecture

The problem of class imbalance Object detection is an imbalanced problem: the number of negative boxes far outweighs the number of objects in a typical image. This imbalance causes two problems:

1. The "easy negatives" can overwhelm training and lead to suboptimal models, which are less effective at discriminating between negative and positive examples.

2. Training is inefficient as most locations in a randomly sampled mini-batch are "easy negatives" (negatives with low objectness score) that contribute no useful learning signal.

Two stage classifiers partially and naturally solve these problems, because they implicitly sample "hard negatives" by training only on the top scored proposals, which are more likely to contain hard-to-discriminate regions.

One-stage detectors, instead, are trained on the full set of anchors and, if negative samples are randomly selected, a mini-batch will on average contain most, if not all, easy negatives.

Retina Net proposes to work at the loss level. Let's consider for simplicity a binary classification problem; the usual Cross Entropy loss in the binary case is:

$$\mathcal{L} = -(y \times \ln(p) + (1-y) \times \ln(1-p))$$

$$BCE(p, y) = \begin{cases} -\ln(p) & \text{if } y = 1 \\ -\ln(1-p) & \text{otherwise} \end{cases}$$

Where p is the probability assigned by the model to have label $y = 1$.

We can do a variable change $-\ln(p) \to p$. For notational convenience, we can define the probability of the true class as:

$$p_t = \begin{cases} -p & \text{if } y = 1 \\ -1-p & \text{otherwise} \end{cases}$$

And rewrite $BCE(p, y) = BCE(p_t) = -\ln(p_t)$

Focal Loss and Retina Net The standard CE loss has a non-trivial magnitude even when examples are correctly classified, i.e. when $p_t >> 0.5$, which are the easy examples. When summed over lots of easy examples of the negative class, these small loss values can overwhelm the rarer hard negative examples. To down-weigh easy examples, we can use the (binary) focal loss $BFL(p_t) = -(1-p_t)^\gamma \ln p_t$. Where γ is a tunable focusing hyper parameter (which is usually left to 2, with $\gamma = 0$ we would have the binary crossentropy).

If the model training is going good, the loss is small but most of the loss is on the easy negatives. γ helps to keep the loss dominated by the hard targets to classify. For example, with $\gamma = 2$:

- $p_t = 0.9$ then $(1-p_t)^2 = (0.1)^2 = 0.01 = \frac{1}{100}$. The Focal Loss is $\times 100$ smaller than Cross Entropy loss, so when we are highly confident and doing good predictions the loss is weighted down.
- $p_t = 0.6$ then $(1-p_t)^2 = (0.4)^2 = 0.16 = \frac{1}{6}$.
- $p_t = 0.1$ then $(1-p_t)^2 = (0.9)^2 = 0.81 = \frac{4}{5}$. The FL is similar to CE loss.

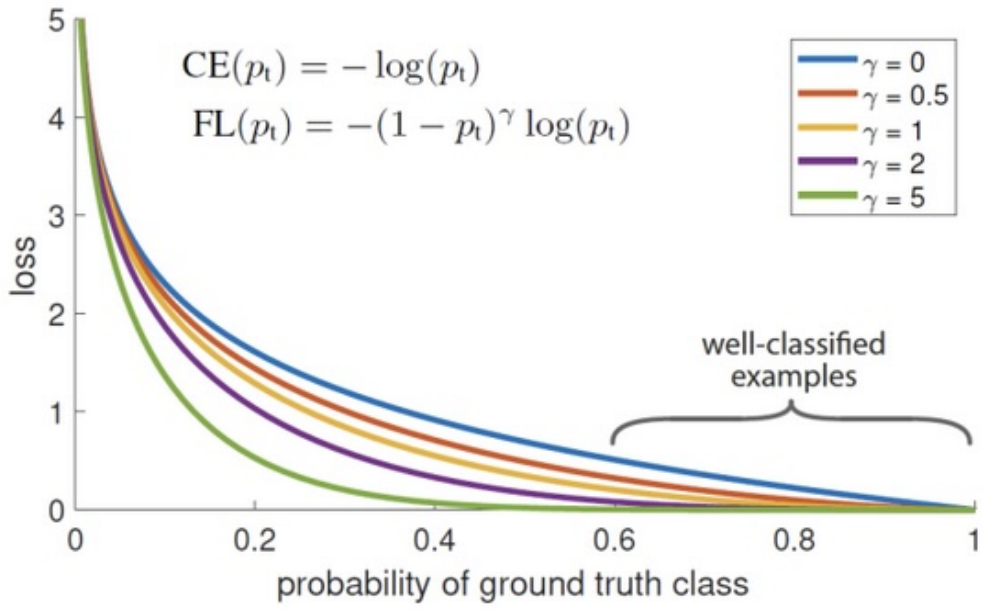


Figure 74: Focal Loss

Class weights Imbalanced classes are usually handled by class weights. Class weights serve a complementary purpose with respect to the focal loss: class weights balance the relative importance of errors in the positive or negative class; focal loss lets the training focus on hard example within each class.

In Retina Net we use both class weights and focal loss to have a better training.

3.4.13 Multi-label classification

In YOLOv3 the assignment of the class to the box is treated as a multi-label classification problem, instead of a multi class problem: each detected box can have from 1 to C classes, which are not assumed to be mutually exclusive. Hence, given the scores predicted by the classification head, YOLOv3 does not use a softmax function to compute one probability function over the set of classes, but C independent sigmoids, to compute the probability of the box belonging to each class independently.

The classification box then becomes the sum of C binary cross entropy terms. Therefore, we don't need to add a background class, since it's represented by a ground-truth vector of 0s, and it's detected at test time when all classification scores are below the detection threshold.

3.4.14 CenterNet

Anchors-based detectors, either two or one stage ones, have some limits:

- Anchors are a brute force approach to detection: we are enumerating a subset of all possible boxes, which is very inefficient. Moreover, the more anchors the better, but it's not clear what the best way to use a subset is.
- We obtain a lot of duplicated entries for an object, which must be post-processed with NMS, so the detector is in practice not end-to-end differentiable.
- Assignment of anchors to ground truths for training is based on manually selected thresholds and hand-crafted rules.

To overcome these limits, CenterNet proposed to represent objects as points (which is also the name of the paper which introduced the network), and regress their size or other properties. It's like the keypoints in Difference of Gaussians, but at a higher level, since the keypoints should be the center of my object.

Given an image of size $3 \times H \times W$, the aim of the network is to produce a heatmap $\hat{Y}$ with values $\in [0, 1]$ and size $C \times \frac{H}{R} \times \frac{W}{R}$ with output stride R. To recover the discretization error caused by the output stride, the network also predicts an offset for each center point. Finally, it also predicts the bounding box size $\hat{S}$ of size $2 \times \frac{H}{R} \times \frac{W}{R}$.

The backbone used to produce the convolutional features are fully convolutional encoder-decoder architectures devised for keypoint detection or semantic segmentation. All the 3 outputs are predicted from the last activation with a dedicated head.

Each ground truth keypoint (object center) $p = (x_p, y_p)$ of class c, is projected in the lower-resolution output heatmap $\tilde{p} = \lfloor \frac{p}{R} \rfloor$. Then, the target heatmap for its class Y_C is set to 1 at $\tilde{p}$ and updated with an unnormalized Gaussian kernel. Points can be seen as a special case of anchor: a single, shape-agnostic, anchor. Detection is performed by finding local maxima in a heatmap, with one box per object (so without using NMS), and based solely on location, not box overlap.

3.5 Image segmentation

In image segmentation we need to do per-pixel labelling, which is a costly task, since the input is the RGB image of size $W \times H$ and the output is **a category c_{uv} for each pixel** $p = (u, v)$. With $c_{u,v} \in [1, ..., C]$ which is a fixed list of categories.

We can try to use synthetic data to harvest the labels, i.e. with a videogame engine, where we exactly know what the objects are and where they are on the screen. But this has the problem of domain shift (we can't learn to do segmentation on GTA V and then try to apply the same network on the real world).

3.5.1 Generalized IoU and other measures

The intersection over union can be generalized to segmentation masks by counting the pixels. For a class $c = 1, ..., C$ we can define the Intersection over Union for the class as $IoU_C = \frac{\text{area of intersection}}{\text{area of union}}$. To compute the mean IoU score for a dataset, we average IoU_C over classes. $mIoU = \frac{1}{C} \sum_{C=1}^{C} IoU_C$.

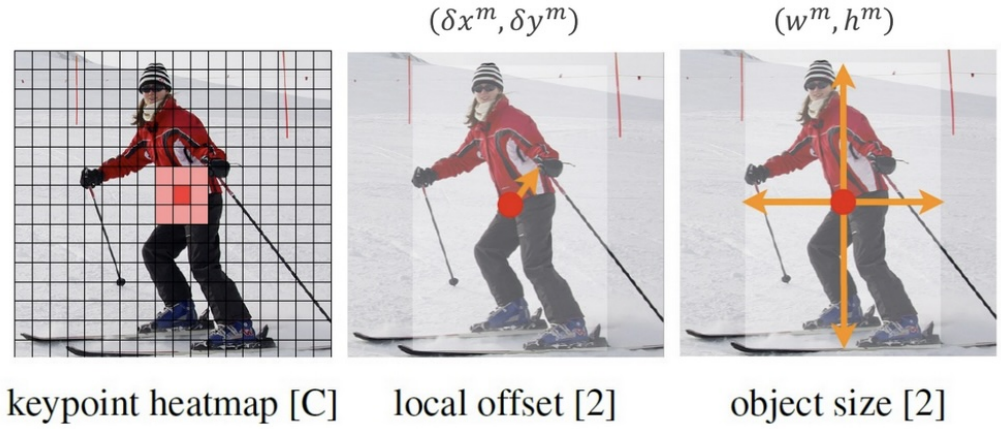


Figure 75: At inference time, given a spatial local maxima in the channel $\hat{Y}_C$ at position (x^m, y^m), the box centered at $(x^m + \delta x^m, x^m + \delta y^m)$ of size (w^m, h^m) and class c is detected, without any further post-processing.

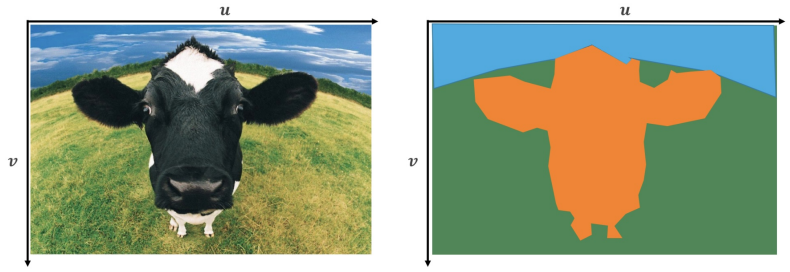


Figure 76: On the right there is the output of the image segmentation task

The $mIoU$ is the main measure to rank semantic segmentation algorithms. Other measures are:

- Pixel accuracy: the fraction of pixels correctly classified $\frac{\sum_C TP_C}{\#\text{ pixels in the dataset}}$. This has the problem of being biased towards the largest object.
- Mean accuracy: the average of the accuracy for each class $\frac{1}{C}\sum_C \frac{TP_C}{\#\text{ pixels of class } C \text{ in the dataset}}$.
- Frequency weighted IoU: weighted average of IoU_C for each class, with weights given by the frequency of a class in the dataset $\sum_C \frac{\#\text{ pixels of class } C}{\#\text{ pixels in the dataset}} IoU_C$.

3.5.2 Slow R-CNN for segmentation

We can apply the same ideas used in R-CNN: we slide the window at all possible positions. There are no proposals, since we must process each pixel, which is even slower than R-CNN for detection.

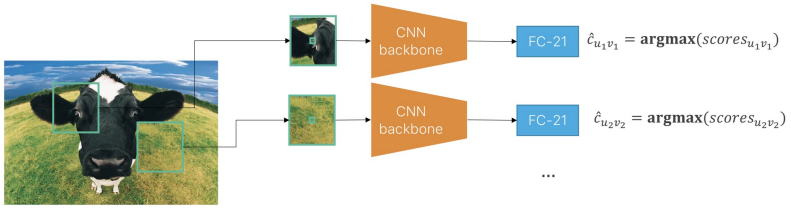


Figure 77: Network architecture for Slow R-CNN for segmentation

As the loss we use the sum of the standard multi-class loss over all pixels:

$$L(\theta, x^{(i)}, y^{(i)}) = \sum_{u,v} CE\left(\text{softmax}(scores_{u,v}), \mathbf{1}(c_{u,v})\right)$$

With $\mathbf{1}$ which is the one-hot encoded ground truth class label for the pixel (u, v).

The idea is that, as we did for detection, we can use a convolutional feature extractor and process the image through a CNN to get some activations and from that compute our output. We

have to go from the activations of the extractor to an output that has as many channels as the number of classes and the same resolution as the input.

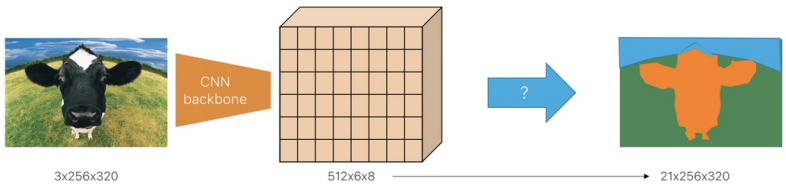


Figure 78: We have to go from 512 channels to 21 channels with a 1×1 conv. We have to go from a resolution of 6×8 to the original resolution of 256×320.

One way to perform the upsampling can be to use standard, not-learned, image processing operators (like bilinear interpolation).

3.5.3 FCN-32s

The FCN-32s (Fully Convolutional Network) is built on top of a CNN backbone. It uses the deepest feature map of the CNN, and then applies a 1×1 convolution to reduce the number of channels to the number of classes. Then the network uses a single upsampling step by a factor of 32 (hence the name 32s) to bring it back to the input image size.

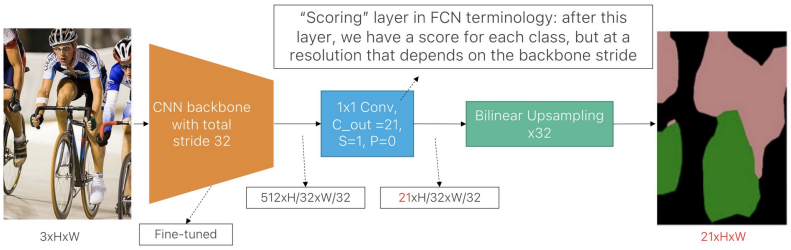


Figure 79: Network architecture for FCN-32s

The problem is that without learning a non-linear upsampling transformation, we can only uniformly spread the coarse information in the final convolutional activation, obtaining very coarse masks. The solution is to upsample multiple activations at different resolutions (like with FPN).

3.5.4 FCN-16s

The FCN-16s adds additional connections between the output and the internal layers, which are referred to as skips. This results to better spatial detail, with more accurate segmentation than FCN-32s.

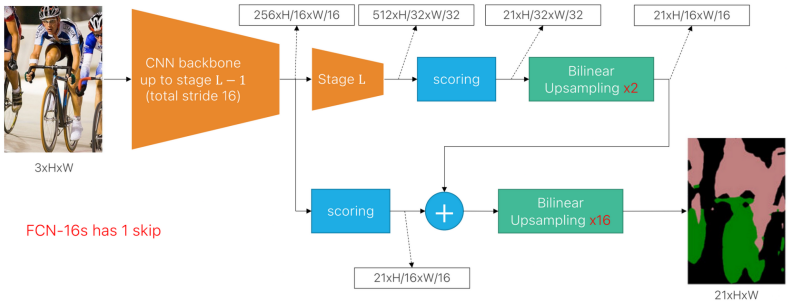


Figure 80: Network architecture for FCN-16s

3.5.5 FCN-8s

In the FCN-8s we can consider one more intermediate layer (2 skip connections) with respect to the FCN-16s.

With the VGG backbone, no more improvements were found after predicting from stride 8 activations (2 skip connections).

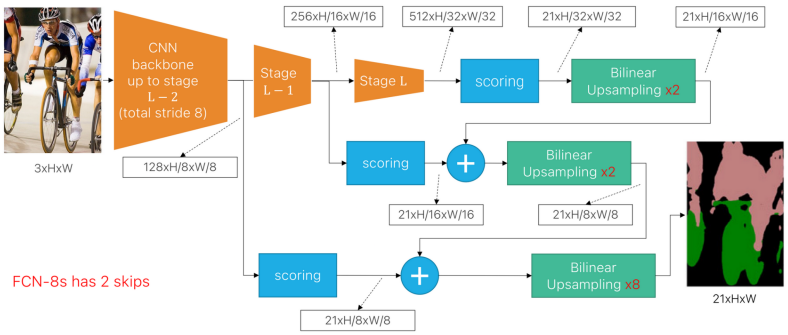


Figure 81: Network architecture for FCN-8s

3.5.6 Transposed Convolutions

Standard convolutions with stride $s > 1$ perform learnable downsampling by skipping $s - 1$ input positions between successive kernel applications. To invert this process and learn upsampling, we can use *transposed convolutions*, sometimes also called *fractionally strided convolutions* or *upconvolutions*.

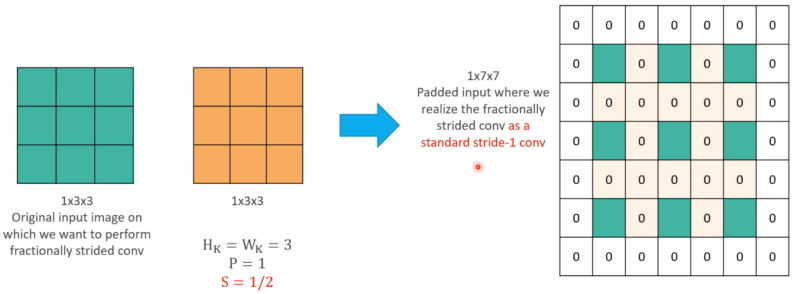


Figure 82: Inserting zeros between input pixels (fractional stride) yields a larger feature map before convolution.

Conceptually, one can imagine upsampling by first "expanding" the input: we insert $s - 1$ zeros between each pair of original pixels (both horizontally and vertically), producing a larger intermediate grid, and then apply a standard convolution with stride 1. For example, inserting zeros around a 3×3 input and convolving with a 3×3 kernel can produce a 5×5 output (Figure 82).

Rather than explicitly inserting zeros, a transposed convolution implements the same computation by *moving the kernel over the output grid*. With stride s, the kernel is shifted by s positions on the output for each move of one position on the input. Each input pixel "casts" a weighted patch (given by the kernel) onto the output grid, and overlapping patches are summed. This reconstructs exactly the same enlarged feature map as the zero-insertion view, but in a single learnable layer.

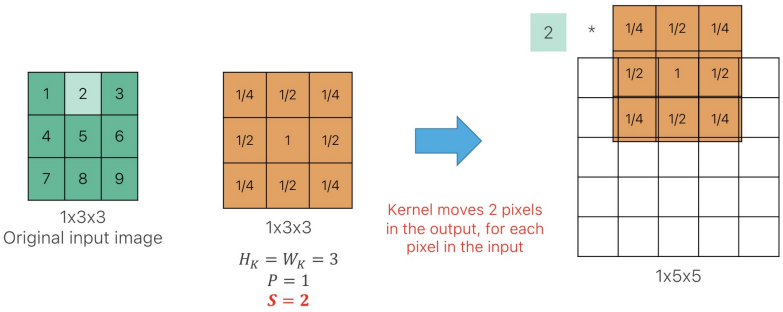


Figure 83: Each input pixel is multiplied by the kernel and "sprayed" onto the output grid with stride 2; overlaps sum to form the final activation.

By learning the kernel weights, transposed convolutions perform upsampling in a way that can adapt to the data. They are widely used in generative and segmentation architectures to increase spatial resolution. While they can introduce checkerboard artifacts in some settings, these artifacts are usually not problematic in well-designed segmentation networks.

3.5.7 U-net

U-net was one of the first networks used for segmentation. They used transposed convolutions to do upsampling, and skip connections which combine features from the encoder to features from the decoder. Skip connections use concatenation instead of summation. The encoder compresses in a semantically rich representation the input, and the decoder takes the latent representation and goes back to either the original input or a segmentation map.

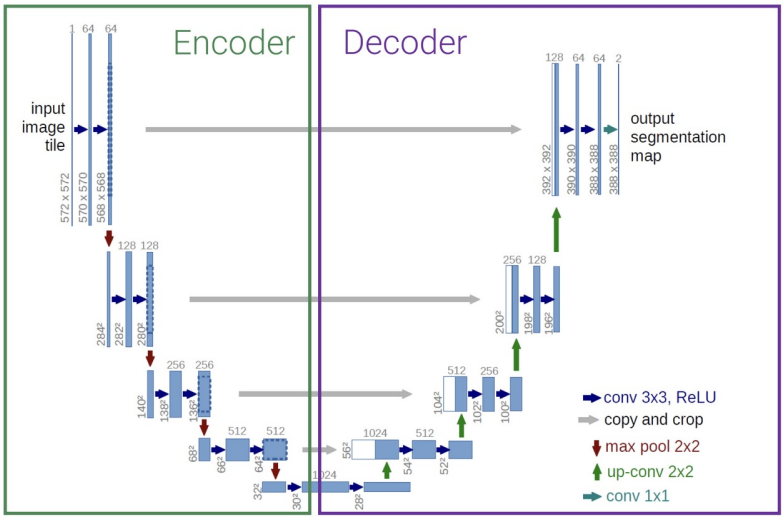


Figure 84: The U-net network architecture.

The operations are mainly three:

- convolutions between features of the same dimensions.
- downsampling with pooling.
- upsampling with transposed convolutions.

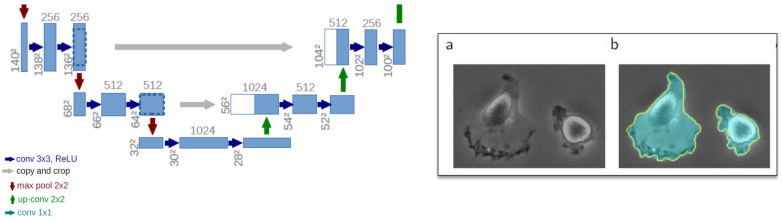


Figure 85: A zoom of the U-net middle layers

3.5.8 Dilated convolutions

This semantically rich feature map at a certain stage comes from the fact that a position in a feature map collects a lot of spatial information in the channel through the concept of receptive field. The problem of previous stages being less semantically rich comes from their receptive field, because they consider less information. The idea of dilated convolutions is to try to address the issue at a different layer enlarging the receptive field without touching the computation. The idea was introduced with the concept of dilated convolutions. Dilated convolutions consider a large receptive field by taking positions far away from the center and putting zeros in between.

We recall that Resnet in each stage has combination of blocks that perform convolutions with batch normalization. The first layer of each stage of Resnet lowers the resolution with striding and doubles the channel. Instead of using striding they kept the same striding but changed the dilation rate. Within the same stage now the blocks don't change resolution (same striding). If we don't halve the resolution we are doing a lot more computation with lots of more channels. Also, the output feature map is bigger because we have no striding. But this is good, since we are doing segmentation and need more semantic information.

They applied this at different stages of the Resnet backbone, to increase the receptive field. This uses much more memory but has the same computational cost.

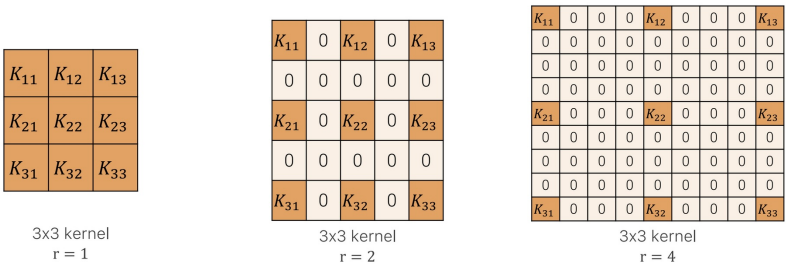


Figure 86: A simple 3×3 kernel has a dilation factor of 1. A dilation rate of 2 means that i introduced a layer of zeros in the feature map.

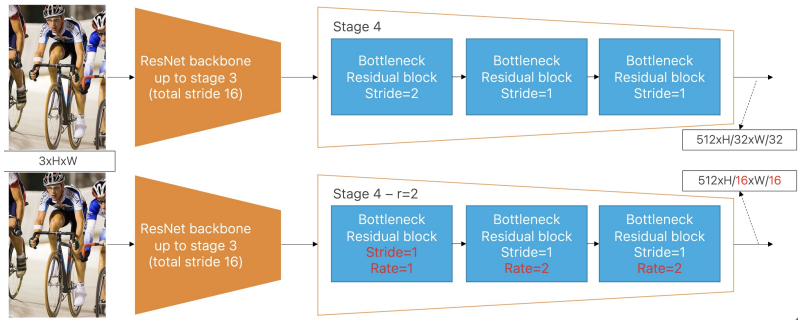


Figure 87: Normal Resnet vs the one with dilated convolutions

If we modify all the stages with dilated convolutions this becomes too costly since we aren't modifying the input size. While the operation we perform is constant the size of the channel still doubles.

3.5.9 DeepLab

ResNet with dilated convolutions are the backbones used in DeepLab, the reference model for semantic segmentation, to control the resolution of the output feature map.

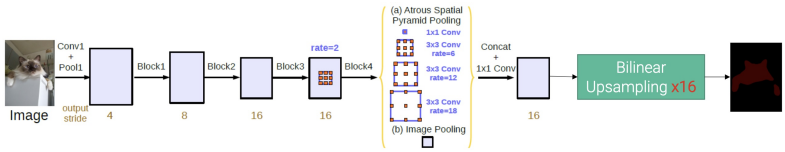


Figure 88: Deeplab v3 network architecture

To handle the problem of the existence of objects at multiple scales, DeepLab introduces a block of operations known as spatial pyramid pooling based on dilated convolutions.

This combination of operations took inspiration from a concept known as spatial pyramid pooling from DeepLab v2. Spatial Pyramid Pooling can be seen as an extension of global average pooling that preserves spatial information. DeepLab emulates the property of the Spatial Pyramid Pooling layer to capture spatial context at multiple scales by leveraging again atrous convolutions. In DeepLab v2, the module performs 3×3 convolutions with increasing dilation rates. Outputs are concatenated and aggregated by a scoring 1×1 convoltuion to produce the output scores map (we can do this because we are not changing the resolution).

When the dilation rate grows, the number of positions where all the 9 weights of the kernel are used shrinks. It was discovered that a dilation rate of 24 was too big, since only one weight was used, so it was removed.

They also added a context feature in DeepLab v3, so they took all the whole feature map and applied global average pooling.

3.5.10 Instance segmentation

Semantic segmentation separates different classes at the pixel level, but doesn't separate different instances of the same class. Object detection separates instances but provides only a crude ap-

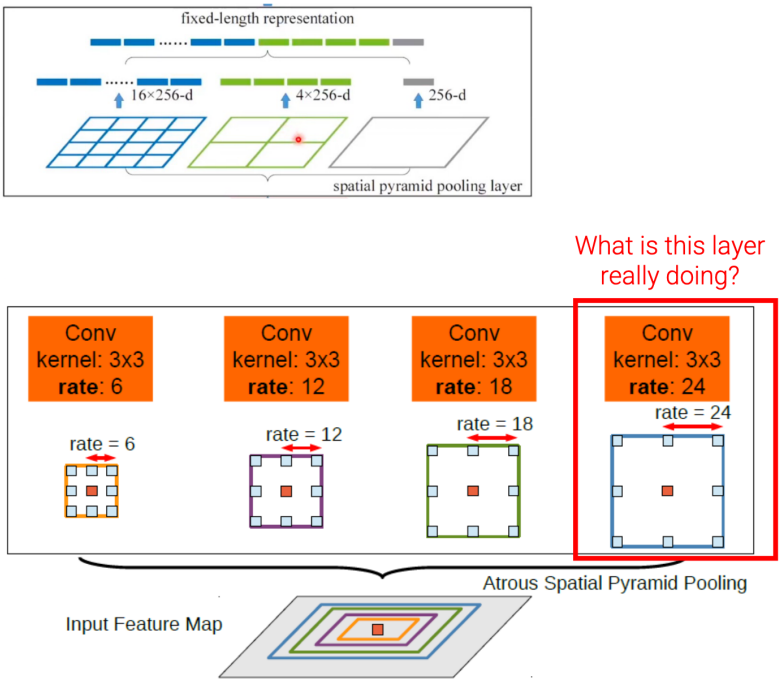


Figure 89: Deeplab v2 network architecture

proximation of the instance shape (the bounding box). The task of instance segmentation lays at the intersection of the two. It can be defined as the task of:

1. Detecting all instances of the objects of interest in an image and classifying them
2. Segmenting them from the background at the pixel level.

Figure 90: difference between object detection, instance segmentation and semantic segmentation.

3.5.11 Mask R-CNN

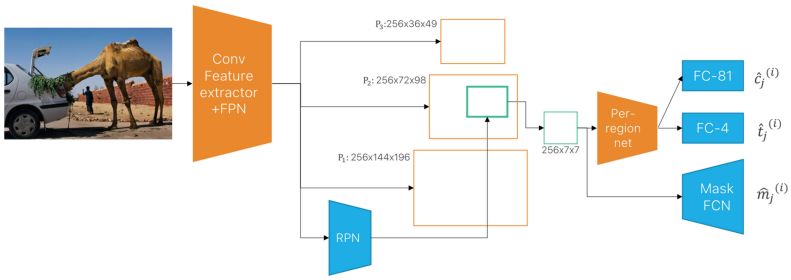


Figure 91: Mask R-CNN network architecture

Mask R-CNN modifies Faster R-CNN, which is a 2 stage detector, by adding a convolutional network called Mask FCN which does segmentation on the proposed region. Another change is the RoI align instead of the RoI pool. Remember that feature maps are spatially continuous, but RoIs have floating-point coordinates after scaling, and we need to align them to the grid. The

RoI align doesn't snap the RoI to the grid, but it divides into equally sized subregions, it samples feature values at a regular grid of points within each RoI cell with bilinear interpolation, and it pools samples feature values in each sub-region.

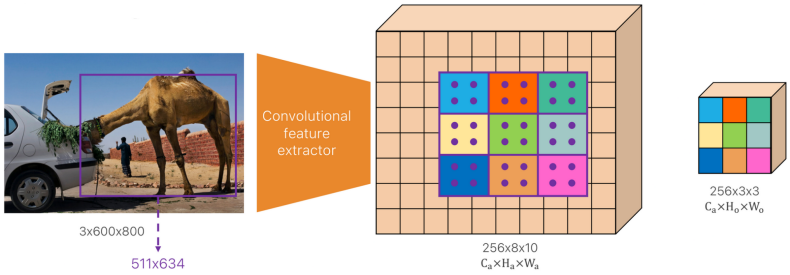


Figure 92: RoI align

3.6 Metric Learning

By removing the classification head of an image classification network we can compute a low-dimesnional representation (a low dimensional embedding) of the input images. Performing nearest neighbor search on such embedding vectors is very effective: we get semantically similar images as neighbors.

3.6.1 Face recognition

Given a query face face recognition is a one to many matching problem. A database of faces can have millions of identities, but few images per subject. A match msust be robust to facial expression, changed hair style, aging, glasses, scarfs...

We can't just train a classification network for face images. If we take imagenet as an example, which has 1000 classes and 1.4 M images, we would need 1.4 B images to train if we want to distinguish between 1 M different faces. Face recognition is also an open-world problem, since we could want to add or remove an user. But this requires to throw away the last layer (since we change the output dimension by adding/removing people) and re-train the full network.

The idea is to use transfer learning, which is based on reusing the representation learned once on a reasonably large dataset as feature extractor with a $k-$NN classifier on the embedding to tackle face recognition.

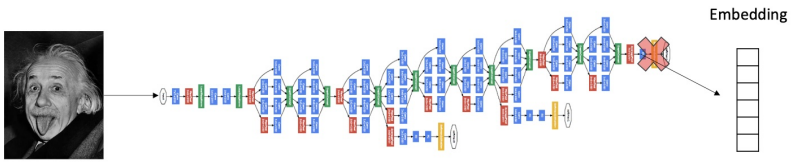


Figure 93: To change the number of images we want to recognize we also have to change the embedding size

Classification embeddings The cross-entropy loss used in classification guides the network to learn high-level and semantically rich embeddings. However, they are an intermediate representation used to classify correctly the input image with the subsequent fully connected linear layer.

In the embedding space the classes are linearly separable. Distances between elements of the same class can be arbitrarily large, and distances between elements of different classes can be arbitrarily small. This is not a great space in which perform nearest neighbor search, since a point can be closer to lots of points of the other classes and not to the ones of the same class.

3.6.2 Face verification

A closely related problem to face recognition is: given two images, confirm that they depict the same person. This is usually solved by learning a similarity function between images and a threshold.

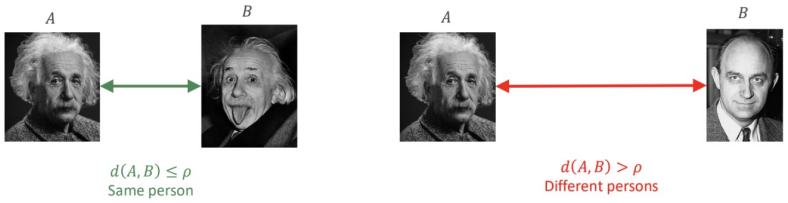


Figure 94: The distance between the images must be under the threshold

3.6.3 Metric Learning

The idea of metric learning, or similarity learning, is to use a specific loss at training time to guide the feature extractor to favor a clustered structure of the embedding such that:

1. The distance between faces of the same person (the intra-class distance) is minimized.
2. The distance between faces of different persons (the inter-class distance) is maximized.

We have to learn discriminative embeddings:

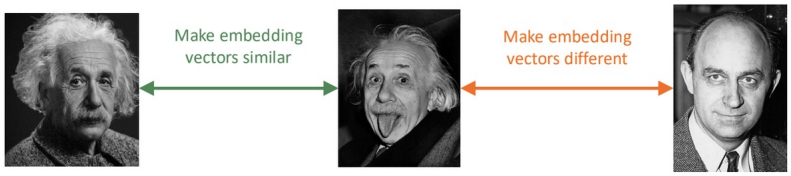


Figure 95: The key concept of metric learning

3.6.4 Siamese network training

A siamese network learns to compare two inputs. It consists in two identical subnetworks (with the same architecture and shared weights) that process two different inputs and then compare their outputs to determine how similar or different the inputs are.

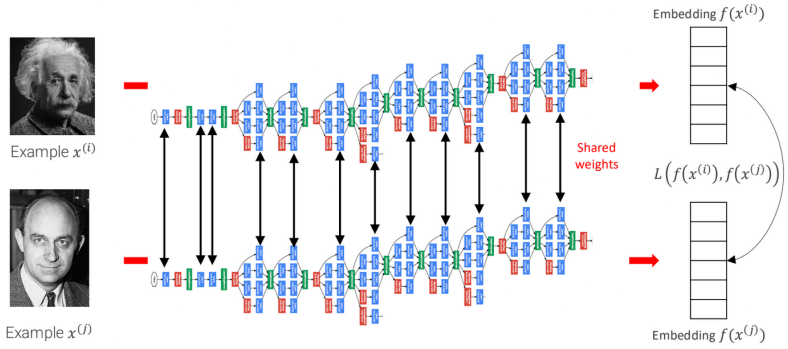


Figure 96: Siamese network architecture

The goal is to compute a loss that depends on two input examples. The embeddings are compared using a loss function $L(f(x^{(i)}), f(x^{(j)}))$.

After producing this embedding space, we can pass a new test image through the network, and then apply a $k-$NN classifier on the embedding space. By doing this, adding new identities is easy, since the embedding space doesn't have to be changed.

Contrastive loss Treating face verification as classification may not generalize well in open-world problems. A loss enforcing directly a clustered embedding structure should:

- make $d(x^{(i)}, x^{(j)})$ small, if $x^{(i)}$ and $x^{(j)}$ depict the same person.
- make $d(x^{(i)}, x^{(j)})$ large, if $x^{(i)}$ and $x^{(j)}$ depict two different persons.

If the Euclidean norm is used $d(x^{(i)}, x^{(j)}) = \|f(x^{(i)}) - f(x^{(j)})\|_2$ a simple loss that capture both requirements is:

$$L\left(f(x^{(i)}), f(x^{(j)})\right) = \begin{cases} \|f(x^{(i)}) - f(x^{(j)})\|_2^2 & \text{if } y^{(i,j)} = +1 \\ -\|f(x^{(i)}) - f(x^{(j)})\|_2^2 & \text{if } y^{(i,j)} = 0 \end{cases}$$

We can note that the first term is bounded from below at 0, but the second one is unbounded (the clusters can be infinitely different). There is practically no gain in pushing clusters further away when they are already far "enough".

A formulation less prone to overfitting uses a margin m to stop pushing clusters of different classes apart from each other, i.e.

$$L\left(f(x^{(i)}), f(x^{(j)})\right) = \begin{cases} \|f(x^{(i)}) - f(x^{(j)})\|_2^2 & \text{if } y^{(i,j)} = +1 \\ \max(0, m - \|f(x^{(i)}) - f(x^{(j)})\|_2^2) & \text{if } y^{(i,j)} = 0 \end{cases}$$

Triplet loss To obtain an effective embedding, we must make embedding vectors similar for the same person, and different for different persons. And we must satisfy both conditions at once.

The contrastive loss achieves this effect indirectly, by reasoning on every pair in the triplet of images. The triplet loss, instead, optimizes the embedding to better fulfil both requirements at once.

$$\|f(P) - f(A)\|_2^2 < \|f(N) - f(A)\|_2^2$$

But the triplet loss isn't robust, since it doesn't guarantee that inter-class distances are large. It also risks training collapse: if the embedding is a constant value (e.g. all 0s) the criteria is satisfied.

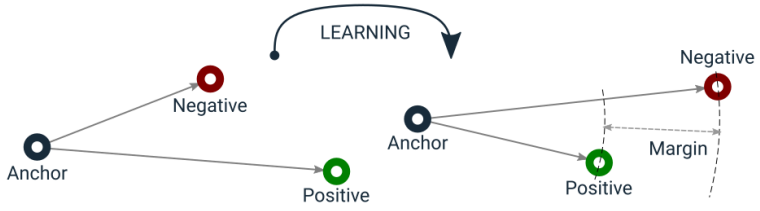


Figure 97: Triplet loss

As we can see, the distance between the green and the red dot is smaller than the distance between the green and the blue one. But the blue and the green one should be closer. With a margin, triplet loss learns to rank.

$$\|f(P) - f(A)\|_2^2 + m < \|f(N) - f(A)\|_2^2$$

It doesn't impose all the pictures of the same person to collapse in a point, hence it may be easier to learn than contrastive loss.

The most important part in training embeddings with the triplet loss it to effectively form the triplets, because the choice of negative examples is key. For most triplets, the constraint $\|f(P) - f(A)\|_2^2 + m < \|f(N) - f(A)\|_2^2$ will be already satisfied. We want to compute the loss only on triplets which will contribute to learning.

Hence, a large mini-batch of B samples is formed by picking a fixed number of images for D identities, to ensure that a significant representation of the anchor-positive distance can be formed, and randomly sampled negatives are added to complete the batch. We have:

- Hard triplets: triplets where the negative is closer to the anchor than the positive
- Semi hard triplets: triplets where the negative is not closer to the anchor than the positive, but which still have positive loss.

Semi-hard negatives are the best to train on. To do semi-hard negative mining, at the beginning of each epoch:

- Compute all the embeddings on the training set (using the network trained so far).
- Create all the possible (anchors, positive) pairs for each identity.
- Create a triplet for each semi-hard negative N if $\|f(P) - f(A)\|_2^2 < \|f(N) - f(A)\|_2^2 < \|f(P) - f(A)\|_2^2 + m$ (a negative example that is within the margin).

We don't select the hardest negatives (the ones where $\|f(P) - f(A)\|_2^2 > \|f(N) - f(A)\|_2^2$) because it was found to lead to poor training, probably because then training is domintaed by mislabeled or poorly imaged faces.