Image Processing and Computer Vision (Module 2)

Last update: 16 June 2024

Academic Year 2023 – 2024
Alma Mater Studiorum · University of Bologna

Contents

1 Camera calibration

World reference frame (WRF) Coordinate system (X_W, Y_W, Z_W) of the real world relative to a reference point (e.g. a corner). World reference frame (WRF)

Camera reference frame (CRF) Coordinate system (X_C, Y_C, Z_C) that characterizes a camera. Camera reference frame (CRF)

Image reference frame (IRF) Coordinate system (U, V) of the image. They are obtained as a perspective projection of CRF coordinates as: Image reference frame (IRF)

$$u = \frac{f}{z} x_C \qquad v = \frac{f}{z} y_C$$

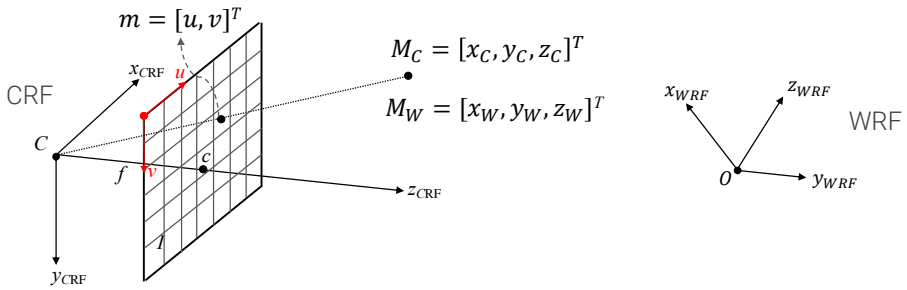


Figure 1.1: Example of WRF, CRF and IRF

1.1 Forward imaging model

1.1.1 Image pixelization (CRF to IRF)

The conversion from the camera reference frame to the image reference frame is done in two steps: Image pixelization

Discretization Given the sizes (in mm) Δu and Δv of the pixels, it is sufficient to modify the perspective projection to map CRF coordinates into a discrete grid: Discretization

$$u = \frac{1}{\Delta u} \frac{f}{z_C} x_C \qquad v = \frac{1}{\Delta v} \frac{f}{z_C} y_C$$

Origin translation To avoid negative pixels, the origin of the image has to be translated from the piercing point c to the top-left corner. This is done by adding an offset (u_0, v_0) to the projection (in the new system, $c = (u_0, v_0)$): Origin translation

$$u = \frac{1}{\Delta u} \frac{f}{z_C} x_C + u_0 \qquad v = \frac{1}{\Delta v} \frac{f}{z_C} y_C + v_0$$

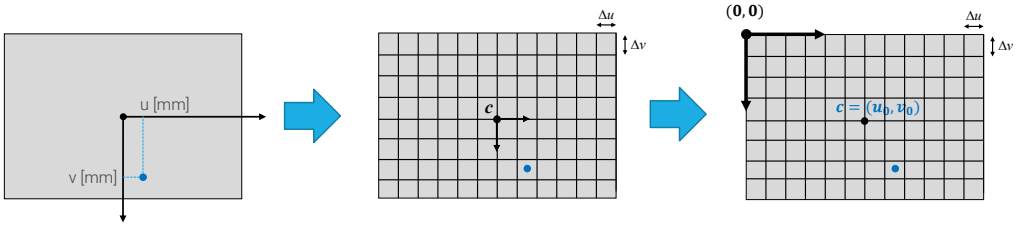


Figure 1.2: Pixelization process

Intrinsic parameters By fixing $f_u = \frac{f}{\Delta u}$ and $f_v = \frac{f}{\Delta v}$, the projection can be rewritten as: Intrinsic parameters

$$u = f_u \frac{x_C}{z_C} + u_0 \qquad v = f_v \frac{y_C}{z_C} + v_0$$

Therefore, there is a total of 4 parameters: f_u, f_v, u_0 and v_0.

Remark. A more general model includes a further parameter (skew) to account for non-orthogonality between the axes of the image sensor such as:

- Misplacement of the sensor so that it is not perpendicular to the optical axis.
- Manufacturing issues.

Nevertheless, in practice skew is always 0.

1.1.2 Roto-translation (WRF to CRF)

The conversion from the world reference system to the camera reference system is done through a roto-translation w.r.t. the optical center. Roto-translation
Given:

- A WRF point $\mathbf{M}_W = (x_W, y_W, z_W)$,
- A rotation matrix $\boldsymbol{R}$,
- A translation vector $\mathbf{t}$,

the coordinates $\mathbf{M}_C$ in CRF corresponding to $\mathbf{M}_W$ are given by:

$$\mathbf{M}_C = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} = \boldsymbol{R}\mathbf{M}_W + \mathbf{t} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

Remark. The coordinates $\mathbf{C}_W$ of the optical center $\mathbf{C} = \bar{\mathbf{0}}$ are obtained as:

$$\bar{\mathbf{0}} = \boldsymbol{R}\mathbf{C}_W + \mathbf{t} \iff (\bar{\mathbf{0}} - \mathbf{t}) = \boldsymbol{R}\mathbf{C}_W \iff \mathbf{C}_W = \boldsymbol{R}^T(\bar{\mathbf{0}} - \mathbf{t}) \iff \mathbf{C}_W = -\boldsymbol{R}^T\mathbf{t}$$

Extrinsic parameters Extrinsic parameters

- The rotation matrix $\boldsymbol{R}$ has 9 elements of which 3 are independent (i.e. the rotation angles around the axes).
- The translation matrix $\mathbf{t}$ has 3 elements.

Therefore, there is a total of 6 parameters.

Remark. It is not possible to combine the intrinsic camera model and the extrinsic roto-translation to create a linear model for the forward imaging model.

$$u = f_u \frac{r_{1,1}x_W + r_{1,2}y_W + r_{1,3}z_W + t_1}{r_{3,1}x_W + r_{3,2}y_W + r_{3,3}z_W + t_3} + u_0 \qquad v = f_v \frac{r_{2,1}x_W + r_{2,2}y_W + r_{2,3}z_W + t_2}{r_{3,1}x_W + r_{3,2}y_W + r_{3,3}z_W + t_3} + v_0$$

1.2 Projective space

Remark. In the 2D Euclidean plane $\mathbb{R}^2$, parallel lines never intersect and points at infinity cannot be represented.

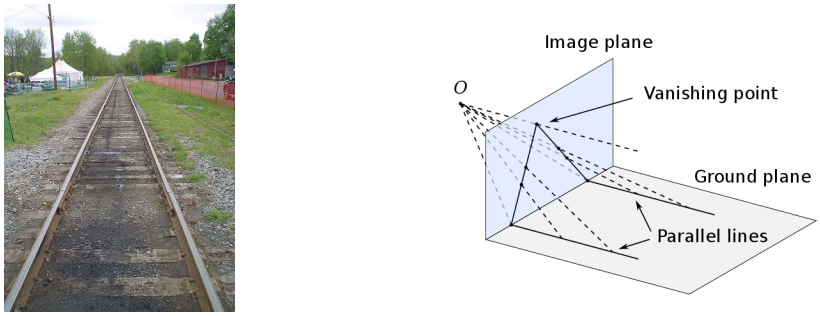


Figure 1.3: Example of point at infinity

Remark. Point at infinity is a point in space while the vanishing point is in the image plane.

Homogeneous coordinates Without loss of generality, consider the 2D Euclidean space $\mathbb{R}^2$. Homogeneous coordinates

Given a coordinate (u, v) in Euclidean space, its homogeneous coordinates have an additional dimension such that:

$$(u, v) \equiv (ku, kv, k) \, \forall k \neq 0$$

In other words, a 2D Euclidean point is represented by an equivalence class of 3D points.

Projective space Space $\mathbb{P}^n$ associated with the homogeneous coordinates of an Euclidean space $\mathbb{R}^n$. Projective space

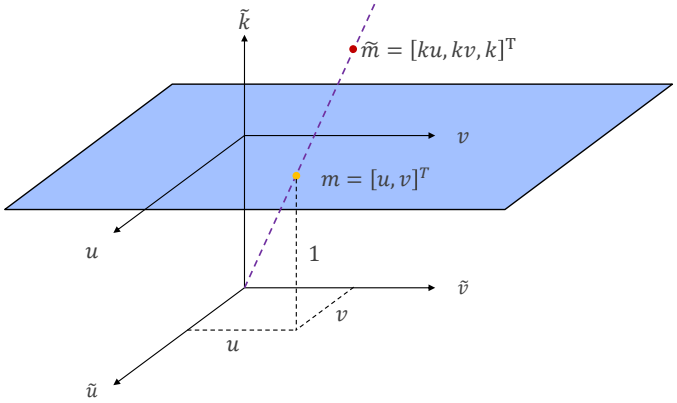


Figure 1.4: Example of projective space $\mathbb{P}^2$

Remark. $\bar{\mathbf{0}}$ is not a valid point in $\mathbb{P}^n$.

Remark. A projective space allows to homogeneously handle both ordinary (image) and ideal (scene) points without introducing additional complexity.

Point at infinity Given the parametric equation of a 2D line defined as: Point at infinity

$$\mathbf{m} = \mathbf{m}_0 + \lambda \mathbf{d} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \lambda \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} u_0 + \lambda a \\ v_0 + \lambda b \end{bmatrix}$$

It is possible to define a generic point in the projective space along the line m as:

$$\tilde{\mathbf{m}} \equiv \begin{bmatrix} \mathbf{m} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} u_0 + \lambda a \\ v_0 + \lambda b \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \frac{u_0}{\lambda} + a \\ \frac{v_0}{\lambda} + b \\ \frac{1}{\lambda} \end{bmatrix}$$

The projective coordinates $\tilde{\mathbf{m}}_\infty$ of the point at infinity of a line m is given by:

$$\tilde{\mathbf{m}}_\infty = \lim_{\lambda \to \infty} \tilde{\mathbf{m}} \equiv \begin{bmatrix} a \\ b \\ 0 \end{bmatrix}$$

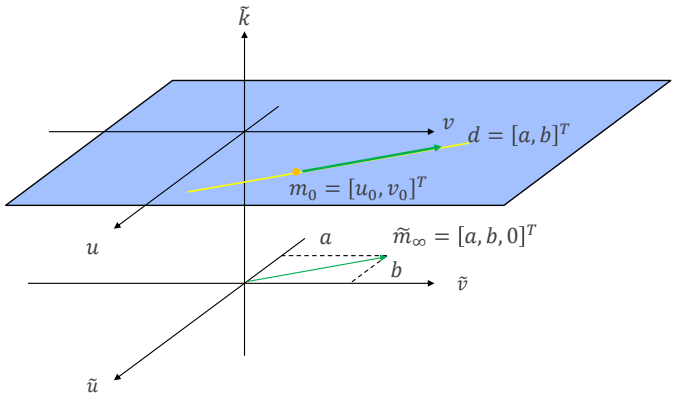


Figure 1.5: Example of infinity point in $\mathbb{P}^2$

In 3D, the definition is trivially extended as:

$$\tilde{\mathbf{M}}_\infty = \lim_{\lambda \to \infty} \begin{bmatrix} \frac{x_0}{\lambda} + a \\ \frac{y_0}{\lambda} + b \\ \frac{z_0}{\lambda} + c \\ \frac{1}{\lambda} \end{bmatrix} \equiv \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix}$$

Perspective projection Given a point $\mathbf{M}_C = (x_C, y_C, z_C)$ in the CRF and its corresponding point $\mathbf{m} = (u, v)$ in the image, the non-linear perspective projection in Euclidean space can be done linearly in the projective space as: Perspective projection in projective space

$$\tilde{\mathbf{m}} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_u \frac{x_C}{z_C} + u_0 \\ f_v \frac{y_C}{z_C} + v_0 \\ 1 \end{bmatrix} \equiv z_C \begin{bmatrix} f_u \frac{x_C}{z_C} + u_0 \\ f_v \frac{y_C}{z_C} + v_0 \\ 1 \end{bmatrix}$$

$$\equiv \begin{bmatrix} f_u x_C + z_C u_0 \\ f_v y_C + z_C v_0 \\ z_C \end{bmatrix} \equiv \begin{bmatrix} f_u & 0 & u_0 & 0 \\ 0 & f_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \\ 1 \end{bmatrix} \equiv \boldsymbol{P}_{\text{int}} \tilde{\mathbf{M}}_C$$

Remark. The equation can be written to take account of the arbitrary scale factor k as:

$$k\tilde{\mathbf{m}} = \boldsymbol{P}_{\text{int}}\tilde{\mathbf{M}}_C$$

or, if k is omitted, as:

$$\tilde{\mathbf{m}} \approx \boldsymbol{P}_{\text{int}}\tilde{\mathbf{M}}_C$$

Remark. In projective space, we can also project in Euclidean space the point at infinity of parallel 3D lines in CRF with direction (a, b, c):

$$\tilde{\mathbf{m}}_\infty \equiv \boldsymbol{P}_{\text{int}} \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix} \equiv \begin{bmatrix} f_u & 0 & u_0 & 0 \\ 0 & f_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix} \equiv \begin{bmatrix} f_u a + c u_0 \\ f_v b + c v_0 \\ c \end{bmatrix} \equiv c \begin{bmatrix} f_u \frac{a}{c} + u_0 \\ f_v \frac{b}{c} + v_0 \\ 1 \end{bmatrix}$$

Therefore, the Euclidean coordinates are:

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

Note that this is not possible when $c = 0$ (i.e. the line is parallel to the image plane).

Intrinsic parameter matrix The intrinsic transformation can be expressed through a matrix:

Intrinsic parameter matrix

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

$\boldsymbol{A}$ is always upper right triangular and models the characteristics of the imaging device.

Remark. If skew is considered, it would be at position $(1, 2)$.

Extrinsic parameter matrix The extrinsic transformation can be expressed through a matrix:

Extrinsic parameter matrix

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \bar{\mathbf{0}} & 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective projection matrix (PPM) As the following hold:

Perspective projection matrix

$$\boldsymbol{P}_{\text{int}} = [\boldsymbol{A}|\bar{\mathbf{0}}] \qquad \tilde{\mathbf{M}}_C \equiv \boldsymbol{G}\tilde{\mathbf{M}}_W$$

The perspective projection can be represented in matrix form as:

$$\tilde{\mathbf{m}} \equiv \boldsymbol{P}_{\text{int}}\tilde{\mathbf{M}}_C \equiv \boldsymbol{P}_{\text{int}}\boldsymbol{G}\tilde{\mathbf{M}}_W \equiv \boldsymbol{P}\tilde{\mathbf{M}}_W$$

where $\boldsymbol{P} = \boldsymbol{P}_{\text{int}}\boldsymbol{G}$ is the perspective projection matrix. It is full-rank and has shape 3×4.

Remark. Every full-rank 3×4 matrix is a PPM.

Canonical perspective projection PPM of form:

Canonical perspective projection

$$\boldsymbol{P} \equiv [\boldsymbol{I}|\bar{\mathbf{0}}]$$

It is useful to represent the core operations carried out by a perspective projection as any general PPM can be factorized as:

$$\boldsymbol{P} \equiv \boldsymbol{A}[\boldsymbol{I}|\bar{\mathbf{0}}]\boldsymbol{G}$$

where:

- $\boldsymbol{G}$ converts from WRT to CRF.
- $[\boldsymbol{I}|\bar{\mathbf{0}}]$ performs the canonical perspective projection (i.e. divide by the third coordinate).
- $\boldsymbol{A}$ applies camera specific transformations.

A further factorization is:

$$\boldsymbol{P} \equiv \boldsymbol{A}[\boldsymbol{I}|\bar{\mathbf{0}}]\boldsymbol{G} \equiv \boldsymbol{A}[\boldsymbol{I}|\bar{\mathbf{0}}] \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \bar{\mathbf{0}} & 1 \end{bmatrix} \equiv \boldsymbol{A}[\boldsymbol{R}|\mathbf{t}]$$

1.3 Lens distortion

The PPM is based on the pinhole model and is unable to capture distortions that a lens introduces.

Radial distortion Deviation from the ideal pinhole caused by the lens curvature. Radial distortion

Barrel distortion Defect associated with wide-angle lenses that causes straight lines to bend outwards. Barrel distortion

Pincushion distortion Defect associated with telephoto lenses that causes straight lines to bend inwards. Pincushion distortion

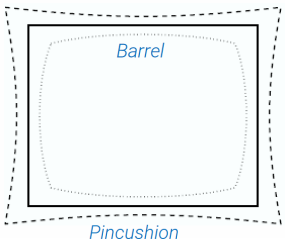


Figure 1.6: Example of distortions w.r.t. a perfect rectangle

Tangential distortion Second-order effects caused by misalignment or defects of the lens (i.e. capture distortions that are not considered in radial distortion).

1.3.1 Modeling lens distortion

Lens distortion can be modeled using a non-linear transformation that maps ideal (undistorted) image coordinates $(x_{\text{undist}}, y_{\text{undist}})$ into the observed (distorted) coordinates (x, y): Modeling lens distortion

$$\begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{L(r) \begin{bmatrix} x_{\text{undist}} \\ y_{\text{undist}} \end{bmatrix}}_{\text{Radial distortion}} + \underbrace{\begin{bmatrix} dx(x_{\text{undist}}, y_{\text{undist}}, r) \\ dy(x_{\text{undist}}, y_{\text{undist}}, r) \end{bmatrix}}_{\text{Tangential distortion}}$$

where:

- r is the distance from the distortion center which is usually assumed to be the piercing point $c = (0, 0)$. Therefore, $r = \sqrt{(x_{\text{undist}})^2 + (y_{\text{undist}})^2}$.
- $L(r)$ is the radial distortion function which is a linear operator defined for positive r only and is approximated using the Taylor series:

$$L(0) = 1 \qquad L(r) = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots$$

  where k_i are additional intrinsic parameters.
- The tangential distortion is approximated as:

$$\begin{bmatrix} dx(x_{\text{undist}}, y_{\text{undist}}, r) \\ dy(x_{\text{undist}}, y_{\text{undist}}, r) \end{bmatrix} = \begin{bmatrix} 2p_1 x_{\text{undist}} y_{\text{undist}} + p_2(r^2 + 2(x_{\text{undist}})^2) \\ 2p_1 y_{\text{undist}} x_{\text{undist}} + p_2(r^2 + 2(y_{\text{undist}})^2) \end{bmatrix}$$

  where p_1 and p_2 are additional intrinsic parameters.

Remark. This approximation has been empirically shown to work.

Remark. The additivity of the two distortions in an assumption. Other models might add arbitrary complexity.

1.3.2 Image formation with lens distortion

Image formation with lens distortion

Lens distortion is applied after the canonical perspective projection. Therefore, the complete workflow for image formation becomes the following:

1. Transform points from WRF to CRF:

$$\boldsymbol{G}\tilde{\mathbf{M}}_W \equiv \begin{bmatrix} x_C & y_C & z_C & 1 \end{bmatrix}^T$$

2. Apply the canonical perspective projection:

$$\begin{bmatrix} \frac{x_C}{z_C} & \frac{y_C}{z_C} \end{bmatrix}^T = \begin{bmatrix} x_{\text{undist}} & y_{\text{undist}} \end{bmatrix}^T$$

3. Apply the lens distortion non-linear mapping:

$$L(r) \begin{bmatrix} x_{\text{undist}} \\ y_{\text{undist}} \end{bmatrix} + \begin{bmatrix} dx(x_{\text{undist}}, y_{\text{undist}}, r) \\ dy(x_{\text{undist}}, y_{\text{undist}}, r) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

4. Transform points from CRF to IRF:

$$\boldsymbol{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \mapsto \begin{bmatrix} u \\ v \end{bmatrix}$$

1.4 Zhang's method

Calibration patterns There are two approaches to camera calibration: Calibration patterns

- Use a single image of a 3D calibration object (i.e. image with at least 2 planes with a known pattern).
- Use multiple (at least 3) images of the same planar pattern (e.g. a chessboard).

Remark. In practice, it is easier to get multiple images of the same pattern.

Algebraic error Error minimized to estimate an initial guess for a subsequent refinement step. It should be cheap to compute. Algebraic error

Geometric error Error minimized to match the actual geometrical location of a problem. Geometric error

Zhang's method Algorithm to determine the intrinsic and extrinsic parameters of a camera setup given multiple images of a pattern. Zhang's method

Image acquisition Acquire n images of a planar pattern with c internal corners.

Consider a chessboard for which we have prior knowledge of:

- The number of internal corners,
- The size of the squares.

> **Remark.** To avoid ambiguity, the number of internal corners should be odd along one axis and even along the other (otherwise, a 180° rotation of the board would be indistinguishable).

The WRF can be defined such that:

- The origin is always at the same corner of the chessboard.
- The z-axis is at the same level of the pattern so that $z = 0$ when referring to points of the chessboard.
- The x and y axes are aligned to the grid of the chessboard. x is aligned along the short axis and y to the long axis.

> **Remark.** As each image has its own extrinsic parameters, during the execution of the algorithm, for each image i will be computed an estimate of its own extrinsic parameters $\boldsymbol{R}_i$ and $\mathbf{t}_i$.

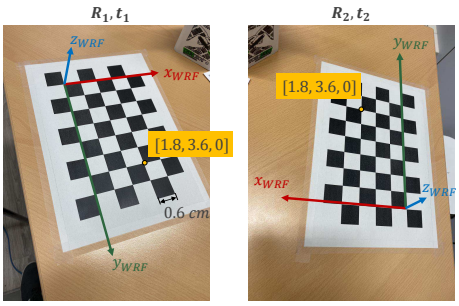


Figure 1.7: Example of two acquired images

Initial homographies guess For each image i, compute an initial guess of its homography $\boldsymbol{H}_i$. Due to the choice of the z-axis position, the perspective projection matrix and the WRF points can be simplified:

$$k\tilde{\mathbf{m}} = k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P}\tilde{\mathbf{M}}_W = \begin{bmatrix} p_{1,1} & p_{1,2} & \cancel{p_{1,3}} & p_{1,4} \\ p_{2,1} & p_{2,2} & \cancel{p_{2,3}} & p_{2,4} \\ p_{3,1} & p_{3,2} & \cancel{p_{3,3}} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ \emptyset \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{H}\tilde{\mathbf{w}}$$

where $\boldsymbol{H}$ is a homography and represents a general transformation between projective planes.

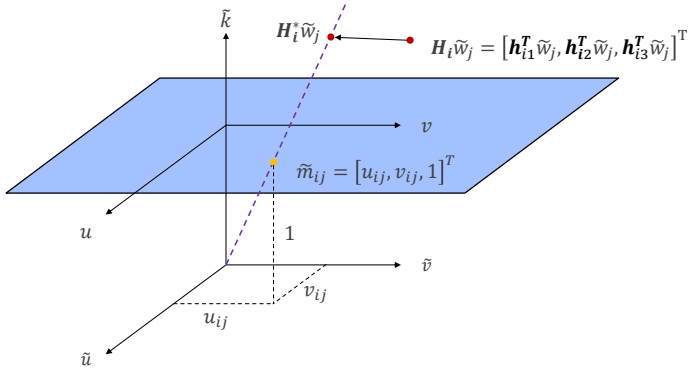
DLT algorithm Consider the i-th image with its c corners. For each corner j, we have prior knowledge of:

- Its 3D coordinates in the WRF.
- Its 2D coordinates in the IRF.

Then, for each corner j, we can define 3 linear equations where the homography $\boldsymbol{H}_i$ of the i-th image is the unknown:

$$\tilde{\mathbf{m}}_{i,j} \equiv \begin{bmatrix} u_{i,j} \\ v_{i,j} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{i,1,1} & p_{i,1,2} & p_{i,1,4} \\ p_{i,2,1} & p_{i,2,2} & p_{i,2,4} \\ p_{i,3,1} & p_{i,3,2} & p_{i,3,4} \end{bmatrix} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} \equiv \boldsymbol{H}_i \tilde{\mathbf{w}}_j \equiv \underbrace{\begin{bmatrix} \mathbf{h}_{i,1}^T \\ \mathbf{h}_{i,2}^T \\ \mathbf{h}_{i,3}^T \end{bmatrix}}_{\mathbb{R}^{3\times 3}} \tilde{\mathbf{w}}_j \equiv \underbrace{\begin{bmatrix} \mathbf{h}_{i,1}^T \tilde{\mathbf{w}}_j \\ \mathbf{h}_{i,2}^T \tilde{\mathbf{w}}_j \\ \mathbf{h}_{i,3}^T \tilde{\mathbf{w}}_j \end{bmatrix}}_{\mathbb{R}^{3\times 1}}$$

Geometrically, we can interpret $\boldsymbol{H}_i \tilde{\mathbf{w}}_j$ as a point in $\mathbb{P}^2$ that we want to align to the projection of $(u_{i,j}, v_{i,j})$ by tweaking $\boldsymbol{H}_i$ (i.e. find $\boldsymbol{H}_i^*$ such that $\boldsymbol{H}_i^* \tilde{\mathbf{w}}_j \equiv k \begin{bmatrix} u_{i,j} & v_{i,j} & 1 \end{bmatrix}^T$).



It can be shown that two vectors have the same direction if their cross product is $\bar{\mathbf{0}}$:

$$\tilde{\mathbf{m}}_{i,j} \equiv \boldsymbol{H}_i \tilde{\mathbf{w}}_j \iff \tilde{\mathbf{m}}_{i,j} \times \boldsymbol{H}_i \tilde{\mathbf{w}}_j = \bar{\mathbf{0}} \iff \begin{bmatrix} u_{i,j} \\ v_{i,j} \\ 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{h}_{i,1}^T \tilde{\mathbf{w}}_j \\ \mathbf{h}_{i,2}^T \tilde{\mathbf{w}}_j \\ \mathbf{h}_{i,3}^T \tilde{\mathbf{w}}_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\iff \begin{bmatrix} v_{i,j} \mathbf{h}_{i,3}^T \tilde{\mathbf{w}}_j - \mathbf{h}_{i,2}^T \tilde{\mathbf{w}}_j \\ \mathbf{h}_{i,1}^T \tilde{\mathbf{w}}_j - u_{i,j} \mathbf{h}_{i,3}^T \tilde{\mathbf{w}}_j \\ u_{i,j} \mathbf{h}_{i,2}^T \tilde{\mathbf{w}}_j - v_{i,j} \mathbf{h}_{i,1}^T \tilde{\mathbf{w}}_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\iff \underbrace{\begin{bmatrix} \bar{\mathbf{0}}_{1\times 3} & -\tilde{\mathbf{w}}_j^T & v_{i,j} \tilde{\mathbf{w}}_j^T \\ \tilde{\mathbf{w}}_j^T & \bar{\mathbf{0}}_{1\times 3} & -u_{i,j} \tilde{\mathbf{w}}_j^T \\ -v_{i,j} \tilde{\mathbf{w}}_j^T & u_{i,j} \tilde{\mathbf{w}}_j^T & \bar{\mathbf{0}}_{1\times 3} \end{bmatrix}}_{\mathbb{R}^{3\times 9}} \underbrace{\begin{bmatrix} \mathbf{h}_{i,1} \\ \mathbf{h}_{i,2} \\ \mathbf{h}_{i,3} \end{bmatrix}}_{\mathbb{R}^{9\times 1}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{h}_*^T \tilde{\mathbf{w}}_j = \tilde{\mathbf{w}}_j^T \mathbf{h}_* \text{ and factorization}$$

$$\iff \underbrace{\begin{bmatrix} \bar{\mathbf{0}}_{1\times 3} & -\tilde{\mathbf{w}}_j^T & v_{i,j} \tilde{\mathbf{w}}_j^T \\ \tilde{\mathbf{w}}_j^T & \bar{\mathbf{0}}_{1\times 3} & -u_{i,j} \tilde{\mathbf{w}}_j^T \end{bmatrix}}_{\mathbb{R}^{2\times 9}} \underbrace{\begin{bmatrix} \mathbf{h}_{i,1} \\ \mathbf{h}_{i,2} \\ \mathbf{h}_{i,3} \end{bmatrix}}_{\mathbb{R}^{9\times 1}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{only the first two equations are linearly independent}$$

Given c corners, a homogeneous overdetermined linear system of $2c$ equations to estimate the (vectorized) homography $\mathbf{h}_i$ is defined as follows:

$$\underbrace{\begin{bmatrix} \bar{\mathbf{0}}_{1\times 3} & -\tilde{\mathbf{w}}_1^T & v_{i,1}\tilde{\mathbf{w}}_1^T \\ \tilde{\mathbf{w}}_1^T & \bar{\mathbf{0}}_{1\times 3} & -u_{i,1}\tilde{\mathbf{w}}_1^T \\ \vdots & \vdots & \vdots \\ \bar{\mathbf{0}}_{1\times 3} & -\tilde{\mathbf{w}}_c^T & v_{i,c}\tilde{\mathbf{w}}_c^T \\ \tilde{\mathbf{w}}_c^T & \bar{\mathbf{0}}_{1\times 3} & -u_{i,c}\tilde{\mathbf{w}}_c^T \end{bmatrix}}_{\mathbb{R}^{2c\times 9}} \underbrace{\begin{bmatrix} \mathbf{h}_{i,1} \\ \mathbf{h}_{i,2} \\ \mathbf{h}_{i,3} \end{bmatrix}}_{\mathbb{R}^{9\times 1}} = \bar{\mathbf{0}}_{2c\times 1} \Rightarrow \boldsymbol{L}_i\mathbf{h}_i = \bar{\mathbf{0}}$$

With the constraint $\|\mathbf{h}_i\| = 1$ to avoid the trivial solution $\mathbf{h}_i = \bar{\mathbf{0}}$.

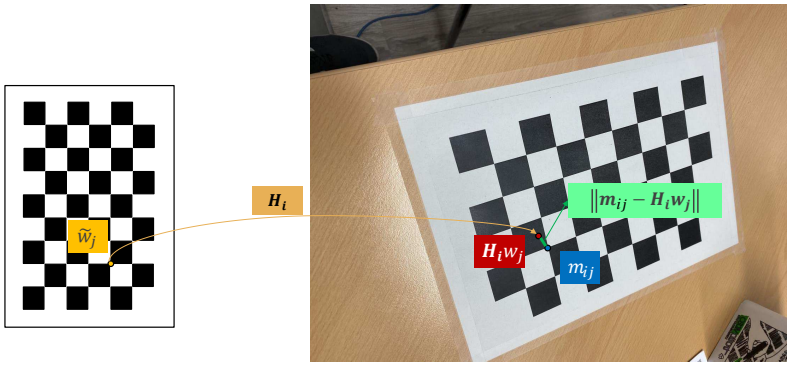
The solution $\mathbf{h}_i^*$ is found by minimizing the norm of $\boldsymbol{L}_i\mathbf{h}_i$:

$$\mathbf{h}_i^* = \arg\min_{\mathbf{h_i}\in\mathbb{R}^9} \|\boldsymbol{L}_i\mathbf{h}_i\| \text{ subject to } \|\mathbf{h}_i\| = 1$$

$\mathbf{h}_i^*$ can be found using the singular value decomposition of $\boldsymbol{L}_i = \boldsymbol{U}_i\boldsymbol{D}_i\boldsymbol{V}_i^T$. It can be shown that $\mathbf{h}_i^* = \mathbf{v}_9$ where $\mathbf{v}_9$ is the last column of $\boldsymbol{V}_i$, associated with the smallest singular value.

| **Remark.** This step minimizes an algebraic error.

Homographies non-linear refinement The homographies $\boldsymbol{H}_i$ estimated at the previous step are obtained using a linear method and need to be refined as, for each image i, the IRF coordinates $\boldsymbol{H}_i\mathbf{w}_j = \left(\frac{h_{i,1}^T\tilde{\mathbf{w}}_j}{h_{i,3}^T\tilde{\mathbf{w}}_j}, \frac{h_{i,2}^T\tilde{\mathbf{w}}_j}{h_{i,3}^T\tilde{\mathbf{w}}_j}\right)$ of the world point $\mathbf{w}_j$ are still not matching the known IRF coordinates $\mathbf{m}_{i,j}$ of the j-th corner in the i-th image.



Given an initial guess for the homography $\boldsymbol{H}_i$, we can refine it through a non-linear minimization problem:

$$\boldsymbol{H}_i^* = \arg\min_{\boldsymbol{H}_i} \sum_{j=1}^{c} \|\mathbf{m}_{i,j} - \boldsymbol{H}_i\mathbf{w}_j\|^2$$

This can be solved using an iterative algorithm (e.g. Levenberg-Marquardt algorithm).

| **Remark.** This step minimizes a geometric error.

Initial intrinsic parameters guess From the PPM, the following relationship between intrinsic and extrinsic parameters can be established:

$$\begin{aligned}
&\boldsymbol{P}_i \equiv \boldsymbol{A}[\boldsymbol{R}_i|\mathbf{t}_i] = \boldsymbol{A}\begin{bmatrix}\mathbf{r}_{i,1} & \mathbf{r}_{i,2} & \mathbf{r}_{i,3} & \mathbf{t}_i\end{bmatrix} \\
&\quad \Rightarrow \boldsymbol{H}_i = \begin{bmatrix}\mathbf{h}_{i,1} & \mathbf{h}_{i,2} & \mathbf{h}_{i,3}\end{bmatrix} = \begin{bmatrix}k\boldsymbol{A}\mathbf{r}_{i,1} & k\boldsymbol{A}\mathbf{r}_{i,2} & k\boldsymbol{A}\mathbf{t}_i\end{bmatrix} \quad \text{By definition of } \boldsymbol{H}_i \\
&\quad \Rightarrow (k\mathbf{r}_{i,1} = \boldsymbol{A}^{-1}\mathbf{h}_{i,1}) \wedge (k\mathbf{r}_{i,2} = \boldsymbol{A}^{-1}\mathbf{h}_{i,2})
\end{aligned}$$

Moreover, as $\boldsymbol{R}_i$ is an orthogonal matrix, the following two constraints must hold:

$$\begin{aligned}
\langle \mathbf{r}_{i,1}, \mathbf{r}_{i,2}\rangle = \bar{\mathbf{0}} &\Rightarrow \langle \boldsymbol{A}^{-1}\mathbf{h}_{i,1}, \boldsymbol{A}^{-1}\mathbf{h}_{i,2}\rangle = \bar{\mathbf{0}} \\
&\Rightarrow \mathbf{h}_{i,1}^T(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}\mathbf{h}_{i,2} = \bar{\mathbf{0}} \\
\langle \mathbf{r}_{i,1}, \mathbf{r}_{i,1}\rangle = \langle \mathbf{r}_{i,2}, \mathbf{r}_{i,2}\rangle &\Rightarrow \langle \boldsymbol{A}^{-1}\mathbf{h}_{i,1}, \boldsymbol{A}^{-1}\mathbf{h}_{i,1}\rangle = \langle \boldsymbol{A}^{-1}\mathbf{h}_{i,2}, \boldsymbol{A}^{-1}\mathbf{h}_{i,2}\rangle \\
&\Rightarrow \mathbf{h}_{i,1}^T(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}\mathbf{h}_{i,1} = \mathbf{h}_{i,2}^T(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}\mathbf{h}_{i,2}
\end{aligned}$$

where $\langle\cdot,\cdot\rangle$ is the dot product.

If at least 3 images have been collected, by stacking the two constraints for each image, we obtain a homogeneous system of equations that can be solved with SVD over the unknown $(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}$.

Note that $(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}$ is symmetric, therefore reducing the number of independent parameters to 5 (6 with skew).

Once $(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}$ has been estimated, the actual values of $\boldsymbol{A}$ can be found by solving a traditional system of equations using the structure and results in $(\boldsymbol{A}^{-1})^T\boldsymbol{A}^{-1}$.

| **Remark.** This step minimizes an algebraic error.

Initial extrinsic parameters guess For each image, given the estimated intrinsic matrix $\boldsymbol{A}$ and the homography $\boldsymbol{H}_i$, it holds that:

$$\begin{aligned}
\boldsymbol{H}_i &= \begin{bmatrix}\mathbf{h}_{i,1} & \mathbf{h}_{i,2} & \mathbf{h}_{i,3}\end{bmatrix} = \begin{bmatrix}k\boldsymbol{A}\mathbf{r}_{i,1} & k\boldsymbol{A}\mathbf{r}_{i,2} & k\boldsymbol{A}\mathbf{t}_i\end{bmatrix} \\
&\Rightarrow \mathbf{r}_{i,1} = \frac{\boldsymbol{A}^{-1}\mathbf{h}_{i,1}}{k}
\end{aligned}$$

Then, as $\mathbf{r}_{i,1}$ is a unit vector, it must be that $k = \|\boldsymbol{A}^{-1}\mathbf{h}_{i,1}\|$.

Now, with k estimated, $\mathbf{r}_{i,2}$ and $\mathbf{t}_i$ can be computed:

$$\mathbf{r}_{i,2} = \frac{\boldsymbol{A}^{-1}\mathbf{h}_{i,2}}{k} \qquad \mathbf{t}_i = \frac{\boldsymbol{A}^{-1}\mathbf{h}_{i,3}}{k}$$

Finally, $\mathbf{r}_{i,3}$ can be computed as:

$$\mathbf{r}_{i,3} = \mathbf{r}_{i,1} \times \mathbf{r}_{i,2}$$

where $\times$ is the cross-product. It holds that:

- $\mathbf{r}_{i,3}$ is orthogonal to $\mathbf{r}_{i,1}$ and $\mathbf{r}_{i,2}$.
- $\|\mathbf{r}_{i,3}\| = 1$ as the cross-product computes the area of the square defined by $\mathbf{r}_{i,1}$ and $\mathbf{r}_{i,2}$ (both unit vectors).

Note that the resulting rotation matrix $\boldsymbol{R}_i$ is not exactly orthogonal as:

- $\mathbf{r}_{i,1}$ and $\mathbf{r}_{i,2}$ are not necessarily orthogonal.
- $\mathbf{r}_{i,2}$ does not necessarily have unit length as k was computed considering $\mathbf{r}_{i,1}$.

SVD for $\boldsymbol{R}_i$ can be used to find the closest orthogonal matrix by substituting the singular value matrix $\boldsymbol{D}$ with the identity $\boldsymbol{I}$.

Remark. This step minimizes an algebraic error.

Initial distortion parameters guess The current estimate of the homographies $\boldsymbol{H}_i$ project WRF points into ideal (undistorted) IRF coordinates $\mathbf{m}_{\text{undist}}$ (this is an assumption). On the other hand, the coordinates $\mathbf{m}$ of the corners in the actual image are distorted.

The original algorithm estimates the parameters of the radial distortion function defined as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = L(r) \begin{bmatrix} x_{\text{undist}} \\ y_{\text{undist}} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} x_{\text{undist}} \\ y_{\text{undist}} \end{bmatrix}$$

where k_1 and k_2 are parameters.

Remark. OpenCV uses a different method to estimate:

- 3 parameters k_1, k_2, k_3 for radial distortion.
- 2 parameters p_1, p_2 for tangential distortion.

Using the estimated intrinsic matrix $\boldsymbol{A}$, it is possible to obtain the CRF coordinates (x, y) from the IRF coordinates (u, v) of $\mathbf{m}$ or $\mathbf{m}_{\text{undist}}$:

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \equiv \boldsymbol{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \equiv \begin{bmatrix} f_u x + u_0 \\ f_v y + v_0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{u - u_0}{f_u} \\ \frac{v - v_0}{f_v} \end{bmatrix}$$

Then, the distortion equation can be rewritten in IRF coordinates as:

$$\begin{aligned}
\begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix} &= (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} \frac{u_{\text{undist}}-u_0}{f_u} \\ \frac{v_{\text{undist}}-v_0}{f_v} \end{bmatrix} \\
\Rightarrow \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} &= (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{\text{undist}} - u_0 \\ v_{\text{undist}} - v_0 \end{bmatrix} \\
\Rightarrow \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} - \begin{bmatrix} u_{\text{undist}} - u_0 \\ v_{\text{undist}} - v_0 \end{bmatrix} &= (k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{\text{undist}} - u_0 \\ v_{\text{undist}} - v_0 \end{bmatrix} \\
\Rightarrow \begin{bmatrix} u - u_{\text{undist}} \\ v - v_{\text{undist}} \end{bmatrix} &= \begin{bmatrix} (u_{\text{undist}} - u_0) r^2 & (u_{\text{undist}} - u_0) r^4 \\ (v_{\text{undist}} - v_0) r^2 & (v_{\text{undist}} - v_0) r^4 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}
\end{aligned}$$

With n images with c corners each, we obtain $2nc$ equations to form a system $\mathbf{d} = \boldsymbol{D}\mathbf{k}$ in 2 unknowns $\mathbf{k} = \begin{bmatrix} k_1 & k_2 \end{bmatrix}^T$. This can be solved in a least squares approach as:

$$\mathbf{k}^* = \min_k \|\boldsymbol{D}\mathbf{k} - \mathbf{d}\|_2 = \boldsymbol{D}^\dagger \mathbf{d} = (\boldsymbol{D}^T \boldsymbol{D})^{-1} \boldsymbol{D}^T \mathbf{d}$$

where $\boldsymbol{D}^\dagger$ is the pseudo-inverse of $\boldsymbol{D}$.

Remark. This step minimizes an algebraic error.

Parameters non-linear refinement A final non-linear refinement of all the estimated parameters is done to obtain a solution closer to their physical meaning.

Assuming i.i.d. noise, this is done through the maximum likelihood estimate (MLE) using the estimated parameters as starting point:

$$\boldsymbol{A}^*, \mathbf{k}^*, \boldsymbol{R}_i^*, \mathbf{t}_i^* = \arg \min_{\boldsymbol{A}, \mathbf{k}, \boldsymbol{R}_i, \mathbf{t}_i} \sum_{i=1}^{n} \sum_{j=1}^{c} \|\tilde{\mathbf{m}}_{i,j} - \hat{\mathbf{m}}(\boldsymbol{A}, \mathbf{k}, \boldsymbol{R}_i, \mathbf{t}_i, \tilde{\mathbf{w}}_j)\|^2$$

where $\tilde{\mathbf{m}}_{i,j}$ are the known IRF coordinates in projective space of the j-th corner in the i-th image and $\hat{\mathbf{m}}(\cdot)$ is the projection from WRF to IRF coordinates using the estimated parameters.

This can be solved using iterative algorithms.

Remark. This step minimizes a geometric error.

1.5 Warping

Warp Transformation of an image on the spatial domain. Warp

Given an image I, warping can be seen as a function w that computes the new coordinates of each pixel:

$$u' = w_u(u, v) \qquad v' = w_v(u, v)$$

The transformation can be:

Rotation $\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$

Full homography $k \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$

Remark. Differently from warping, filtering transforms the pixel intensities of an image.

1.5.1 Forward mapping

Starting from the input image coordinates, apply the warping function to obtain the output image. Forward mapping

Output coordinates might be continuous and need to be discretized (e.g. truncated or rounded). This might give rise to two problems:

Fold More than one input pixel ends up in the same output pixel.

Hole An output pixel does not have a corresponding input pixel.

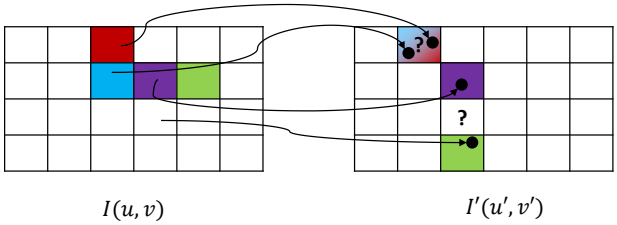
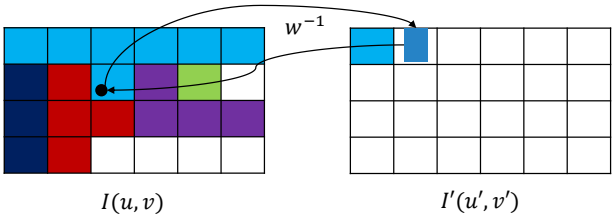


Figure 1.8: Example of fold and hole

1.5.2 Backward mapping

Starting from the output image coordinates, use the inverse of the warping function w^{-1} to find its corresponding input coordinates. Backward mapping

$$u = w_u^{-1}(u', v') \qquad v = w_v^{-1}(u', v')$$



The computed input coordinates might be continuous. Possible discretization strategies are:

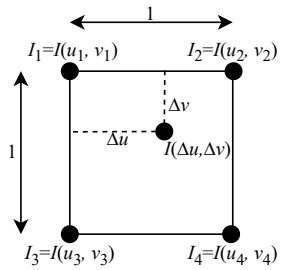
- Truncation.
- Nearest neighbor.
- Interpolation between the 4 closest pixels of the continuous point (e.g. bilinear, bicubic, ...).

Bilinear interpolation

Bilinear interpolation Given a continuous coordinate (u, v) and its closest four pixels $(u_1, v_1), \ldots, (u_4, v_4)$ with intensities denoted for simplicity as $I_i = I(u_i, v_i)$, bilinear interpolation works as follows:

1. Compute the offset of (u, v) w.r.t. the top-left pixel:

$$\Delta u = u - u_1 \qquad \Delta v = v - v_1$$



2. Interpolate a point (u_a, v_a) between (u_1, v_1) and (u_2, v_2) in such a way that it is perpendicular to (u, v). Do the same for a point (u_b, v_b) between (u_3, v_3) and (u_4, v_4). The intensities of the new points are computed by interpolating the intensities of their extrema:

$$I_a = I_1 + (I_2 - I_1)\Delta u \qquad I_b = I_3 + (I_4 - I_3)\Delta u$$

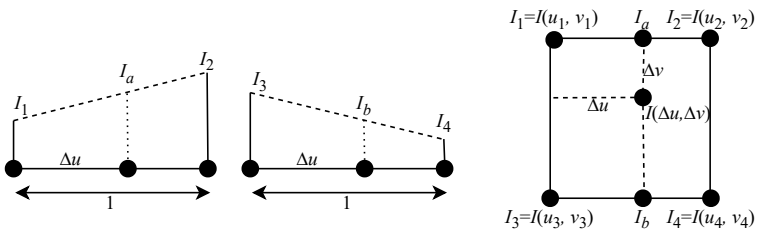
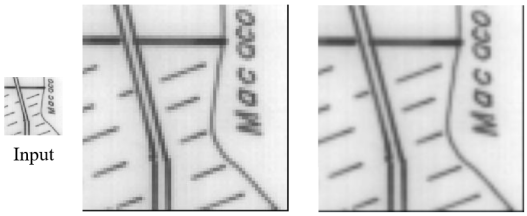


Figure 1.9: In the figure, it is assumed that $I_1 < I_2$ and $I_3 > I_4$

3. The intensity $I(\Delta u, \Delta v) = I'(u', v')$ in the warped image is obtained by interpolating the intensities of I_a and I_b:

$$\begin{aligned} I'(u', v') &= I_a + (I_b - I_a)\Delta v \\ &= \Big(I_1 + (I_2 - I_1)\Delta u\Big) + \Big(\big(I_3 + (I_4 - I_3)\Delta u\big) - \big(I_1 + (I_2 - I_1)\Delta u\big)\Big)\Delta v \\ &= (1 - \Delta u)(1 - \Delta v)I_1 + \Delta u(1 - \Delta v)I_2 + (1 - \Delta u)\Delta v I_3 + \Delta u \Delta v I_4 \end{aligned}$$

Remark (Zoom)**.** Zooming using nearest-neighbor produces sharper edges while bilinear interpolation results in smoother images.



Remark. Nearest-neighbor is suited to preserve transition (e.g. zoom a binary mask while maintaining the 0s and 1s).

1.5.3 Undistort warping

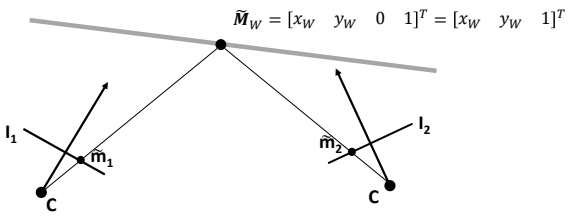
Once a camera has been calibrated using Zhang's method, the lens distortion parameters can be used to obtain the undistorted image through backward warping.

$$\begin{aligned} w_u &= u_{\text{undist}} + (k_1 r^2 + k_2 r^4)(u_{\text{undist}} - u_0) \\ w_v &= v_{\text{undist}} + (k_1 r^2 + k_2 r^4)(v_{\text{undist}} - v_0) \end{aligned}$$

$$I'(u_{\text{undist}}, v_{\text{undist}}) = I\big(w_u^{-1}(u_{\text{undist}}, v_{\text{undist}}), w_v^{-1}(u_{\text{undist}}, v_{\text{undist}})\big)$$

Undistorted images enjoy some properties:

Planar warping Any two images without lens distortion of a planar world scene ($z_W = 0$) are related by a homography. Planar warping



Given two images containing the same world point, their image points (in projective space) are respectively given by a homography $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ (note that with $z_w = 0$, the PPM is a 3×3 matrix and therefore a homography):

$$\tilde{\mathbf{m}}_1 = \boldsymbol{H}_1 \tilde{\mathbf{M}}_W \qquad \tilde{\mathbf{m}}_2 = \boldsymbol{H}_2 \tilde{\mathbf{M}}_W$$

$$\tilde{\mathbf{m}}_1 = \boldsymbol{H}_1 \boldsymbol{H}_2^{-1} \tilde{\mathbf{m}}_2 \qquad \tilde{\mathbf{m}}_2 = \boldsymbol{H}_2 \boldsymbol{H}_1^{-1} \tilde{\mathbf{m}}_1$$

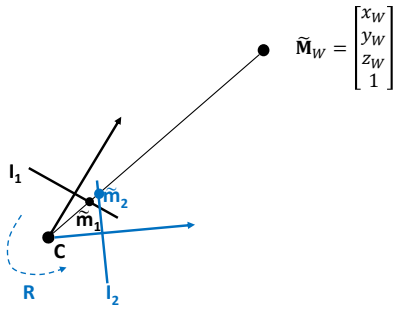
Then, $\boldsymbol{H}_1 \boldsymbol{H}_2^{-1} = \boldsymbol{H}_{21} = \boldsymbol{H}_{12}^{-1}$ is the homography that relates $\tilde{\mathbf{m}}_2$ to $\tilde{\mathbf{m}}_1$ and $\boldsymbol{H}_2 \boldsymbol{H}_1^{-1} = \boldsymbol{H}_{12} = \boldsymbol{H}_{21}^{-1}$ relates $\tilde{\mathbf{m}}_1$ to $\tilde{\mathbf{m}}_2$.

Remark. Only ground points on the planar section of the image can be correctly warped.

Example (Inverse Perspective Mapping)**.** In autonomous driving, it is usually useful to have a bird-eye view of the road.
In a controlled environment, a calibrated camera can be mounted on a car to take a picture of the road in front of it. Then, a (virtual) image of the road viewed from above is generated. By finding the homography that relates the two images, it is possible to produce a bird-eye view of the road from the camera mounted on the vehicle.
Note that the homography needs to be computed only once.

Rotation warping Any two images without lens distortion taken by rotating the camera about its optical center are related by a homography.

Rotation warping



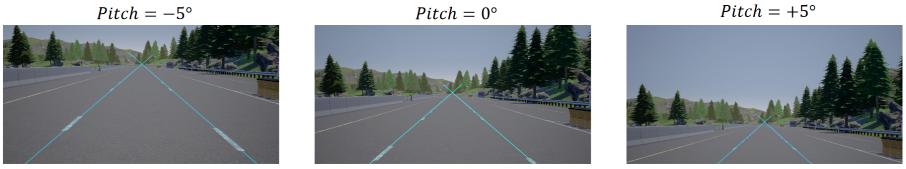
It is assumed that the first image is taken in such a way that the WRF and CRF are the same (i.e. no extrinsic parameters). Then, a second image is taken by rotating the camera about its optical center. It holds that:

$$\tilde{\mathbf{m}}_1 = \boldsymbol{A}[\boldsymbol{I}|\bar{\mathbf{0}}]\tilde{\mathbf{M}}_W = \boldsymbol{A}\tilde{\mathbf{M}}_W \qquad \tilde{\mathbf{m}}_2 = \boldsymbol{A}[\boldsymbol{R}|\bar{\mathbf{0}}]\tilde{\mathbf{M}}_W = \boldsymbol{A}\boldsymbol{R}\tilde{\mathbf{M}}_W$$
$$\tilde{\mathbf{m}}_1 = \boldsymbol{A}\boldsymbol{R}^{-1}\boldsymbol{A}^{-1}\tilde{\mathbf{m}}_2 \qquad \tilde{\mathbf{m}}_2 = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^{-1}\tilde{\mathbf{m}}_1$$

Then, $\boldsymbol{A}\boldsymbol{R}^{-1}\boldsymbol{A}^{-1} = \boldsymbol{H}_{21} = \boldsymbol{H}_{12}^{-1}$ is the homography that relates $\tilde{\mathbf{m}}_2$ to $\tilde{\mathbf{m}}_1$ and $\boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^{-1} = \boldsymbol{H}_{12} = \boldsymbol{H}_{21}^{-1}$ relates $\tilde{\mathbf{m}}_1$ to $\tilde{\mathbf{m}}_2$.

Remark. Any point of the image can be correctly warped.

Example (Compensate pitch or yaw)**.** In autonomous driving, cameras should be ideally mounted with the optical axis parallel to the road plane and aligned with the direction of motion. It is usually very difficult to physically obtain perfect alignment but a calibrated camera can help to compensate pitch (i.e. rotation around the x-axis) and yaw (i.e. rotation around the y-axis) by estimating the vanishing point of the lane lines.



It is assumed that the vehicle is driving straight w.r.t. the lines and that the WRF is attached to the vehicle in such a way that the z-axis is pointing in front of the vehicle. It holds that any line parallel to the z-axis has direction $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$ and their point at infinity in perspective space is at $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T$.
The coordinates of the vanishing point are then obtained as:

$$\mathbf{m}_\infty \equiv \boldsymbol{A}[\boldsymbol{R}|0] \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \equiv \boldsymbol{A}\mathbf{r}_3 \equiv \boldsymbol{A} \begin{bmatrix}0\\\sin\beta\\\cos\beta\end{bmatrix}$$

where $\mathbf{r}_3$ is the third column of the rotation matrix $\boldsymbol{R}_{\text{pitch}} = \begin{bmatrix}1 & 0 & 0\\0 & \cos\beta & \sin\beta\\0 & -\sin\beta & \cos\beta\end{bmatrix}$ that applies a rotation of β degree around the x-axis.
By computing the point at infinity, it is possible to estimate $\mathbf{r}_3 = \frac{\boldsymbol{A}^{-1}\mathbf{m}_\infty}{\|\boldsymbol{A}^{-1}\mathbf{m}_\infty\|_2}$ (as $\mathbf{r}_3$ is a unit vector) and from it we can find the entire rotation matrix $\boldsymbol{R}_{\text{pitch}}$.
Finally, the homography $\boldsymbol{A}\boldsymbol{R}_{\text{pitch}}\boldsymbol{A}^{-1}$ relates the pitched image to the ideal image.

Remark. The same procedure can be done for the yaw.

2 Image classification

2.1 Supervised datasets

Dataset Given a set of labeled data, it can be split into: Dataset

Train set $D^{\text{train}} = \{(\mathrm{x}^{(i)}_{\text{train}}, y^{(i)}_{\text{train}}) \mid i = 1, \dots, N\}$.

Test set $D^{\text{test}} = \{(\mathrm{x}^{(i)}_{\text{test}}, y^{(i)}_{\text{test}}) \mid i = 1, \dots, M\}$.

It is assumed that the two sets contain i.i.d. samples drawn from the same unknown distribution.

2.1.1 Modified NIST (MNIST)

Content Handwritten digits from 0 to 9.

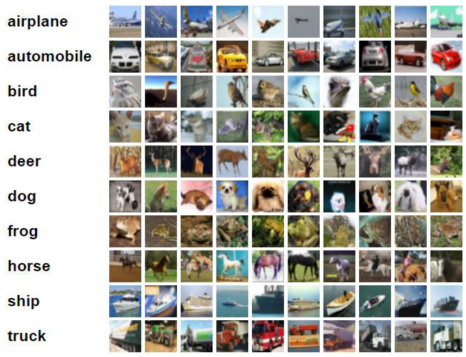
Number of classes 10.

Train set size 50k.

Test set size 10k.

Image format 28×28 grayscale.

2.1.2 CIFAR10



Content Objects of various categories.

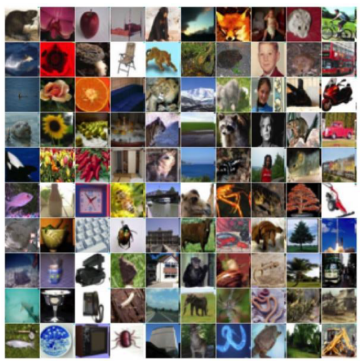
Number of classes 10.

Train set size 50k.

Test set size 10k.

Image size 32×32 RGB.

2.1.3 CIFAR100



Content Objects of various categories.

Number of classes 100 (20 super-classed with 5 sub-classes).

Train set size 50k.

Test set size 10k.

Image size 32×32 RGB.

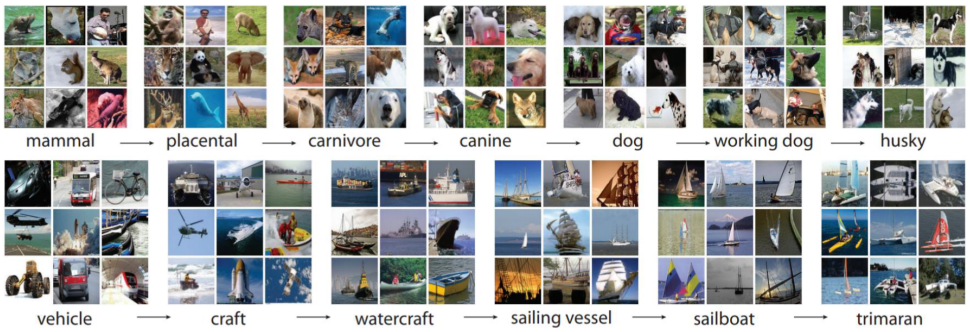
2.1.4 ImageNet 21k

Content Objects of various categories.

Number of classes 21k synsets from WordNet organized hierarchically.

Dataset size 14 millions.

Image size Variable resolution RGB. Average size of 400×350.



2.1.5 ImageNet 1k

Content Objects of various categories.

Number of classes 1000.

Train set size 1.3 millions.

Validation set size 50k.

Test set size 100k.

Image size Variable resolution RGB. Often resized to 256×256.

Remark. Performance is usually measured as top-5 accuracy as making a single prediction might be ambiguous due to the fact that the images can contain multiple objects.

2.2 Learning

Learning problem Find the best model h^* from the hypothesis space $\mathbb{H}$ that minimizes a loss function $\mathcal{L}$: Learning problem

$$h^* = \arg\min_{h \in \mathbb{H}} \mathcal{L}(h, \boldsymbol{D}^{\text{train}})$$

In machine learning, models are usually parametrized. The problem then becomes to find the best set of parameters $\boldsymbol{\theta}^*$ from the parameter space Θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{D}^{\text{train}})$$

2.2.1 Loss function

Loss function Easy to optimize function that acts as a proxy to measure the goodness of a model. Loss function

The loss computed on a dataset is usually obtained as the average of the values of the single samples:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{D}^{\text{train}}) = \frac{1}{N} \sum_{i}^{|\boldsymbol{D}^{\text{train}}|} \mathcal{L}\big(\boldsymbol{\theta}, (\mathbf{x}^{(i)}, y^{(i)})\big)$$

0-1 loss Loss computed as the number of misclassifications: 0-1 loss

$$\mathcal{L}\big(\boldsymbol{\theta}, (\mathbf{x}^{(i)}, y^{(i)})\big) = |\text{misclassifications}|$$

This loss is not ideal as it is insensitive to small (or even large) changes in the parameters. Moreover, it does not tell in which direction should the parameters be modified to reduce the loss.

Remark. This loss can be minimized using a combinatorial optimization approach but it does not scale well with large datasets.

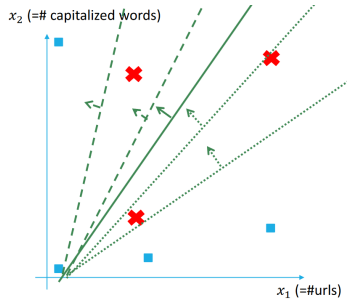


Figure 2.1: Example of linear classifier for spam detection. Small changes on the boundary line do not change the 0-1 loss. The loss itself does not tell which is the best direction to move the line.

Root mean square error Loss computed as the direct comparison between the prediction and target label: Root mean square error

$$\mathcal{L}\big(\boldsymbol{\theta}, (\mathbf{x}^{(i)}, y^{(i)})\big) = \|f(\mathbf{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)}\|_2$$

Note that $y^{(i)}$ might be encoded (e.g. one-hot).

Cross-entropy loss Transform the logits of a model into a probability distribution and estimate the parameters through MLE. Cross-entropy loss

Softmax Function that converts its input into a probability distribution. Given the logits $\mathbf{s} \in \mathbb{R}^c$, the score $\mathbf{s}_j$ of class j is converted into a probability as follows: Softmax

$$\mathcal{P}_{\text{model}}(Y = j | X = \mathbf{x}^{(i)}; \boldsymbol{\theta}) = \texttt{softmax}_j(\mathbf{s}) = \frac{\exp(\mathbf{s}_j)}{\sum_{k=1}^{c} \exp(\mathbf{s}_k)}$$

For numerical stability, `softmax` is usually computed as:

$$\begin{aligned}\texttt{softmax}_j(\mathbf{s} - \max\{\mathbf{s}\}) &= \frac{\exp(\mathbf{s}_j - \max\{\mathbf{s}\})}{\sum_{k=1}^{c} \exp(\mathbf{s}_k - \max\{\mathbf{s}\})} \\ &= \frac{\cancel{\exp(-\max\{\mathbf{s}\})}\exp(\mathbf{s}_j)}{\cancel{\exp(-\max\{\mathbf{s}\})}\sum_{k=1}^{c} \exp(\mathbf{s}_k)} = \texttt{softmax}_j(\mathbf{s})\end{aligned}$$

Maximum likelihood estimation Use MLE to estimate the parameters on the probability distribution outputted by the `softmax` function: Cross-entropy loss

$$\begin{aligned}
\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \mathcal{P}_{\text{model}}(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} \mathcal{P}_{\text{model}}(Y = y^{(i)} | X = \mathbf{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log \mathcal{P}_{\text{model}}(Y = y^{(i)} | X = \mathbf{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} -\log \mathcal{P}_{\text{model}}(Y = y^{(i)} | X = \mathbf{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} -\log \left(\frac{\exp(\mathbf{s}_{y^{(i)}})}{\sum_{k=1}^{c} \exp(\mathbf{s}_k)} \right) \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} -\log \left(\exp(\mathbf{s}_{y^{(i)}}) \right) + \log \left(\sum_{k=1}^{c} \exp(\mathbf{s}_k) \right) \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} -\mathbf{s}_{y^{(i)}} + \log \left(\sum_{k=1}^{c} \exp(\mathbf{s}_k) \right)
\end{aligned}$$

The second term ($\log\left(\sum_{k=1}^{c} \exp(\mathbf{s}_k)\right)$) is called `logsumexp` and approximates the max function. Therefore, the loss can be seen as:

$$\mathcal{L}\big(\boldsymbol{\theta}, (\mathbf{x}^{(i)}, y^{(i)})\big) = -\mathbf{s}_{y^{(i)}} + \log \left(\sum_{k=1}^{c} \exp(\mathbf{s}_k) \right) \approx -\mathbf{s}_{y^{(i)}} + \max\{\mathbf{s}\}$$

2.2.2 Gradient descent

Gradient descent An epoch e of gradient descent does the following: Gradient descent

1. Classify all training data to obtain the predictions $\hat{y}^{(i)} = f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(e-1)})$ and the loss $\mathcal{L}(\boldsymbol{\theta}^{(e-1)}, \boldsymbol{D}^{\text{train}})$.
2. Compute the gradient $\nabla\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(e-1)}, \boldsymbol{D}^{\text{train}})$.
3. Update the parameters $\boldsymbol{\theta}^{(e)} = \boldsymbol{\theta}^{(e-1)} - \texttt{lr} \cdot \nabla\mathcal{L}$.

Stochastic gradient descent Reduce the computational cost of gradient descent by computing the gradient of a single sample. An epoch e of SGD does the following: Stochastic gradient descent

1. Shuffle the training data $\boldsymbol{D}^{\text{train}}$.
2. For $i = 0, \dots, N-1$:
   a) Classify $\mathbf{x}^{(i)}$ to obtain the prediction $\hat{y}^{(i)} = f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(e*N+i)})$ and the loss $\mathcal{L}\big(\boldsymbol{\theta}^{(e*N+i)}, (\mathbf{x}^{(i)}, y^{(i)})\big)$.
   b) Compute the gradient $\nabla\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\big(\boldsymbol{\theta}^{(e*N+i)}, (\mathbf{x}^{(i)}, y^{(i)})\big)$.
   c) Update the parameters $\boldsymbol{\theta}^{(e*N+i+1)} = \boldsymbol{\theta}^{(e*N+i)} - \texttt{lr} \cdot \nabla\mathcal{L}$.

SGD with mini-batches Increase the update accuracy of SGD by using a mini-batch. An epoch e of SGD with mini-batches of size B does the following: SGD with mini-batches

1. Shuffle the training data $\boldsymbol{D}^{\text{train}}$.
2. For $u = 0, \dots, U$, with $U = \lceil \frac{N}{B} \rceil$:
   a) Classify the examples $\boldsymbol{X}^{(u)} = \{\mathbf{x}^{(Bu)}, \dots, \mathbf{x}^{(B(u+1)-1)}\}$ to obtain the predictions $\hat{Y}^{(u)} = f(\mathbf{X}^{(u)}; \boldsymbol{\theta}^{(e*U+u)})$ and the loss $\mathcal{L}\big(\boldsymbol{\theta}^{(e*U+u)}, (\boldsymbol{X}^{(u)}, \hat{Y}^{(u)})\big)$.
   b) Compute the gradient $\nabla\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\big(\boldsymbol{\theta}^{(e*U+u)}, (\boldsymbol{X}^{(u)}, \hat{Y}^{(u)})\big)$.
   c) Update the parameters $\boldsymbol{\theta}^{(e*U+u+1)} = \boldsymbol{\theta}^{(e*U+u)} - \texttt{lr} \cdot \nabla\mathcal{L}$.

The following properties generally hold:

- Larger batches provide a smoother estimation of the gradient and allow to better exploit parallel hardware (below a certain limit, there is no gain in time).
- Smaller batches require more iterations to train but might have a regularization effect for better generalization.

Gradient computation Gradients can be computed: Gradient computation

Numerically Slow and approximate but easy to implement.

Analytically Using the chain rule.

Automatically Using automatic differentiation (e.g. backpropagation).

2.3 Linear classifier

Determine the class by computing a linear combination of the input. Linear classifier
Given c classes and a flattened image $\mathbf{x} \in \mathbb{R}^i$, a linear classifier f parametrized on $\boldsymbol{W} \in \mathbb{R}^{c \times i}$ is defined as:

$$f(\mathbf{x}; \boldsymbol{W}) = \boldsymbol{W}\mathbf{x} = \texttt{logits}$$

where the $\texttt{logits} \in \mathbb{R}^c$ vector contains a score for each class.
The prediction is obtained as the index of the maximum score.

Remark. Directly predicting the integer encoded classes is not ideal as it would give a (probably) inexistent semantic ordering (e.g. if 2 encodes bird and 3 encodes cat, 2.5 should not mean half bird and half cat).

Remark. Linear classifiers can be seen as a template-matching method. Each row of $\boldsymbol{W} \in \mathbb{R}^{c \times i}$ is a class template that is cross-correlated with the image to obtain a score.

Remark. In practice, a linear classifier is actually an affine classifier parametrized on $\theta = (\boldsymbol{W} \in \mathbb{R}^{c \times i}, \mathbf{b} \in \mathbb{R}^c)$: Affine classifier

$$f(\mathbf{x}; \theta) = \boldsymbol{W}\mathbf{x} + \mathbf{b} = \texttt{logits}$$

Remark. Linear classifiers are limited by the expressiveness of the input data as pixels alone do not contain relevant features.

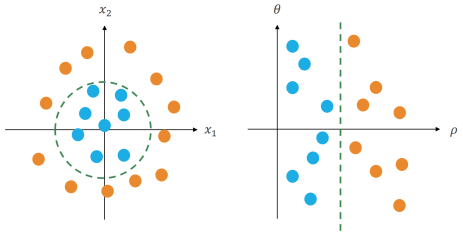


Figure 2.2: Example of non-linearly separable data points that become linearly separable in polar coordinates

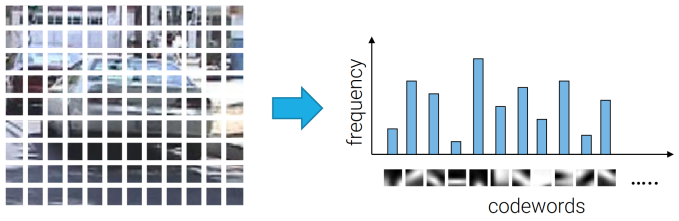
2.4 Bag of visual words

Codeword Visual feature (e.g. an edge with a particular direction) that appears in an image.

Codeword

Bag of visual words (BOVW) Encoding of an image into a histogram of codeword frequencies.

Bag of visual words (BOVW)



2.5 Neural networks

Shallow neural network Linear transformations with an activation function:

Shallow neural network

$$\begin{aligned} f(\mathbf{x}, \boldsymbol{\theta}) &= \boldsymbol{W}_2\mathbf{h} + \mathbf{b}_2 \\ &= \boldsymbol{W}_2\phi(\boldsymbol{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = \mathbf{s} \end{aligned}$$

where:

- $\boldsymbol{\theta} = (\boldsymbol{W}_1 \in \mathbb{R}^{h\times i}, \mathbf{b}_1 \in \mathbb{R}^h, \boldsymbol{W}_2 \in \mathbb{R}^{c\times h}, \mathbf{b}_2 \in \mathbb{R}^c)$ are the parameters of the linear transformations with an inner representation of size h.
- ϕ is an activation function.
- $\mathbf{h}$ and $\mathbf{s}$ are activations.

Activation function Function to introduce non-linearity.

Activation function

> **Remark.** Without an activation function, a neural network is equivalent to a plain linear transformation.

Examples of activation functions are:

Sigmoid Defined as:

$$\sigma(a) = \frac{1}{1+\exp(-a)} \qquad \frac{\partial\sigma(a)}{\partial a} = \sigma(a)\big(1-\sigma(a)\big)$$

It is subject to the vanishing gradient problem.

Rectified linear unit (ReLU) Defined as:

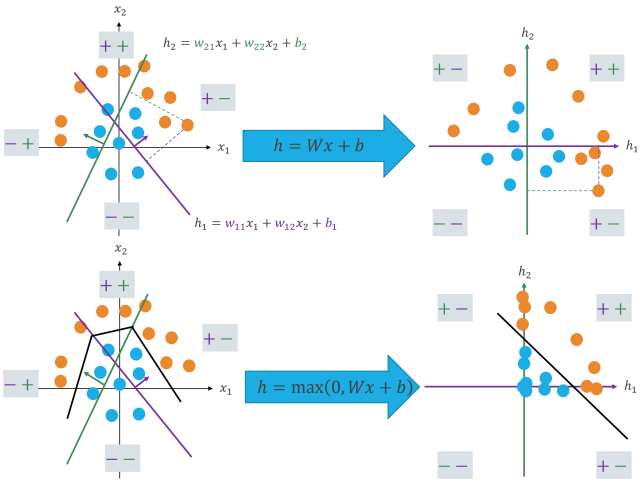
$$\texttt{ReLU}(a) = \max\{0, a\} \qquad \frac{\partial\texttt{ReLU}(a)}{\partial a} = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

It is subject to the dead neuron problem for negative inputs.

Leaky ReLU Defined as:

$$\texttt{leaky\_ReLU}(a) = \begin{cases} a & \text{if } a \geq 0 \\ 0.01 \cdot a & \text{otherwise} \end{cases} \qquad \frac{\partial\texttt{leaky\_ReLU}(a)}{\partial a} = \begin{cases} 1 & \text{if } a \geq 0 \\ 0.01 & \text{otherwise} \end{cases}$$

Example (Linear separability)**.** Linear transformations do not change the linear separability of the data points. A non-linear function can make linear separation possible.



Deep neural network Multiple layers of linear transformations and activation functions: Deep neural network

$$\begin{aligned} f(\mathbf{x}, \boldsymbol{\theta}) &= \boldsymbol{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L \\ &= \boldsymbol{W}_L \phi_L(\boldsymbol{W}_{L-1}\mathbf{h}_{L-2} + \mathbf{b}_{L-1}) + \mathbf{b}_L \\ &= \boldsymbol{W}_L \phi_L(\boldsymbol{W}_{L-1}\phi_{L-1}(\cdots \phi_1(\boldsymbol{W}_1 \mathbf{x} + \mathbf{b}_1)\cdots) + \mathbf{b}_{L-1}) + \mathbf{b}_L = \mathbf{s} \end{aligned}$$

Depth Number of layers.

Width Number of activations at each layer.

2.6 Convolutional neural networks

2.6.1 Image filtering

Consider the case of vertical edge detection. Image filtering can be implemented through:

Fully-connected layer Use an FC layer to transform the image. Image filtering with fully-connected layers

Given an image of size $H \times W$, the layer requires:

- $(H \cdot W) \cdot (H \cdot (W-1)) \approx H^2W^2$ parameters.
- $2(H \cdot W) \cdot (H \cdot (W-1)) \approx 2H^2W^2$ FLOPs (multiplications and additions).

Convolution/Correlation Use a convolution (more precisely, a cross-correlation) to transform the image. Image filtering with convolutions

Remark. Convolutions preserve the spatial structure of the image, have shared parameters and extract local features.

Given an image of size $H \times W$, a convolution requires:

- 2 parameters (in the case of edge detection).
- $3(H \cdot (W-1)) \approx 3HW$ FLOPs.

Convolution matrix A convolution can be expressed as a multiplication matrix such that:

- The parameters are shared across rows.

- The resulting matrix is sparse.
- It adapts to varying input sizes.
- It is equivariant to translation (but not w.r.t. rotation and scale).

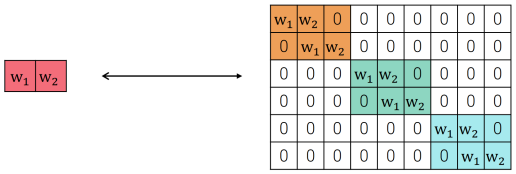


Figure 2.4: Multiplication matrix of a 1×2 convolution

2.6.2 Convolutional layer

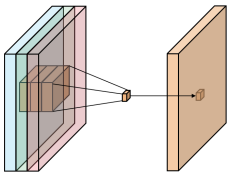
Multi-channel convolution

Multi-channel convolution On inputs with multiple channels (e.g. RGB images), different 2D convolutions are applied across the different channels.

Given a $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$ image I, a convolution kernel K will have shape $C_{\text{in}} \times H_K \times W_K$ and the output activation at each pixel is computed as:

$$[K * I](j,i) = \sum_{n=1}^{C_{\text{in}}} \sum_{m=-\lfloor \frac{H_K}{2} \rfloor}^{\lfloor \frac{H_K}{2} \rfloor} \sum_{l=-\lfloor \frac{W_K}{2} \rfloor}^{\lfloor \frac{W_K}{2} \rfloor} K_n(m,l) I_n(j-m, i-l) + b$$

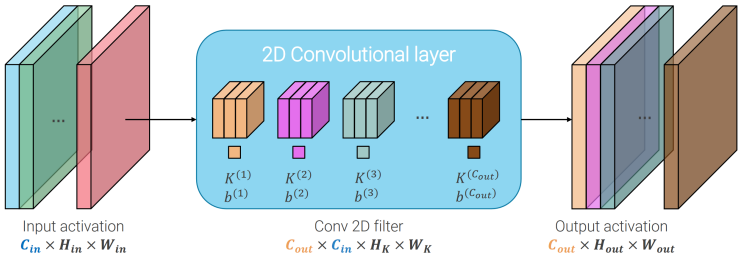
where b is a bias term associated with the filter.



2D convolutional layer

2D convolutional layer Given a $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$ image I and a desired number of channels C_{out} in the output activation, multiple different convolution kernels $K^{(i)}$ are applied and their results are stacked:

$$[K * I]_k(j,i) = \sum_{n=1}^{C_{\text{in}}} \sum_m \sum_l K_n^{(k)}(m,l) I_n(j-m, i-l) + b^{(k)} \quad \text{for } k = 1, \ldots, C_{\text{out}}$$



Remark. Only applying convolutions results in a linear transformation of the input. Therefore, an activation function is applied after convolving.

Padding

No padding Convolutions are only applied at pixels on which they do not overflow. No padding

Given a $H_{\text{in}} \times W_{\text{in}}$ image and a $H_K \times W_K$ kernel, the output shape is:

$$H_{\text{out}} = H_{\text{in}} - H_K + 1 \qquad W_{\text{out}} = W_{\text{in}} - W_K + 1$$

| **Remark.** This type of padding is referred to as `valid`.

Zero padding Zeros are added around the image. Zero padding

Given a $H_{\text{in}} \times W_{\text{in}}$ image and a $H_K \times W_K$ kernel, the padding is usually $P = \frac{H_K - 1}{2}$ (for odd square kernels) and the output shape is:

$$H_{\text{out}} = H_{\text{in}} - H_K + 1 + 2P \qquad W_{\text{out}} = W_{\text{in}} - W_K + 1 + 2P$$

| **Remark.** This type of padding is referred to as `same`.

Stride Amount of pixels the convolution kernel is slid after each application. This is useful for downsampling the image. Stride

Given a $H_{\text{in}} \times W_{\text{in}}$ image and a $H_K \times W_K$ kernel, the output with stride S and padding P has shape:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - H_K + 2P}{S} \right\rfloor + 1 \qquad W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - W_K + 2P}{S} \right\rfloor + 1$$

Receptive field Number of pixels in the input image that affects a hidden unit. Receptive field

Given a $H_K \times W_K$ kernel, without stride, the receptive field of a neuron at the L-th layer is:

$$r_L = \big(1 + L \cdot (H_K - 1)\big) \cdot \big(1 + L \cdot (W_K - 1)\big)$$

If each layer has a stride S_l, then the receptive field of the L-th activation is:

$$r_L = \left(1 + \sum_{l=1}^{L} \left((H_K - 1) \prod_{i=1}^{l-1} S_i\right)\right) \cdot \left(1 + \sum_{l=1}^{L} \left((W_K - 1) \prod_{i=1}^{l-1} S_i\right)\right)$$

| **Remark.** Without stride, the receptive field grows linearly with the number of layers. With the same stride (> 1) across all the layers, the growth becomes exponential as $\prod_{i=1}^{l-1} S_i = S^{l-1}$.

Computational cost Computational cost

Parameters Given a $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$ image, a kernel $H_K \times W_K$ and a desired number of output channels C_{out}, the corresponding convolutional layer has the following number of parameters:

$$C_{\text{out}}(C_{\text{in}} H_K W_K + 1)$$

Floating-point operations Given a $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$ input image, a kernel $H_K \times W_K$ and the corresponding output image of size $C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}}$, the number of FLOPs (multiplications and additions) is:

$$2(C_{\text{out}} H_{\text{out}} W_{\text{out}})(C_{\text{in}} H_K W_K)$$

Multiply-accumulate operations A MAC operation implemented in hardware allows to perform a multiplication and an addition in a single clock cycle. Therefore, the number of MACs is:

$$\not{2}(C_{\text{out}} H_{\text{out}} W_{\text{out}})(C_{\text{in}} H_K W_K)$$

Other convolutional layers

1D convolutional layer Suitable for time series. 1D convolutional layer

$$[K * S]_k(i) = \sum_{n=1}^{C_{\text{in}}} \sum_l K_n^{(k)}(l) S_n(i-l) + b^{(k)}$$

3D convolutional layer Suitable for videos. 3D convolutional layer

$$[K * V]_k(h,j,i) = \sum_{n=1}^{C_{\text{in}}} \sum_p \sum_m \sum_l K_n^{(k)}(p,m,l) V_n(h-p, j-m, i-l) + b^{(k)}$$

2.6.3 Pooling layer

Kernel that aggregates several values through a fixed function into one output. Each input channel is processed independently (i.e. $C_{\text{in}} = C_{\text{out}}$). Pooling layer

Remark. Traditionally, pooling layers were used for downsampling. Therefore, the stride is usually > 1.

Max pooling Select the maximum within the kernel. Max pooling

Remark. Max pooling is invariant to small (depending on the receptive field, it can also be big w.r.t the input image) spatial translations.

Remark. Mean pooling can be represented through normal convolutions.

2.6.4 Batch normalization layer

Normalize the output of a layer during training in such a way that it has zero mean and unit variance. Batch normalization layer

Training During training, normalization is done on the current batch. Given the B activations of a batch $\{\mathbf{a}^{(i)} \in \mathbb{R}^D \mid i = 1, \dots, B\}$, mean and variance are computed as:

$$\boldsymbol{\mu}_j = \frac{1}{B} \sum_{i=1}^{B} \mathbf{a}_j^{(i)} \qquad \mathbf{v}_j = \frac{1}{B} \sum_{i=1}^{B} \left(\mathbf{a}_j^{(i)} - \boldsymbol{\mu}_j \right)^2 \qquad \text{for } j = 1, \dots, D$$

Then, the normalized activation is computed as:

$$\hat{\mathbf{a}}_j^{(i)} = \frac{\mathbf{a}_j^{(i)} - \boldsymbol{\mu}_j}{\sqrt{\mathbf{v}_j + \varepsilon}} \qquad \text{for } j = 1, \dots, D$$

where ε is a small constant.

To introduce some flexibility, the final activation $\mathbf{s}^{(i)}$ is learned as:

$$\mathbf{s}_j^{(i)} = \boldsymbol{\gamma}_j \hat{\mathbf{a}}_j^{(i)} + \boldsymbol{\beta}_j \qquad \text{for } j = 1, \dots, D$$

where $\boldsymbol{\gamma}_j$ and $\boldsymbol{\beta}_j$ are parameters.

To estimate the mean and variance of the entire dataset to use during inference, their running averages are also computed. At the t-th step, the running averages of mean and variance are computed as:

$$\boldsymbol{\mu}_j^{(t)} = (1-\beta)\boldsymbol{\mu}_j^{(t-1)} + \beta \boldsymbol{\mu}_j \qquad \mathbf{v}_j^{(t)} = (1-\beta)\mathbf{v}_j^{(t-1)} + \beta \mathbf{v}_j \qquad \text{for } j = 1, \dots, D$$

where β is the momentum (usually $\beta = 0.1$).

Remark. All training steps of batch normalization are differentiable and can be integrated into gradient descent. If normalization is done outside gradient descent, the optimization process might undo it.

Remark. For convolutional layers, mean and variance are computed along the spatial dimension (i.e. pixels in the same output channel are normalized in the same way).

Inference During inference, the final running averages of mean $\boldsymbol{\mu}$ and variance $\mathbf{v}$ are used to normalize the activations (i.e. they are considered constants). Given the learned parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, an activation is normalized as follows:

$$\begin{aligned}\mathbf{s}_j^{(i)} &= \boldsymbol{\gamma}_j \frac{\mathbf{a}_j^{(i)} - \boldsymbol{\mu}_j}{\sqrt{\mathbf{v}_j + \varepsilon}} + \boldsymbol{\beta}_j \\ &= \left(\frac{\boldsymbol{\gamma}_j}{\sqrt{\mathbf{v}_j + \varepsilon}}\right) \mathbf{a}_j^{(i)} + \left(\boldsymbol{\beta}_j - \frac{\boldsymbol{\gamma}_j \boldsymbol{\mu}_j}{\sqrt{\mathbf{v}_j + \varepsilon}}\right)\end{aligned} \qquad \text{for } j = 1, \dots, D$$

Remark. Normalization during inference can be seen as a linear transformation. Therefore, it can be merged with the previous layer.

Properties The advantages of batch normalization are:

- It allows to use a higher learning rate and makes initialization less important.
- Training becomes non-deterministic (i.e. adds noise) acting as some form of regularization.
- During inference, there is no overhead as it can be merged with the previous layer.

The disadvantages are:

- It is not clear why it works.
- Training and inference work differently.
- It does not scale with batches that are too small.

Remark (Internal covariate shift)**.** A possible motivation for batch normalization is that each layer of a neural network expects an input distribution that changes at each training iteration. On the other hand, the distribution of the input itself depends on the previous layer and it also changes at each iteration. Therefore, each layer is disrupted by the update of the previous one. Batch normalization aims to minimize this by maintaining a fixed distribution.

3 Successful architectures

3.1 Preliminaries

Stem layer First convolutional layer(s) of a CNN that aims to reduce the spatial size of the activations for memory and computational purposes but also to rapidly increase the receptive field.

Stem layer

Model parallelism Train a model by splitting its weights on multiple computational units, each receiving the same data.

Model parallelism

Data parallelism Train a model by distributing the data on multiple computational units, each with a copy of the weights of the model.

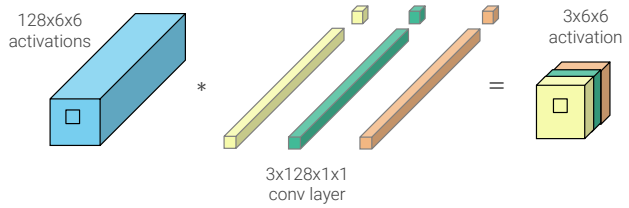
Data parallelism

Overlapping pooling Pooling layer with kernel size and stride chosen in such a way that some pixels at a step have also been considered at the previous one (e.g. 3×3 kernel with stride 2).

Overlapping pooling

1×1 **convolution** Convolution used to change the depth of the activation while maintaining its spatial dimension. It can be seen as a linear fully connected layer at each spatial dimension.

1×1 convolution



Remark. Stacking multiple 1×1 convolutions is equivalent to a multi-layer perceptron (i.e. universal function approximator).

Parameters computation

Parameters computation

Input layer Given an input image of shape $W_{\text{in}} \times H_{\text{in}} \times C_{\text{in}}$, the number of activations in the input layer is given by:

$$\texttt{\#activations} = W_{\text{in}} \cdot H_{\text{in}} \cdot C_{\text{in}}$$

Convolutional layer Given:

- An input of shape $W_{\text{in}} \times H_{\text{in}} \times C_{\text{in}}$,
- A kernel of shape $W_{\text{K}} \times H_{\text{K}}$ with padding P and stride S,
- A desired number of output channels C_{out},

the number of parameters of a convolutional layer (see Section 2.6.2) is given by:

$$\texttt{\#params} = \big((W_{\text{K}} \cdot H_{\text{K}} \cdot C_{\text{in}}) + 1\big) \cdot C_{\text{out}}$$

The output shape (see Section 2.6.2) and the number of activations is given by:

$$\texttt{activ\_w} = \left\lfloor \frac{W_{\text{in}} - W_{\text{K}} + 2P}{S} \right\rfloor + 1 \qquad \texttt{activ\_h} = \left\lfloor \frac{H_{\text{in}} - H_{\text{K}} + 2P}{S} \right\rfloor + 1$$

$$\texttt{\#activations} = \texttt{activ\_w} \cdot \texttt{activ\_h} \cdot C_{\text{out}}$$

The number of FLOPs for a single example of the batch is given by:

$$\texttt{FLOPs} = \texttt{\#activations} \cdot (W_{\text{K}} \cdot H_{\text{K}} \cdot C_{\text{in}}) \cdot 2$$

Pooling layer Given:

- An input of shape $W_{\text{in}} \times H_{\text{in}} \times C_{\text{in}}$,
- A kernel of shape $W_{\text{K}} \times H_{\text{K}}$ with padding P and stride S,

the number of activations in a pooling layer is computed as above with $C_{\text{in}} = C_{\text{out}}$:

$$\texttt{\#activations} = \texttt{activ\_w} \cdot \texttt{activ\_h} \cdot C_{\text{in}}$$

The number of FLOPs for a single example of the batch is given by:

$$\texttt{FLOPs} = \texttt{\#activations} \cdot (W_{\text{K}} \cdot H_{\text{K}})$$

Fully-connected layer Given:

- An activation of shape C_{in},
- The number of neurons $C_{\text{out}} = \texttt{\#activations}$,

the number of parameters of a fully-connected layer is:

$$\texttt{\#params} = (C_{\text{in}} \cdot C_{\text{out}}) + C_{\text{out}}$$

The number of FLOPs for a single example of the batch is given by:

$$\texttt{FLOPs} = 2 \cdot C_{\text{in}} \cdot C_{\text{out}}$$

Memory usage Given:

- The batch size B,
- The activation size `#activations`,
- The number of parameters `#params`,

to estimate the memory consumption, the following have to be taken into account:

- To compute the gradient of the loss, every intermediate activation for every example in the batch has to be stored.
- For every parameter, we have to store its value and the gradient of the loss w.r.t. it.
- Optimizers with momentum need to store more values per parameter.

It is therefore hard to estimate memory requirements. A rule of thumb estimates a lower bound as twice the activation size and 3-4 times the number of parameters. Assuming that `float32` (4 bytes) are used, memory consumption is computed as:

$$\texttt{memory\_activ\_bytes} = 2 \cdot (B \cdot \texttt{\#activations} \cdot 4)$$

$$\texttt{memory\_params\_bytes} = 3 \cdot (\texttt{\#params} \cdot 4)$$

3.2 LeNet-5

LeNet-5 is one of the first convolutional neural networks. The network has the following properties: LeNet-5

- At each layer, the number of channels increases and the spatial dimension decreases.
- Convolutions have the following characteristics: 5×5 kernels, no padding and average pooling for down-sampling.
- `Sigmoid` and `tanh` activation functions are used, with carefully selected amplitudes (i.e. they are scaled).
- The last layers are fully connected with a radial basis function as the output activation.
- There are no residual connections and normalization layers.

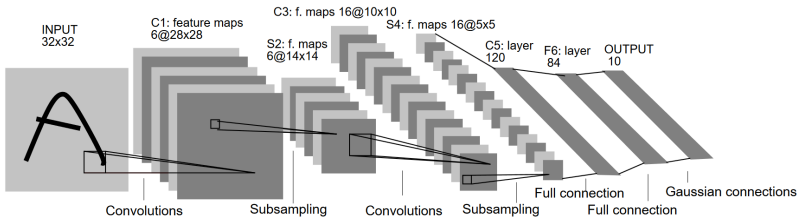


Figure 3.1: LeNet-5 architecture

3.3 AlexNet

AlexNet is the first CNN that broke the stagnation of the field. AlexNet

3.3.1 Architecture

AlexNet is composed of:

- 5 convolutional layers (convolution + `ReLU`, sometimes with max-pooling).
- 3 feed-forward layers.

Remark. Some layers are normalized using local response normalization (more active neurons are enhanced and the others are inhibited).

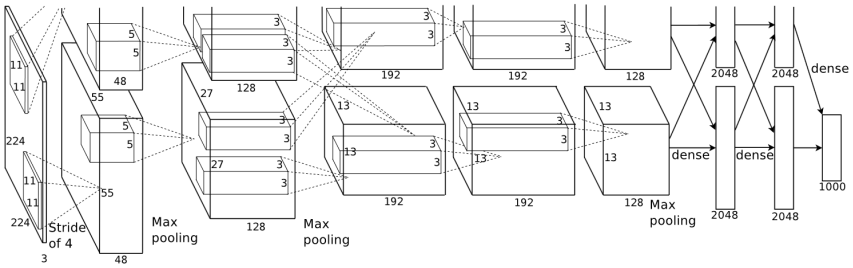


Figure 3.2: AlexNet architecture

3.3.2 Training

The network was trained on ImageNet 1k (for the ILSVRC 2012) with a batch size of 128. Due to GPU memory limitations, training was split into two parallel lines on two GPUs (model parallelism).

Grouped convolution

Grouped convolution A convolution is split into two sets of weights, each trained independently on a computational line. At some steps (e.g. `conv3`), the two GPUs are allowed to communicate. In this case, the result of the operation can be virtually seen as if it was done by a single computational unit (i.e. the operation is done on the full set of weights).

Remark. At the time, training took 5-6 days on two NVIDIA GTX 580.

3.3.3 Properties

AlexNet has the following trends:

- The first convolutional layer is a stem layer.
- The majority of the parameters are in the fully connected layers (even though they have to process an activation of 4096 elements).
- The largest memory consumption for activations is at the first convolutional layer.
- The largest amount of FLOPs is required by the convolutional layers.
- The total number of parameters and activations at training time are relatively comparable.
- 2.2 GFLOPs are required to process an image at training time.

Table 3.1: Parameters of AlexNet (batch size of 128)

	Convolution				Single activation			Batch requirements		Parameters	
Layer	**Channels**	**H/W**	**Stride**	**Padding**	**H/W**	**Channels**	**#activations**	**MFLOPs**	**Activ. mem.**	**Amount**	**Memory**
`input`	–	–	–	–	227	3	154 587	–	75.5 MB	0	0.0
`conv1`	96	11	4	0	55	96	290 400	26 986.3	283.6 MB	35 K	0.4 MB
`pool1`	1	3	2	0	27	96	69 984	80.6	68.3 MB	0	0.0
`conv2`	256	5	1	2	27	256	186 624	114 661.8	182.3 MB	615 K	7.0 MB
`pool2`	1	3	2	0	13	256	43 264	49.8	42.3 MB	0	0.0
`conv3`	384	3	1	1	13	384	64 896	38 277.2	63.4 MB	885 K	10.1 MB
`conv4`	384	3	1	1	13	384	64 896	57 415.8	63.4 MB	1327 K	15.2 MB
`conv5`	256	3	1	1	13	256	43 264	38 277.2	42.3 MB	885 K	10.1 MB
`pool3`	1	3	2	0	6	256	9216	10.6	9.0 MB	0	0.0
`flatten`	0	0	0	0	1	9216	9216	0.0	0.0	0	0.0
`fc6`	4096	1	1	0	1	4096	4096	9663.7	4.0 MB	37 758 K	432.0 MB
`fc7`	4096	1	1	0	1	4096	4096	4295.0	4.0 MB	16 781 K	192.0 MB
`fc8`	1000	1	1	0	1	1000	1000	1048.6	1.0 MB	4097 K	46.9 MB
							Total	290 851	1406 MB	62 378 K	714 MB

3.4 ZFNet/Clarifai

ZFNet/Clarifai

The aggressive stem layer of AlexNet causes dead neurons that do not specialize in recognizing anything.
Ablation and visualization studies found out that the first stem layer works better if split into two layers respectively with a 7×7 and 5×5 kernel, both with stride 2.

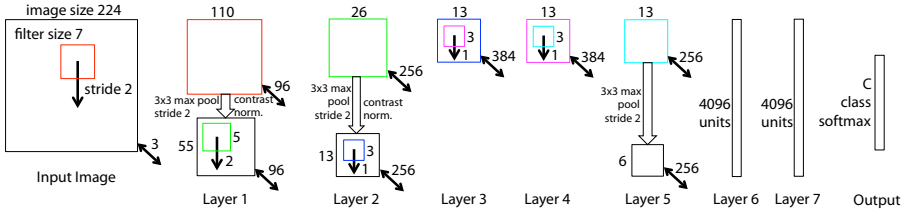


Figure 3.3: ZFNet architecture

(a) AlexNet (b) ZFNet

Figure 3.4: First layer activations comparison

3.5 VGG

3.5.1 Architecture

VGG

Network with a higher depth and smaller components. The authors constrained the layers to:

- 3×3 convolutions with stride 1 and padding 1.
- 2×2 max-pooling with stride 2 and padding 0.
- Number of channels that doubles after each pool.

Remark. It has been found out that deeper networks work better.

Stage Fixed combination of layers that process inputs of the same spatial resolution.

Stage

VGG stages are:

- `conv` $\mapsto$ `conv` $\mapsto$ `pool`.
- `conv` $\mapsto$ `conv` $\mapsto$ `conv` $\mapsto$ `pool`.
- `conv` $\mapsto$ `conv` $\mapsto$ `conv` $\mapsto$ `conv` $\mapsto$ `pool`.

Remark. One stage has the same receptive field of a single larger convolution but has fewer parameters, requires less computation and adds more non-linearity.
On the other hand, two activations are computed and both need to be stored for backpropagation.

Example.

Convolutional layer	Parameters	FLOPs	Activations
$C \times C \times 5 \times 5$, $S = 1$, $P = 2$	$25C^2 + C$	$50C^2 \cdot W_{\text{in}} \cdot H_{\text{in}}$	$C \cdot W_{\text{in}} \cdot H_{\text{in}}$
Two stacked $C \times C \times 3 \times 3$, $S = 1$, $P = 1$	$18C^2 + 2C$	$36C^2 \cdot W_{\text{in}} \cdot H_{\text{in}}$	$2 \cdot C \cdot W_{\text{in}} \cdot H_{\text{in}}$

Remark. Local response normalization was experimented with and dropped. As batch normalization had not been invented yet, weights at deeper layers were initialized from shallower architectures.

Table 3.2: Architecture of various versions of VGG

A (11 weight layers)	**B** (11 weight layers)	**C** (13 weight layers)	**D** (16 weight layers)	**VGG-16** (16 weight layers)	**VGG-19** (19 weight layers)
Input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
max-pool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
max-pool					
conv3-256 conv3-256	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
max-pool					
conv3-512 conv3-512	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
max-pool					
conv3-512 conv3-512	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
max-pool					
FC-4096 FC-4096 FC-1000 `softmax`					

3.5.2 Properties

VGG-16 has the following trends:

- Most of the parameters are concentrated at the fully connected layers.
- Most of the computation is required by the convolutions.
- Most of the memory is required to store the activations as there are no stem layers.
- Training was done on 4 GPUs with data parallelism for 2-3 weeks.

Table 3.3: Parameters of VGG-16 (batch size of 128)

Layer	Convolution				Single activation			Batch requirements		Parameters	
	Channels	H/W	Stride	Padding	H/W	Channels	#activations	MFLOPs	Activ. mem.	Amount	Memory
input	–	–	–	–	224	3	150 528	–	73.5 MB	0	0.0
conv1	64	3	1	1	224	64	3 211 264	22 196.3	3136.0 MB	2 K	0.0
conv2	64	3	1	1	224	64	3 211 264	473 520.1	3136.0 MB	37 K	0.4 MB
pool1	1	2	2	0	112	64	802 816	411.0	784.0 MB	0	0.0
conv3	128	3	1	1	112	128	1 605 632	236 760.1	1568.0 MB	74 K	0.8 MB
conv4	128	3	1	1	112	128	1 605 632	473 520.1	1568.0 MB	148 K	1.7 MB
pool2	1	2	2	0	56	128	401 408	205.5	392.0 MB	0	0.0
conv5	256	3	1	1	56	256	802 816	236 760.1	784.0 MB	295 K	3.4 MB
conv6	256	3	1	1	56	256	802 816	473 520.1	784.0 MB	590 K	6.8 MB
conv7	256	3	1	1	56	256	802 816	473 520.1	784.0 MB	590 K	6.8 MB
pool3	1	2	2	0	28	256	200 704	102.8	196.0 MB	0	0.0
conv8	512	3	1	1	28	512	401 408	236 760.1	392.0 MB	1180 K	13.5 MB
conv9	512	3	1	1	28	512	401 408	473 520.1	392.0 MB	2360 K	27.0 MB
conv10	512	3	1	1	28	512	401 408	473 520.1	392.0 MB	2360 K	27.0 MB
pool4	1	2	2	0	14	512	100 352	51.4	98.0 MB	0	0.0
conv11	512	3	1	1	14	512	100 352	118 380.0	98.0 MB	2360 K	27.0 MB
conv12	512	3	1	1	14	512	100 352	118 380.0	98.0 MB	2360 K	27.0 MB
conv13	512	3	1	1	14	512	100 352	118 380.0	98.0 MB	2360 K	27.0 MB
pool5	1	2	2	0	7	512	25 088	12.8	24.5 MB	0	0.0
flatten	1	1	1	0	1	25 088	25 088	0.0	0.0	0	0.0
fc14	4096	1	1	0	1	4096	4096	26 306.7	4.0 MB	102 786 K	1176.3 MB
fc15	4096	1	1	0	1	4096	4096	4295.0	4.0 MB	16 781 K	192.0 MB
fc16	1000	1	1	0	1	1000	1000	1048.6	1.0 MB	4100 K	46.9 MB
							Total	3 961 171	14 733 MB	138 382 K	1584 MB

3.6 Inception-v1 (GoogLeNet)

Network that aims to optimize computing resources.

Inception-v1 (GoogLeNet)

3.6.1 Architecture

Stem layers Down-sample the image from a shape of 224 to 28. As in ZFNet, multiple layers are used (5) and the largest convolution is of shape 7×7 with stride 2.

Inception module Main component of Inception-v1 that computes multiple convolutions on the input.

Inception module

Naive approach Given the input, the output is the concatenation of:

- A 5×5 convolution with stride 1 and padding 2.
- A 3×3 convolution with stride 1 and padding 1.
- A 1×1 convolution with stride 1 and padding 0.
- A 3×3 max-pooling with stride 1 and padding 1.

By using this approach, two problems arise:

- The max-pooling layer outputs a large number of channels (same as input).

- The convolutions are computationally expensive due to the large number of input channels.

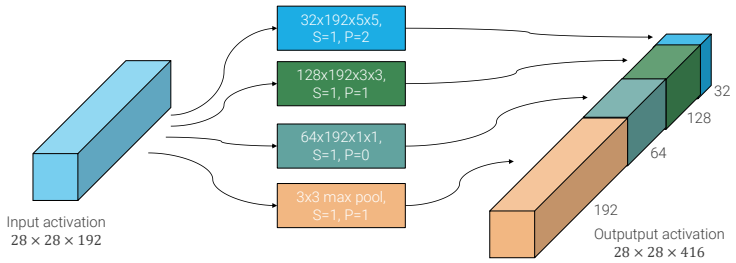


Figure 3.5: Naive inception module on the output of the stem layers

Actual approach Same as the naive approach, but max-pooling, 5×5 and 3×3 convolutions are preceded by 1×1 convolutions.

> **Remark.** For max-pooling, the 1×1 convolution can indifferently be placed before or after.

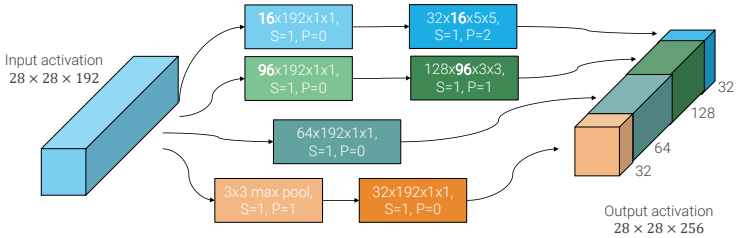


Figure 3.6: Actual inception module on the output of the stem layers

> **Remark.** The multiple convolutions of an inception module can be seen as decision components.

Auxiliary `softmax` Intermediate `softmax`s are used to ensure that hidden features are good enough. They also act as regularizers.

During inference, they are discarded.

Global average pooling classifier Instead of flattening between the convolutional and fully connected layers, global average pooling is used to reduce the number of parameters.

Global average pooling classifier

> **Remark.** If the kernel size of the pooling layer is computed by the layer itself (e.g. `AdaptiveAvgPool2d`), the network will be able to process inputs of any size (but this does not guarantee the quality of classification for all the image shapes).

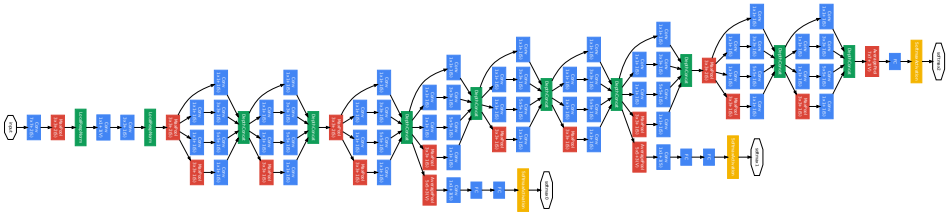


Figure 3.7: Architecture of Inception-v1

3.6.2 Properties

- The fully connected layer has a relatively small amount of parameters and a negligible number of FLOPs.
- Metrics were measured using test-time augmentation (the input image is split into random small chunks and each is processed by the network separately. The final result is the average of the results as in ensemble models). Strictly speaking, this makes results difficult to compare to other models that only do a single pass.

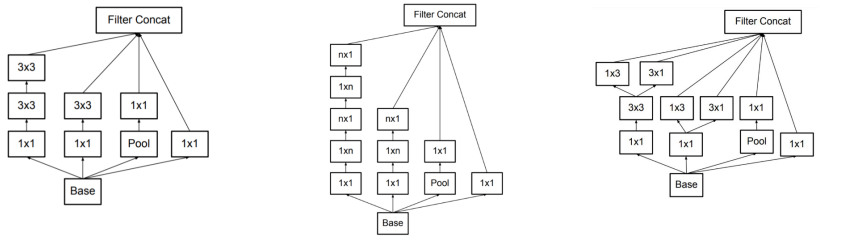
Table 3.4: Parameters of Inception-v1 (batch size of 128)

	Incep. 1 × 1 Other conv.				Incep. 3 × 3			Incep. 5 × 5			Max-pool		Single activ.			Batch requir.		Parameters	
Layer	**Channels**	**H/W**	**Stride**	**Padding**	**Channels**	**1 × 1 ch.s**	**H/W**	**Channels**	**1 × 1 ch.s**	**H/W**	**1 × 1 ch.s**	**H/W**	**H/W**	**Channels**	**#activations**	**MFLOPs**	**Activ. mem.**	**Amount**	**Memory**
input	–	–	–	–	–	–	–	–	–	–	–	–	224	3	150 528	–	73.5 MB	0	0.0
conv1	64	7	2	3	–	–	–	–	–	–	–	–	112	64	802 816	30 211.6	784.0 MB	9 K	0.1 MB
pool1	1	3	2	1	–	–	–	–	–	–	–	–	56	64	200 704	231.2	196.0 MB	0	0.0
conv2	64	1	1	0	–	–	–	–	–	–	–	–	56	64	200 704	3288.3	196.0 MB	4 K	0.0
conv3	192	3	1	1	–	–	–	–	–	–	–	–	56	192	602 112	88 785.0	588.0 MB	111 K	1.3 MB
pool2	1	3	2	1	–	–	–	–	–	–	–	–	28	192	150 528	173.4	147.0 MB	0	0.0
incep1	64	1	1	0	128	96	3	32	16	5	32	3	28	256	200 704	31 380.5	196.0 MB	163 K	1.9 MB
incep2	128	1	1	0	192	128	3	96	32	5	64	3	28	480	376 320	75 683.1	367.5 MB	388 K	4.4 MB
pool3	1	3	2	1	–	–	–	–	–	–	–	–	14	480	94 080	108.4	91.9 MB	0	0.0
incep3	192	1	1	0	208	96	3	48	16	5	64	3	14	512	100 352	17 403.4	98.0 MB	376 K	4.3 MB
incep4	160	1	1	0	224	112	3	64	24	5	64	3	14	512	100 352	20 577.8	98.0 MB	449 K	5.1 MB
incep5	128	1	1	0	256	128	3	64	24	5	64	3	14	512	100 352	23 609.2	98.0 MB	509 K	5.8 MB
incep5	112	1	1	0	288	144	3	64	32	5	64	3	14	528	103 488	28 233.4	101.1 MB	605 K	6.9 MB
incep6	256	1	1	0	320	160	3	128	32	5	128	3	14	832	163 072	41 445.4	159.3 MB	867 K	9.9 MB
pool4	1	3	2	1	–	–	–	–	–	–	–	–	7	832	40 768	47.0	39.8 MB	0	0.0
incep7	256	1	1	0	320	160	3	128	32	5	128	3	7	832	40 768	11 860.0	39.8 MB	1042 K	11.9 MB
incep8	384	1	1	0	384	192	3	128	48	5	128	3	7	1024	50 176	16 689.7	49.0 MB	1443 K	16.5 MB
avgpool	1	1	1	0	–	–	–	–	–	–	–	–	1	1024	1024	6.4	1.0 MB	0	0.0
fc1	1000	1	1	0	–	–	–	–	–	–	–	–	1	1000	1000	262.1	1.0 MB	1025 K	11.7 MB
															Total	389 996	3251 MB	6992 K	80 MB

3.6.3 Inception-v3

Uses convolution factorization to improve computational efficiency, reduce the number of parameters and make training more disentangled and easier. Different modules are used depending on the activation shape.

Inception-v3



Factorization A, used for fine-scale activations (35x35)

Factorization B, used for mid-scale activations (17x17)

Factorization C, used for coarse-scale activations (8x8)

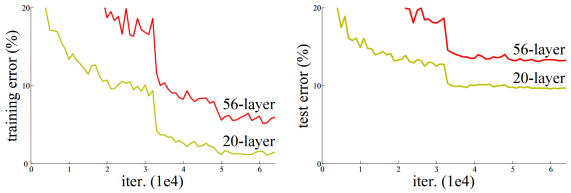
Figure 3.8: Inception-v3 modules

3.6.4 Inception-v4

A larger version of Inception-v3 with more complicated stem layers. Inception-v4

3.7 Residual networks

Remark. Training a very deep network from scratch might result in a solution that performs worse than a shallower layer. One could expect that the network simply overfits, but in reality, it underfits as gradient descent is not able to find a solution.

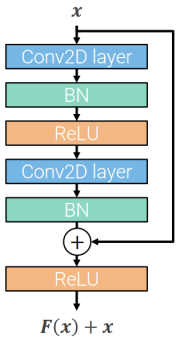


Remark. Technically, a deep neural network that has similar performance to a small network $\mathcal{N}$ can always be constructed. It is sufficient to take $\mathcal{N}$ as the starting network and add an arbitrary amount of identity layers.

Standard residual block Block that allows to easily learn the identity function through a skip connection. The output of a residual block with input x and a series of convolutional layers F is: Standard residual block

$$F(x;\boldsymbol{\theta}) + x$$

Skip connection Connection that skips a certain number of layers (e.g. 2 convolutional blocks). Skip connection



Remark. Training starts with small weights so that the network starts as the identity function. Updates can be seen as perturbations of the identity function.

Remark. Batch normalization is heavily used.

Remark. Skip connections are applied before the activation function (ReLU) as otherwise it would be summed to all positive values making the perturbation of the identity function less effective.

3.7.1 ResNet

VGG-inspired network with residual blocks. It has the following properties: ResNet-18

- A stage is composed of residual blocks.
- A residual block is composed of two 3×3 convolutions followed by batch normalization.
- The first residual block of each stage halves the spatial dimension and doubles the number of channels (there is no pooling).
- Stem layers are less aggressive than GoogLeNet (`conv + pool`. Input reduced to a shape of 56×56).

- Global average pooling is used instead of flattening.

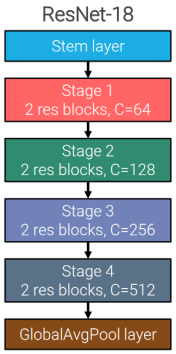


Figure 3.9: Architecture of ResNet-18

Skip connection reshape Due to the shape mismatch, the output of a stage cannot be directly used as the skip connection of the next stage. Possible solutions are:

- Apply stride 2 and zero-pad the missing channels of the skip connection (this does not add new parameters).
- The output of the previous stage is passed through a 1×1 convolution with stride 2 and $2C$ output channels (shown to work slightly better).

Bottleneck residual network Variant of residual blocks that uses more layers with approximately the same number of parameters and FLOPs of the standard residual block. Instead of using two 3×3 convolutions, bottleneck residual network has the following structure:

Bottleneck residual network

- 1×1 convolution to compress the channels of the input by an order of 4 (and the spatial dimension by 2 if it is the first block of a stage, as in normal ResNet).
- 3×3 convolution.
- 1×1 convolution to match the shape of the skip connection.

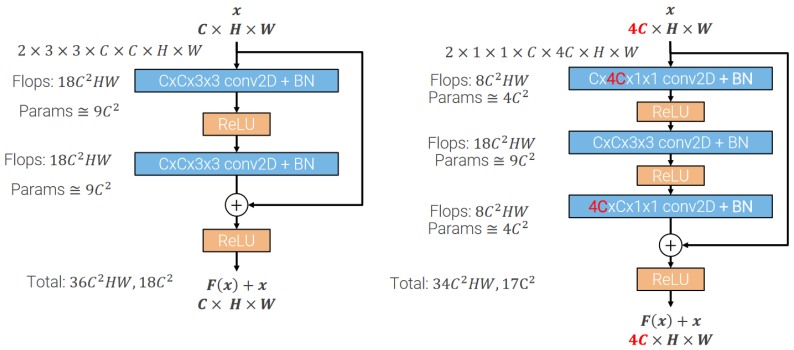


Figure 3.10: Standard residual block (left) and bottleneck block (right)

Remark. ResNet improves the results of a deeper network but, beyond a certain depth, the gain is negligible.

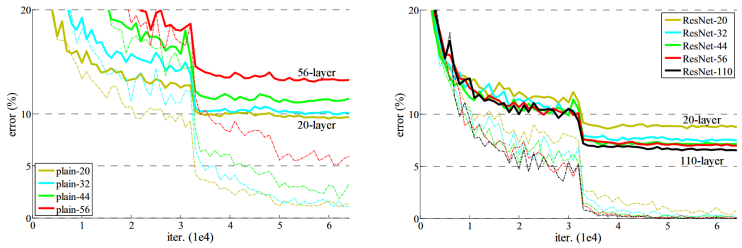
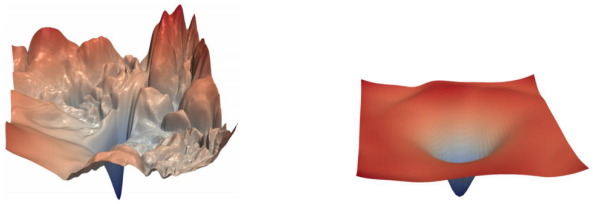


Table 3.5: Variants of ResNet

ResNet-18	**ResNet-34**	**ResNet-50**	**ResNet-101**	**ResNet-152**
Stem layers				
2 residual blocks ($C = 64$)	3 residual blocks ($C = 64$)	3 bottleneck blocks ($C = 256$)	3 bottleneck blocks ($C = 256$)	3 bottleneck blocks ($C = 256$)
2 residual blocks ($C = 128$)	4 residual blocks ($C = 128$)	4 bottleneck blocks ($C = 512$)	4 bottleneck blocks ($C = 512$)	8 bottleneck blocks ($C = 512$)
2 residual blocks ($C = 256$)	6 residual blocks ($C = 256$)	6 bottleneck blocks ($C = 1024$)	23 bottleneck blocks ($C = 1024$)	36 bottleneck blocks ($C = 1024$)
2 residual blocks ($C = 512$)	3 residual blocks ($C = 512$)	3 bottleneck blocks ($C = 2048$)	3 bottleneck blocks ($C = 2048$)	3 bottleneck blocks ($C = 2048$)
Average pooling + Fully-connected				

Remark. Residual connection creates a smother loss surface.



(a) Without skip connections (b) With skip connections

Figure 3.11: Loss surface visualized through dimensionality reduction

Remark (ResNet as ensemble)**.** Skip connections can be seen as a way to create an ensemble-like network as it allows the network to ignore blocks by learning the identity function.

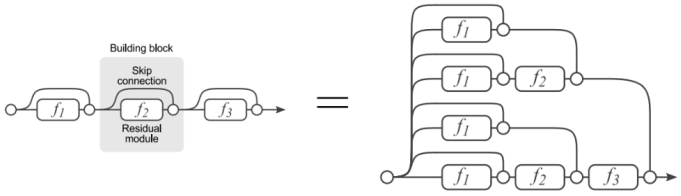
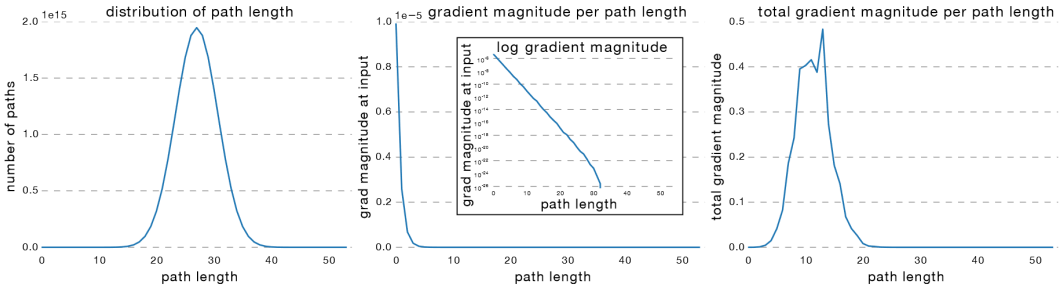


Figure 3.12: Possible paths of a residual network with three blocks

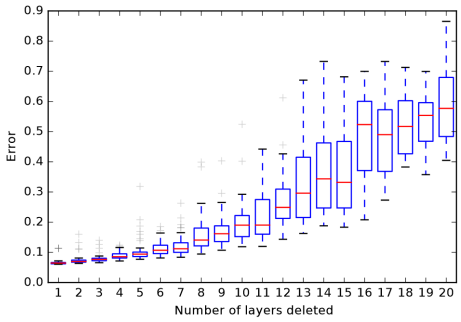
Studies show that, in ResNet-56, the gradient updates a subset of the blocks at the time. This is due to the fact that:

- The majority of the possible paths have a length of ~ 30.
- The gradient magnitude is significant at the first layers (i.e. in shorter paths).

By multiplying the values of the two points above, results show that the total gradient magnitude is significant only up until paths of length ~ 20.



Further experiments show that by randomly deleting layers of the network, the drop in performance, as expected in ensemble models, becomes significant only after a certain number of layers.



Therefore, skip connections do not directly solve the vanishing problem but go around it by creating an ensemble of smaller networks.

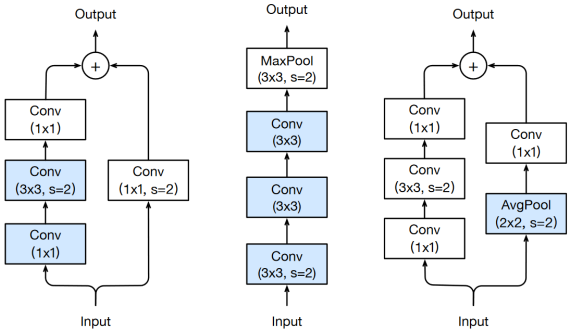
3.7.2 ResNet-v2

Several improvements over the original ResNet have been made:

ResNet-B As the first block of each stage is responsible for halving the spatial dimension, in a bottleneck block this is done by the first 1×1 convolution. This causes to lose $\frac{3}{4}$ of the input activations as a 1×1 convolution with stride ≥ 2 does not have spatial extent. To solve this issue, the halving of the input image is done by the second 3×3 convolution. ResNet-B

ResNet-C The 7×7 convolution in stem layers is replaced by a sequence of three 3×3 convolutions, the first one with stride 2. ResNet-C

ResNet-D Similarly to ResNet-B, the 1×1 convolution used to match the shape of the skip connection has a stride of 2 and causes a loss of activations. Therefore, stride is dropped and a 2×2 average pooling with stride 2 is added before the convolution. ResNet-D



(a) ResNet-B (b) ResNet-C (c) ResNet-D

3.7.3 Inception-ResNet-v4

Network with bottleneck-block-inspired inception modules.

Inception-ResNet-A Three 1×1 convolutions are used to compress the input channels. Each of them leads to a different path: Inception-ResNet-A

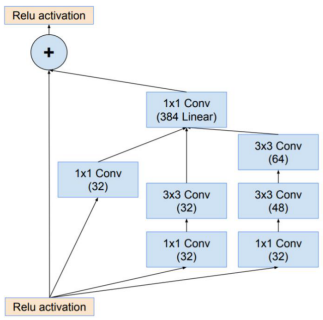
- Directly to the final concatenation.
- To a 3×3 convolution.
- To two 3×3 convolutions (i.e. a factorized 5×5 convolution).

The final concatenation is passed through a 1×1 convolution to match the skip connection shape.

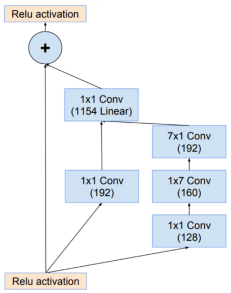
Inception-ResNet-B Three 1×1 convolutions are used to compress the input channels. Each of them leads to: Inception-ResNet-B

- Directly to the final concatenation.
- A 1×7 and 7×1 convolutions (i.e. a factorized 7×7 convolution).

The final concatenation is passed through a 1×1 convolution to match the skip connection shape.



Inception-resnet A, used for fine-scale activations (35x35)



Inception-resnet B, used for mid-scale activations (17x17)

3.8 Transfer learning

Adapt a pre-trained network on a new dataset. There are mainly two approaches: Transfer learning

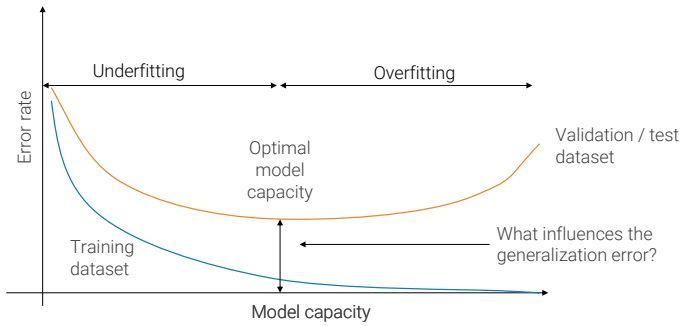
Frozen CNN The weights of the pre-trained CNN are kept frozen and a new trainable classification head is added at the end of the model. In this case, the pre-trained model only acts as a feature extractor.

Fine-tuning After transferring using the frozen CNN approach, the pre-trained model can be unfrozen (entirely or only the higher layers) and further trained together with the new classification head, usually for a few steps.

4 Training recipes

Model capacity Capability of a network to fit the train set. It depends on the architecture of the model. Model capacity

> **Remark.** By varying the architecture of a model, the resulting network might overfit or underfit.



Effective capacity Actual capacity of a model that depends on the training hyperparameters (e.g. learning rate, epochs, ...). Effective capacity

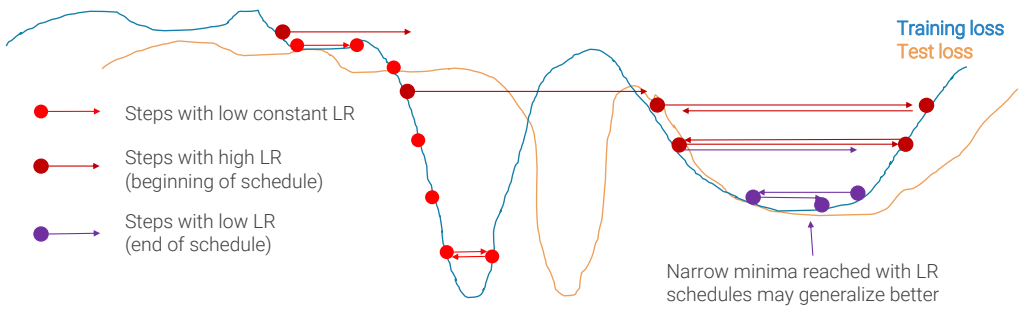
> **Remark.** In practice, a model with a large theoretical capacity is used and it is tuned on its effective capacity.

4.1 Learning rate schedule

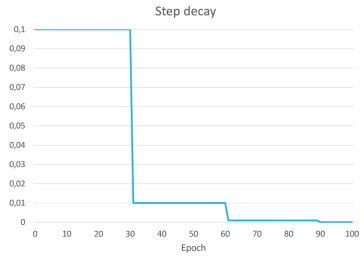
Mixture of high and low learning rates to find a good compromise between speed and accuracy. Learning rate schedule

> **Remark.** If the learning rate is too high, after the first iteration, updates might get stuck in a valley.

> **Remark.** Intuitively, learning rate schedulers allow to find wide minima (i.e. skip narrow minima and reach a basin with a minimum more "compatible" with the step size). Moreover, the optimal loss of the test set is usually a shifted and distorted version of the train loss. Therefore, a wider minimum results in a more robust model.



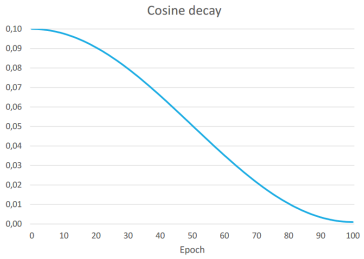
Step Start with a high learning rate and divide it by 10 when reaching a plateau.



Cosine Continuous decay of the learning rate that follows the cosine function. Given the number of training epochs E, the starting learning rate $\texttt{lr}_0$ and ending learning rate $\texttt{lr}_E$, the learning rate at epoch e is given by:

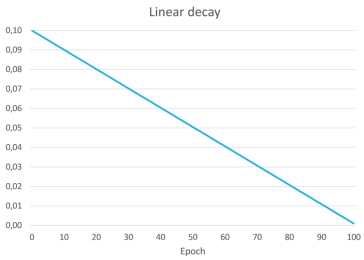
$$\texttt{lr}_e = \texttt{lr}_E + \frac{1}{2}(\texttt{lr}_0 - \texttt{lr}_E)\left(1 + \cos\left(\frac{e\pi}{E}\right)\right)$$

> **Remark.** Compared to the step scheduler, it only requires two hyperparameters (starting and ending learning rate).



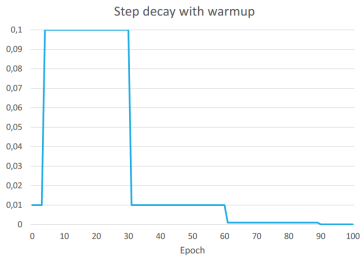
Linear Continuous decay of the learning rate that follows a linear function. Given the number of training epochs E, the starting learning rate $\texttt{lr}_0$ and ending learning rate $\texttt{lr}_E$, the learning rate at epoch e is given by:

$$\texttt{lr}_e = \texttt{lr}_E + (\texttt{lr}_0 - \texttt{lr}_E)\left(1 + \frac{e}{E}\right)$$



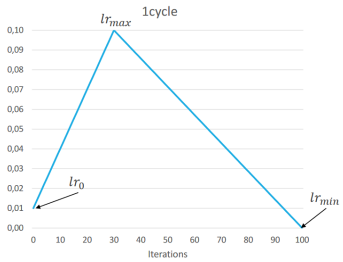
Warm-up Start with a small learning rate for a few steps (a few epochs or batch steps) before growing and progressively decaying.

> **Remark.** This is useful for large networks where poor initialization and high learning rates at the beginning might slow down convergence.



One cycle Scheduler defined on batch steps. It starts with a small learning rate that grows until a maximum is reached after which decay starts.
Given:

- The number of training iterations I,
- The starting learning rate $\texttt{lr}_0$,
- The peak learning rate $\texttt{lr}_{\max}$,
- The ending learning rate $\texttt{lr}_{\min}$,
- The percentage of iterations p for which the learning rate should increase,

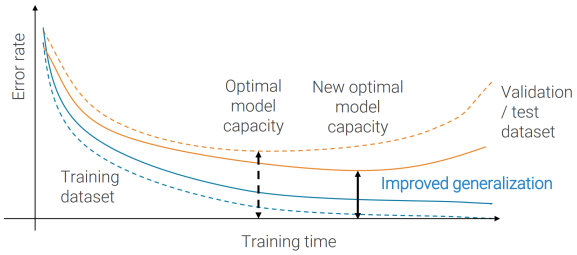


The learning rate at iteration i is given by:

$$\texttt{lr}_i = \begin{cases} \texttt{lr}_{\max} + (\texttt{lr}_0 - \texttt{lr}_{\max})\left(1 - \frac{i}{pI}\right) & \text{if } i < pI \\ \texttt{lr}_{\min} + (\texttt{lr}_0 - \texttt{lr}_{\min})\left(1 - \frac{i-pI}{I-pI}\right) & \text{if } i \geq pI \end{cases}$$

4.2 Regularization

Regularization Modifications to the network that aim to improve its generalization capacity without introducing overfitting. Regularization



4.2.1 Parameter norm penalties

Add a regularization term to the loss:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) = \mathcal{L}^{(\text{task})}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) + \lambda \mathcal{L}^{(\text{reg})}(\boldsymbol{\theta})$$

Remark. Intuitively, normalization forces parameters to be small, therefore, limiting the effective capacity of the model and improving generalization.

L1 regularization The regularization term is:

$$\mathcal{L}^{(\text{reg})}(\boldsymbol{\theta}) = \sum_i |\boldsymbol{\theta}|$$

L2 regularization The regularization term is:

$$\mathcal{L}^{(\text{reg})}(\boldsymbol{\theta}) = \sum_i \boldsymbol{\theta}^2 = \|\boldsymbol{\theta}\|_2^2$$

Remark. When using plain SGD, L2 regularization is also called weight decay. In fact, given the loss:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) = \mathcal{L}^{(\text{task})}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

The gradient update is:

$$\begin{aligned} \boldsymbol{\theta}^{(i+1)} &= \boldsymbol{\theta}^{(i)} - \texttt{lr}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(i)}; \mathcal{D}^{(\text{train})}) \\ &= \boldsymbol{\theta}^{(i)} - \texttt{lr}\nabla_{\boldsymbol{\theta}}\left[\mathcal{L}^{(\text{task})}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) + \frac{\lambda}{2}\|\boldsymbol{\theta}^{(i)}\|_2^2\right] \\ &= \boldsymbol{\theta}^{(i)} - \texttt{lr}\left[\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(\text{task})}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})}) + \lambda\boldsymbol{\theta}^{(i)}\right] \\ &= \underbrace{(1 - \texttt{lr}\lambda)\boldsymbol{\theta}^{(i)}}_{\text{Decayed parameter vector}} - \underbrace{\texttt{lr}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(\text{task})}(\boldsymbol{\theta}; \mathcal{D}^{(\text{train})})}_{\text{Standard gradient descent step}} \end{aligned}$$

Remark. The `weight_decay` parameter of more advanced optimizers (e.g. Adam) is not always the L2 regularization.
In PyTorch, `Adam` implements L2 regularization while `AdamW` uses another type of regularizer.

4.2.2 Early stopping

Early stopping

Monitor performance on the validation set and either:

- Select the checkpoint with the best validation set after a maximum number of epochs.
- Set a hyperparameter (patience) that stops training if, after a certain number of steps, validation performance does not improve.

Remark. If possible, the first option is preferable as validation metrics might improve after a few steps of stagnation or decrease.

4.2.3 Label smoothing

Label smoothing

When using cross-entropy with softmax, the model has to push the correct logit to $+\infty$ and the others to $-\infty$ so that the softmax outputs 1 for the correct label. This is unnecessary as it might cause overfitting.
Given C classes, labels can be smoothed by assuming a small uniform noise ε:

$$\mathbf{y}^{(i)} = \begin{cases} 1 - \frac{\varepsilon}{C}(C-1) & \text{if } i \text{ is the correct label} \\ \frac{\varepsilon}{C} & \text{otherwise} \end{cases}$$

4.2.4 Dropout

Dropout

Train time For each batch step, generate a mask that sets some activations (usually $10\% - 50\%$) to zero during the forward pass.

Remark. Intuitively, activations that are not dropped should learn to not rely on the other neurons and therefore avoiding focusing on selective information.

Remark. Dropout can be seen as an ensemble of models where each mask is a new model.

Test time With dropout, the output of the network is non-deterministic and methods to make inference more deterministic should be applied:

Naive approach A naive workaround is to sample some random masks and average the outputs of the network on these masks:

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{m}}[f(\mathbf{x};\boldsymbol{\theta},\boldsymbol{m})] = \sum_{\boldsymbol{m}} p(\boldsymbol{m}) f(\mathbf{x};\boldsymbol{\theta},\boldsymbol{m})$$

Although more accurate, this method is computationally expensive.

Weight scaling Without loss of generality, consider an activation a obtained as a linear combination of two neurons x_1 and x_2. During test time, the activation is:

$$a^{(\text{test})} = w_1 x_1 + w_2 x_2$$

Assuming a dropout with $p = 0.5$ (a neuron has a 50% probability of being dropped), the expected value of the activation $a^{(\text{train})}$ at train time is:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{m}}[a^{(\text{train})}] &= \frac{1}{4}(w_1x_1 + w_2x_2) + \frac{1}{4}(w_1x_1 + 0) + \frac{1}{4}(0 + w_2x_2) + \frac{1}{4}(0 + 0) \\ &= \frac{1}{2}(w_1x_1 + w_2x_2) = pa^{(\text{test})}\end{aligned}$$

Therefore, there is a p factor of discrepancy between $a^{(\text{train})}$ and $a^{(\text{test})}$ that might disrupt the distribution of the activations. Two approaches can be taken:

- Rescale the value at test time:

$$pa^{(\text{test})} = \mathbb{E}_{\boldsymbol{m}}\left[a^{(\text{train})}\right]$$

- Rescale the value at train time (inverted dropout):

$$a^{(\text{test})} = \mathbb{E}_{\boldsymbol{m}}\left[\frac{a^{(\text{train})}}{p}\right]$$

  This approach is preferred as it leaves test time unchanged.

Remark. Weight scaling is exact only for linear layers. Still, with a non-linear activation, this method is a fast approximation of the output of the network with dropout.

Remark. Dropout on convolutional layers are usually not necessary as they already have a strong inductive bias.

Remark. Dropout and batch normalization show a general pattern for regularization:

- At train time, some randomness is added.
- At test time, inference is done by averaging or approximating the output of the network.

4.2.5 Data augmentation

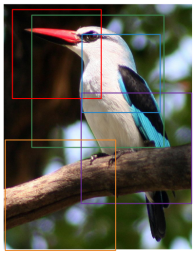
Increase the size of a dataset by manipulating (e.g. flipping) the existing examples. Data augmentation

Remark. More data always reduces variance and is therefore always a positive thing.

Remark. Transformations should be label-preserving (e.g. a 180° rotation should not be applied on a 6 or 9).

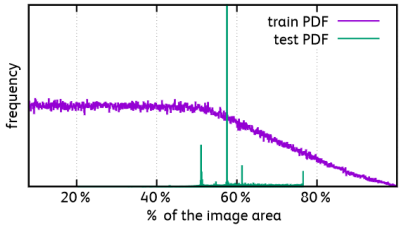
Multi-scale training Sample random crops and scales: Multi-scale training

1. Choose a random size S in range $[S_{\min}, S_{\max}]$.
2. Isotropically (i.e. preserve the aspect ratio) scale the training image so that the short side is of size S.
3. Sample random patches of a given size.



Remark. For S close to the patch size, crops will capture whole-image statistics. For S bigger than the patch size, crops will cover small portions of the image (with the risk of getting areas where the target content is not present).

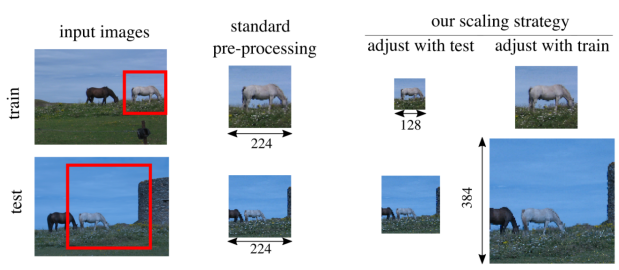
FixRes At test time, the image is usually presented with the target at the center (photographer bias) at a specific size. However, using data augmentation at train time, the size of the crops of the target varies following a distribution that is different from the one of the test set and usually involves smaller crops of the input image. FixRes

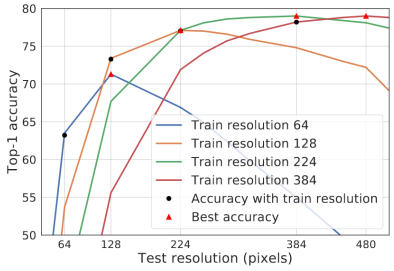


Therefore, there is a discrepancy between train and test images as, during training, the network sees objects that are bigger due to the rescaling of the input image.

It has been shown that a possible solution to close this discrepancy is to train the model using images with a lower resolution than the test set. The possible alternatives are:

- Reduce train-time resolution.
- Reduce test-time resolution.





<end of course>