Machine Learning for Computer Vision

Last update: 20 December 2024

Academic Year 2024 – 2025

Alma Mater Studiorum · University of Bologna

Contents

[1] Excerpt from IPCV2
[2] Excerpt from IPCV2

1 Optimizers

1.1 Stochastic gradient descent with mini-batches

SGD

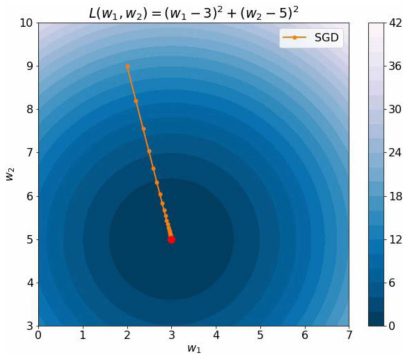
Stochastic gradient descent (SGD) Gradient descent based on a noisy approximation of the gradient computed on mini-batches of B data samples.

An epoch e of SGD with mini-batches of size B does the following:

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 $\boldsymbol{\theta}^{(e*U+u+1)} = \boldsymbol{\theta}^{(e*U+u)} - \texttt{lr} \cdot \nabla\mathcal{L}$.

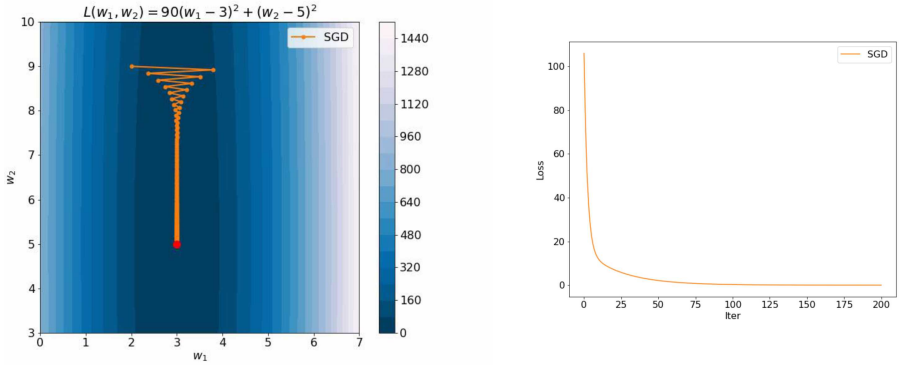
SGD spheres

Remark (Spheres)**.** GD/SGD works better on convex functions (e.g., paraboloids) as there are no preferred directions to reach a minimum. Moreover, faster convergence can be obtained by using a higher learning rate.

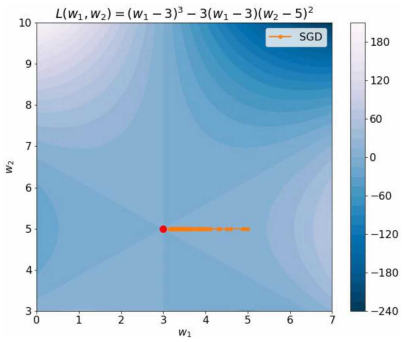


SGD canyons

Remark (Canyons)**.** A function has a canyon shape if it grows faster in some directions. The trajectory of SGD oscillates in a canyon (the steep area) and a smaller learning rate is required to reach convergence. Note that, even though there are oscillations, the loss alone decreases and is unable to show the oscillating behavior.



Remark (Local minima)**.** GD/SGD converges to a critical point. Therefore, it might end up in a saddle point or local minima. SGD local minima



1.2 Second-order methods

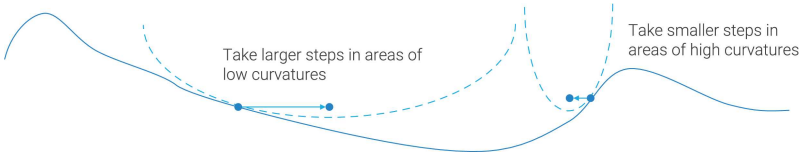
Methods that also consider the second-order derivatives when determining the step.

Newton's method Second-order method for the 1D case based on the Taylor expansion: Newton's method

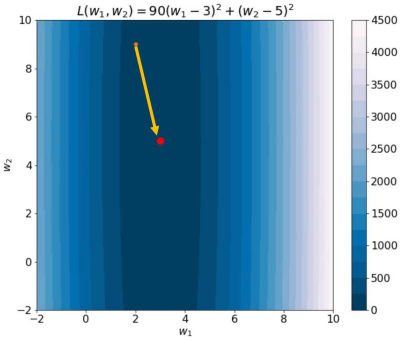
$$f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\Delta x + \frac{1}{2} f''(x_t)\Delta x^2$$

which can be seen as a paraboloid over the variable Δx.

Given a function f and a point x_t, the method fits a paraboloid at x_t with the same slope and curvature at $f(x_t)$. The update is determined as the step required to reach the minimum of the paraboloid from x_t. It can be shown that this step is $-\frac{f'(x_t)}{f''(x_t)}$.



Remark. For quadratic functions, second-order methods converge in one step.



General second-order method For a generic multivariate non-quadratic function, the update is: General second-order method

$$-\texttt{lr} \cdot \boldsymbol{H}_f^{-1}(\mathbf{x}_t)\nabla f(\mathbf{x}_t)$$

where $\boldsymbol{H}_f$ is the Hessian matrix.

Remark. Given k variables, $\boldsymbol{H}$ requires $O(k^2)$ memory. Moreover, inverting a matrix has time complexity $O(k^3)$. Therefore, in practice second-order methods are not applicable for large models.

1.3 Momentum

Standard momentum Add a velocity term $v^{(t)}$ to account for past gradient updates: Standard momentum

$$v^{(t+1)} = \mu v^{(t)} - \texttt{lr}\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + v^{(t+1)}$$

where $\mu \in [0, 1[$ is the momentum coefficient.

In other words, $v^{(t+1)}$ represents a weighted average of the update steps done up until time t.

Remark. Momentum helps to counteract a poor conditioning of the Hessian matrix when working with canyons.

Remark. Momentum helps to reduce the effect of variance of the approximated gradients (i.e., acts as a low-pass filter).

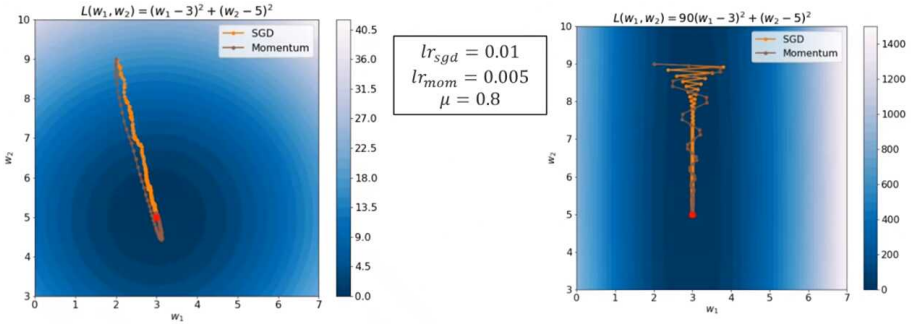


Figure 1.1: Plain SGD vs momentum SGD in a sphere and a canyon. In both cases, momentum converges before SGD.

Nesterov momentum Variation of the standard momentum that computes the gradient step considering the velocity term: Nesterov momentum

$$v^{(t+1)} = \mu v^{(t)} - \texttt{lr}\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)} + \mu v^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + v^{(t+1)}$$

Remark. The key idea is that, once $\mu v^{(t)}$ is summed to $\boldsymbol{\theta}^{(t)}$, the gradient computed at $\boldsymbol{\theta}^{(t)}$ is obsolete as it has been partially updated.

Remark. In practice, there are methods to formulate Nesterov momentum without the need of computing the gradient at $\boldsymbol{\theta}^{(t)} + \mu v^{(t)}$.

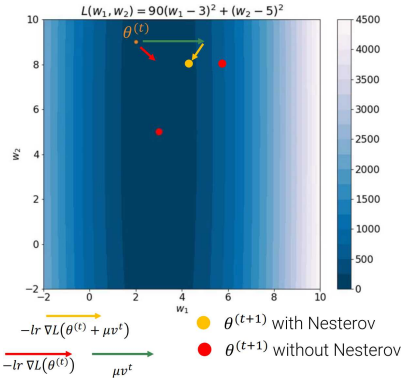


Figure 1.2: Visualization of the step in Nesterov momentum

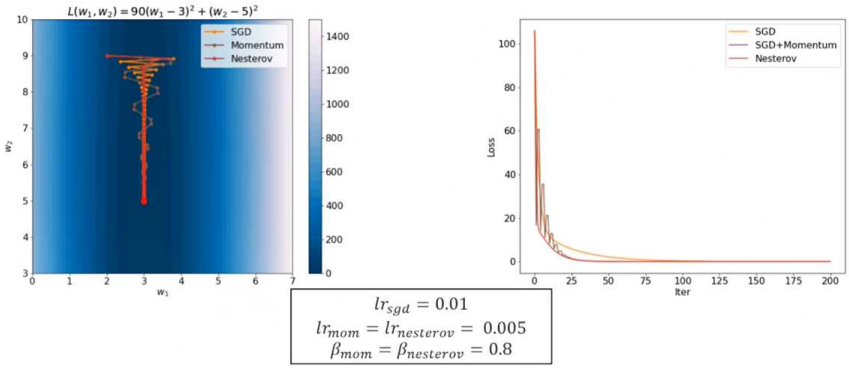


Figure 1.3: Plain SGD vs standard momentum vs Nesterov momentum

1.4 Adaptive learning rates methods

Adaptive learning rates Methods to define per-parameter adaptive learning rates.

Adaptive learning rates

Ideally, assuming that the changes in the curvature of the loss are axis-aligned (i.e., the parameters are independent), it is reasonable to obtain a faster convergence by:

- Reducing the learning rate along the dimension where the gradient is large.
- Increasing the learning rate along the dimension where the gradient is small.

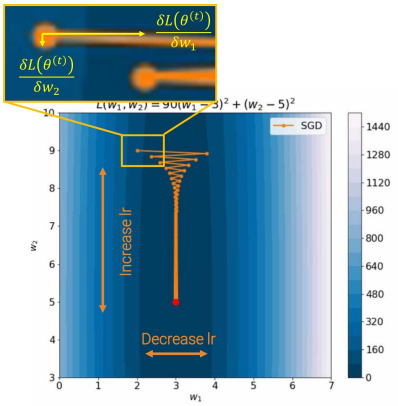


Figure 1.4: Loss where the w_1 parameter has a larger gradient, while w_2 has a smaller gradient

Remark. As the landscape of a high-dimensional loss cannot be seen, automatic methods to adjust the learning rates must be used.

1.4.1 AdaGrad

Adaptive gradient (AdaGrad) Each entry of the gradient is rescaled by the inverse of the history of its squared values: Adaptive gradient (AdaGrad)

$$\mathbb{R}^{N\times 1} \ni \mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} + \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\mathbb{R}^{N\times 1} \ni \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{\mathbf{s}^{(t+1)}} + \varepsilon} \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$

where:

- $\odot$ is the element-wise product.
- Division and square root are element-wise.
- ε is a small constant.

Remark. By how it is defined, $\mathbf{s}^{(t)}$ is monotonically increasing which might reduce the learning rate too early when the minimum is still far away.

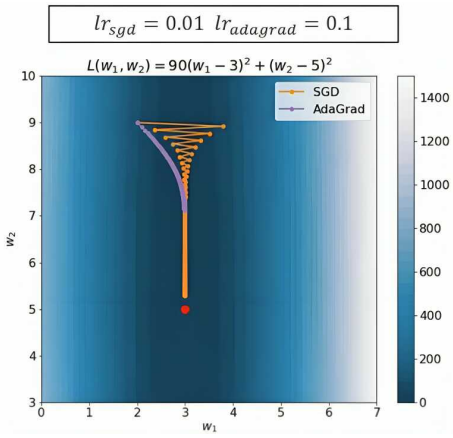


Figure 1.5: SGD vs AdaGrad. AdaGrad stops before getting close to the minimum.

1.4.2 RMSProp

RMSProp Modified version of AdaGrad that down-weighs the gradient history $s^{(t)}$: RMSProp

$$\mathbf{s}^{(t+1)} = \beta\mathbf{s}^{(t)} + (1-\beta)\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{\mathbf{s}^{(t+1)}} + \varepsilon} \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$

where $\beta \in [0, 1]$ (typically 0.9 or higher) makes $s^{(t)}$ an exponential moving average.

Remark. RMSProp is faster than SGD at the beginning before slowing down and reaching similar performances as SGD.

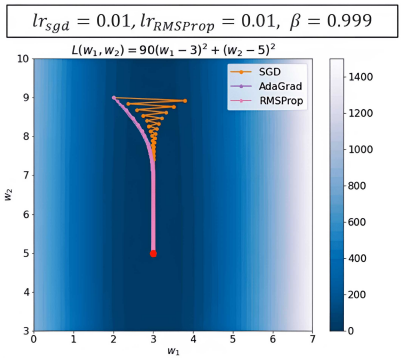


Figure 1.6: SGD vs AdaGrad vs RMSProp

1.4.3 Adam

Adaptive moments (Adam) Extends RMSProp by also considering a running average for the gradients:

Adaptive moments (Adam)

$$\mathbf{g}^{(t+1)} = \beta_1 \mathbf{g}^{(t)} + (1 - \beta_1)\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\mathbf{s}^{(t+1)} = \beta_2 \mathbf{s}^{(t)} + (1 - \beta_2)\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$

where $\beta_1, \beta_2 \in [0, 1]$ (typically $\beta_1 = 0.9$ and $\beta_2 = 0.999$).

Moreover, as $\mathbf{g}^{(0)} = 0$, $\mathbf{s}^{(0)} = 0$, and β_1, β_2 are typically large (i.e., past history weighs more), Adam starts by taking small steps (e.g., $\mathbf{g}^{(1)} = (1 - \beta_1)\nabla\mathcal{L}(\boldsymbol{\theta}^{(0)})$ is simply rescaling the gradient for no reason). To cope with this, a debiased formulation of $\mathbf{g}$ and $\mathbf{s}$ is used:

$$\mathbf{g}^{(t)}_{\text{debiased}} = \frac{\mathbf{g}^{(t+1)}}{1 - \beta_1^{t+1}} \qquad \mathbf{s}^{(t)}_{\text{debiased}} = \frac{\mathbf{s}^{(t+1)}}{1 - \beta_2^{t+1}}$$

where the denominators $(1 - \beta_i^{t+1}) \to 1$ for increasing values of t.

Finally, the update is defined as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{s^{(t)}_{\text{debiased}}} + \varepsilon} \odot g^{(t)}_{\text{debiased}}$$

Remark. It can be shown that $\frac{g^{(t)}_{\text{debiased}}}{\sqrt{s^{(t)}_{\text{debiased}}}}$ has a bounded domain, making it more controlled than RMSProp.

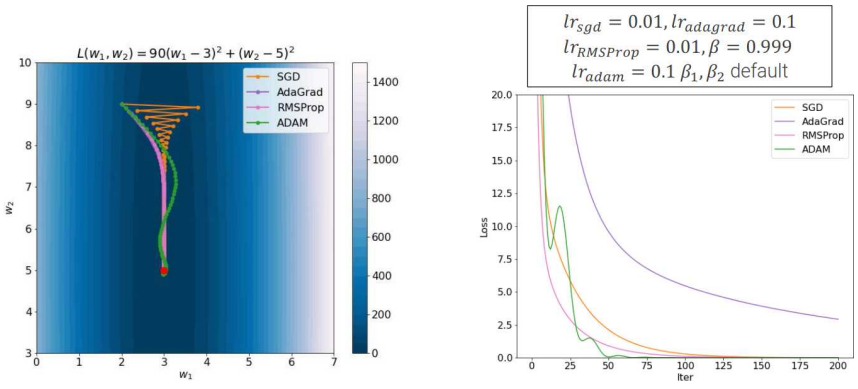


Figure 1.7: SGD vs AdaGrad vs RMSProp vs Adam

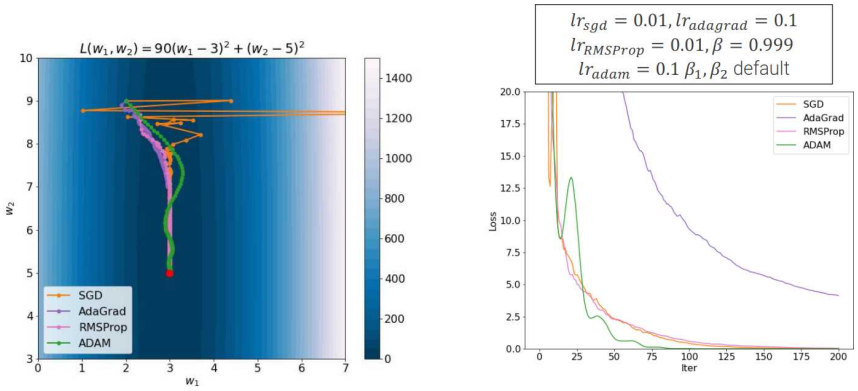
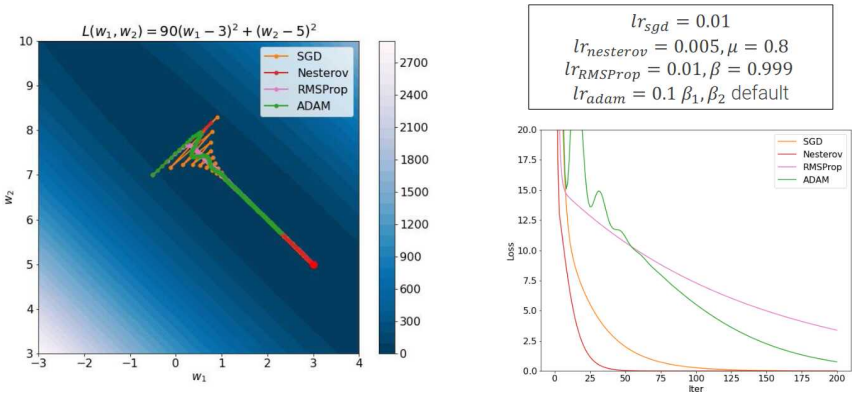


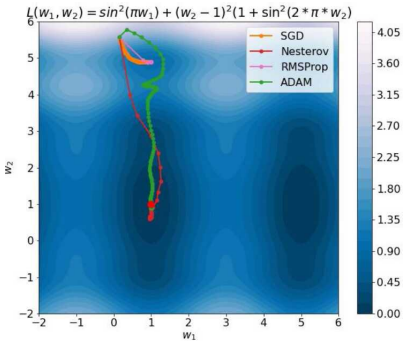
Figure 1.8: SGD vs AdaGrad vs RMSProp vs Adam with a smaller batch size

Remark. Adam is based on the assumption of unrelated parameters (i.e., axis-aligned). If this does not actually hold, it might be slower to converge.



Remark. Empirically, in computer vision Nesterov momentum (properly tuned) works better than Adam.

Remark. Momentum based approaches tend to prefer large basins. Intuitively, by accumulating momentum, it is possible to "escape" smaller ones.



1.4.4 AdamW

Adam with weight decay (AdamW) Modification on the gradient update of Adam to include weight decay:

Adam with weight decay (AdamW)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{s^{(t)}_{\text{debiased}}} + \varepsilon} \odot g^{(t)}_{\text{debiased}} - \lambda \boldsymbol{\theta}^{(t)}$$

Remark. Differently from SGD, L2 regularization on Adam is not equivalent to applying weight decay. In fact, by definition, the regularization term is applied to the gradient and not on the update step:

$$\nabla_{\text{actual}}\mathcal{L}(\boldsymbol{\theta}^{(t)}) = \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) + \lambda\boldsymbol{\theta}^{(t)}$$

where $\nabla_{\text{actual}}\mathcal{L}(\boldsymbol{\theta}^{(t)})$ is the actual gradient used to compute the running averages $\mathbf{g}$ and $\mathbf{s}$.

2 Architectures

2.1 Inception-v1 (GoogLeNet)[1]

Network that aims to optimize computing resources (i.e., small amount of parameters and FLOPs). *Inception-v1 (GoogLeNet)*

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*

Given the input activation, the output is the concatenation of:

- A 1×1 (stride 1) and a 5×5 (stride 1, padding 2) convolution.
- A 1×1 (stride 1) and a 3×3 (stride 1 and padding 1) convolution.
- A 1×1 (stride 1 and padding 0) convolution.
- A 1×1 (stride 1) convolution and a 3×3 (stride 1 and padding 1) max-pooling.

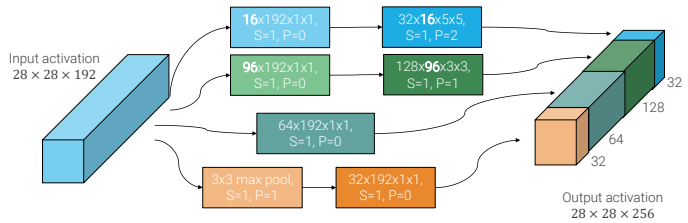


Figure 2.1: Inception module on the output of the stem layers

> **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*

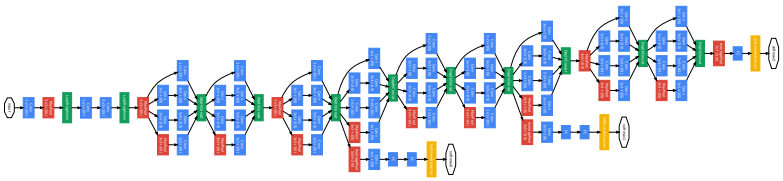


Figure 2.2: Architecture of Inception-v1

[1] Excerpt from IPCV2

2.2 Residual networks[2]

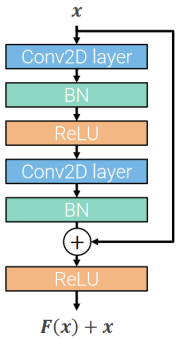
Standard residual block Block that allows to easily learn the identity function through a skip connection. The output of a residual block with input x and a series of convolutional layers F is:

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.

2.2.1 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).

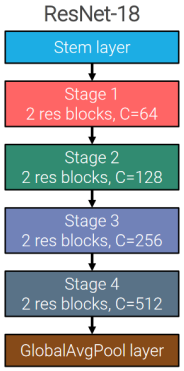


Figure 2.3: Architecture of ResNet-18

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.

[2]Excerpt from IPCV2

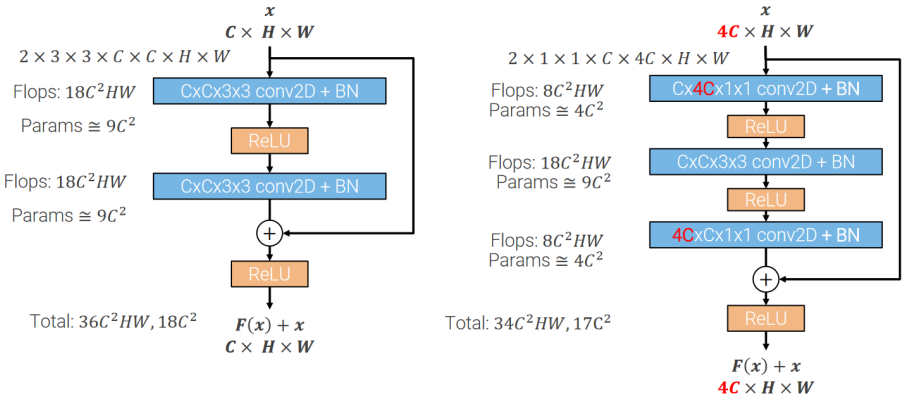


Figure 2.4: Standard residual block (left) and bottleneck block (right)

2.2.2 Inception-ResNet-v4

Network with bottleneck-block-inspired inception modules.

Inception-ResNet-A

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

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

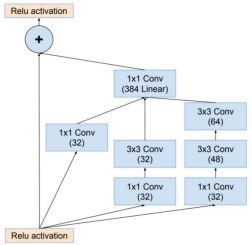
The final concatenation is passed through a 1×1 convolution to match the skip connection shape.

Inception-ResNet-B

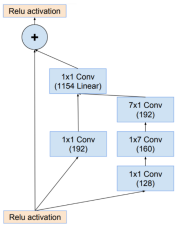
Inception-ResNet-B Three 1×1 convolutions are used to compress the input channels. Each of them leads to:

- 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)

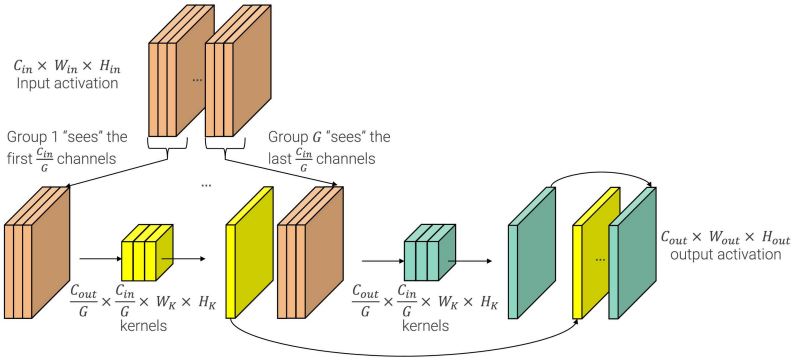
2.3 ResNeXt

Remark. Inception and Inception-ResNet modules are multi-branch architectures and can be interpreted as a split-transform-merge paradigm. Moreover, their architectures have been specifically "hand" designed.

Grouped convolution Given: Grouped convolution

- The input activation of shape $C_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}$,
- The desired number of output channels C_{out},
- The number of groups G,

a grouped convolution splits the input into G chunks of $\frac{C_{\text{in}}}{G}$ channels and processes each with a dedicated set of kernels of shape $\frac{C_{\text{out}}}{G} \times \frac{C_{\text{in}}}{G} \times W_K \times H_K$. The output activation is obtained by stacking the outputs of each group.

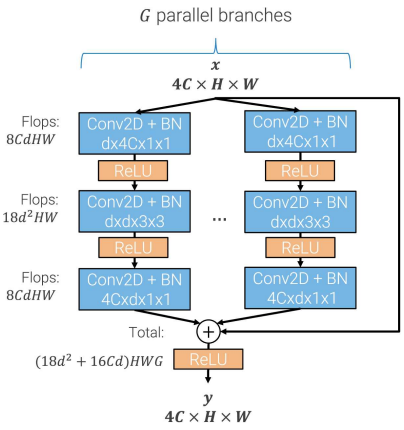


By processing the input in smaller chunks, there are the following gains:

- The number of parameters is G times less.
- The number of FLOPs is G times less.

Remark. Grouped convolutions are trivially less expressive than convolving on the full input activation. However, as convolutions are expected to build a hierarchy of features, it is reasonable to process the input in chunks as, probably, not all of it is needed.

ResNetXt block Given the number of branches G and the number of intermediate channels d, a ResNeXt block decomposes a bottleneck residual block into G parallel branches that are summed out at the end. ResNetXt block



Remark. The branching in a ResNeXt block should not be confused with grouped convolutions.

Remark. Parametrizing G and d allows obtaining configurations that are FLOP-wise comparable with the original ResNet by fixing G and solving a second-order equation over d.

Equivalent formulation Given an input activation $\mathbf{x}$ of shape $4C \times H \times W$, each layer of the ResNeXt block can be reformulated as follows:

Second 1×1 convolution Without loss of generality, consider a ResNeXt block with $G = 2$ branches.

The output $\mathbf{y}_k$ at each channel $k = 1, \ldots, 4C$ is obtained as:

$$\mathbf{y}_k = \mathbf{y}_k^{(1)} + \mathbf{y}_k^{(2)} + \mathbf{x}_k$$

where the output $\mathbf{y}_k^{(b)}$ of a branch b is computed as:

$$\begin{aligned}
\mathbf{y}_k^{(b)}(j,i) &= \left[\mathbf{w}^{(b)} * \mathbf{a}^{(b)}\right]_k (j,i) \\
&= \mathbf{w}_k^{(b)} \cdot \mathbf{a}^{(b)}(j,i) \\
&= \mathbf{w}_k^{(b)}(1)\mathbf{a}^{(b)}(j,i,1) + \cdots + \mathbf{w}_k^{(b)}(d)\mathbf{a}^{(b)}(j,i,d)
\end{aligned}$$

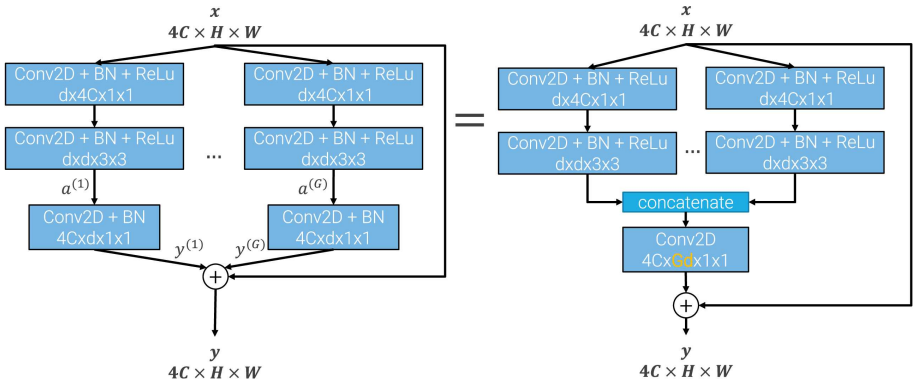
where:

- $*$ represents a convolution,
- $\mathbf{a}^{(b)}$ is the input activation with d channels from the previous layer.
- $\mathbf{w}^{(b)}$ is the convolutional kernel. $\mathbf{w}_k^{(b)} \in \mathbb{R}^d$ is the kernel used to obtain the k-th output channel.

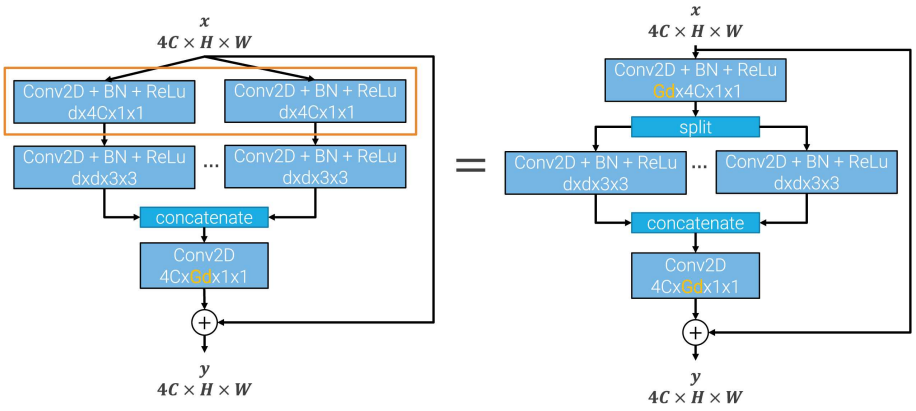
By putting everything together:

$$\begin{aligned}
\mathbf{y}_k(j,i) &= \mathbf{w}_k^{(1)} \cdot \mathbf{a}^{(1)}(j,i) + \mathbf{w}_k^{(2)} \cdot \mathbf{a}^{(2)}(j,i) + \mathbf{x}_k \\
&= \underbrace{\left[\mathbf{w}_k^{(1)} \mathbf{w}_k^{(2)}\right]}_{\text{by stacking, this is a } 1\times 1 \text{ convolution with } 2d \text{ channels}} \cdot \underbrace{\left[\mathbf{a}^{(1)}(j,i)\mathbf{a}^{(2)}(j,i)\right]}_{\text{by stacking depth-wise, this is an activation with } 2d \text{ channels}} + \mathbf{x}_k
\end{aligned}$$

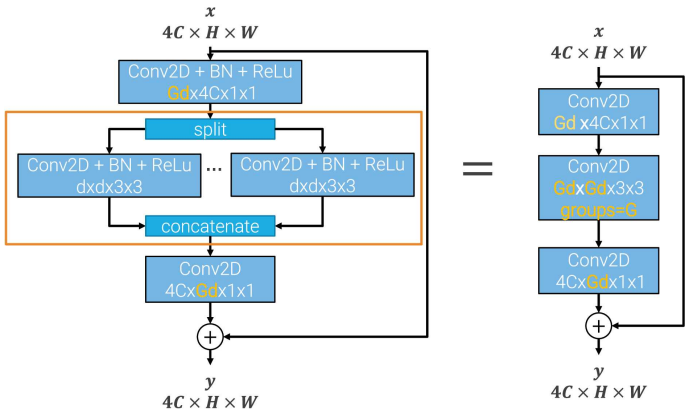
Therefore, the last ResNeXt layer with G branches is equivalent to a single convolution with Gd input channels that processes the concatenation of the activations of the previous layer.



First 1×1 convolution The G 1×1 convolutions at the first layer of ResNeXt all process the same input $\mathbf{x}$. Trivially, this can also be represented using a single 1×1 convolution with G times more output channels that can be split afterwards.



3×3 **convolution** By putting together the previous two equivalences, the middle layer has the same definition of a grouped convolution with G groups. Therefore, it can be seen as a single grouped convolution with G groups and Gd input and output channels.



Remark. Therefore, a ResNeXt block is similar to a bottleneck block.

Remark. It has been empirically seen that, with the same FLOPs, it is better to have more groups (i.e., wider activations).

2.4 Squeeze-and-excitation network (SENet)

Squeeze-and-excitation module

Squeeze-and-excitation module Block that weighs the channels of the input activation. Given the c-th channel of the input activation $\mathbf{x}_c$, the output $\tilde{\mathbf{x}}_c$ is computed as:

$$\tilde{\mathbf{x}}_c = s_c \mathbf{x}_c$$

where $s_c \in [0, 1]$ is the scaling factor.

The two operations of a squeeze-and-excitation block are:

Squeeze Global average pooling to obtain a channel-wise vector.

Excitation Feed-forward network that first compresses the input channels by a ratio r (typically 16) and then restores them. A final sigmoid gives the channel weights.

Squeeze-and-excitation network (SENet) Deep ResNet/ResNeXt with squeeze-and-excitation modules.

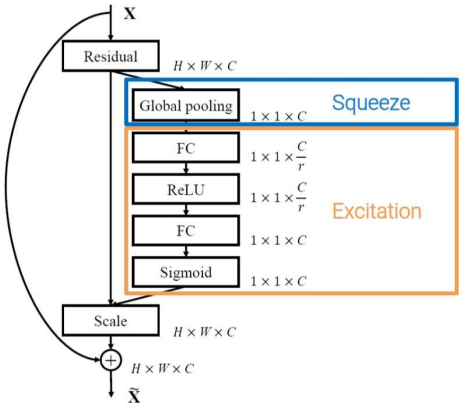


Figure 2.5: SE-ResNet module

2.5 MobileNetV2

Depth-wise separable convolution Use grouped convolutions to reduce the computational cost of standard convolutions. The operations of filtering and combining features are split:

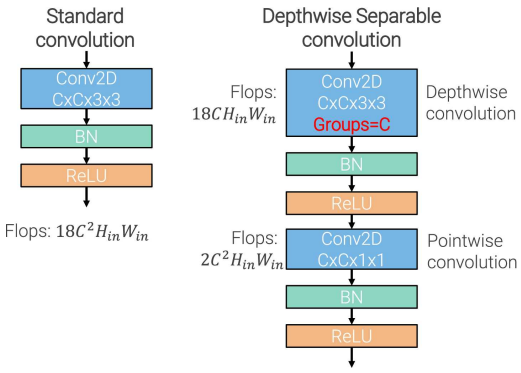
Depth-wise separable convolution

Depth-wise convolution Processes each channel in isolation. In other words, a grouped convolution with groups equal to the number of input channels is applied.

Context point-wise convolution 1×1 convolution applied after the depth-wise convolution to reproduce the channel-wide effect of standard convolutions.

Remark. The gain in computation is up to 10 times the FLOPs of normal convolutions.

Remark. Depth-wise convolutions are less expressive than normal convolutions.



Remark. The 3×3 convolution in bottleneck residual blocks process a compressed version of the input activation, which might cause loss of information when passing through the ReLUs.

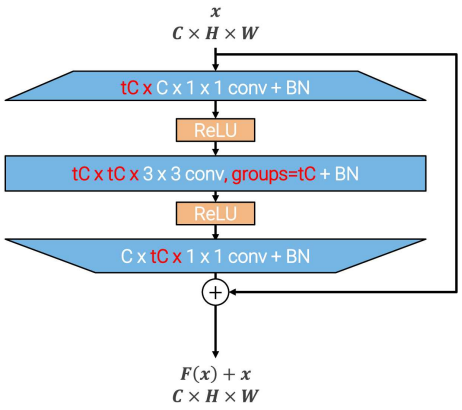
Inverted residual block Modified bottleneck block composed by:

Inverted residual block

1. A 1×1 convolution to expand the input channels by a factor of t.

2. A 3×3 depth-wise convolution.
3. A 1×1 convolution to compress the channels back to the original shape.

Moreover, non-linearity between residual blocks is removed as a result of theoretical studies.



MobileNetV2 Stack of inverted residual blocks. MobileNetV2

- The number of channels grows slower compared to other architectures.
- The stem layer is lightweight due to the low number of intermediate channels.
- Due to the small number of channels, the number of channels in the activation are expanded before passing to the fully-connected layers.

Remark. Stride 2 is applied to the middle 3×3 convolution when downsampling is needed.

Table 2.1: Architecture of MobileNetV2 with expansion factor (t), number of channels (c), number of times a block is repeated (n), and stride (s).

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

2.6 Model scaling

Single dimension scaling Scaling a baseline model by width, depth, or resolution generally improves the accuracy.

Width scaling Increase the number of channels. Width scaling

Depth scaling Increase the number of blocks. Depth scaling

Resolution scaling Increase the spatial dimension of the activations. Resolution scaling

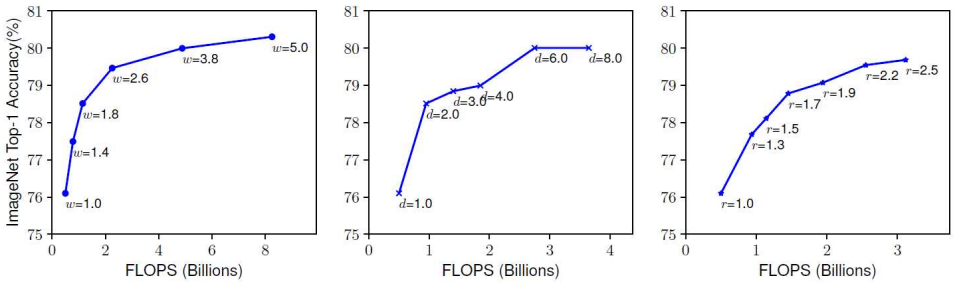


Figure 2.6: Top-1 accuracy variation with width, depth, and resolution scaling on EfficientNet

Compound scaling Scaling across multiple dimensions. Compound scaling

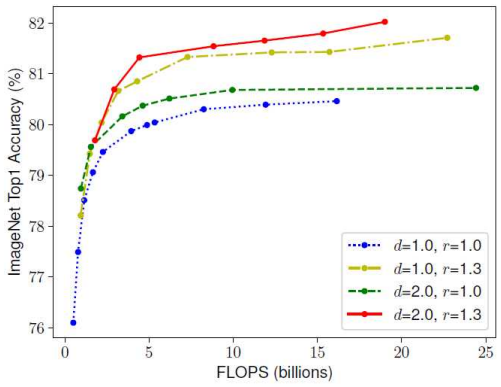


Figure 2.7: Width scaling for different fixed depths and resolutions

Compound scaling coefficient Use a compound coefficient ϕ to scale dimensions and systematically control the FLOPs increase.

> **Remark.** $\phi = 0$ represents the baseline model.

The multiplier for depth (d), width (w), and resolution (r) are determined as:

$$d = \alpha^\phi \qquad w = \beta^\phi \qquad r = \gamma^\phi$$

where α, β, and γ are subject to:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \qquad \text{with } \alpha, \beta, \gamma \geq 1$$

By enforcing this constraint, FLOPs will approximately grow by 2^ϕ (i.e., double) for each increase of ϕ.

In practice, α, β, and γ are determined through grid search.

> **Remark.** The constraint is formulated in this way as FLOPS scales linearly by depth but quadratically by width and resolution.

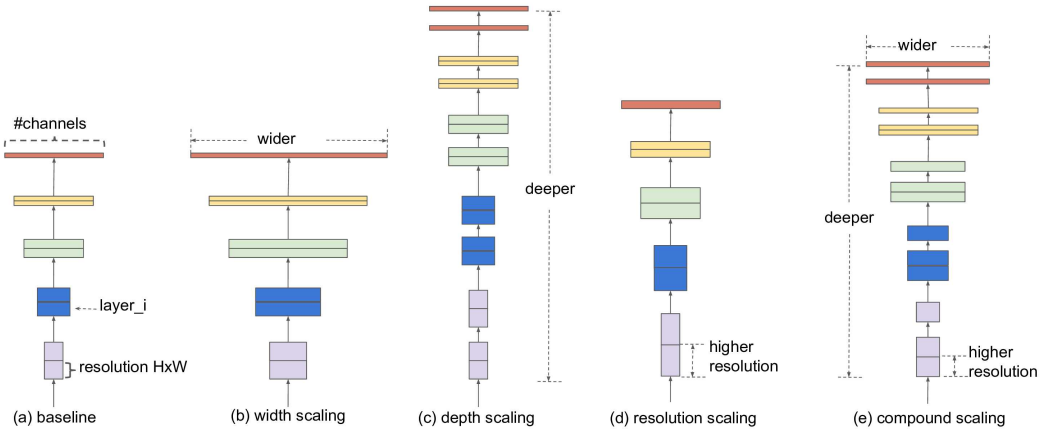
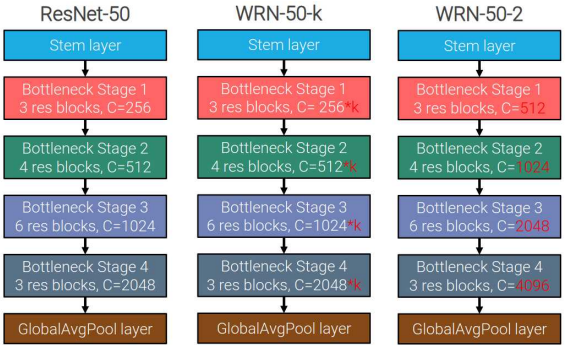


Figure 2.8: Model scaling approaches

2.6.1 Wide ResNet

Wide ResNet (WRN) ResNet scaled width-wise.

Wide ResNet (WRN)

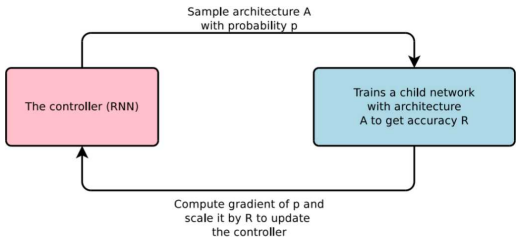


Remark. Wider layers are easier to parallelize on GPUs.

2.7 EfficientNet

Neural architecture search (NAS) Train a controller neural network using gradient policy to output network architectures.

Neural architecture search (NAS)

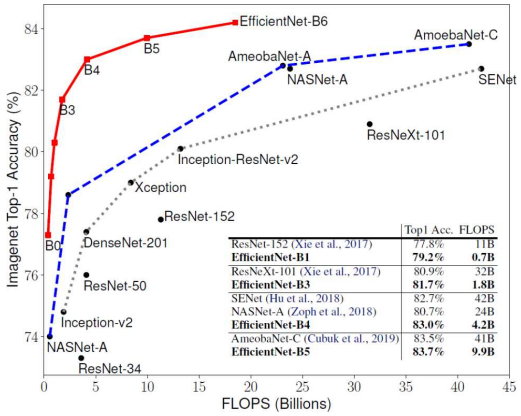


Remark. Although effective, we usually cannot extract guiding principles from the architecture outputted by NAS.

EfficientNet-B0 Architecture obtained through neural architecture search starting from MobileNet.

EfficientNet-B0

Scaling the baseline model (B0) allowed obtaining high accuracies with a controlled number of FLOPs.



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.2%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.7%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**83.0%**	**4.2B**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.7%**	**9.9B**

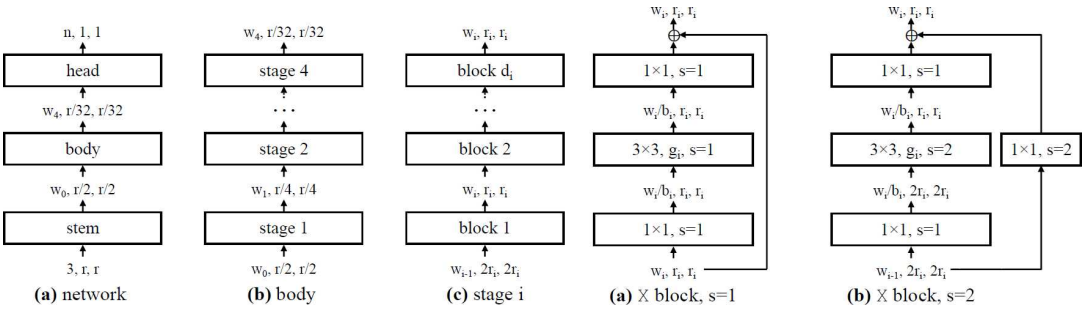
2.8 RegNet

Design space Space of a parametrized population of neural network architectures. By sampling networks from a design space, it is possible to determine a distribution and evaluate it using statistical tools. Design space

> **Remark.** Comparing distributions is more robust than searching for a single well performing architecture (as in NAS).

RegNet Classic stem-body-head architecture (similar to ResNeXt with fewer constraints) with four stages. Each stage i has the following parameters: RegNet

- Number of blocks (i.e., depth) d_i.
- Width of the blocks w_i (so each stage does not necessarily double the number of channels).
- Number of groups of each block g_i.
- Bottleneck ratio of each block b_i.



In other words, RegNet defines a 16-dimensional design space. To evaluate the architectures, the following is done:

1. Sample $n = 500$ models from the design space and train them on a low-epoch training regime.
2. Determine the error empirical cumulative distribution function F computed as the fraction of models with an error less than e:

$$F(e) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[e_i < e]$$

3. Evaluate the design space by plotting F.

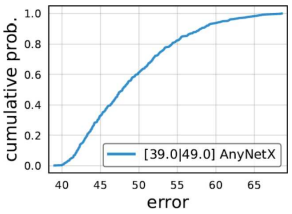


Figure 2.9: Example of cumulative distribution

Remark. Similarly to the ROC curve, the plot of the perfect design space is a straight line at 1.0 probability starting from 0% error rate.

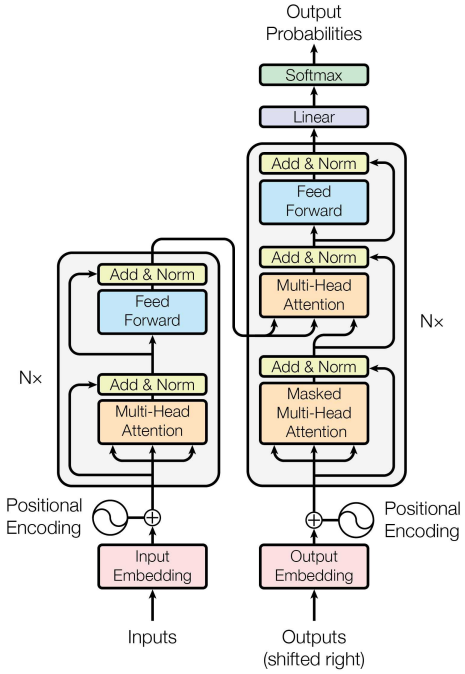
4. Repeat by fixing or finding relationships between parameters (i.e., try to reduce the search space).

Remark. In the original paper, RegNet outperformed EfficientNet. However, results were computed by retraining EfficientNet using the same hyperparameter configuration of RegNet, while the original paper of EfficientNet explicitly tuned its hyperparameters to maximize the results.

3 Transformers in computer vision

3.1 Transformer

Transformer Neural architecture designed for NLP sequence-to-sequence tasks. It heavily relies on the attention mechanism. Transformer



Autoregressive generation A transformer generates the output sequence progressively given the input sequence and the past outputted tokens. At the beginning, the first token provided as the past output is a special start-of-sequence token (`<SoS>`). Generation is terminated when a special end-of-sequence token (`<EoS>`) is generated. Autoregressive generation

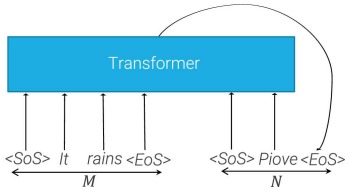


Figure 3.1: Example of autoregressive generation

3.1.1 Attention mechanism

Traditional attention Matrix computed by a neural network to weigh each token of a sequence against the tokens of another one. Traditional attention

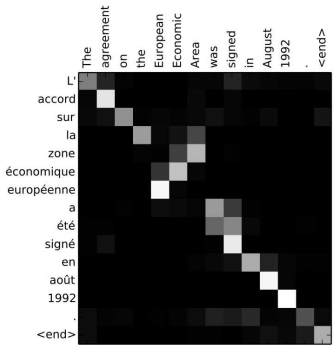


Figure 3.2: Attention weights for machine translation

Remark. Before attention, for tasks such as machine translation, the whole input sequence was mapped into an embedding that is used to influence the generation of the output.

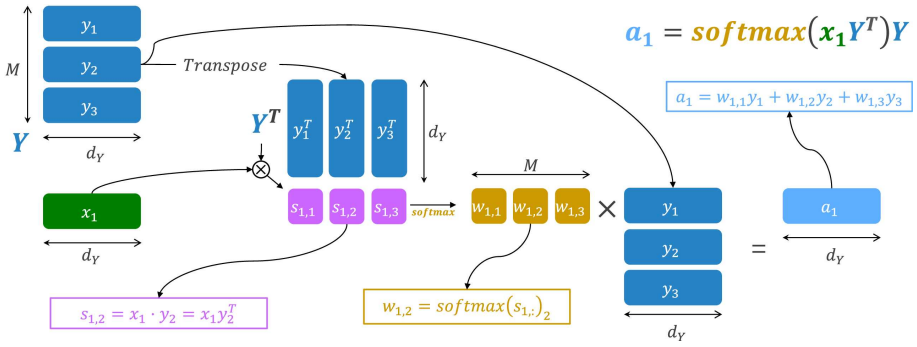
Remark. This is not the same attention of transformers as they do not directly compute attention weights between inputs and outputs.

Dot-product attention

Dot-product attention Given M input tokens $\boldsymbol{Y} \in \mathbb{R}^{M \times d_Y}$ and a vector $\mathbf{x}_1 \in \mathbb{R}^{d_Y}$, dot-product attention computes a linear combination of $\boldsymbol{Y}$ where each component is weighted based on a similarity score between $\boldsymbol{Y}$ and $\mathbf{x}_1$.

This is done as follows:

1. Determine the similarity scores of the inputs as the dot-product between $\mathbf{x}_1$ and $\boldsymbol{Y}^T$:
$$\mathbf{x}_1 \boldsymbol{Y}^T \in \mathbb{R}^M$$
2. Compute the attention weights by applying the `softmax` function on the similarity scores:
$$\texttt{softmax}(\mathbf{x}_1 \boldsymbol{Y}^T) \in \mathbb{R}^M$$
3. Determine the output activation $\mathbf{a}_1$ as the dot-product between the attention weights and the input $\boldsymbol{Y}$:
$$\mathbb{R}^{d_Y} \ni \mathbf{a}_1 = \texttt{softmax}(\mathbf{x}_1 \boldsymbol{Y}^T) \boldsymbol{Y}$$



Scaled dot-product attention

Scaled dot-product attention To add more flexibility, a linear transformation can be applied on the inputs $\boldsymbol{Y}$ and $\mathbf{x}_1$ to obtain:

Keys With the projection $\boldsymbol{W}_K \in \mathbb{R}^{d_Y \times d_K}$ such that $\mathbb{R}^{M \times d_K} \ni \boldsymbol{K} = \boldsymbol{Y}\boldsymbol{W}_K$, where d_K is the dimension of the keys.

Query With the projection $\boldsymbol{W}_Q \in \mathbb{R}^{d_X \times d_K}$ such that $\mathbb{R}^{d_K} \ni \mathbf{q}_1 = \mathbf{x}_1\boldsymbol{W}_X$, where d_X is the length of $\mathbf{x}_1$ that is no longer required to be d_Y as there is a projection.

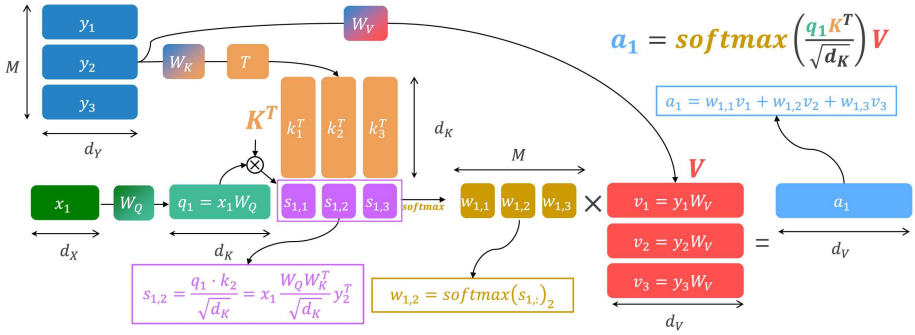
Values With the projection $\boldsymbol{W}_V \in \mathbb{R}^{d_Y \times d_V}$ such that $\mathbb{R}^{M \times d_V} \ni \boldsymbol{V} = \boldsymbol{Y}\boldsymbol{W}_K$, where d_V is the dimension of the values.

The attention mechanism is then defined as:

$$\mathbf{a}_1 = \texttt{softmax}(\mathbf{q}_1\boldsymbol{K}^T)\boldsymbol{V}$$

To obtain smoother attention weights when working with high-dimensional activations (i.e., avoid a one-hot vector from `softmax`), a temperature of $\sqrt{d_K}$ is applied to the similarity scores:

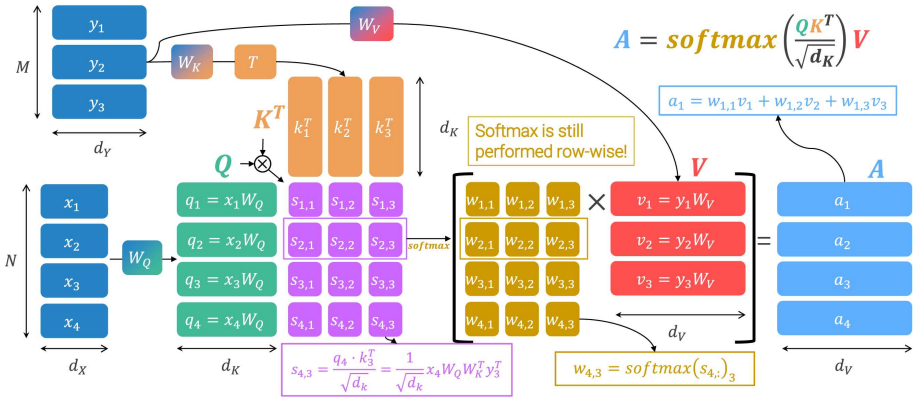
$$\mathbf{a}_1 = \texttt{softmax}\left(\frac{\mathbf{q}_1\boldsymbol{K}^T}{\sqrt{d_K}}\right)\boldsymbol{V}$$



Finally, due to the linear projections, instead of a single vector, there can be an arbitrary number N of inputs $\boldsymbol{X} \in \mathbb{R}^{N \times d_X}$ to compute the queries $\mathbb{R}^{N \times d_K} \ni \boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$. This change affects the similarity scores $\boldsymbol{Q}\boldsymbol{K}^T \in \mathbb{R}^{N \times M}$ and the output activations $\boldsymbol{A} \in \mathbb{R}^{N \times d_V}$.

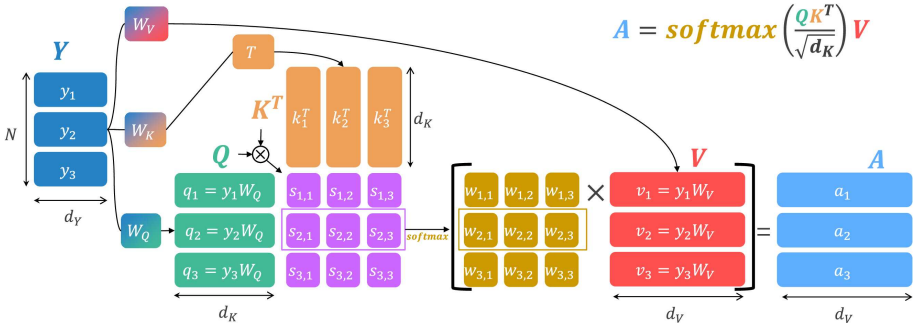
The overall attention mechanism can be defined as:

$$\boldsymbol{A} = \texttt{softmax}_{\texttt{row-wise}}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_K}}\right)\boldsymbol{V}$$



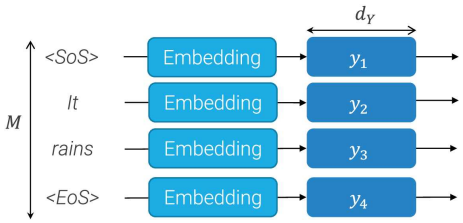
Self-attention Scaled dot-product attention mechanism where the inputs to compute keys, queries, and values are the same. Self-attention

Given an input $\boldsymbol{Y} \in \mathbb{R}^{N \times d_Y}$, the shape of each component is: $\boldsymbol{K} \in \mathbb{R}^{N \times d_K}$, $\boldsymbol{Q} \in \mathbb{R}^{N \times d_K}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d_V}$, and $\boldsymbol{A} \in \mathbb{R}^{N \times d_V}$.



3.1.2 Embeddings

Embedding layer Converts input tokens into their corresponding learned embeddings of shape d_Y (usually denoted as d_{model}). Embedding layer



3.1.3 Encoder

Encoder components A transformer encoder is composed of:

Multi-head self-attention (`MHSA`) Given an input $\boldsymbol{Y} \in \mathbb{R}^{M \times d_Y}$, a `MHSA` block parallelly passes it through h different self-attention blocks to obtain the activations $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(h)}$. The output $\boldsymbol{A}$ of the block is obtained as a linear projection of the column-wise concatenation of the activations $\boldsymbol{A}^{(i)}$: Multi-head self-attention

$$\mathbb{R}^{M \times d_Y} \ni \boldsymbol{A} = \left[A^{(1)} | \ldots | A^{(h)}\right] \boldsymbol{W}_O$$

where $\boldsymbol{W}_O \in \mathbb{R}^{hd_V \times d_Y}$ is the projection matrix.

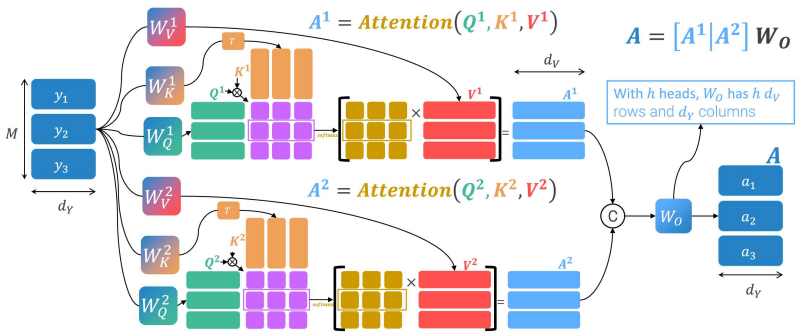


Figure 3.3: `MHSA` with two heads

Remark. The idea of multiple attention heads is to allow the model to attend to information from different representation subspaces.

Remark. Even though they can be freely set, the dimensions for queries, keys, and values of each attention head usually are $d_K = d_V = d_Y/h$.

Layer normalization (`LN`) Normalize each input activation independently to have zero mean and unit variance, regardless of the other activations in the batch. Layer normalization

Given B activations $\mathbf{a}^{(i)} \in \mathbb{R}^D$, mean and variance of each activation $i = 1, \dots, B$ are computed as:

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

Each component j of the normalized activation $\hat{\mathbf{a}}^{(i)}$ is computed as:

$$\hat{\mathbf{a}}_j^{(i)} = \frac{\mathbf{a}_j^{(i)} - \mu^{(i)}}{\sqrt{v^{(i)} + \varepsilon}}$$

As in batch normalization, the actual output activation $\mathbf{s}^{(i)}$ of each input $\mathbf{a}^{(i)}$ is scaled and offset by learned values:

$$\mathbf{s}_j^{(i)} = \boldsymbol{\gamma}_j \hat{\mathbf{a}}_j^{(i)} + \boldsymbol{\beta}_j$$

Remark. Differently from computer vision, in NLP the input is not always of the same length and padding is needed. Therefore, batch normalization do not always work well.

Remark. Layer normalization is easier to distribute on multiple computation units and has the same behavior at both training and inference time.

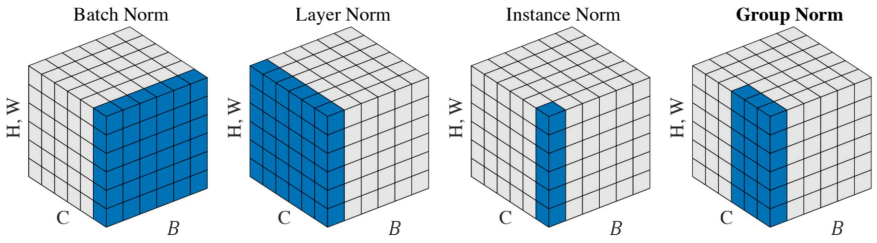


Figure 3.4: Affected axis of normalization methods

Feed-forward network (`FFN`) MLP with one hidden layer applied to each token independently. ReLU or one of its variant is used as activation function: Feed-forward network

$$\texttt{FFN}(\mathbf{x}) = \texttt{relu}(\mathbf{x}\boldsymbol{W}_1 + \mathbf{b}_1)\boldsymbol{W}_2 + \mathbf{b}_2$$

Remark. It can be implemented using two 1D convolutions with kernel size 1.

Residual connection Around the `MHSA` and `FFN` modules. Residual connection

Encoder stack Composed of L encoder layers. Encoder stack

Encoder layer Layer to compute a higher level representation of each input token while maintaining the same length of d_Y. Encoder layer

Given the input tokens $\boldsymbol{H}^{(i)} = [\mathbf{h}_1^{(i)}, \dots, \mathbf{h}_N^{(i)}]$, depending on the position of layer normalization, an encoder layer computes the following:

Post-norm transformer Normalization is done after the residual connection:

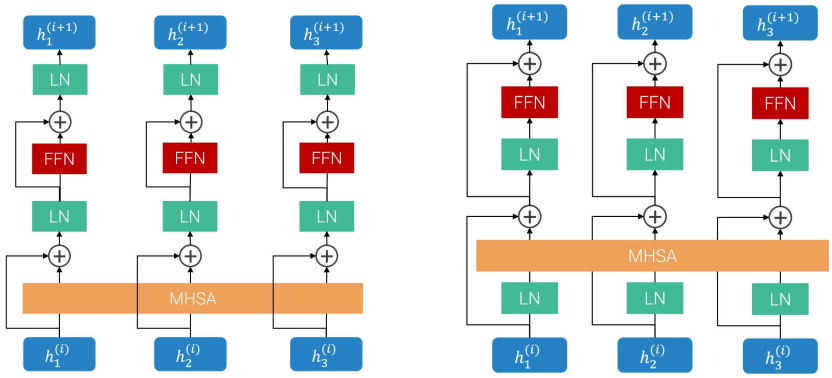
$$\bar{\mathbf{h}}_j^{(i)} = \texttt{LN}\left(\mathbf{h}_j^{(i)} + \texttt{MHSA}_{\boldsymbol{H}^{(i)}}(\mathbf{h}_j^{(i)})\right)$$
$$\mathbf{h}_j^{(i+1)} = \texttt{LN}\left(\bar{\mathbf{h}}_j^{(i)} + \texttt{FNN}(\bar{\mathbf{h}}_j^{(i)})\right)$$

Remark. In post-norm transformers, residual connections are "disrupted" by layer normalization.

Pre-norm transformer Normalization is done inside the residual connection:

$$\bar{\mathbf{h}}_j^{(i)} = \mathbf{h}_j^{(i)} + \texttt{MHSA}_{\boldsymbol{H}^{(i)}}\left(\texttt{LN}(\mathbf{h}_j^{(i)})\right)$$
$$\mathbf{h}_j^{(i+1)} = \bar{\mathbf{h}}_j^{(i)} + \texttt{FNN}\left(\texttt{LN}(\bar{\mathbf{h}}_j^{(i)})\right)$$

Remark. In practice, with pre-norm transformer training is more stable.



(a) Encoder in post-norm transformer (b) Encoder in pre-norm transformer

Remark. Of all the components in an encoder, attention heads are the only one that allow interaction between tokens.

3.1.4 Decoder

Decoder stack Composed of L decoder layers. Decoder stack

Decoder layer Layer to autoregressively generate tokens. Decoder layer

Its main components are:

Multi-head self-attention Processes the input tokens.

Encoder-decoder multi-head attention/Cross-attention Uses as query the output of the previous `MHSA` layer, and as keys and values the output of the encoder stack. In other words, it allows the tokens passed through the decoder to attend the input sequence. Cross-attention

Remark. The output of cross-attention can be seen as an additive delta to improve the activations $\mathbf{z}_j^{(i)}$ obtained from the first MHSA layer.

Remark. As queries are independent to each other, and keys and values are constants coming from the encoder, cross-attention works in a token-wise fashion.

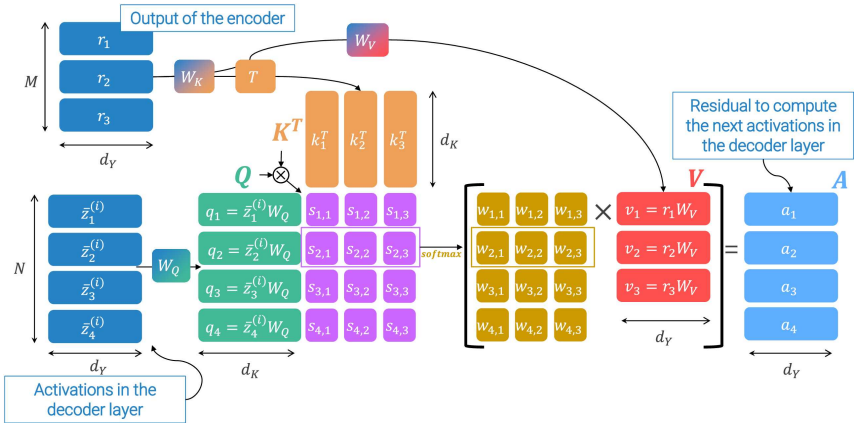


Figure 3.6: Cross-attention data flow

Feed-forward network MLP applied after cross-attention.

Remark. As for the encoder, there is a post-norm and pre-norm formulation.

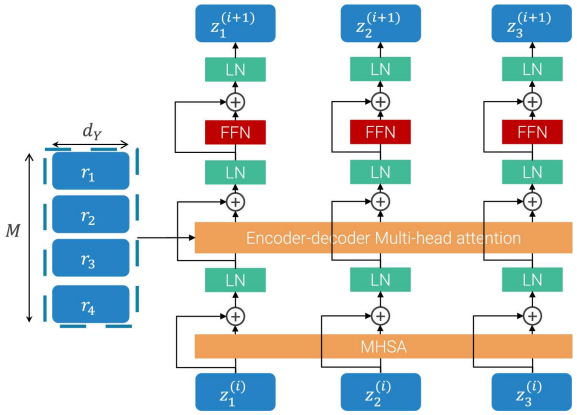


Figure 3.7: Decoder in post-norm transformer

Parallel training When training, as the ground truth is known, it is possible to train all decoder outputs in a single pass. Given a target sequence `[<SoS>`, $t_1, \ldots, t_n$, `<EoS>]`, it is processed by the decoder in the following way:

- The input is `[<SoS>`, $t_1, \ldots, t_n$`]` (i.e., without end-of-sequence token).
- The expected output [$t_1, \ldots, t_n$, `<EoS>]` (i.e., without start-of-sequence token).

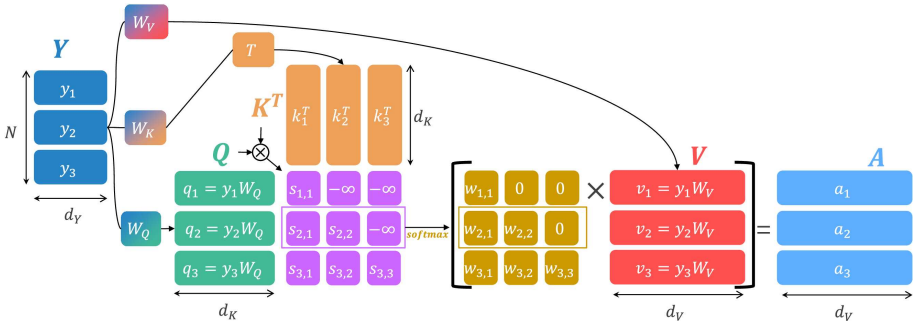
In other words, with a single pass, it is expected that each input token generates the correct output token.

Remark. Without changes to the self-attention layer, a token at position i in the input is able to attend to future tokens at position $\geq i+1$. This causes a data leak as, during inference, autoregressive generation do not have access to future tokens.

Masked self-attention Modification to self-attention to prevent tokens from attending at future positions (i.e., at their right). This can be done by either setting

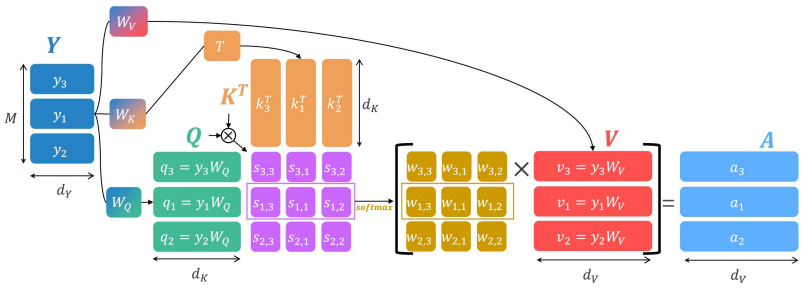
Masked self-attention

the similarity scores with future tokens to $-\infty$ or directly setting the corresponding attention weights to 0 (i.e., make the attention weights a triangular matrix).



3.1.5 Positional encoding

Remark (Self-attention equivariance to permutation)**.** By permuting the input sequence of a self-attention layer, the corresponding outputs will be the same as if it were the original sequence, but it is affected by the same permutation. Therefore, self-attention alone does not have information on the ordering of the tokens.



Positional encoding Vector of shape d_Y added to the embeddings to encode positional information. Positional encoding can be: Positional encoding

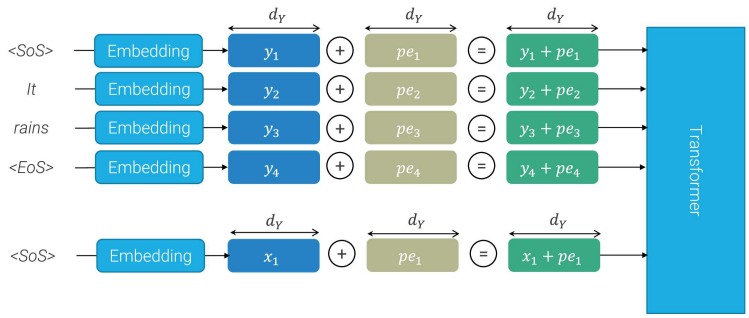
Fixed The vector associated to each position is fixed and known before training.

Example. The original transformer paper proposed the following encoding:

$$\texttt{pe}_{\texttt{pos},2i} = \sin\left(\frac{\texttt{pos}}{10000^{2i/d_Y}}\right) \qquad \texttt{pe}_{\texttt{pos},2i+1} = \cos\left(\frac{\texttt{pos}}{10000^{2i/d_Y}}\right)$$

where `pos` indicates the position of the token and i is the dimension of the position encoding vector (i.e., even indexes use sin and odd indexes use cos).

Learned The vector for position encoding is learned alongside the other parameters.



Remark (Transformer vs recurrent neural networks)**.** Given a sequence of n tokens with d-dimensional embeddings, self-attention and RNN can be compared as follows:

- The computational complexity of self-attention is $O(n^2 \cdot d)$ whereas for RNN is $O(n \cdot d^2)$. Depending on the task, n might be a big value.
- The number of sequential operations for training is $O(1)$ for self-attention (parallel training) and $O(n)$ for RNN (not parallelizable).
- The maximum path length (i.e., maximum number of operations before a token can attend to all the others) is $O(1)$ for self-attention (through the multi-head self-attention layer) and $O(n)$ for RNN (it needs to process each token individually while maintaining a memory).

3.2 Vision transformer

Remark. Using single pixels as tokens is unfeasible due to the complexity scaling of transformers as an $H \times W$ image results in an attention matrix of $(HW)^2$ entries.

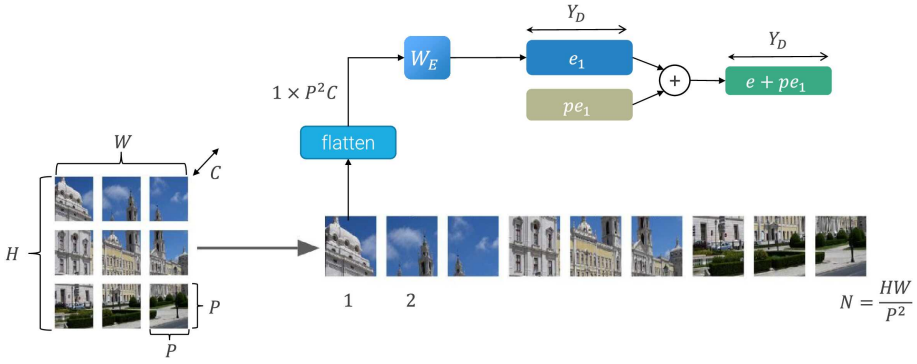
Example. Consider an ImageNet image with shape 224×224. The attention weights will have $(224^2)^2 = 2.5$ bln entries which would require 5 GB to store them in half-precision. A classic 12 layers with 8 heads transformer would require 483 GB of memory to only store all attention matrices.

Remark. Compared to text, image pixels are more redundant and less semantically rich. Therefore, processing all of them individually is not strictly necessary.

Patch

Patch Given an image of size $C \times H \times W$, it is divided into patches of size $P \times P$ along the spatial dimension. Each patch is converted into a Y_D-dimensional embedding for a transformer as follows:

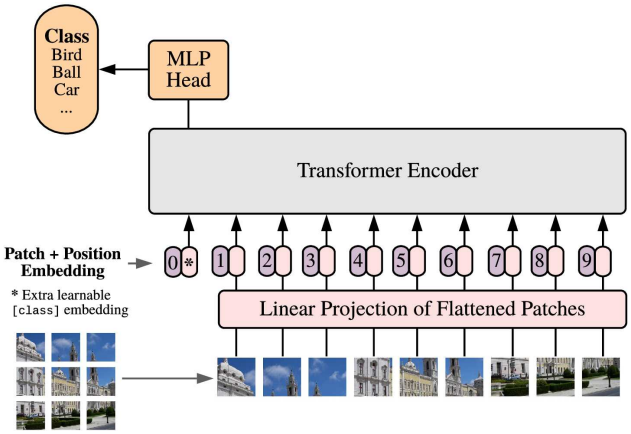
1. Flatten the patch into a P^2C vector.
2. Linearly transform it through a learned projection matrix $W_E \in \mathbb{R}^{P^2C \times Y_D}$.
3. Add positional information.



Vision transformer (ViT)

Vision transformer (ViT) Transformer encoder that processes embedded patches. A special classification token (`[CLS]`, as in BERT) is appended at the beginning of the sequence to encode the image representation and its embedding is passed through a traditional classifier to obtain the logits.

Remark. The (pre-norm) transformer encoder used in vision is the same one as in NLP.



Remark. Differently from convolutional neural networks where convolutions are the major source of FLOPs, in ViT the number of FLOPs heavily depends on the length of the input sequence due to the quadratic complexity of the attention mechanism.

ViT variants The main size-wise variants of ViT are the following:

Model	Layers	Heads	Hidden size	MLP size	Parameters
ViT-base	12	12	768	3072	86 M
ViT-large	24	16	1024	4096	307 M
ViT-huge	32	16	1280	5120	632 M

Note that, by convention, the MLP size is four times the hidden size.

Moreover, ViT models can also vary depending on the size of the input patch.

The overall notation to denote size and patch is: ViT-`<size>`/`<patch size>`.

Results The main experimental observations and results using vision transformer are the following:

- The first embedding projection W_E for RGB images shows a similar behavior to convolutions as they tend to recognize edges and color variations.

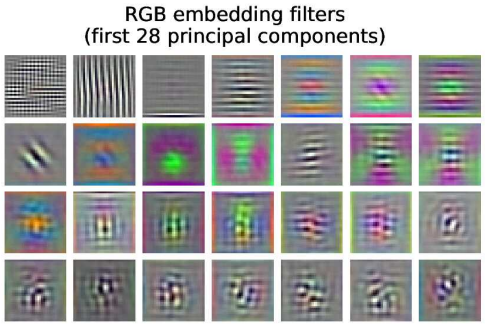


Figure 3.8: Visualization of the columns of the patches linear projection matrix W_E. Each column has shape $3P^2$ and can be reshaped to be a $3 \times P \times P$ image.

- The learned positional embeddings are able to encode information about the row and column positioning of the patches.

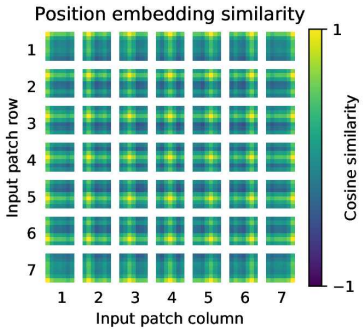


Figure 3.9: Cosine similarity of the positional encoding of each patch compared to all the others

- Attention heads at the lower layers attend at both positions around the patch and far from them. Higher layers, as with convolutions, attend to distant patches.

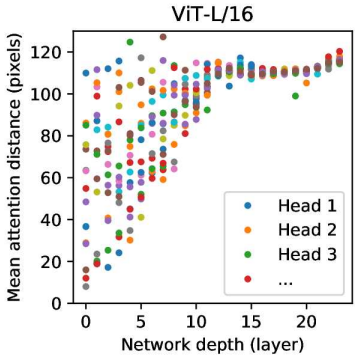


Figure 3.10: Mean attention distance of the heads of ViT-large/16

- On ImageNet top-1 accuracy, ViT outperforms a large ResNet only when pre-trained on a large dataset.

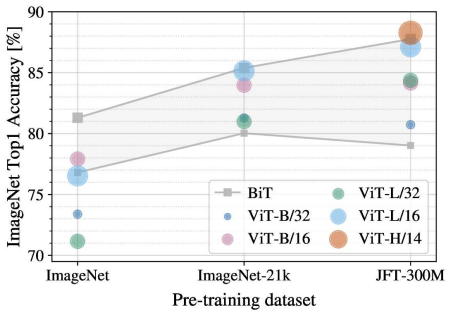


Figure 3.11: ImageNet top-1 accuracy with different pre-training datasets. BiT represents ResNet (two variants).

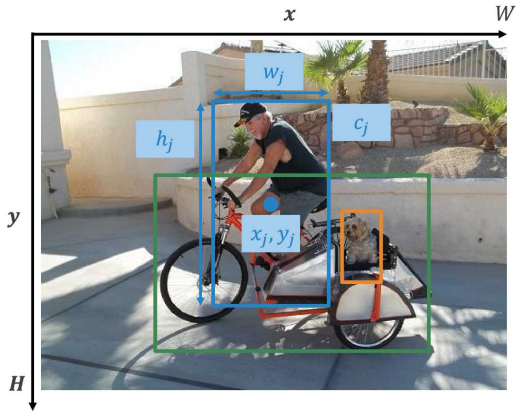
Remark. Comparison between convolutional neural networks and vision transformer is not straightforward.

Remark. On an execution efficiency point-of-view, the currently more common inference hardware is more optimized for convolutions.

4 Object detection

Object detection Given an RGB $W \times H$ image, determine a set of objects $\{o_1, \dots, o_n\}$ contained in it. Each object o_j is described by: Object detection

- A category $c_j \in \{1, \dots, C\}$ as in image classification.
- A bounding box $BB_j = [x_j, y_j, w_j, h_j]$ where $x_j, w_j \in [0, W-1]$ and $y_j, h_j \in [0, H-1]$. (x_j, y_j) is the center and (w_j, h_j) is the size of the box.
- A confidence score ρ_j.



Remark. Differently from classification, a model has to:

- Be able to output a variable number of results.
- Output both categorical and spatial information.
- Work on high resolution input images.

4.1 Metrics

Intersection over union (`IoU`) Measures the amount of overlap between two boxes computed as the ratio of the area of intersection over the area of union: Intersection over union (`IoU`)

$$\texttt{IoU}(BB_i, BB_j) = \frac{|BB_i \cap BB_j|}{|BB_i| + |BB_j| - |BB_i \cup BB_j|}$$

True/false positive criteria Given a threshold $\rho_{\texttt{IoU}}$, a detection BB_i is a true positive (`TP`) w.r.t. a ground-truth $\widehat{BB_j}$ if it is classified with the same class and:

$$\texttt{IoU}(BB_i, \widehat{BB_j}) > \rho_{\texttt{IoU}}$$

Remark. Confidence can also be considered when determining a match through a threshold $\rho_{\min}$.

Recall Measures the number of ground-truth objects that have been found:

$$\texttt{recall} = \frac{|\texttt{TP}|}{|\text{ground-truth boxes}|}$$

Precision Measures the number of correct detections among all the predictions:

$$\texttt{precision} = \frac{|\texttt{TP}|}{|\text{model detections}|}$$

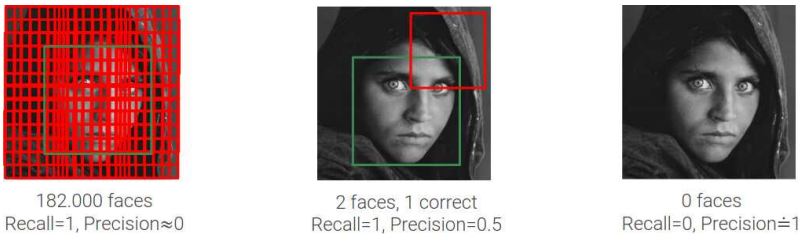


Figure 4.1: Recall and precision in different scenarios

Precision-recall curve Plot that relates all possible precisions and recalls of a detector.

Example. Consider the following image and the bounding boxes found by a detector:

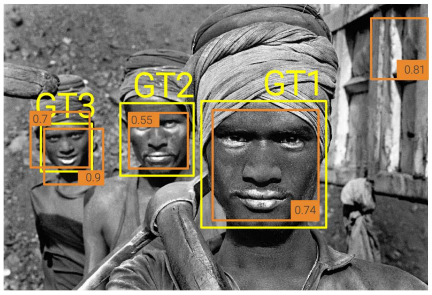
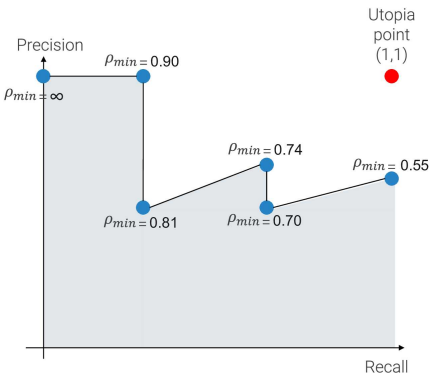


Figure 4.2: Ground-truth (yellow boxes) and predictions (orange boxes) with their confidence score

By sorting the confidence scores, it is possible to plot the precision-recall curve by varying the threshold ρ_{min}:



Remark. Recall is monotonically decreasing, while precision can both decrease and increase.

Average precision (AP) Area under the precision-recall curve.

Average precision (AP)

Mean average precision (mAP) Mean AP over the possible classes.

Mean AP (mAP)

COCO mean average precision Compute for each class the average AP over varying $\rho_{\texttt{IoU}}$ (e.g., in the original paper, $\rho_{\texttt{IoU}} \in [0.5, 0.95]$ with 0.05 steps) and further average them over the possible classes.

COCO mAP

> **Remark.** Higher COCO mAP indicates a detector with good localization capabilities.

4.2 Viola-Jones

Viola-Jones General framework for object detection, mainly applied to faces.

Viola-Jones object detection

It is one of the first successful applications of machine learning in computer vision and has the following basis:

- Use AdaBoost to learn an ensemble of features.
- Use multi-scale rectangular features computed efficiently using integral images.
- Cascade to obtain real-time speed.

4.2.1 Boosting

Weak learner Classifier with an error rate slightly higher than a random classifier (i.e., in a balanced binary task, accuracy slightly higher than 50%).

Weak learner

Decision stump Classifier that learns a threshold for a single feature (i.e., decision tree with depth 1).

Decision stump

Strong learner Classifier with an accuracy strongly correlated with the ground-truth.

Strong learner

Adaptive boosting (AdaBoost) Ensemble of M weak learners $\texttt{WL}_i$ that creates a strong learner $\texttt{SL}$ as the linear combination of their predictions (i.e., weighted majority vote):

Adaptive boosting (AdaBoost)

$$\texttt{SL}(x) = \left(\sum_{i=1}^{M} \alpha_i \texttt{WL}_i(x) > 0 \right)$$

Training Given N training samples $(x^{(i)}, y^{(i)})$ and M untrained weak learners $\texttt{WL}_i$, training is done sequentially by tuning a learner at the time:

Boosting training

1. Uniformly weigh each sample: $w^{(i)} = \frac{1}{N}$.
2. For each weak learner $\texttt{WL}_j$ $(j = 1, \dots, M)$:
   a) Fit the weak learner on the weighted training data.
   b) Compute its error rate:

$$\varepsilon_j = \sum_{i : x^{(i)} \text{ misclassified}} w^{(i)}$$

   c) Compute the reweigh factor:

$$\beta_j = \frac{1 - \varepsilon_j}{\varepsilon_j}$$

d) Increase the weight of misclassified samples:

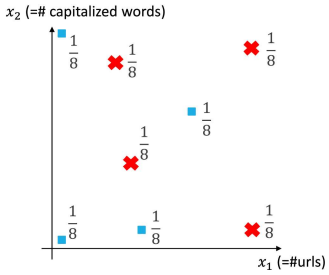
$$w^{(i)} = w^{(i)}\beta_j$$

and re-normalize all samples so that their weights sum to 1.

3. Define the strong classifier as:

$$\texttt{SL}(x) = \left(\sum_j \ln(\beta_j)\texttt{WL}_j(x) > 0 \right)$$

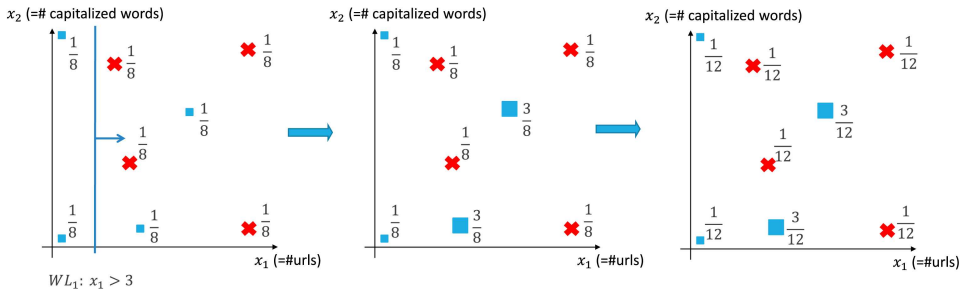
Example. Consider the problem of spam detection with two features x_1 and x_2 (number of URLs and capitalized words, respectively). The training samples and their initial weights are the following:



We want to train an ensemble of 3 decision stumps $\texttt{WL}_j$.
Let's say that the first weak classifier learns to detect spam using the criteria $x_1 > 3$. The error rate and reweigh factor are:

$$\varepsilon_1 = \frac{1}{8} + \frac{1}{8} \qquad \beta_1 = \frac{1-\varepsilon_1}{\varepsilon_1} = 3$$

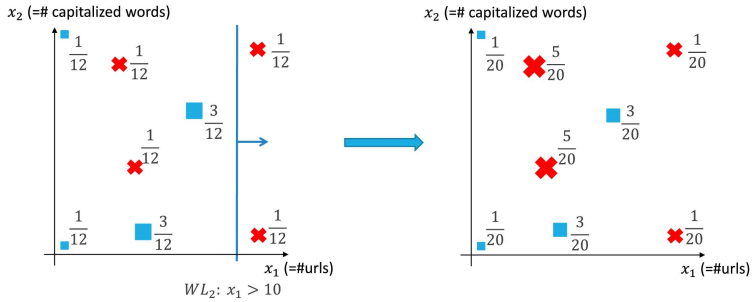
The new reweighed and normalized samples are:



Now, assume that the second classifier learns $x_1 > 10$. The error rate and reweigh factor are:

$$\varepsilon_2 = \frac{1}{12} + \frac{1}{12} \qquad \beta_2 = \frac{1-\varepsilon_2}{\varepsilon_2} = 5$$

The new reweighed and normalized samples are:



Finally, the third classifier learns $x_2 > 20$. The error rate and reweigh factor are:

$$\varepsilon_3 = \frac{1}{20} + \frac{1}{20} + \frac{3}{20} \qquad \beta_3 = \frac{1-\varepsilon_3}{\varepsilon_3} = 3$$

The strong classifier is defined as:

$$\texttt{SL}(x) = \begin{cases} 1 & \text{if } \big(\ln(3)\texttt{WL}_1(x) + \ln(5)\texttt{WL}_2(x) + \ln(3)\texttt{WL}_3(x)\big) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

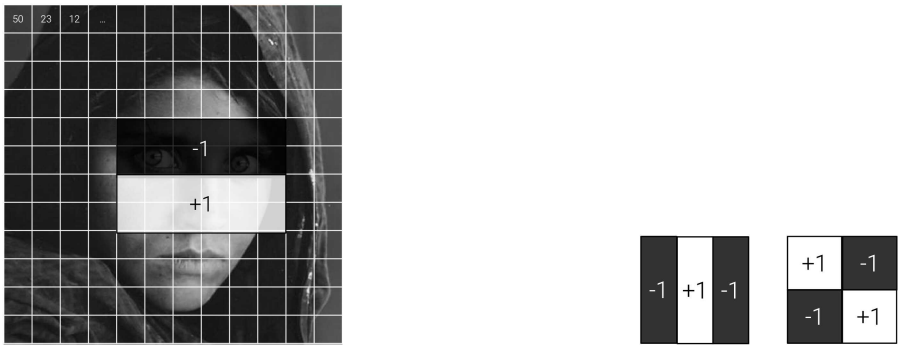
Haar-like features For face detection, a 24×24 patch of the image is considered (for now) and the weak classifiers define rectangular filters composed of 2 to 4 subsections applied at fixed positions of the patch. Haar-like features

Given a patch x, a weak learned $\texttt{WL}_j$ classifies it as:

$$\texttt{WL}_j(x) = \begin{cases} 1 & \text{if } s_j f_j \geq s_j \rho_j \\ -1 & \text{otherwise} \end{cases}$$

where the learned parameters are:

- The size and position of the filter (f_j is the result of applying the filter).
- The polarity s_j.
- The threshold ρ_j.



(a) Filter applied on a patch

(b) Other possible filters

Figure 4.3: Example of filters

Remark. AdaBoost is used to select a subset of the most effective filters.

4.2.2 Integral images

Integral image Given an image I, its corresponding integral image II is defined as: Integral image

$$II(i,j) = \sum_{i' \leq i, j' \leq j} I(i', j')$$

In other words, the value at coordinates (i, j) in the integral image is the sum of all the pixels of the original image in an area that starts from the top-left corner and has as bottom-right corner the pixel at (i, j).

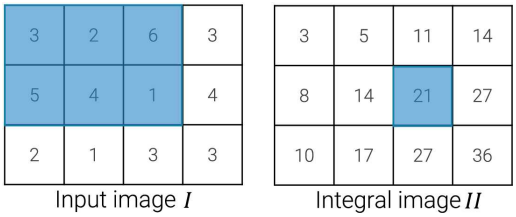


Figure 4.4: Example of integral image

Remark. In practice, the integral image can be computed recursively as:

$$II(i,j) = II(i,j-1) + II(i-1,j) - II(i-1,j-1) + I(i,j)$$

Fast feature computation Given an image I and its integral image II, the sum of the pixels in a rectangular area of I can be computed in constant time as: Fast feature computation

$$II(A) - II(B) - II(C) + II(D)$$

where A, B, C, and D are coordinates defined as in Figure 4.5.

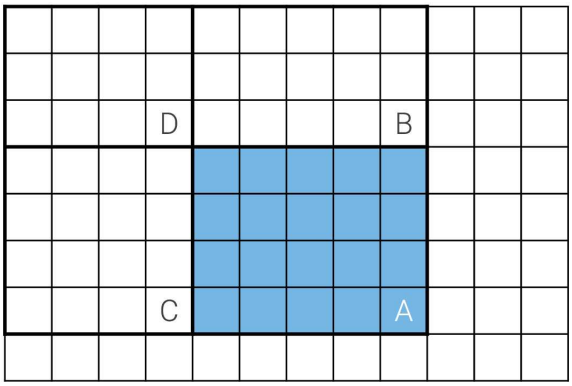


Figure 4.5: Summation of the pixels in the blue area

Multi-scale sliding window During inference, Viola-Jones is a sliding window detector that scans the image considering patches of fixed size. Multi-scale sliding window

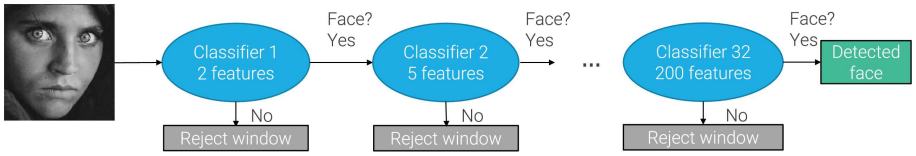
To achieve scale-invariance, patches of different size are used, scaling the rectangular filters accordingly.

Remark. The integral image allows to compute the features in constant time independently of the patch size.

4.2.3 Cascade

Cascade To obtain real-time predictions, a hierarchy of classifiers is used to quickly reject background patches. The first classifier considers a few features while the following ones use more. Cascade

| **Remark.** The simpler classifiers have a high recall so that they do not discard faces.



4.2.4 Non-maximum suppression

Non-maximum suppression (NMS) Algorithm to obtain a single bounding box from several overlapping ones. Given the set of all the bounding boxes with their confidence that a detector found, NMS works as follows: Non-maximum suppression (NMS)

1. Until there are unchecked boxes:
   a) Consider the bounding box with the highest confidence.
   b) Eliminate all boxes with overlap higher than a chosen threshold (e.g., $\texttt{IoU} > 0.5$).

| **Remark.** If two objects are close, NMS might detect them as a single instance.

4.3 CNN for object detection

4.3.1 Object localization

Object localization Subset of object detection problems where it is assumed that there is only a single object to detect. Object localization

CNN for object localization A pre-trained CNN can be used as feature extractor with two heads: CNN for object localization

Classification head Used to determine the class.

Regression head Used to determine the bounding box.

Given:

- The ground-truth class $c^{(i)}$ and bounding box $BB^{(i)}$,
- The predicted class logits $\texttt{scores}^{(i)}$ and bounding box $\widehat{BB}^{(i)}$,

training is a multi-task learning problem with two losses:

$$\mathcal{L}^{(i)} = \mathcal{L}_{\text{CE}}\left(\texttt{softmax}(\texttt{scores}^{(i)}), \mathbb{1}[c^{(i)}]\right) + \lambda \mathcal{L}_{\text{MSE}}\left(\widehat{BB}^{(i)}, BB^{(i)}\right)$$

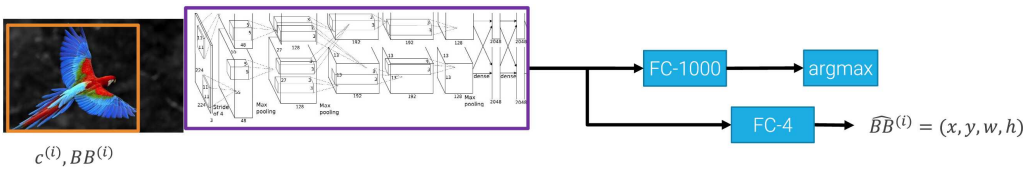


Figure 4.6: Localizer with AlexNet as feature extractor and 1000 classes

Remark. A localization CNN can be used as a sliding window detector to detect multiple objects.
An additional background class (`bg`) has to be added to mark patches without an object. Moreover, when a patch belongs to the background, the loss related to the bounding box should be ignored. Therefore, the loss becomes:

$$\mathcal{L}^{(i)} = \mathcal{L}_{\text{CE}}\left(\texttt{softmax}(\texttt{scores}^{(i)}), \mathbb{1}[c^{(i)}]\right) + \lambda \mathbb{1}[c^{(i)} \neq \texttt{bg}]\mathcal{L}_{\text{MSE}}\left(\widehat{BB}^{(i)}, BB^{(i)}\right)$$

where $\mathbb{1}[c^{(i)} \neq \texttt{bg}]$ is 1 iff the ground-truth class $c^{(i)}$ is not the background class.
This approach has two main problems:

- Background patches are usually more frequent, requiring additional work to balance the dataset or mini-batch.
- There are too many patches to check.

4.3.2 Region proposal detectors

Region proposal Class of algorithms to find regions likely to contain an object. Region proposal

Selective search Region proposal algorithm that works as follows: Selective search

1. Segment the image into superpixels (i.e., uniform regions).
2. Merge superpixels based on similarity of color, texture, or size. Each aggregation generates a proposed region.
3. Repeat until everything collapses in a single region.

Remark. Region proposal algorithms should have a high recall.

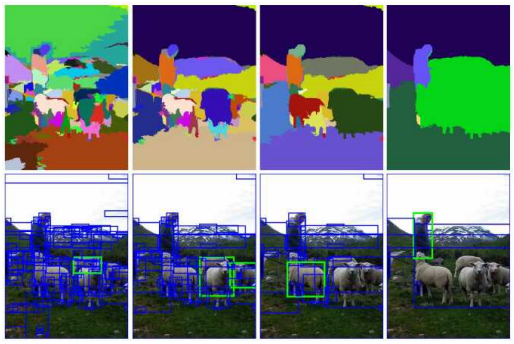


Figure 4.7: Example of some iterations of selective search

Region-based CNN (R-CNN) Use a CNN for object localization with selective search. The workflow is the following: Region-based CNN (R-CNN)

1. Run selective search to get the proposals.
2. For each proposal:
   a) Warp the proposed crop to the input shape of the CNN.
   b) Feed the warped crop to the CNN to get:
      - A class prediction.
      - A bounding box correction (as selective search already gives a box).

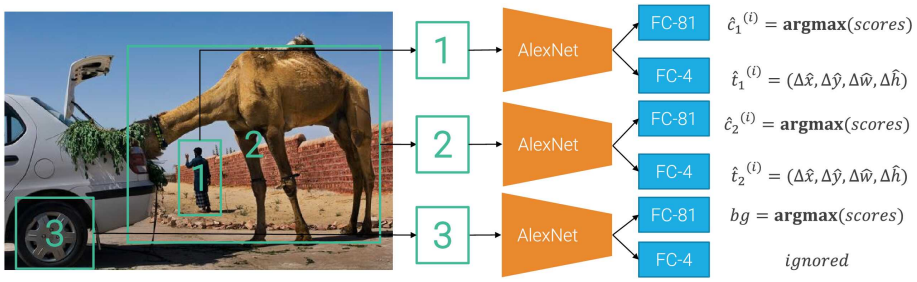


Figure 4.8: Example of R-CNN using AlexNet

Bounding box correction Given a selective search bounding box BB_{SS} and the network predicted correction $\hat{t}$: Bounding box correction

$$BB_{\text{SS}} = (x_{\text{SS}}, y_{\text{SS}}, w_{\text{SS}}, h_{\text{SS}}) \qquad \hat{t} = (\Delta\hat{x}, \Delta\hat{y}, \Delta\hat{w}, \Delta\hat{h})$$

the output box BB_{out} is given by:

$$BB_{\text{out}} = (x_{\text{SS}} + w_{\text{SS}}\Delta\hat{x}, y_{\text{SS}} + h_{\text{SS}}\Delta\hat{y}, w_{\text{SS}}\exp(\Delta\hat{w}), h_{\text{SS}}\exp(\Delta\hat{h}))$$

where the center is a translation relative to the box size and the dimensions are log-space scaled.

Remark. This formulation is due to the fact that a neural network tend to output smaller values, so overall it results an easier task to learn.

Training Given a training sample $x^{(i)}$ with class $c^{(i)}$ and bounding box $BB^{(i)} = [x_{\text{GT}}, y_{\text{GT}}, w_{\text{GT}}, h_{\text{GT}}]$, the selective search box $BB_{\text{SS}}^{(i)}$ associated to it during training is the one with the most overlap, while the others are considered background. The target correction $t^{(i)} = [\Delta x, \Delta y, \Delta w, \Delta h]$ is computed as:

$$\Delta x = \frac{x_{\text{GT}} - x_{\text{SS}}}{w_{\text{SS}}} \quad \Delta y = \frac{y_{\text{GT}} - y_{\text{SS}}}{h_{\text{SS}}} \quad \Delta w = \ln\left(\frac{w_{\text{GT}}}{w_{\text{SS}}}\right) \quad \Delta w = \ln\left(\frac{h_{\text{GT}}}{h_{\text{SS}}}\right)$$

The loss is then defined as:

$$\mathcal{L}^{(i)} = \mathcal{L}_{\text{CE}}\left(\texttt{softmax}(\texttt{scores}^{(i)}), \mathbb{1}[c^{(i)}]\right) + \lambda\mathbb{1}[c^{(i)} \neq \texttt{bg}]\mathcal{L}_{\text{MSE}}\left(\hat{t}^{(i)}, t^{(i)}\right)$$

Remark. Empirically, it has been observed that feature computation, fine-tuning, bounding box correction, and architecture are important to increase mAP.

Remark. Instead of AlexNet, any other CNN can potentially be used.

Remark. R-CNN is slow as it requires to process each proposed crop.

Fast R-CNN Optimization to R-CNNs that avoids processing overlapping pixels of the proposed crops with the CNN multiple times: Fast R-CNN

1. Process the original image with the feature extractor section of the CNN.
2. Compute the proposed crops from the feature extractor activations and adjust to the correct shapes through pooling.
3. Feed each crop to the remaining fully-connected layers of the CNN and the task-specific heads.

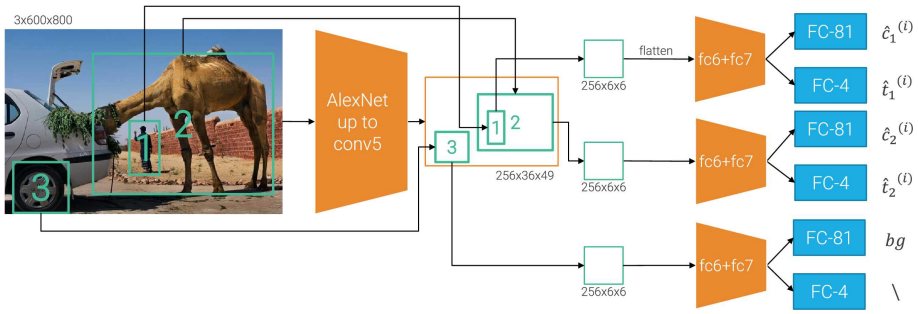


Figure 4.9: Example of fast R-CNN using AlexNet

Region of interest pool (RoIPool)

Region of interest pool (RoIPool) Given an input activation of shape $C_a \times H_a \times W_a$ and the desired output spatial dimension $H_o \times W_o$, RoIPool allows to obtain an output of shape $C_a \times H_o \times W_o$ as follows:

1. Project the proposed region from the original image to the feature extractor activations.
2. Snap the projection to grid (i.e., apply rounding) to obtain a spatial dimension of $H_r \times W_r$.

   Remark. As a single pixel in the activation encodes multiple pixels of the input image, snapping might lose some information.

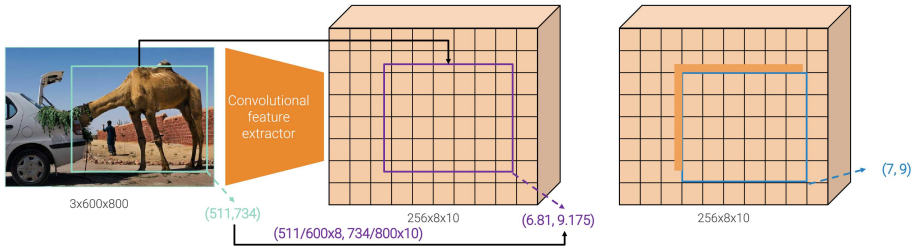


Figure 4.10: Project and snap operations

3. Apply max pooling with kernel of approximately size $\left\lceil \frac{H_r}{H_o} \right\rceil \times \left\lceil \frac{W_r}{W_o} \right\rceil$ and stride approximately $\left\lfloor \frac{H_r}{H_o} \right\rfloor \times \left\lfloor \frac{W_r}{W_o} \right\rfloor$.

   Remark. Approximations are needed as the spatial dimension of the crop might not be directly convertible to the desired output shape. So, some iterations might not use the precise kernel size or stride.

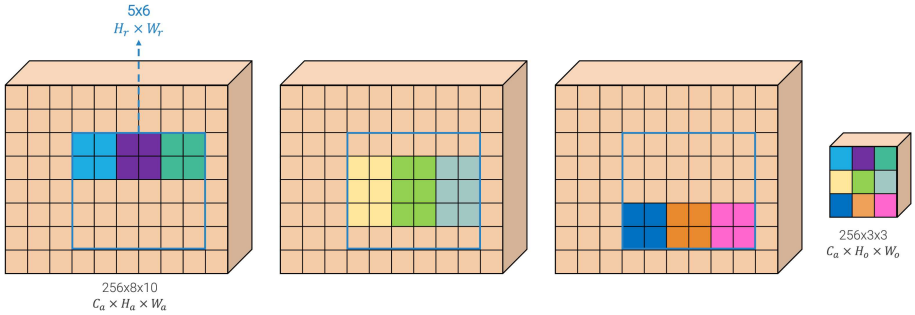


Figure 4.11: Pooling operation with varying kernel size

Remark. Snapping and approximate pooling introduce two sources of quantization.

Huber loss Instead of L2, fast R-CNN uses the Huber (i.e., smooth L1) loss to compare bounding boxes: Huber loss

$$\mathcal{L}_{BB}^{(i)} = \sum_{d \in \{x,y,w,h\}} \mathcal{L}_{\text{huber}}\left(\Delta\hat{d}^{(i)} - \Delta d^{(i)}\right)$$

$$\mathcal{L}_{\text{huber}}(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq 1 \\ |a| - \frac{1}{2} & \text{otherwise} \end{cases}$$

Remark. L2 grows quadratically with the loss which makes it sensitive to outliers. Smooth L1 maintains the gradient constant to 1 for big values.

Remark. Fast R-CNN reduces the number of FLOPs when applying the convolutions but moves the bottleneck to the feed-forward layers.

	Conv FLOPs	FF FLOPs
R-CNN	$n \cdot 2154$ M	$n \cdot 117$ M
Fast R-CNN	16 310 M	$n \cdot 117$ M

Table 4.1: FLOPs comparison with AlexNet as CNN and n proposals

Remark. The slowest component of fast R-CNN is selective search.

Faster R-CNN Selective search is dropped and a region proposal network (RPN) is used: Faster R-CNN

1. Pass the input image through the feature extractor section of the CNN.
2. Feed the activations to the RPN to determine the regions of interest.
3. Continue as in fast R-CNN.

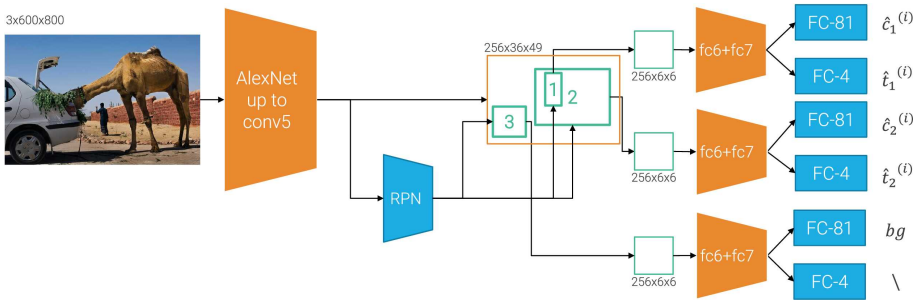


Figure 4.12: Example of faster R-CNN using AlexNet

Region proposal network (RPN) Network that takes as input the image activations of shape $C_L \times H_L \times W_L$ and outputs: Region proposal network (RPN)

- The objectness scores of shape $2 \times H_L \times W_L$.

  Remark. The two channels are due to the fact that the original paper uses a two-way softmax, which in practice is equivalent to a sigmoid.

- The proposed boxes of shape $4 \times H_L \times W_L$.

In other words, an RPN makes a prediction at each input pixel.

Remark. RPN has a small fixed receptive field (that should roughly be the size of an object), but can predict boxes larger than it.

Remark. As is, RPN is basically solving object detection as it has to determine the exact box for the objects, which might be a difficult task.

Anchor Known bounding box with fixed scale and aspect-ratio. Anchor

Anchor correction Make an RPN predict a correction for a known anchor whose center is positioned at the center of the receptive field. Anchor correction

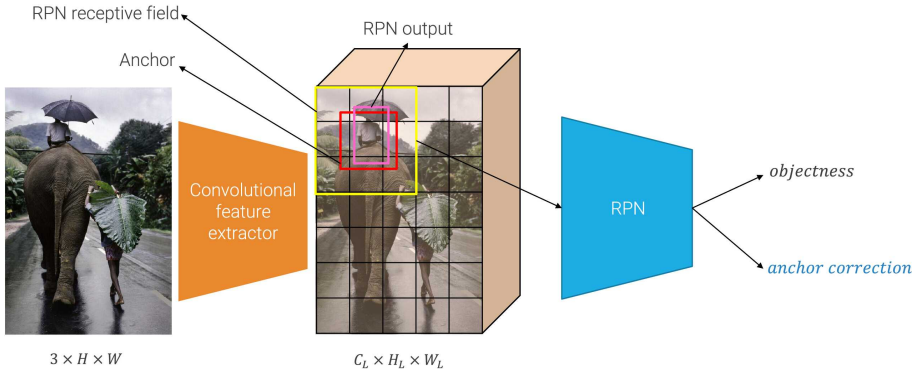


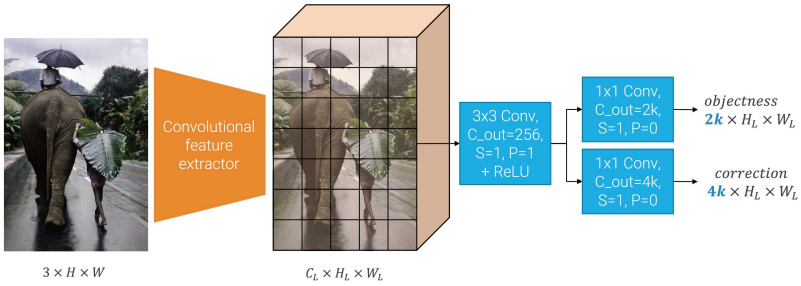
Figure 4.13: Example of an iteration of a 1-anchor RPN

k anchors correction Consider k different anchors so that the RPN outputs k objectness scores (overall shape of $2k \times H_L \times W_L$) and k corrections (overall shape of $4k \times H_L \times W_L$) at each pixel. k anchors correction

Remark. Virtually, this can be seen as putting together the outputs of k different 1-anchor RPN (with different anchors).

Architecture An RPN is implemented as a two-layer CNN:

1. A 3×3 convolution with padding 1, stride 1, 256 output channels, and ReLU as activation.
2. Two parallel 1×1 convolutions with no padding and stride 1 with $2k$ and $4k$ output channels, respectively.



Remark. Only the proposals with the highest objectness scores are considered at training and test time.

Training Given a training image $x^{(i)}$ and a bounding box BB_{GT}, the j-th anchor BB_A can be a:

Negative anchor BB_A has objectness score $o^{(i,j)} = 0$ (i.e., it contains background) if $\texttt{IoU}(BB_{GT}, BB_A) < 0.3$.

Positive anchor BB_A has objectness score $o^{(i,j)} = 1$ (i.e., it contains an object) whether:

- $\texttt{IoU}(BB_{GT}, BB_A) \geq 0.7$.
- $\texttt{IoU}(BB_{GT}, BB_A)$ is the largest and none of the others are ≥ 0.7.

Ignored anchor BB_A is not considered for this sample in all other cases.

A mini-batch is composed of all the positive anchors and it is filled with negative anchors to reach the desired size.

> **Remark.** Differently from R-CNN, multiple boxes have a positive label as it is ambiguous to determine which anchor is responsible for recognizing a particular object.

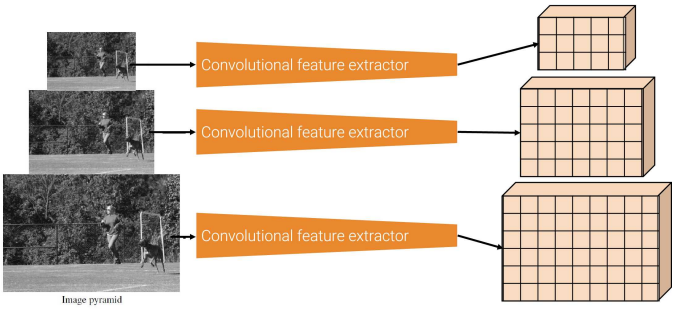
> **Remark.** R-CNN is unable to detect objects smaller than the grid size.

4.3.3 Multi-scale detectors

Image pyramid multi-scale detection Obtain a feature pyramid by feeding the input image to the convolutional feature extractor at different scales.

Image pyramid multi-scale detection

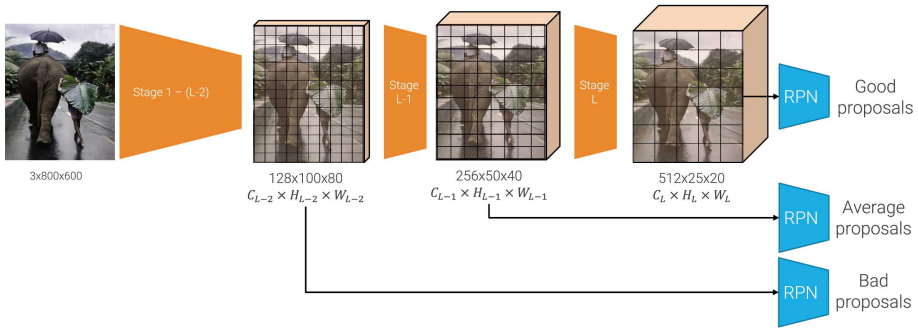
> **Remark.** This approach creates effective features at different scales, but it is computationally expensive.



CNN pyramid multi-scale detection CNNs naturally produce a pyramid of features composed of the activations at each stage.

CNN pyramid multi-scale detection

> **Remark.** This approach do not affect computational cost, but features at smaller scales have bad semantic quality as they are at the beginning of the network.



Feature pyramid network (FPN) Network that enhances small scale features (at the beginning of the network) by combining them with high resolution and semantically rich features (at the end of the network).

Feature pyramid network (FPN)

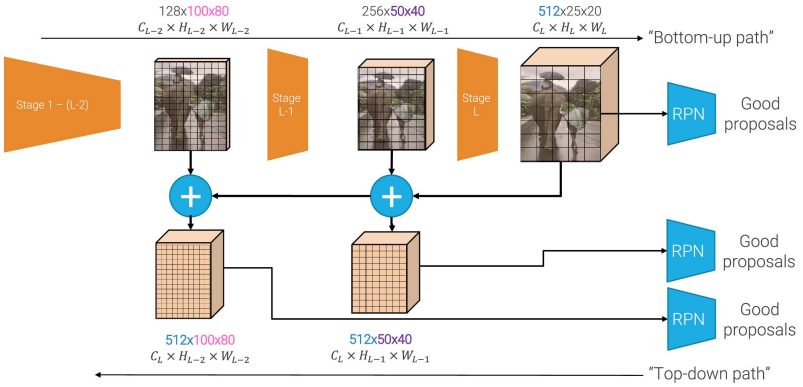


Figure 4.14: General FPN flow

Top-down path Given the activation $A^{(L)}$ at layer L and the activation $A^{(L-1)}$ at layer $L-1$, the top-down path computes the enhanced features as follows:

1. Obtain $\bar{A}^{(L)}$ by upsampling $A^{(L)}$ using nearest neighbor to match the spatial dimension of $A^{(L-1)}$.
2. Obtain $\bar{A}^{(L-1)}$ by applying a 1×1 convolution to $A^{(L-1)}$ to match the number of channels of $A^{(L)}$.
3. Sum $\bar{A}^{(L)}$ and $\bar{A}^{(L-1)}$, and apply a 3×3 convolution to reduce aliasing artifacts caused by upsampling.

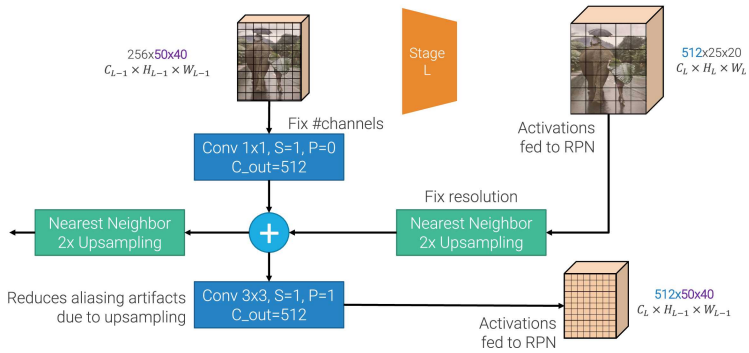


Figure 4.15: FPN top-down flow

Faster R-CNN with FPN The FPN is used with the feature extractor to obtain a pyramid of features $P_1, \ldots, P_n$. A proposal of the RPN is assigned to the most suited level of the pyramid P_k.

Faster R-CNN with FPN

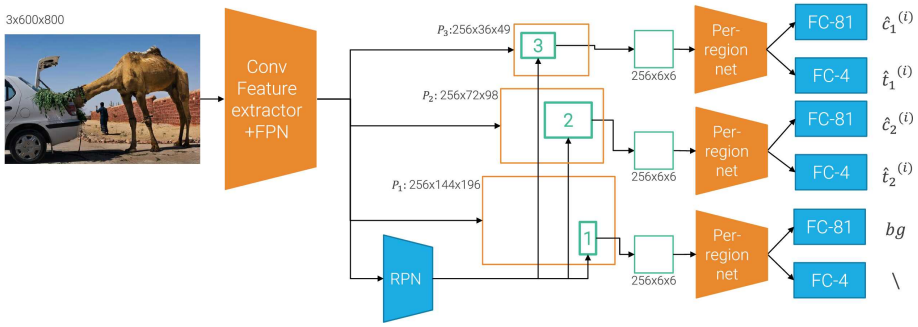


Figure 4.16: Example of faster R-CNN with FPN

Remark. Given a proposal of the RPN with size $w \times h$, the most suited level of the pyramid P_k is determined by the following formula:

$$k = \left\lfloor k_0 + \log_2 \left(\frac{\sqrt{wh}}{224} \right) \right\rfloor$$

where k_0 is the level of the feature map at which a 224×224 proposal should be mapped to.

Remark. R-CNN and its improvements can be seen as a detector with two stages:

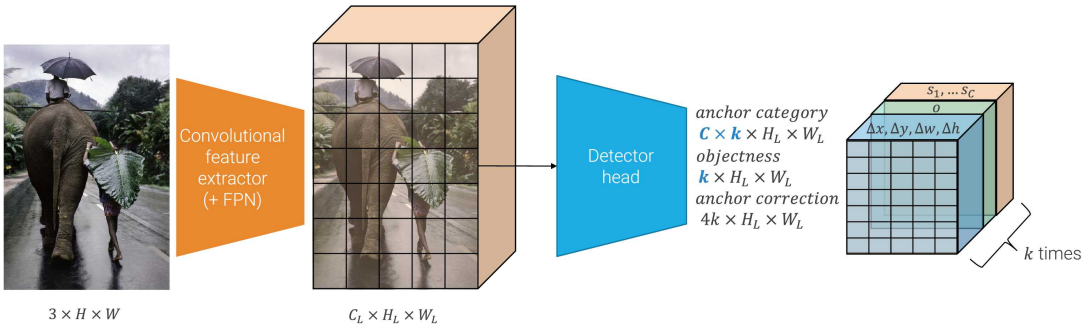
Stage 1 Passes the input image into the feature extractor and RPN to obtain the activations and proposals.

Stage 2 Passes each proposal through RoI pooling and per-region classification and correction.

4.3.4 One-stage detectors

One-stage detector Drop the second stage of R-CNNs and let the RPN determine both the bounding box and the class. One-stage detector

As an objectness score is available, it is used to determine whether the region contains background (i.e., there is no need to add a background class).



Multi-label classification Task in which classes are not mutually exclusive (i.e., multiple classes can be assigned to the same object). Multi-label classification

With C classes, the architecture of a multi-label detector has C independent sigmoid functions at the output layer. Given the predictions for the j-th box of the i-th sample $\mathbf{s}^{(i,j)} \in \mathbb{R}^C$ and the ground-truth $\mathbf{y}^{(i,j)} \in \{0,1\}^C$, the classification loss is defined as:

$$\mathcal{L}(\mathbf{s}^{(i,j)}, \mathbf{y}^{(i,j)}) = \sum_{k=1}^{C} \texttt{BCE}\left(\sigma(\mathbf{s}_k^{(i,j)}), \mathbf{y}_k^{(i,j)}\right)$$

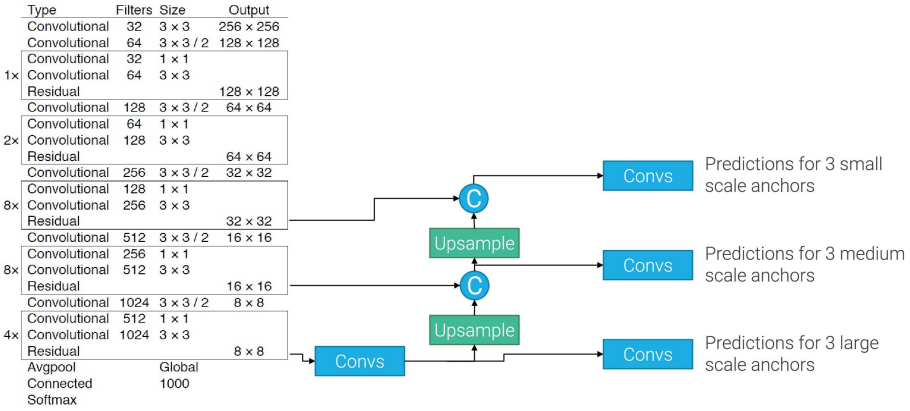
Remark. Not assigning a class (i.e., $\forall k = 1 \dots C : \mathbf{y}_k^{(i,j)} = 0$) can also be used to model the background class.

Remark. Even if the task is multi-class classification, it has been observed that multiple sigmoids perform better.

YOLOv3 One-stage detector that uses DarkNet-53 as backbone for feature extraction and learned anchors. YOLOv3

DarkNet-53 Architecture based on bottleneck residual blocks and multi-scale concatenated features (in FPN they were summed).

It is optimized to obtain a good accuracy-speed trade-off.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Learned anchors Size and aspect-ratio of the anchors are determined using k-means on the training set boxes. The distance between a box BB and a centroid BB_{centroid} is computed as: Learned anchors

$$1 - \texttt{IoU}(BB_{\text{centroid}}, BB)$$

Remark. YOLOv2 uses $k = 5$, while YOLOv3 uses $k = 9$.

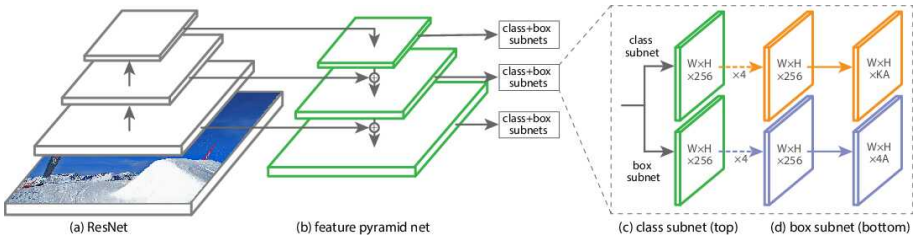
Remark (Class unbalance)**.** The object detection task is usually unbalanced towards (easy) negative boxes (i.e., background). RPNs and one-stage detectors are particularly sensitive to this problem as they always have to evaluate each possible anchor (two-stage detectors are less affected as they only consider top-scored proposals). This unbalance may cause:

- Suboptimal models as easy negative boxes still have a non-zero loss and, being the majority, they make gradient descent unnecessarily consider them.
- Inefficient training due to the fact that random sampled mini-batches will mostly contain easy negative boxes that do not provide useful learning information.

Hard negative mining Sort negative anchors by classification loss and apply NMS. Only top scoring anchors are used in mini-batches.

Remark. This approach allows to reduce the effect of unbalancing, but only a subset of negative examples are used to train the model.

RetinaNet One-stage detector that uses ResNet with FPN as feature extractor. The classification and regression heads are 3×3 convolutions that do not share parameters (differently from the traditional RPN). Moreover, several tweaks have been considered to deal with class unbalance. RetinaNet



Remark. The standard cross-entropy has a non-negligible magnitude even for correctly classified examples. Therefore, easy negative boxes over-weigh the overall loss even if they are already well-classified (i.e., with output probability $p \gg 0.5$).

Binary focal loss Given the output probability p and ground-truth label y, consider the following notation: Binary focal loss

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

$$\texttt{BCE}(p, y) = \texttt{BCE}(p_t) = -\ln(p_t) = \begin{cases} -\ln(p) & \text{if } y = 1 \\ -\ln(1 - p) & \text{if } y = 0 \end{cases}$$

Binary focal loss down-weighs the loss as follows:

$$\texttt{BFL}_\gamma(p_t) = (1 - p_t)^\gamma \texttt{BCE}(p_t) = -(1 - p_t)^\gamma \ln(p_t) = \begin{cases} -(1 - p)^\gamma \ln(p) & \text{if } y = 1 \\ -(p)^\gamma \ln(1 - p) & \text{if } y = 0 \end{cases}$$

where γ is a hyperparameter ($\gamma = 0$ is equivalent to the standard unweighted BCE).

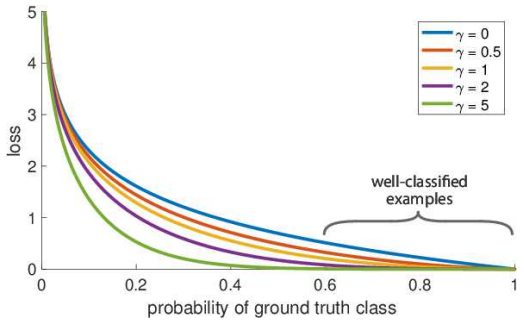


Figure 4.17: Focal loss for varying γ

Example. Consider a focal loss with $\gamma = 2$. The down-weigh factors for varying p_t are:

p_t	$(1 - p_t)^\gamma$
0.9	$(0.1)^2 = 0.01 = \frac{1}{100}$
0.6	$(0.4)^2 = 0.16 \approx \frac{1}{6}$
0.4	$(0.6)^2 = 0.36 \approx \frac{1}{3}$
0.1	$(0.9)^2 = 0.81 \approx \frac{4}{5}$

Binary class weights Weigh the loss value of the positive class by $\alpha \in [0, 1]$ and the negative class by $(1 - \alpha)$. Binary class weights

Consider the following notation:

$$\alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{if } y = 0 \end{cases}$$

Binary cross-entropy with class weights can be defined as:

$$\texttt{WBCE}(p_t) = \alpha_t \texttt{BCE}(p_t) = -\alpha_t \ln(p_t) = \begin{cases} -\alpha \ln(p) & \text{if } y = 1 \\ -(1 - \alpha) \ln(1 - p) & \text{if } y = 0 \end{cases}$$

On the same note, α-balanced binary focal loss can be defined as:

$$\texttt{WBFL}_\gamma(p_t) = \alpha_t \texttt{BFL}_\gamma(p_t) = -\alpha_t(1-p_t)^\gamma \ln(p_t) = \begin{cases} -\alpha(1-p)^\gamma \ln(p) & \text{if } y = 1 \\ -(1-\alpha)(p)^\gamma \ln(1-p) & \text{if } y = 0 \end{cases}$$

Remark. Class weights and focal loss are complementary: the former balances the importance of positive and negative classification errors, while the latter focuses on the hard examples of each class.

RetinaNet loss RetinaNet is applied to all anchors A using the α-balanced binary focal loss for classification and the same bounding box loss as R-CNN:

$$\mathcal{L}^{(i)} = \sum_{j=1}^{A} \left(\sum_{k=1}^{C} \texttt{WBFL}_\gamma \left(\sigma(\mathbf{s}_k^{(i,j)}), \mathbf{y}_k^{(i,j)} \right) + \lambda \mathbb{1}[\mathbf{y}^{(i,j)} \neq \bar{\mathbf{0}}] \mathcal{L}_{\text{huber}} \left(\hat{t}^{(i)} - t^{(i)} \right) \right)$$

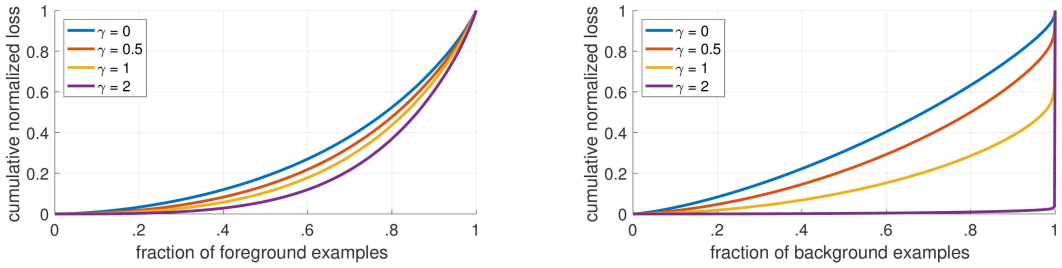


Figure 4.18: Cumulative loss contribution for varying γ of the focal loss. Note that, for the background examples, the contribution to the loss becomes more relevant only when the majority of the samples (i.e., the most difficult ones) has been considered.

Model initialization Instead of initializing biases to 0 (i.e., equiprobable classes), set them to account for unbalanced classes. Model initialization

Consider a newly initialized network with input $\mathbf{x}$, weights $\mathbf{w} \sim \mathcal{N}(\mu = 0, \sigma)$, and bias b. The output activation s is computed as:

$$s = \mathbf{w}\mathbf{x} + b \approx r + b \quad \text{with } r \sim \mathcal{N}(\mu = 0, \sigma)$$

Assume that we want to force the output probability to be ψ, we have that:

$$\begin{aligned} \sigma(r+b) &= \psi \\ \sigma(b) &= \psi && \text{as } \mathbb{E}[r] = 0 \\ \frac{1}{1+e^{-b}} &= \psi \\ b &= -\ln\left(\frac{1-\psi}{\psi}\right) \end{aligned}$$

A reasonable value for ψ is the empirical ratio of the positive class:

$$\psi = \frac{|\text{positives}|}{|\text{positives}| + |\text{negatives}|}$$

Example. Assume an imbalanced dataset with a ratio 1:100 of positive to negative examples. We would like to initialize the bias b so that the model outputs $\psi = \frac{1}{100} = 0.01$ (i.e., more likely to classify as negative at the beginning). Therefore, we have that:

$$b = -\ln\left(\frac{1-0.01}{0.01}\right) \approx -4.6$$

Remark. Anchor-based detectors have the following limitations:

- Anchors are a subset of all possible boxes. Their application is inefficient as they are sort of brute forcing the problem.
- Multiple anchors are viable boxes for the same object and NMS is needed to post-process the predictions.
- During training, the anchor assigned to the ground-truth is selected through hand-crafted rules and thresholds.

4.3.5 Keypoint-based detectors

CenterNet Anchor-free object detector based on keypoints. CenterNet

Given an input image of size $3 \times H \times W$, CenterNet outputs:

Heatmap A matrix $\hat{Y} \in [0,1]^{C \times \frac{H}{R} \times \frac{W}{R}}$ where R is the output stride (usually small, e.g., $R = 4$). Each value scores the "keypoint-ness" of that pixel for a given class.

Offset A matrix $\hat{O} \in \mathbb{R}^{2 \times \frac{H}{R} \times \frac{W}{R}}$ that indicates an offset for each point of the heatmap to map them from the strided shape back to the original one.

Bounding box size A matrix $\hat{S} \in \mathbb{R}^{2 \times \frac{H}{R} \times \frac{W}{R}}$ that indicates the width and height of the bounding box originated from each point of the heatmap.

keypoint heatmap [C] local offset [2] object size [2]

Architecture The backbone of CenterNet is a convolutional encoder-decoder (i.e., first downsample with the encoder and then upsample with the decoder) also used for keypoint detection or semantic segmantation. The output of the backbone is fed to three different branches composed of a 3×3 convolution, ReLU, and a 1×1 convolution.

Inference Predictions are determined as follows:

1. Find the local maxima of the heatmap $\hat{Y}$.
2. For each local maximum at channel $c^{(m)}$ and position $(x^{(m)}, y^{(m)})$ of $\hat{Y}$:
   a) Determine the bounding box center coordinates using the corresponding offsets in $\hat{O}$:

$$(x^{(m)} + \hat{O}_x^{(m)}, y^{(m)} + \hat{O}_y^{(m)})$$

b) Determine the box size using the corresponding values $(\hat{S}_W^{(m)}, \hat{S}_H^{(m)})$ in $\hat{S}$.

Remark. NMS is implicit with the local maxima search operation.

Training Given a ground-truth keypoint (i.e., the center of a box) $p = (x_p, y_p)$ of class c, its coordinates are projected onto the output heatmap coordinate system as $\tilde{p} = (\lfloor \frac{x_p}{R} \rfloor, \lfloor \frac{y_p}{R} \rfloor)$.

The target heatmap $Y_c^{(p)}$ for class c is created as follows:

$$Y_c^{(p)}[x,y] = \begin{cases} 1 & \text{if } (x,y) = \tilde{p} \\ \exp\left(-\frac{(x-x_{\tilde{p}})^2+(y-y_{\tilde{p}})^2}{2\sigma_p^2}\right) & \text{otherwise} \end{cases}$$

In other words, to help training, the heatmap at $\tilde{p}$ is 1 and it smoothly decreases to 0 while moving away from $\tilde{p}$.

The loss is the following:

$$\mathcal{L}^{(p)} = \mathcal{L}_{\text{heatmap}}^{(p)} + \mathcal{L}_{\text{box}}^{(p)}$$

where:

- $\mathcal{L}_{\text{heatmap}}^{(p)}$ is the binary focal loss applied pixel-wise with an additional weighting factor:

$$\mathcal{L}_{\text{heatmap}}^{(p)} = \sum_{x,y,c} \begin{cases} -(1-\hat{Y}_c[x,y])^{\gamma} \ln(\hat{Y}_c[x,y]) & \text{if } Y_c[x,y] = 1 \\ -(1-Y_c[x,y])^{\beta} \cdot (\hat{Y}_c[x,y])^{\gamma} \ln(1-\hat{Y}_c[x,y]) & \text{otherwise} \end{cases}$$

  where $(1-Y_c[x,y])^{\beta}$ reduces the loss at positions close to the ground truth. γ and β are hyperparameters ($\gamma = 2$ and $\beta = 4$ in the original paper).

- $\mathcal{L}_{\text{box}}^{(p)}$ is the Huber loss to compare bounding boxes.

Remark. CenterNet can solve other tasks such as 3D bounding box estimation or pose estimation.

3D size [3] depth [1] orientation [8]

joint locations [$k \times 2$] joint heatmap [k] joint offset [2]

Remark. Keypoints can be seen as a special case of anchors, but:

- They are only based on location and not overlap.
- They do not rely on manual thresholds.
- There is no need for NMS.

However, CenterNet still relies on some hand-crafted design decisions such as the windows size to detect local maxima and the output stride.

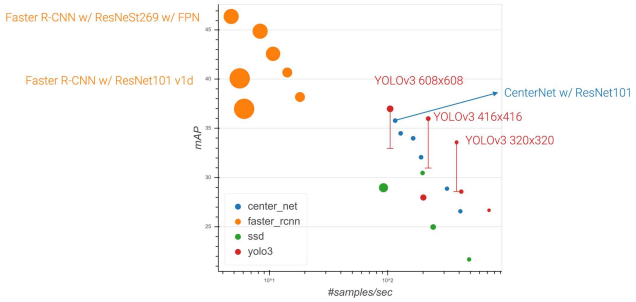


Figure 4.19: mAP – speed comparison of the various object detection approaches

4.3.6 Transformer-based detectors

Detection transformer (DETR)

Detection transformer (DETR) Method based on two ideas:

- Use transformers to predict a set of objects in a single pass (i.e., solve a set prediction problem).
- Use the Hungarian (i.e., bipartite matching) loss as set prediction loss that forces a unique matching between predictions and ground-truths.

Parallel decoding

Parallel decoding Generate all outputs of a decoder in a single pass (instead of doing it autoregressively).

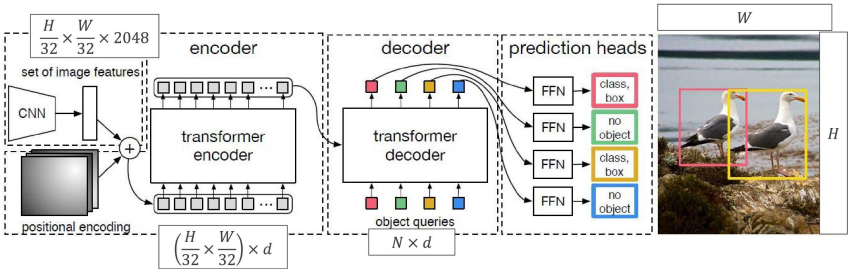
Object queries

Object queries Learned positional encodings that are fed to the decoder.

> **Remark.** In object detection, there are no other generated output tokens to feed into the decoder for parallel decoding. Therefore, only the learned positional encodings are used.

Architecture Given an input image I of size $H \times W \times 3$, DETR does the following:

1. Pass the input I through a CNN feature extractor to obtain an activation A of shape $\frac{H}{32} \times \frac{W}{32} \times 2048$.
2. Use a 1×1 convolution to adjust the number of channels of A from 2048 to d.
3. Add positional encoding to A.
4. Pass the activation A through the transformer encoder to obtain the keys and values for the decoder.
5. Pass the learned object queries through the decoder to obtain the outputs O_i.
6. Pass each output O_i through an MLP to obtain class and box predictions.



Hungarian loss Consider for simplicity a problem with 2 classes (plus background). Given:

- N predictions $\{\hat{y}_i = (\hat{p}_i, \hat{b}_i)\}_{i=1}^{N}$ where $\hat{p}_i$ is the class probability distribution and $\hat{b}_i$ describes the bounding box normalized w.r.t. the image size.
- O ground-truth boxes padded to N with background classes $\{\hat{y}_i = (c_i, b_i)\}_{i=1}^{O} \cup \{\hat{y}_i = \varnothing\}_{i=N-O}^{N}$ where c_i is the class, b_i describes the bounding box normalized w.r.t. the image size, and $\varnothing$ represents the background class.

The Hungarian loss is defined in two steps:

1. Solve the bipartite matching problem of finding the optimal permutation σ^* that associates each prediction to a unique ground-truth box while minimizing the matching loss $\mathcal{L}_{\text{match}}$ defined as follows:

$$\mathcal{L}_{\text{match}}(\hat{y}_i, y_j) = \begin{cases} -\hat{p}_i[c_j] + \mathcal{L}_{\text{box}}(\hat{b}_i, b_j) & \text{if } c_j \neq \varnothing \\ 0 & \text{otherwise} \end{cases}$$

   where $\mathcal{L}_{\text{box}}$ is a loss based on the linear combination of the Huber loss and IoU.

   The overall problem is the following:

$$\sigma^* = \arg\min_{\sigma} \sum_{i=1}^{N} \mathcal{L}_{\text{match}}(\hat{y}_{\sigma(i)}, y_i)$$

2. Given the optimal permutation σ^*, compute the loss as:

$$\mathcal{L}_{\text{hungarian}}(\hat{y}, y) = \sum_{i=1}^{N} \left(-\ln\left(\hat{p}_{\sigma^*(i)}[c_i]\right) + \mathbb{1}[c_i \neq \varnothing] \mathcal{L}_{\text{box}}(\hat{b}_{\sigma^*(i)}, b_i) \right)$$

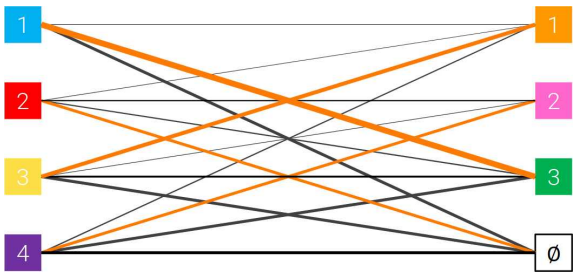


Figure 4.20: Possible permutations and optimal permutation (in orange).

Remark. Results show that faster R-CNN + FPN is better for smaller object and DETR performs best with larger objects.

Remark (Visualization)**.** By analyzing the main components of DETR, the following can be observed:

Encoder The encoder tend to solve a segmentation problem (i.e., determine what the object is).

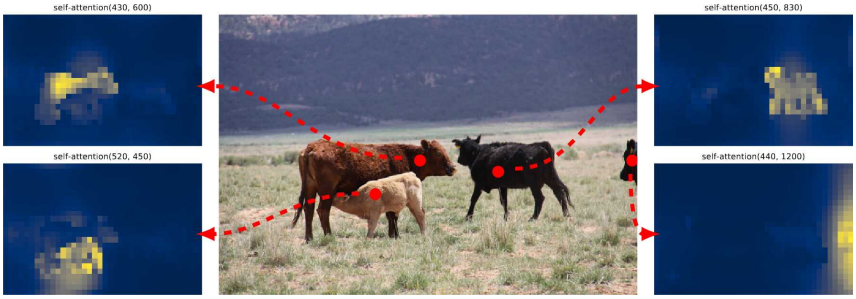


Figure 4.21: Self-attention map of some pixels at the last encoder. Yellow tiles indicate that the analyzed pixel attends to that patch.

Decoder The decoder tend to attend at object boundaries (i.e., determine where the object is).

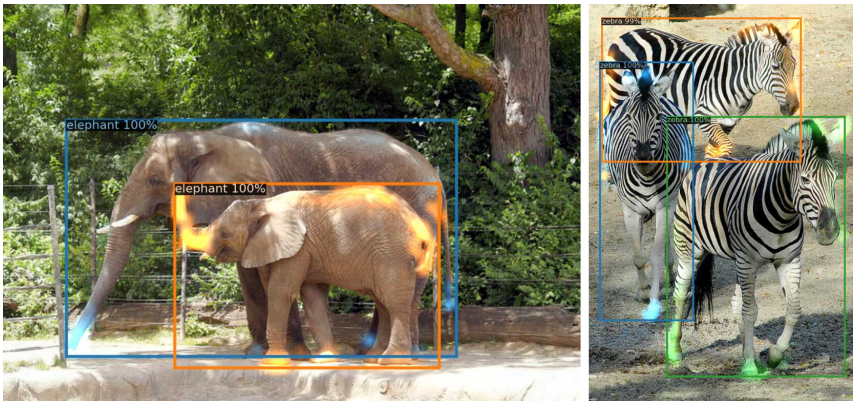


Figure 4.22: Decoder attention. Highlighted areas have a higher attention weight.

Object query Each object query tend to be specialized in recognizing objects in specific areas.

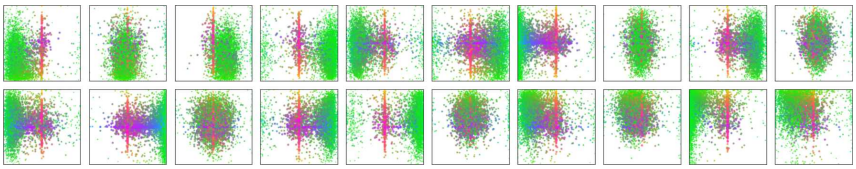


Figure 4.23: Position of the predictions of each object query. Green dots represent small boxes, red large horizontal boxes, and blue large vertical boxes.

4.A Appendix: EfficientDet

Remark. When working with object detection, there are many options to scale a model:

Backbone Change the CNN to refine the input image.

Image resolution Change the resolutions produced by the CNN.

Multi-scale feature representation Change the architecture of the multi-scale detector.

Detector head Change the classification and regression heads.

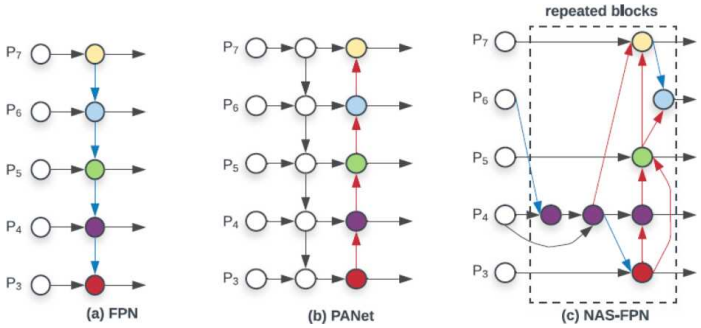
EfficientDet Similarly to MobileNet and EfficientNet, EfficientDet uses a compound scaling coefficient ϕ to scale up the model (using heuristic rules). EfficientDet

Remark. Before EfficientDec, other multi-scale feature representations have been introduced:

FPN As described in Section 4.3.3.

PANet Based on FPN with an additional bottom-up path to merge higher resolution features with coarser ones. PANet

NAS-FPN Base block found using neural architecture search. Multiple blocks are repeated to obtain the multi-scale features. NAS-FPN



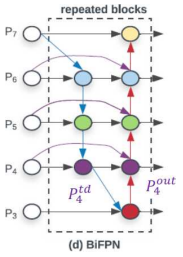
Weighted bi-directional feature pyramid network (BiFPN) Architecture to represent multi-scale features with the following characteristics: Weighted bi-directional feature pyramid network (BiFPN)

- Applied as repeated blocks (i.e., as NAS-FPN).
- Based on PANet with the following changes:
  - Activations generated by only one parent activation are removed and the connections are adjusted.
  - Add a connection between input and output node of a block.
  - Use depth-wise separable convolutions.
  - Add learnable weights to weigh features at different resolutions.

The output P_i^{out} at the i-th scale is therefore computed as:

$$P_i^{\text{out}} = \texttt{conv}\left(\frac{w_{i,1}P_i^{\text{in}} + w_{i,2}P_i^{\text{top-down}} + w_{i,3}\texttt{downsample}(P_{i-1}^{\text{out}})}{w_{i,1} + w_{i,2} + w_{i,3} + \varepsilon}\right)$$

$$P_i^{\text{top-down}} = \texttt{conv}\left(w'_{i,1}P_i^{\text{in}} + w_{i+1,\text{td}}\texttt{upsample}(P_{i+1}^{\text{top-down}})\right)$$



5 Segmentation

5.1 Semantic segmentation

Semantic segmentation Given an input image, output a category for each pixel.

Semantic segmentation

Pixel-wise IoU IoU generalized to pixel-wise segmentation masks. Given a class c, the ground-truths $y^{(i)}$, and predictions $\hat{y}^{(i)}$, the IoU w.r.t. c is computed as follows:

Pixel-wise IoU

$$TP_c = \sum_{i=1} |\text{pixels } (u,v) : y^{(i)}_{(u,v)} = c \wedge \hat{y}^{(i)}_{(u,v)} = c|$$

$$\texttt{IoU}_c = \frac{TP_c}{\sum_{i=1} \Big(|(u,v) : \hat{y}^{(i)}_{(u,v)} = c| + |(u,v) : y^{(i)}_{(u,v)} = c| \Big) - TP_c}$$

The mean IoU is computed as:

$$\texttt{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \texttt{IoU}_c$$

> **Remark.** Mean IoU is not strictly correlated to human's perception of segmentation. A small gain in `mIoU` might correspond to a large visual improvement.

5.1.1 Kinect human pose estimation

Human pose detection Task of detecting the position and orientation of a person.

Human pose detection

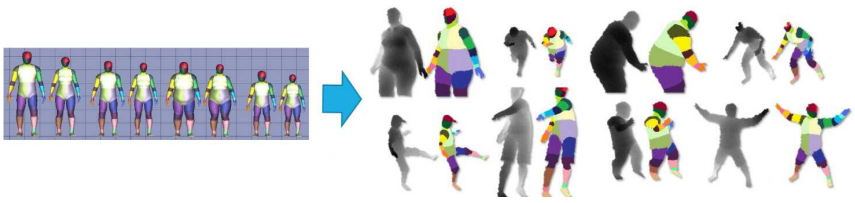
Pipeline Kinect pose detection is done in three phases:

1. Capture a depth image and remove the background to obtain the depth map of a person.
2. Classify each pixel into a body part.
3. Determine the position of the joints (i.e., skeleton) by finding local modes (i.e., center of mass).

Synthetic annotated data The data used to create the model is artificially generated. 100k poses were captured using motion capture devices. Then, different mock-up body models (for which the ground-truth is known) were simulated and recorded through a virtual camera with the same intrinsic parameters of the Kinect camera. This allows to obtain robustness with different body and clothing shapes.

Synthetic annotated data

> **Remark.** The workflow of the original paper consists of iteratively training the model and creating new training data for poses that the current model struggles on.



Depth comparison features Given a depth image D and the offsets $\theta = (\Delta p, \Delta n)$, each pixel x of D produces a feature as follows: Depth comparison features

$$f(x; D, (\Delta p, \Delta n)) = D\left[x + \frac{\Delta p}{D[x]}\right] - D\left[x + \frac{\Delta n}{D[x]}\right]$$

In other words, each x is described by the difference in depth between two points offset from x. The depth at background pixels is a large positive number.

Remark. As real-time processing is required, this approach allows to quickly compute features to discriminate parts of the body. However, it does not always produce a correct response.

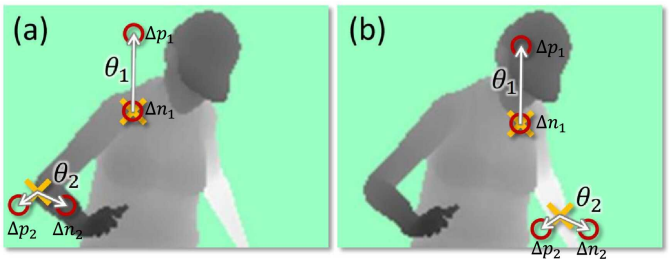
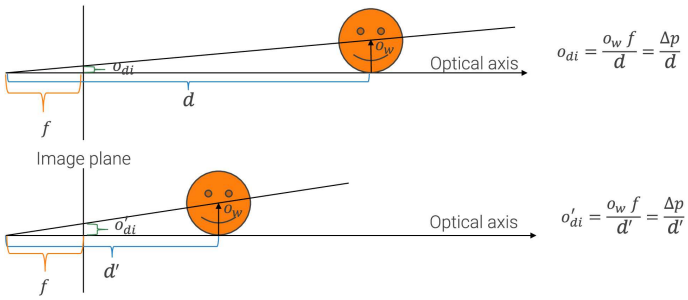


Figure 5.1: Examples of feature computation

Depth-invariant offsets The denominator ($D[x]$) applied to the offsets allows obtaining depth-invariant offsets.

Consider two objects at different depths. The focal length of the camera is f and the world offset we want to apply is o_w. By changing depth d, the offset in the image plane o_{di} changes due to the perspective projection rules. Therefore, to obtain the offset in the image plane, we have that:

$$o_{di} : f = o_w : d \Rightarrow o_{di} = \frac{o_w f}{d} = \frac{\Delta p}{d}$$



Decision tree Depth comparison features are used to train decision trees. Decision tree

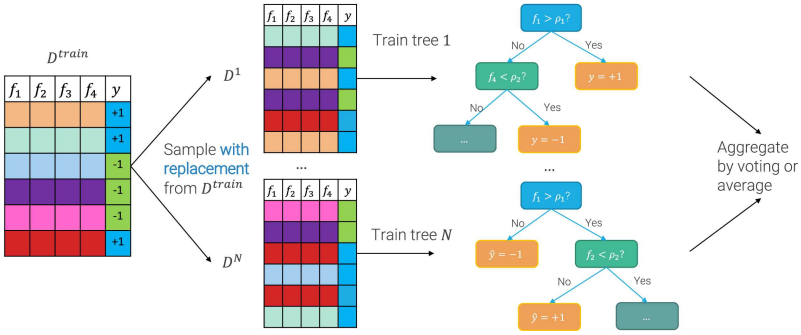
Remark. Decision trees are unstable robust classifiers. They are able to achieve good performance but significantly change in structure if the training data is slightly perturbed (i.e., high variance).

Random forest Ensemble of N decision trees that aims to reduce variance by averaging their predictions. Random forest

Remark. For a random forest to be effective, its decision trees should be uncorrelated so that when averaging, the average of their errors tend to 0.

Bootstrap aggregating (bagging)

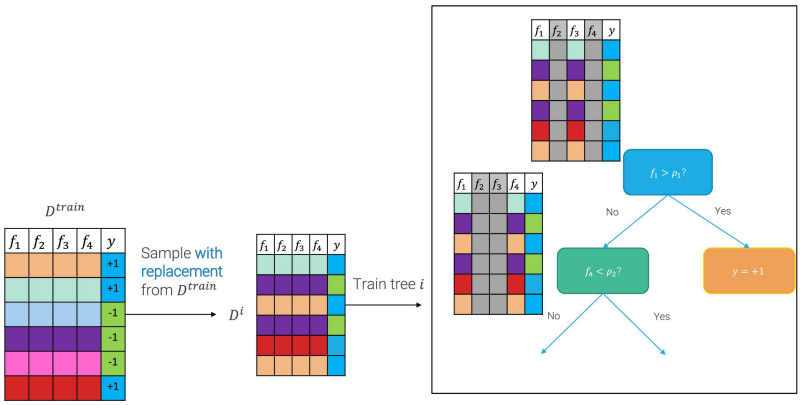
Bootstrap aggregating (bagging) Train each decision tree using a replica of the training set obtained by sampling with replacement.



Random splitting

Random splitting Even though bagging reduces variance, if there is a subset of particularly predictive features, the resulting trees will be correlated.

To avoid this, in a random forest, tree splitting is done on a different random subset of features each time.



Remark. Random forests are:

- Fast and parallelizable at both training and inference.
- Robust to hyperparameters change.
- Interpretable.

5.1.2 R-CNN

R-CNN for segmentation

R-CNN for segmentation Slide a window across each pixel of the input image. Each slice is passed through an R-CNN to determine the class of the pixel at the center of the window.

The loss for an example i is the sum of the cross-entropy losses at each pixel (u, v):

$$\mathcal{L}^{(i)} = \sum_{(u,v)} \mathcal{L}_{\text{CE}} \left(\texttt{softmax}(\texttt{scores}_{(u,v)}), \mathbb{1}\left(c^{(i)}_{(u,v)}\right) \right)$$

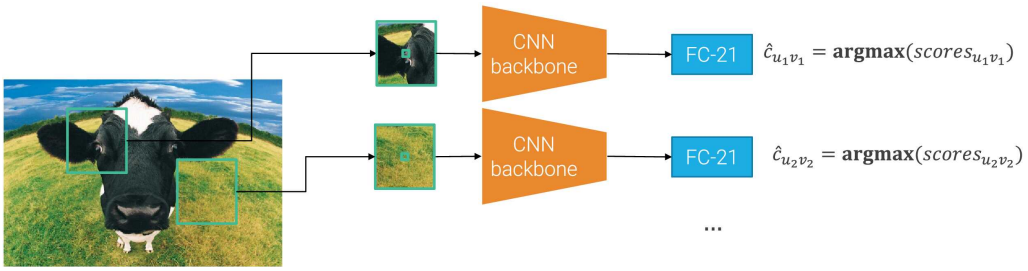


Figure 5.2: R-CNN for segmentation with 20 (+1) classes

5.1.3 Fully convolutional network

Standard up-sampling

Standard up-sampling Non-learned operator to increase the spatial dimension of the input image. Possible approaches are:

- **Nearest neighbor** Fill the new pixels with the value of the nearest one.
- **Bilinear interpolation** Fill the new pixels by interpolating the nearest ones.

Fully convolutional network (FCN)

Fully convolutional network (FCN) Pass the whole input image into a CNN and use the activation to determine the class of each pixel. The flow is the following:

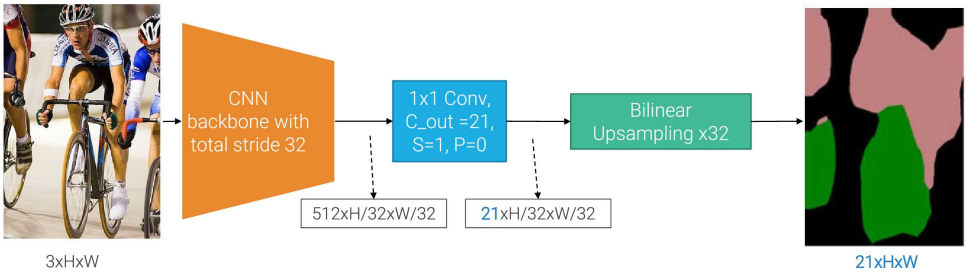
- **Backbone** Pass the input image through a CNN.
- **Scoring layer** Apply a 1×1 convolution on the output activation to obtain the correct number of channels (one per class).
- **Up-sampling** Apply an up-sampling operator to restore the input spatial dimension.

> **Remark.** Without learned non-linear up-sampling operators, the output mask is dependent on the total stride of the CNN. Therefore, a coarse final activation will result in a coarse segmentation mask.

Pyramid of FCNs

Pyramid of FCNs Stack of FCNs with skips (i.e., merges) to up-sample multiple activations at different resolutions.

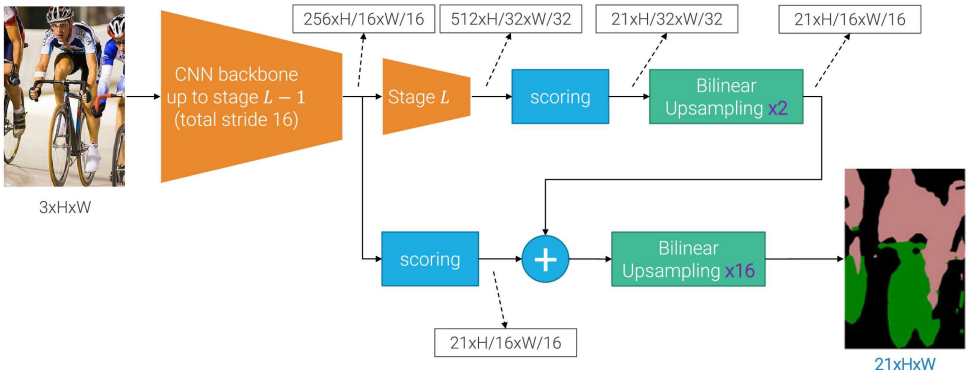
FCN-32S FCN that applies the backbone CNN up until the last layer L obtaining a total stride of 32. This results in a very coarse mask.



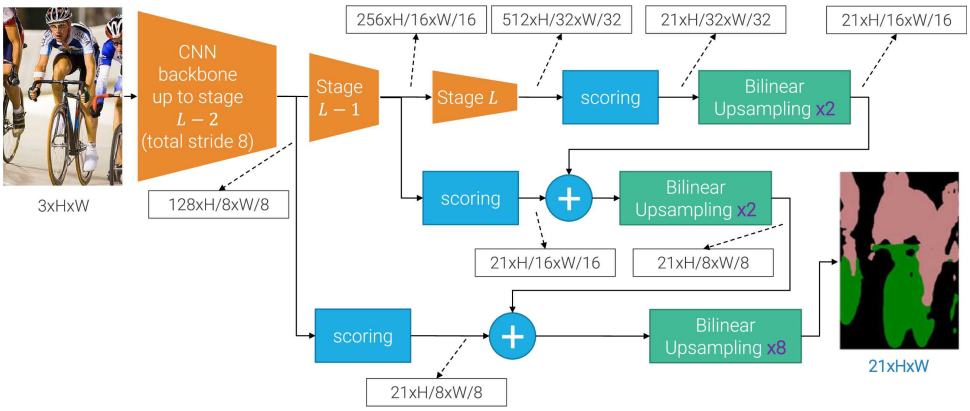
FCN-16S FCN that applies the backbone CNN up until the layer $L-1$ and then branches into two paths:

- A branch continues into the L-th layer as in FCN-32S. Up-sampling is done to match the spatial dimension of the other branch so that it can be used as a skip.

- A branch passes through a scoring layer and is summed to the output of the other branch before up-sampling to the image spatial dimension.



FCN-8S As FCN-16S where the skip comes from FCN-16S.



Remark. Ablation study found out that:

- Results do not show significant improvements after stride 8 (i.e., the initial stages of the backbone CNN are less relevant for the segmentation task).
- Fine-tuning the backbone is important.
- There is no significant difference in training end-to-end (i.e., directly FCN-8S) and coarse-to-fine (i.e., progressively train starting from FCN-32S up to FCN-8S)
- Skips between resolutions to merge them are important.

5.1.4 U-Net

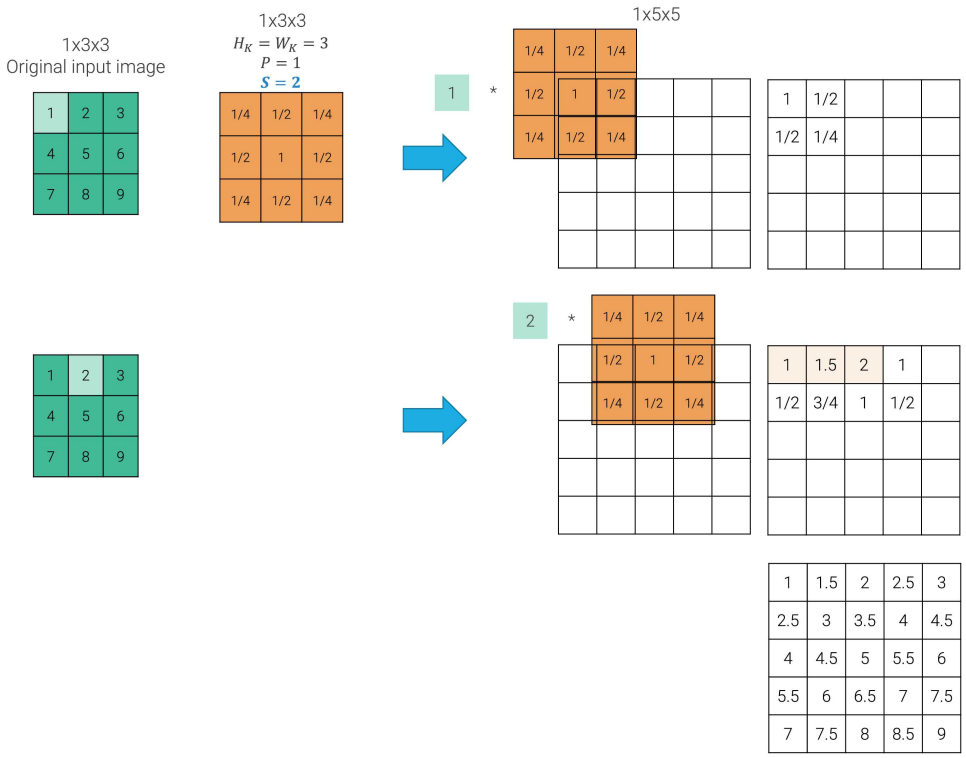
Transposed convolution Operator to invert the spatial down-sampling of a convolution.

Transposed convolution

Given a convolution with convolutional matrix K, a transposed convolution is obtained by applying K^T to the image.

In practice, a transposed convolution can be obtained by sliding the kernel on the output activation (instead of input). The values at the output activation correspond to the product between the input pixels and the kernel. If multiple kernels overlap at the same output pixel, its value is obtained as the sum of all the values that end up in that position.

Example. Consider images with 1 channel. Given a 3×3 input image and a 3×3 transposed convolution kernel with stride 2, the output activation has spatial dimension 5×5 and is obtained as follows:



Remark. A transposed convolution is usually initialized as a bilinear interpolator. For instance, a 3×3 kernel is initialized as:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Remark. Transposed convolutions can be seen as convolutions with a fractional stride.

Remark. Transposed convolutions are also called deconvolutions (this is technically wrong as deconvolutions are the full inverse of convolutions), up-convolutions, fractionally/backward strided convolutions.

Remark. In generative tasks, transposed convolutions generate checkerboard artifacts. Darker pixels are due to the fact that they result from the summation of multiple kernel applications.

U-Net Encoder-decoder architecture for segmentation: U-Net

Encoder Backbone CNN that down-samples the input image.

Decoder Symmetric structure to the encoder that up-samples the activation using transposed convolutions and further refines them with normal 3×3 convolutions.

Each level of the encoder is connected to its corresponding level in the decoder through a skip connection that concatenates the activations.

The scoring layer is only applied at the end to adjust the number of channels.

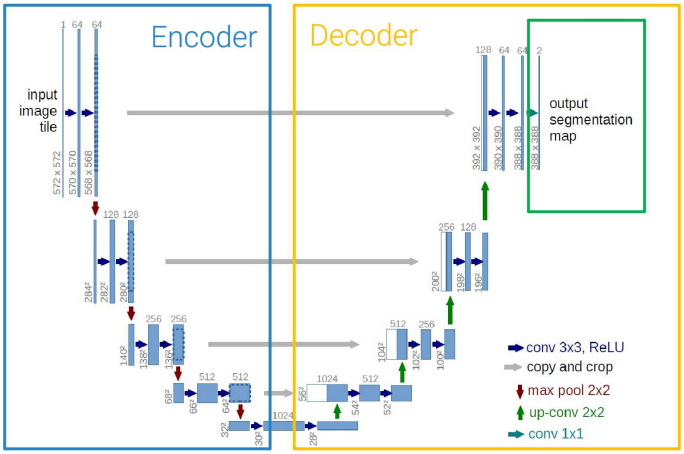


Figure 5.3: U-Net structure. Note that in the original paper, convolutions have padding `valid` and the input is provided from a sliding window. Modern implementations have padding `same` and same input and output spatial dimension.

Remark. Intuitively, the two components of a U-Net do the following:

Encoder Determine what is in the image.

Decoder Determine where the objects are.

5.1.5 Dilated CNN

Remark. The standard classification backbone approach has some shortcomings:

- Predictions made using the activation at higher layers are semantically rich but spatially coarse.
- Predictions made using the activation at lower layers are semantically poorer but have a higher spatial resolution.

Dilated/atrous convolution

Dilated/atrous convolution Given a dilation rate r, a dilated convolution is equivalent to applying a kernel with an $r-1$ gap between weights.

Formally, a dilated convolution with kernel K and dilation rate r applied on an image I is computed as follows:

$$[K * I](j,i) = \sum_{n=1}^{C} \sum_{m} \sum_{l} K_n(m,l) I_n(j - r \cdot m, i - r \cdot l) + b$$

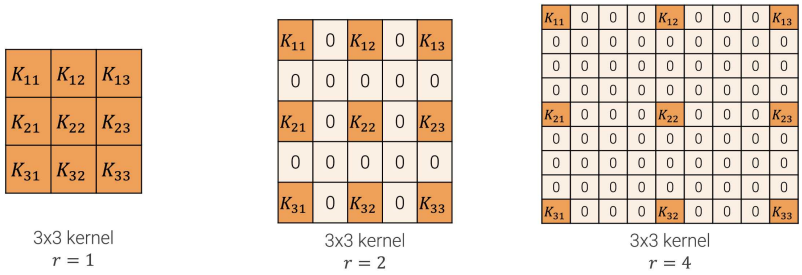
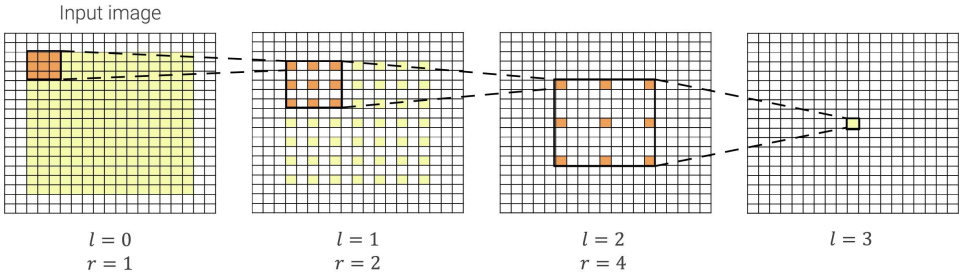


Figure 5.4: Example of 3×3 dilated convolutions with increasing dilation rate

Remark. By stacking dilated convolutions with exponentially increasing dilation rate $r_l = 2^l$, it is possible to achieve the following effects:

- Exponential growth in receptive field. At the l-th level the receptive field is $(2^{l+1} - 1) \times (2^{l+1} - 1)$.
- Linear growth in the number of parameters.
- Unchanged resolution.

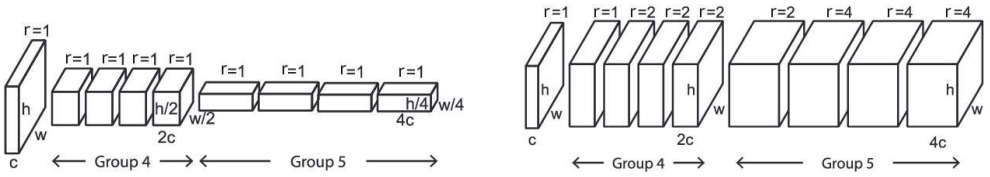


Dilated ResNet ResNet composed of standard stages up until a certain layer. The remaining ones are dilated stages. Dilated ResNet

Dilated bottleneck residual block Standard bottleneck residual block composed of convolutions with kernels $1 \times 1 \mapsto 3 \times 3 \mapsto 1 \times 1$ where the middle 3×3 convolution is a dilated convolution.

Dilated stage Given a dilation rate r, a dilated stage is built as follows:

- The first bottleneck block has stride 1 (instead of 2) and dilation rate $\frac{r}{2}$.
- The remaining blocks have stride 1 and dilation rate r.



(a) ResNet with standard stages

(b) ResNet with two dilated stages

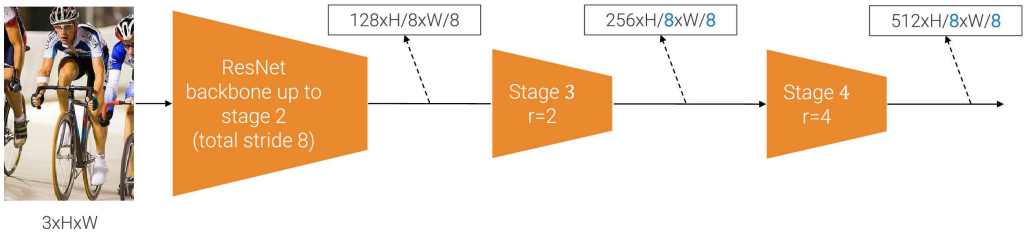


Figure 5.6: Dilated ResNet with total stride 8

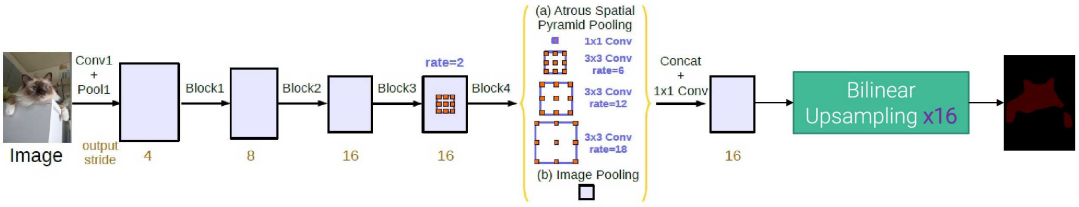
Remark. In principle, it is possible to process the input image without altering the resolution. However, this approach is computationally more expensive as the activations are larger. Therefore, the first standard ResNet stages are kept to apply some stride. As in FCN, a total stride of 8 (up to the second stage) or 16 (up to the third stage) is used.

DeepLab v3 Architecture based on dilated ResNet that considers objects at multiple scales. DeepLab v3

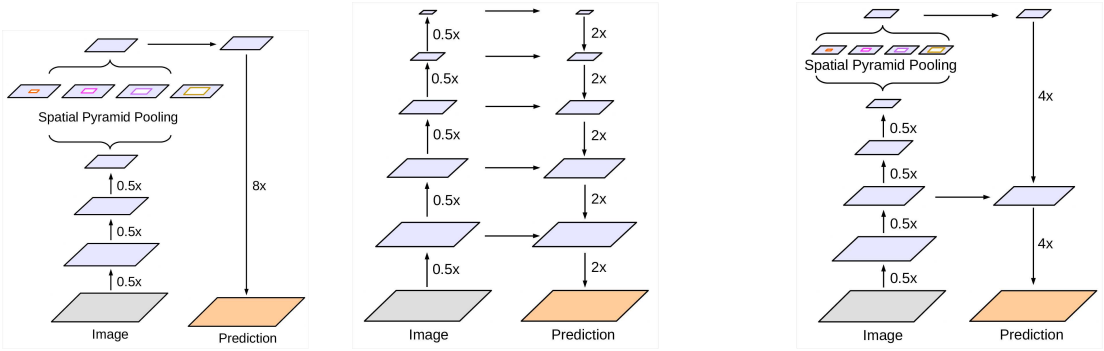
Atrous spatial pyramid pooling (ASPP) Module that takes as input the activation of the dilated ResNet backbone and applies separate 3×3 convolutions with large dilation rates to encode spatial context. The final output is the concatenation of the activations of each convolution. Atrous spatial pyramid pooling (ASPP)

Curriculum training Training is done in two steps:

1. Train with a total stride of 16 (i.e., only the last ResNet stage is dilated). This allows for a faster training with larger batches.
2. Fine-tune with a total stride of 8 (i.e., the last two stages are dilated). This results in spatially larger and more detailed activations.



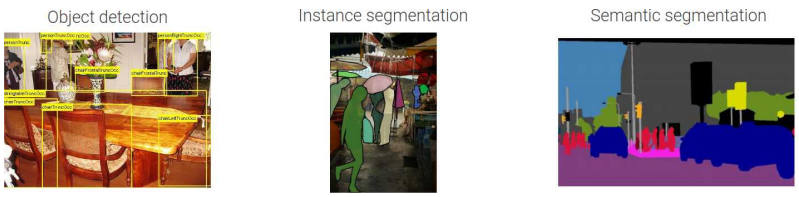
DeepLab v3+ Architecture based on the ASPP of DeepLab v3 and the decoder of U-Net. DeepLab v3+



(a) DeepLab v3 (b) U-Net (c) DeepLab v3+

5.2 Instance segmentation

Instance segmentation Task of segmenting and classifying all instances of the objects of interest (i.e., intersection between object detection and semantic segmentation). Instance segmentation

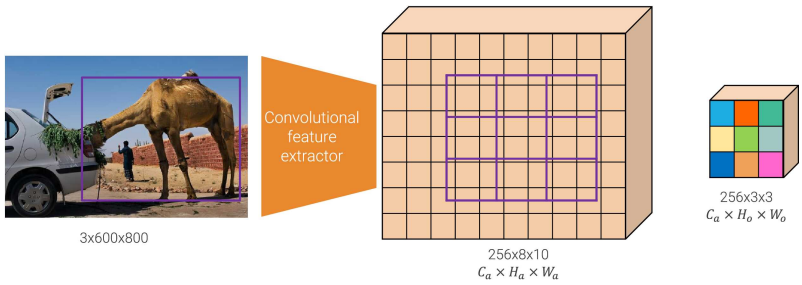


5.2.1 Mask R-CNN

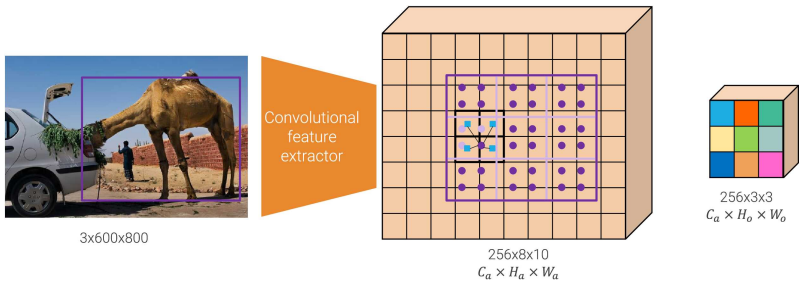
Mask R-CNN Architecture based on faster R-CNN with a modification to RoI pool and the addition of a CNN head to predict the segmentation mask. Mask R-CNN

RoI align Modification to RoI pool to avoid quantization. It works as follows: RoI align

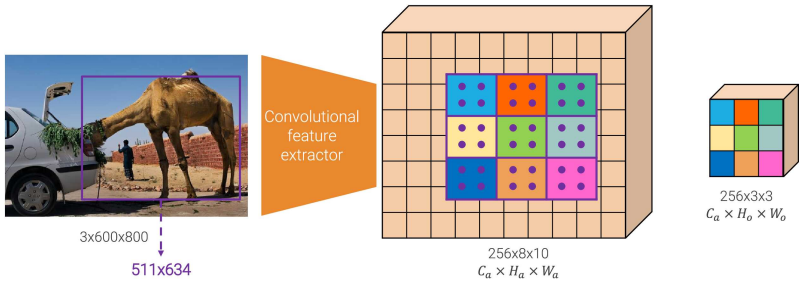
1. Divide the proposal into equal subregions without snapping to grid.



2. Sample some values following a regular grid within each subregion. Use bilinear interpolation to determine the values of the sampled points (as they are most likely not pixel-perfect).



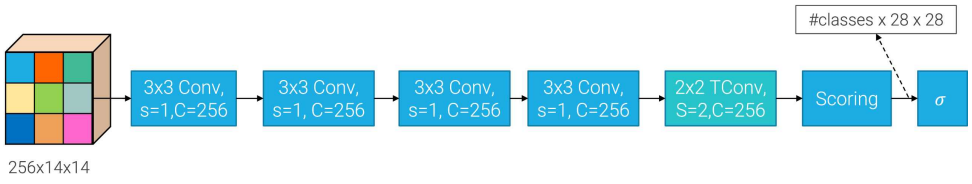
3. Max or average pool the sampled values in each subregion.



Mask head Fully-convolutional network that takes the output of RoI align and predicts a binary mask with resolution 28×28 for each class. It is composed of convolutions, a transposed convolution, and a scoring layer. Mask head

In practice, the bounding box predicted by the standard R-CNN flow is used to determine how to warp the segmentation mask onto the original image.

> **Remark.** Empirical experiments show that solving a multi-label problem (i.e., multiple sigmoids) is better than a multi-class problem (i.e., softmax).



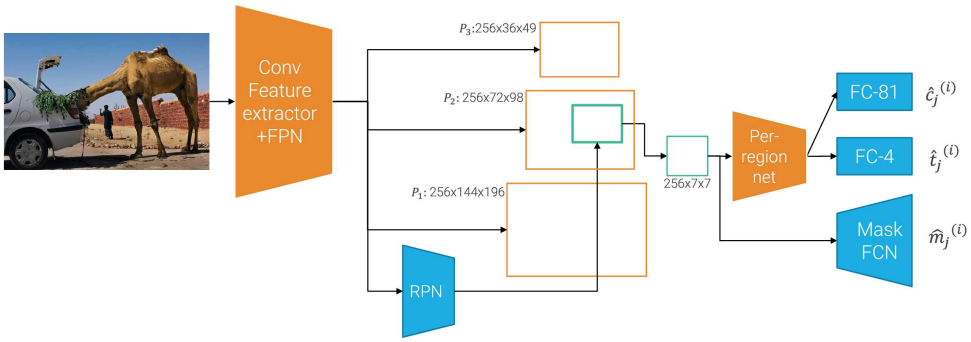


Figure 5.8: Overall architecture of mask R-CNN

Training The R-CNN flow is trained in the standard way.

Given the ground-truth class c and mask m, and the predicted mask $\hat{m}$, the mask head is trained using the following loss:

$$\mathcal{L}_{\text{mask}}(c, m) = \frac{1}{28 \times 28} \sum_{u=0}^{27} \sum_{v=0}^{27} \mathcal{L}_{\text{BCE}}\left(\hat{m}_c[u, v], m[u, v]\right)$$

In other words, the ground-truth mask is compared against the predicted mask at the correct class.

Inference The class prediction of R-CNN is used to select the correct channel of the mask head. The bounding box of R-CNN is used to decide how to warp the segmentation mask onto the image.

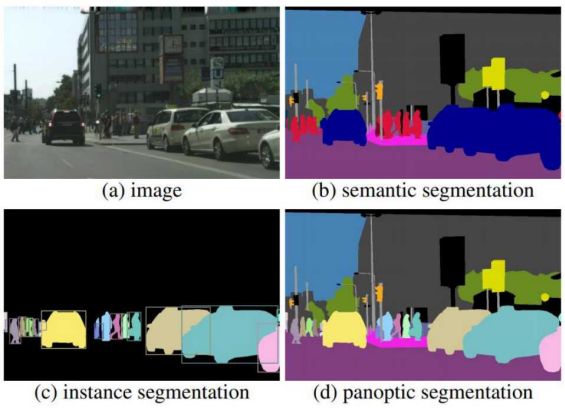
Remark. By attaching a different head to predict keypoints, this framework can be adapted for human pose estimation.

5.3 Panoptic segmentation

Things Countable object categories that can be split into individual instances. Things

Stuff Uncountable amorphous regions (e.g., background) Stuff

Panoptic segmentation Task of classifying each pixel of the image. Objects of interest (i.e., things) are treated as in instance segmentation. Background and non-relevant textures (i.e., stuff) are treated as in semantic segmentation. Panoptic segmentation



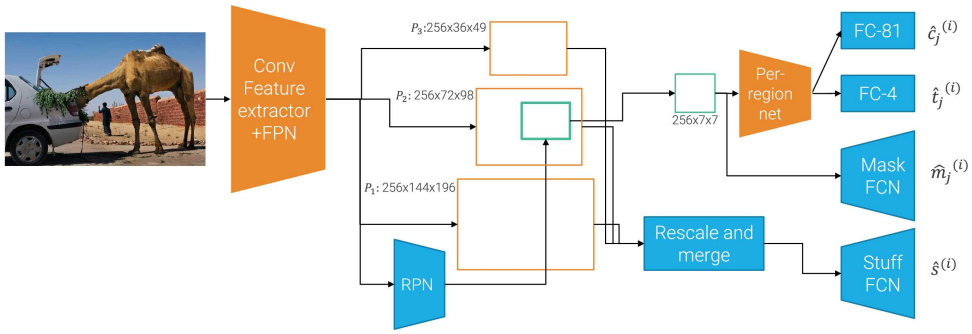
5.3.1 Panoptic feature pyramid network

Panoptic feature pyramid network Modification of mask R-CNN with an additional path for stuff prediction that works as follows:

Panoptic feature pyramid network

1. Rescale and merge FPN feature maps.
2. Pass the result through a fully-convolutional network to predict stuff masks.

When generating the predictions, the mask head (for things prediction) has the priority over the stuff head.



5.3.2 MaskFormer

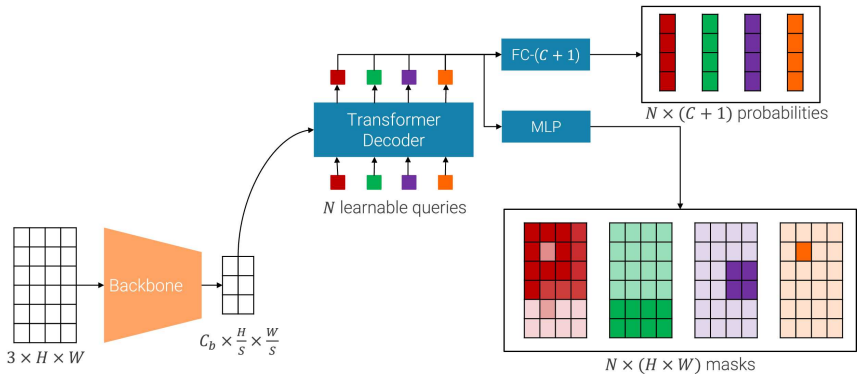
Remark. With convolutions, semantic segmentation has been solved by classifying each pixel into a class. On the other hand, instance and panoptic segmentation partition the image into masks and classify them.
It is intuitive to show that solving a mask classification problem can solve all types of segmentation.

MaskFormer Modification of DETR for pixel-wise predictions.

MaskFormer

Architecture (naive) An additional path is inserted at the output of the transformer decoder that passes each of its output into a multi layer perceptron to predict a binary mask.

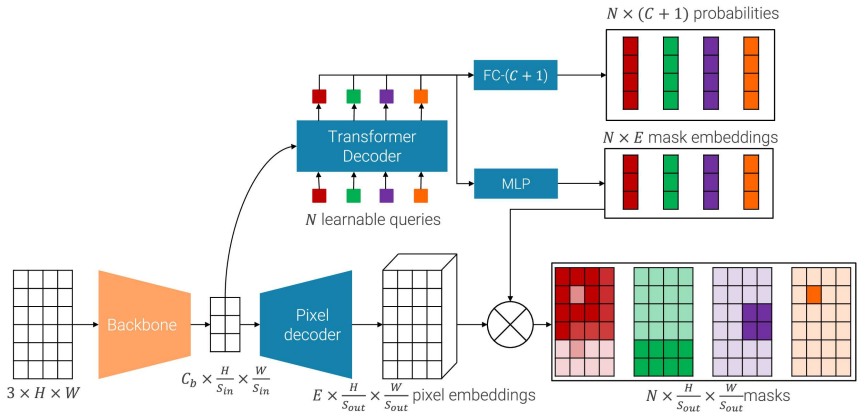
Remark. This approach attempts to predict full-resolution masks from spatially coarse features.



Architecture (pixel decoder) The following operations are added:

1. A pixel decoder is used to compute pixel-wise embeddings from the output of the CNN backbone.

2. An MLP is added to compute mask embeddings from the outputs of the transformer decoder.
3. The dot product between mask and pixel embeddings allows to compute the output binary masks.



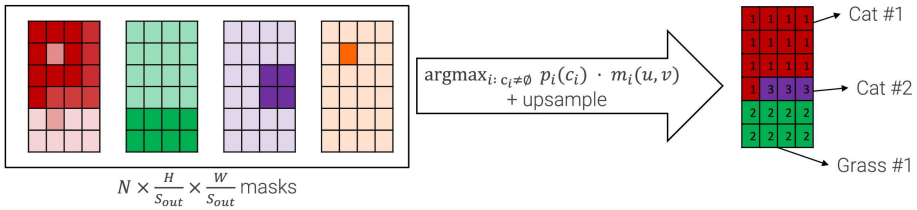
Inference Given the output class probabilities $\mathbf{p}_1, \dots, \mathbf{p}_N$ and masks $\boldsymbol{m}_1, \dots, \boldsymbol{m}_N$, inference is done as follows:

1. Determine the class c_i of the i-th mask from the distribution $\mathbf{p}_i$.
2. Classify each pixel (u, v) as follows:
   a) For each mask i that is not background, compute the following score:

$$s_i = \mathbf{p}_i(c_i) \cdot \boldsymbol{m}_i(u, v)$$

   b) Select the mask with the highest score s_i as the one associated to the pixel.

Remark. Up-sampling might be needed to match the shape of the mask to the original image.



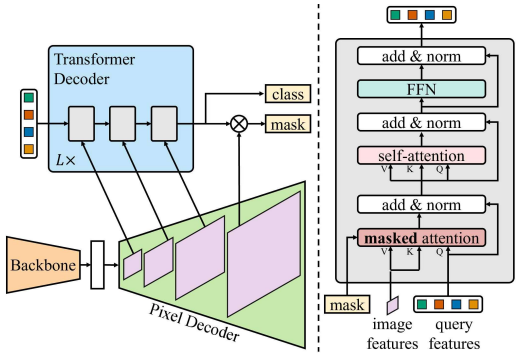
Remark. Although it is able to solve all types of segmentation, MaskFormer does not have state-of-the-art results and it is hard to train.

5.3.3 Masked-attention mask transformer (Mask2Former)

Mask2Former Modification of MaskFormer with the addition of: Mask2Former

- Masked attention in the transformer decoder for faster training and better results.
- Multi-scale high-resolution features in the pixel-decoder for multi-scale detection.

- Training speed-up by not supervising all pixels (i.e., not all pixels of the mask are trained at once, like a sort of dropout).



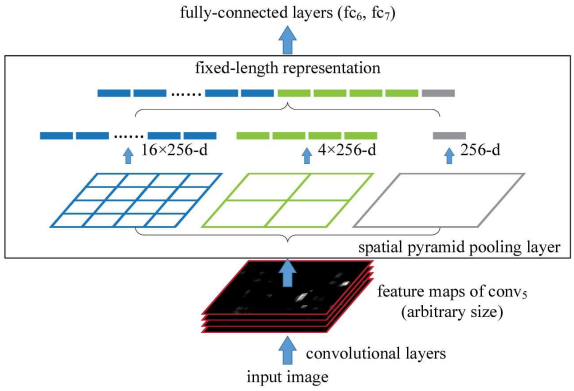
5.A Appendix: Spatial pyramid pooling layer

Spatial pyramid pooling layer (SPP)

Spatial pyramid pooling layer (SPP) Method to obtain a representation with a fixed-resolution containing different scales.

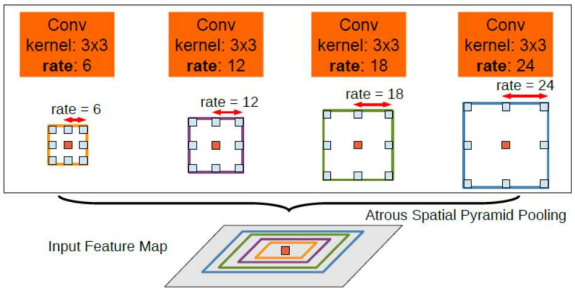
The input image is first passed through the CNN backbone. Then, the feature map is max-pooled with a fixed number of variable-size windows before being flattened.

Remark. This can be seen as an extension of global average pooling to preserve spatially localized information.

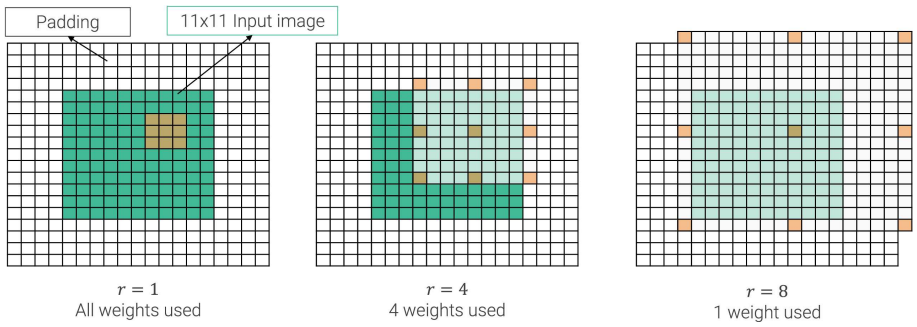


5.A.1 DeepLab v2 ASPP

ASPP in DeepLab v2 emulates SPP through dilated convolutions with increasing rate that are concatenated and aggregated through a 1×1 convolution.



Remark. When the dilation rate grows, the actual number of relevant weights that are not applied to the padding decreases.



5.A.2 DeepLab v3 ASPP

ASPP on DeepLab v3 is formed by dilated convolutions and a path that computes global image features to emulate larger dilation rates (i.e., avoid wasting computation on padding with dilated convolutions with a large rate).

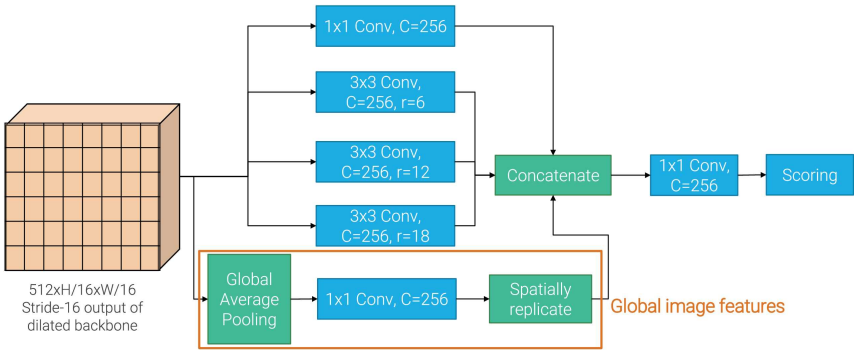


Figure 5.9: ASPP with stride-16. With stride-8, rates are doubled.

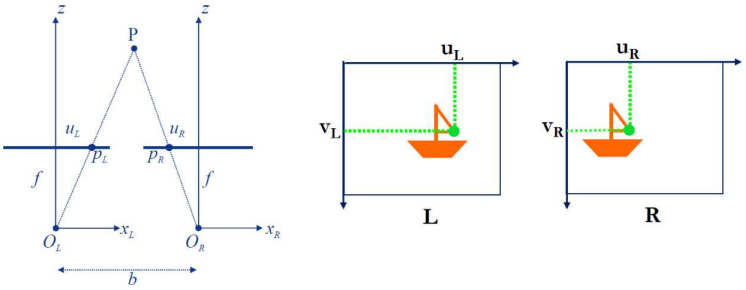
6 Depth estimation

Stereo correspondence Given an ideal stereo setup, the depth of each point in the world can be obtained by solving a correspondence problem along rows. Given two points (u_L, v_L) and $(u_R, v_R = v_L)$ in the left and right image, respectively, representing the projection of the same 3D point, its distance z from the camera can be determined using the disparity d:

Stereo correspondence

$$d = u_L - u_R$$
$$z = \frac{bf}{d}$$

where b is the baseline (i.e., camera distance) and f is the focal length.



Remark. Due to the lack of data, existing models are trained on synthetic data and fine-tuned afterwards on real data.

6.1 Monocular depth estimation

Monocular (single-view) depth estimation Reconstruct the 3D structure of a scene from a single image.

Monocular (single-view) depth estimation

Remark. In principle, this is an ill-posed problem. However, humans are able to solve it through learning.

Remark. Traditional supervised frameworks (e.g., encoder-decoder) to solve monocular depth estimation have some limitations:

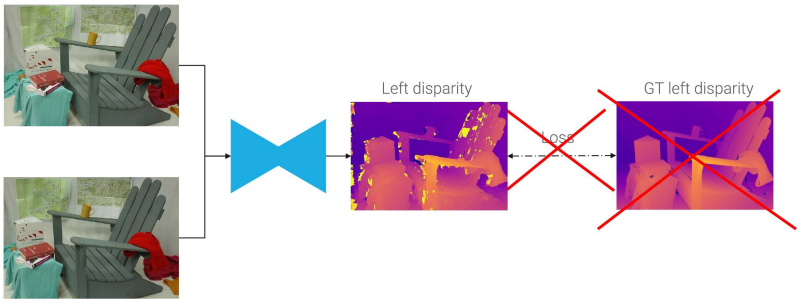
- They require a large amount of realistic synthetic data.
- They require expensive hardware for depth measurement when fine-tuning on real data.

6.1.1 Monodepth

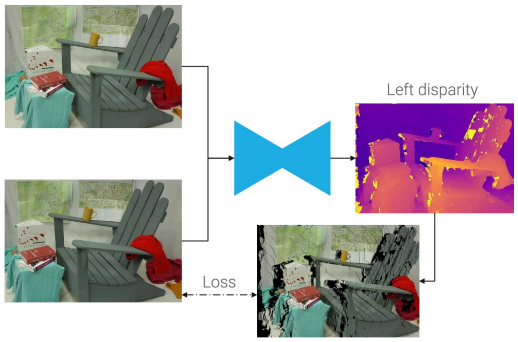
Supervised stereo pipeline

Supervised stereo pipeline

Naive approach A possible solution for depth estimation is to feed a CNN with a pair of synchronized images and make it predict the left (or right) disparity. However, this approach requires to know the ground-truth disparity, which is expensive to obtain.



Reconstruction approach Make the model predict the left disparity which is then used to reconstruct the right image. This works as a pixel (u, v) in the left image with disparity $\tilde{d}$ should appear as the same to the pixel $(u + \tilde{d}, v)$ in the right image.



Monodepth (no left-right) Network that takes as input the left (or right) image of a stereo vision system and predicts the left (or right) disparity.

Monodepth (no left-right)

Training (naive) Once the left disparity has been predicted, it is used to reconstruct the right image.

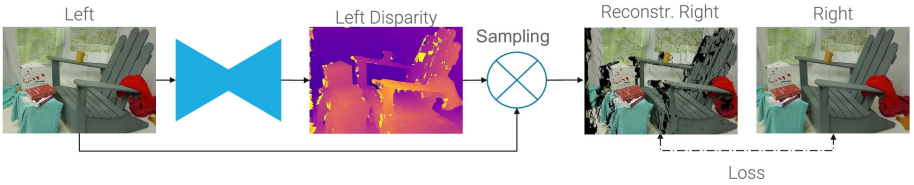


Figure 6.1: Naive training flow

Remark. Forward mapping creates holes and is ambiguous as disparities are non-integer values.

(Backward) bilinear sampling Compute the right image by determining each pixel of the output backwards and by interpolating it in the left image.

Remark. By reconstructing the right image backwards, the estimated disparity will be with respect to the right image, which is not available during inference.

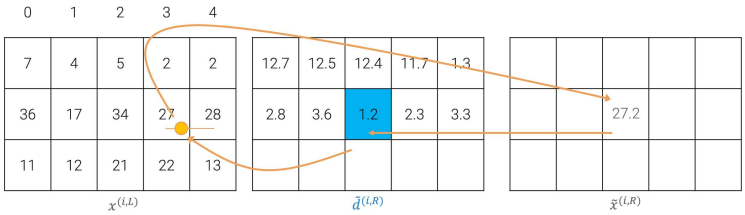


Figure 6.2: Backward reconstruction from the right image

Training (correct) Once the left disparity has been predicted, it is used to reconstruct the left image by backward mapping from the right image (which is available at train time).

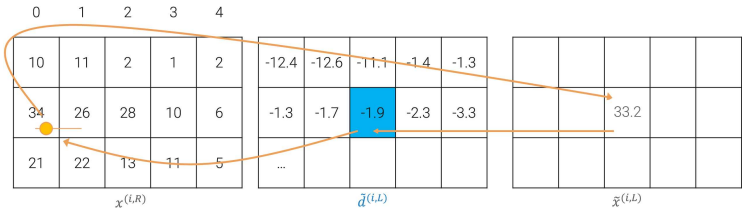


Figure 6.3: Backward reconstruction from the left image

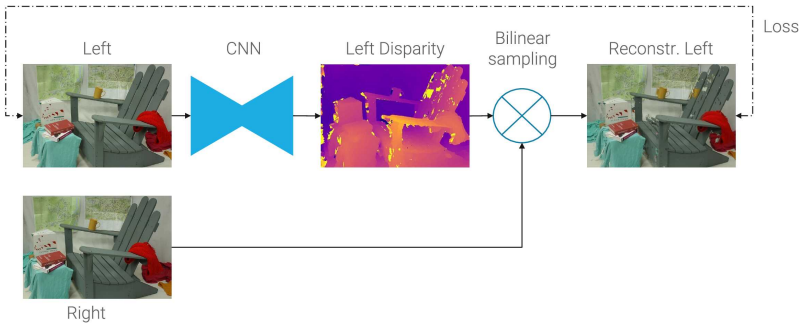


Figure 6.4: Actual training flow

Reconstruction loss Mix between the structural similarity index (SSIM) (which measures a perceptual distance) and L1 norm: Reconstruction loss

$$\mathcal{L}_{\text{ap}}(x^{(i,L)}) = \frac{1}{N}\sum_{(u,v)}\left(\alpha\frac{1-\texttt{SSIM}(x^{(i,L)}_{u,v},\hat{x}^{(i,L)}_{u,v})}{2} + (1-\alpha)\left\|x^{(i,L)}_{u,v}-\hat{x}^{(i,L)}_{u,v}\right\|_1\right)$$

where $x^{(i,L)}$ is the i-th input left image and $\hat{x}^{(i,L)}$ the reconstructed left image.

Disparity smoothness Loss penalty to exploit the fact that disparity tends to be locally smooth and only change at edges: Disparity smoothness

$$\mathcal{L}_{\text{ds}}(x^{(i,L)}) = \frac{1}{N}\sum_{(u,v)}\left(\left|\partial_u d^{(i,L)}_{u,v}\right| e^{-\|\partial_u x^{(i,L)}_{u,v}\|_1} + \left|\partial_v d^{(i,L)}_{u,v}\right| e^{-\|\partial_v x^{(i,L)}_{u,v}\|_1}\right)$$

where $x^{(i,L)}$ is the i-th input left image and $d^{(i,L)}$ the predicted left disparity.

In this way:

- If the gradient of $x_{u,v}^{(i,L)}$ is small (i.e., $e^{-\|\partial_* x_{u,v}^{(i,L)}\|_1} \to 1$), the gradient of $d_{u,v}^{(i,L)}$ is forced to be small too.
- If the gradient of $x_{u,v}^{(i,L)}$ is big (i.e., $e^{-\|\partial_* x_{u,v}^{(i,L)}\|_1} \to 0$), the gradient of $d_{u,v}^{(i,L)}$ can be indifferently large or small.

Remark. Monodepth without left-right processing has fairly good results but it exhibits texture-copy artifacts and errors at depth discontinuities.

Remark. Monodepth in this form requires stereo images but only exploits one of the images.

Monodepth (left-right)

Monodepth (left-right) Make the network predict both left and right disparity and reconstruct both left and right images.

Disparity consistency loss

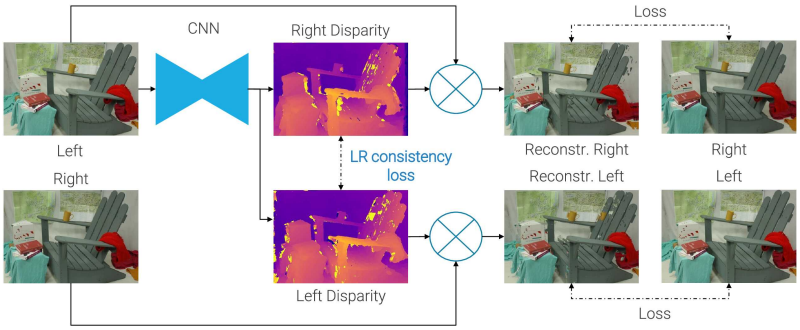
Disparity consistency loss Enforces that the shifts of the two estimated disparities are consistent:

$$\mathcal{L}_{\text{lr}}(x^{(i,L)}, x^{(i,R)}) = \frac{1}{N}\sum_{(u,v)} \left| d_{u,v}^{(i,L)} - d_{u+d_{u,v}^{(i,L)},v}^{(i,R)} \right| + \frac{1}{N}\sum_{(u,v)} \left| d_{u+d_{u,v}^{(i,R)},v}^{(i,L)} - d_{u,v}^{(i,R)} \right|$$

where $d^{(i,L)}$ and $d^{(i,R)}$ are the i-th left and right predicted disparity, respectively.

The overall loss is the following:

$$\begin{aligned}\mathcal{L}(x^{(i,L)}, x^{(i,R)}) = &\alpha_{\text{ap}} \left(\mathcal{L}_{\text{ap}}(x^{(i,L)}) + \mathcal{L}_{\text{ap}}(x^{(i,R)}) \right) \\ &+ \alpha_{\text{ds}} \left(\mathcal{L}_{\text{ds}}(x^{(i,L)}) + \mathcal{L}_{\text{ds}}(x^{(i,R)}) \right) \\ &+ \alpha_{\text{lr}} \mathcal{L}_{\text{lr}}(x^{(i,L)}, x^{(i,R)})\end{aligned}$$



Inference Use the left disparity to determine depth. Everything else is not necessary.

Architecture Monodepth is implemented as a U-Net like network:

- Up-convolutions are substituted with bilinear up-sampling to avoid checkerboard artifacts.
- Disparity maps are computed at several resolutions and processed by the loss for alignment reason.

| **Remark.** It has been seen that better encoders help performance.

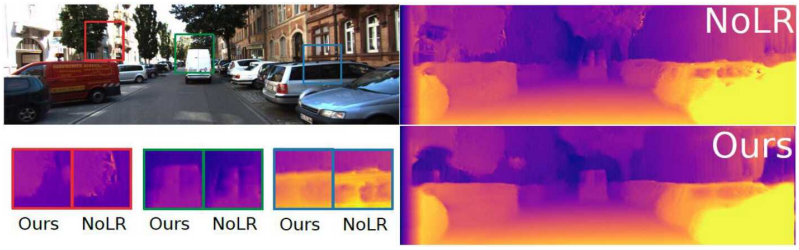


Figure 6.5: Comparison of Monodepth with and without left-right processing

6.1.2 Structure from motion learner

Structure from motion learner (SfMLearner)

Structure from motion learner (SfMLearner) Relaxes the assumption of stereo images by using monocular video frames.

The network takes as input a target image and nearby image(s) and is based on two flows:

Depth CNN Takes as input the target image and estimates its depth map.

Pose CNN Takes as input the target and nearby images, and estimates the camera poses to project from target to nearby image.

The outputs of both networks are used to reconstruct the target image and a reconstruction loss is used for training.

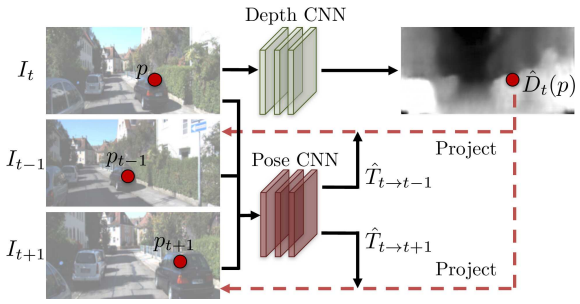


Figure 6.6: SfMLearner with two nearby images

6.1.3 Depth Pro

Depth Pro

Depth Pro Extension of SfMLearner trained using more datasets.

7 Metric learning

Metric learning Task of training a network that produces discriminative embeddings (i.e., with a clustered structure) such that: Metric learning

- The distance of related objects (i.e., intra-class distance) is minimized.
- The distance of different objects (i.e., inter-class distance) is maximized.

7.1 Face recognition

Face recognition Given a database of identities, classify a query face. Face recognition

Open-world setting System where it is easy to add or remove identities.

7.1.1 Face recognition as classification

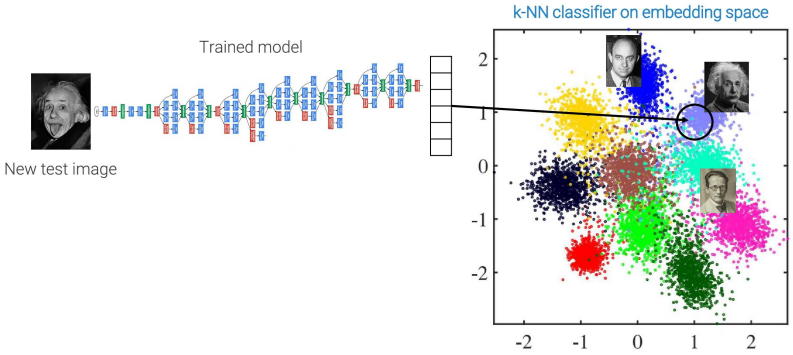
Plain classifier Consider each identity as a class and use a CNN with a softmax head to classify the input image. Plain classifier

> **Remark.** This approach requires a large softmax and do not allow adding or removing identities without retraining.

kNN classifier Use a feature extractor to embed faces and use a kNN classifier to recognize it. kNN classifier

Gallery Set of embeddings of known identities. Gallery

> **Remark.** This approach allows to easily add or remove new identities.



> **Remark.** Feature extractors for classification are trained using the cross-entropy loss to learn semantically rich embeddings. However, when classifying, these embeddings are passed through a final linear layer. Therefore, it is sufficient that they are linearly separable.
> In other words, the distance between elements of the same class can be arbitrarily large and the distance between different classes can be arbitrarily small, as long as they are linearly separable.

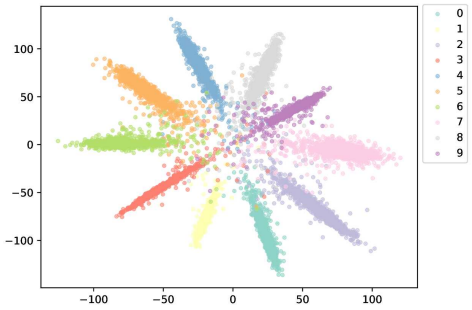


Figure 7.1: MNIST embeddings in 2D

7.2 Metric learning losses

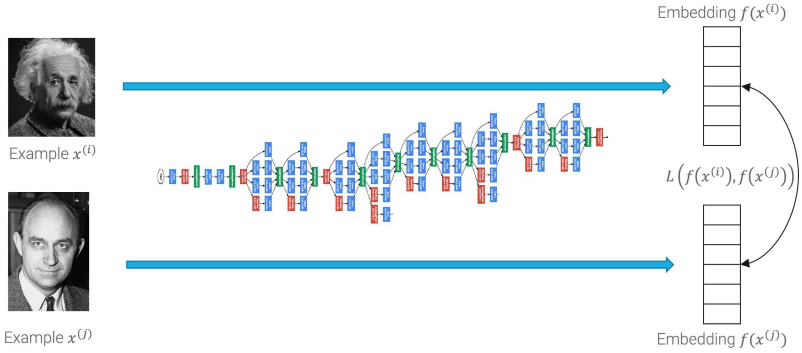
Face verification Task of confirming that two faces represent the same identity. This problem can be solved by either: Face verification

- Using better metrics than the Euclidean distance (e.g., as done in DeepFace).
- Using better embeddings (e.g., as done in DeepID or FaceNet).

Remark. This task can be used to solve face recognition.

7.2.1 Contrastive loss

Siamese network training Train a network by comparing its outputs from two different inputs. This can be virtually seen as training two copies of the same network with shared weights. Siamese network training



Contrastive loss Loss to enforce clustered embeddings. It is defined as follows: Contrastive loss

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

As the second term is not lower-bounded (i.e., it can be arbitrarily small), a margin m is included to prevent pushing different classes too far away:

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

Remark. A margin m^+ can also be added to the positive branch to prevent collapsing all embeddings of the same class to the same point.

Remark. The negative branch $\max\left\{0, m - \|f(x^{(i)}) - f(x^{(j)})\|_2\right\}$ is the hinge loss, which is used in SVM.

Remark. With L2 regularization, the fact that the second term is not lower-bounded is not strictly a problem as the weights lay on a hyper-sphere and therefore set a bound to the output. Still, there is no need to push the embeddings excessively far away.

DeepID2 CNN trained using contrastive and cross-entropy loss. DeepID2

Test-time augmentation The trained network is used as follows:

1. Process 200 fixed crops of the input image.
2. Select the top-25 crops.
3. Use PCA to reduce the concatenation of the output 4000-dimensional vector into a 180-dimensional embedding.

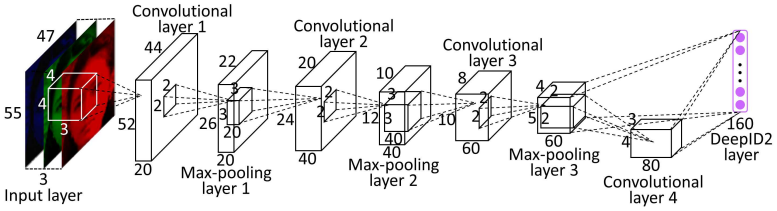


Figure 7.2: DeepID2 on a single crop

Remark. Contrastive loss indirectly (i.e., separately) enforces that similar entities should be close and different ones should be distant.

7.2.2 Triplet loss

Triplet loss Enforces at the same time that embeddings should be close for the same entity and distant for different ones. Triplet loss

Naive formulation Given:

- An anchor image A (i.e., reference image),
- A positive image P,
- A negative image N,

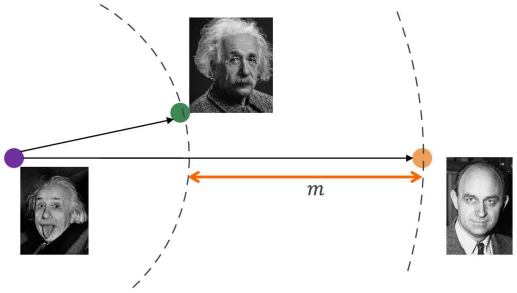
The triplet loss is defined as:

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

Remark. This definition does not guarantee sufficient inter-class distance and also risks training collapse (i.e., constant embeddings close to 0 satisfy the loss).

Margin formulation Given a margin m, the triplet loss is defined as:

$$\|f(P) - f(A)\|_2^2 + m < \|f(N) - f(A)\|_2^2$$
$$\|f(P) - f(A)\|_2^2 - \|f(N) - f(A)\|_2^2 + m < 0$$
$$\mathcal{L}_{\text{triplet}}(A, P, N) = \max\left\{0, \|f(P) - f(A)\|_2^2 - \|f(N) - f(A)\|_2^2 + m\right\}$$



Remark. Differently from contrastive loss, triplet loss naturally prevents collapse using a single hyperparameter m.

Remark. By considering an anchor, a face recognition dataset is composed of a few positive images and lots of negative images. However, if the embedding of a negative image is already far enough, the loss will be 0 and does not affect training, making convergence slow.

Semi-hard negative mining Given an anchor A, a batch of B samples is formed as follows: Semi-hard negative mining

1. Sample $D \ll B$ identities.
2. For each identity, sample a bunch of images.
3. Consider all possible anchor-positive pairs (A, P). Randomly sample a negative image N and create a triplet (A, P, N) if it is semi-hard (i.e., within the margin):

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

Repeat this process until the batch is filled.

Remark. Batches are formed on-line (i.e., during training).

Remark. It has been seen that hard-negatives ($\|f(P) - f(A)\|_2^2 > \|f(N) - f(A)\|_2^2$) make training harder.

FaceNet Architecture based on the backbone of Inception-v1 with the following modifications: FaceNet

- Some max pooling layers are substituted with L2 pooling.
- The output embedding is divided by its L2 norm to normalize it.

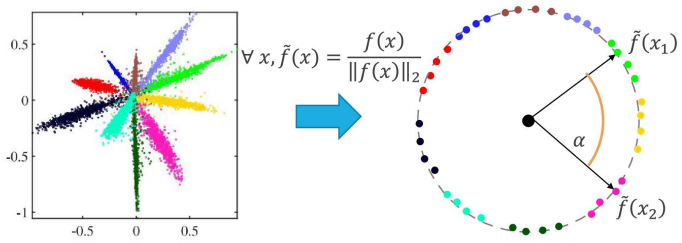
Remark. DeepID2 was trained on a small dataset (CelebFaces+) with test time augmentation. On the other hand, FaceNet was training on a very large dataset (LFW). Ultimately, FaceNet has better performance.

7.2.3 ArcFace loss

Remark. With L2 normalized outputs, the embeddings lay on a hypersphere. Therefore, it is possible to compare them using the cosine similarity:

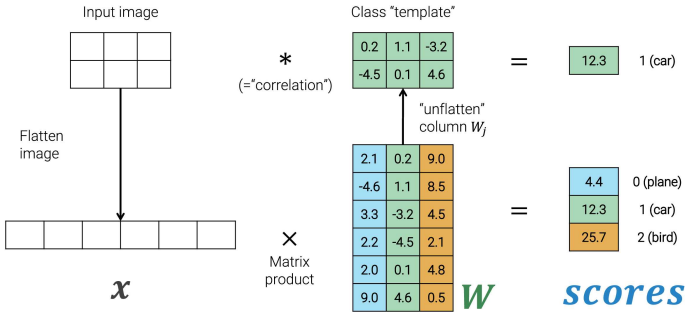
$$\cos(\alpha) = \frac{\langle \tilde{f}(x_1), \tilde{f}(x_2) \rangle}{\|\tilde{f}(x_1)\|_2 \|\tilde{f}(x_2)\|_2} = \frac{\langle \tilde{f}(x_1), \tilde{f}(x_2) \rangle}{1} = \tilde{f}(x_1)^T \tilde{f}(x_2)$$

where $\tilde{f}(x) = \frac{f(x)}{\|f(x)\|}$ is the normalized embedding.



Remark. With normalized embeddings, classification with softmax as pre-training objective becomes more reasonable. The advantage with softmax is that it does not require sampling informative negative examples. However, as is, it is unable to identify well separated cluster.

Remark (Recall: linear classifiers)**.** A linear classifier is equivalent to performing template matching. Each column of the weight matrix represents a template.



ArcFace loss Modification to softmax to enforce clustered classes. ArcFace loss

Consider a classification head with a linear layer added on top of the embedding model during training. The weight matrix $\boldsymbol{W}$ of the linear layer has a column for each identity. If $\boldsymbol{W}$ is L2 normalized ($\tilde{\boldsymbol{W}}$), applying the linear layer is equivalent to computing the cosine similarity between the embedding and each identity template.

(a) Softmax (b) ArcFace

Figure 7.3: Classification results with L2 normalized embeddings. Points represent embeddings and colors are the classes. The points marked by the radii are the templates. Note that with softmax there is no clear distinction between identities.

To enforce the embeddings closer to the correct template, the logit $\mathbf{s}_y$ of the true identity y can be modified with a penalty m:

$$\mathbf{s}_y = \cos\Big(\arccos(\langle \tilde{\boldsymbol{W}}_y, \tilde{f}(x) \rangle) + m\Big)$$

Intuitively, the penalty makes softmax "think" that the error is bigger than what it really is, making it push the embedding closer to the correct template.

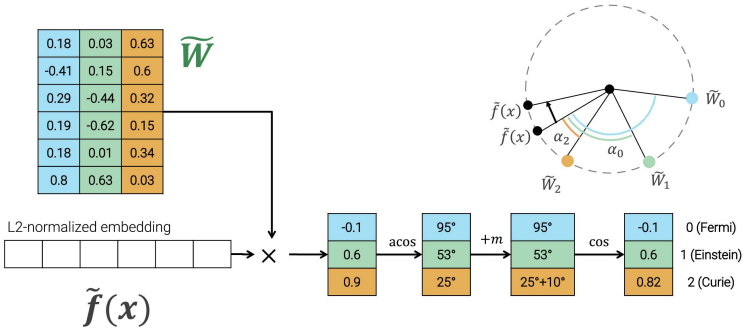


Figure 7.4: Penalty application with `Curie` as the correct class

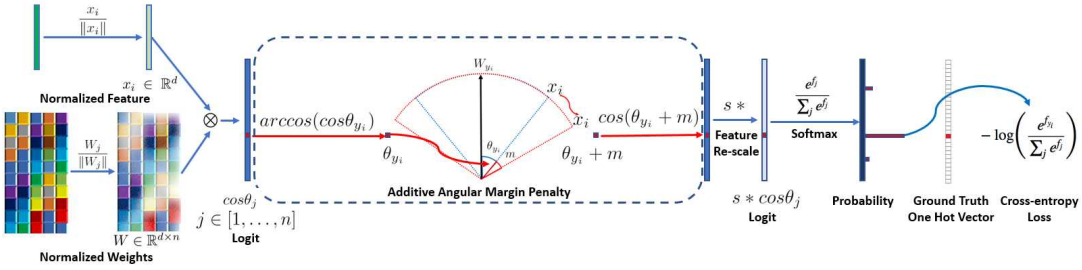


Figure 7.5: Overall ArcFace flow

7.2.4 NT-Xent / N-pairs / InfoNCE loss

Angular triplet loss Consider the triplet loss (without margin) defined in Section 7.2.2 rewritten in angular form: Angular triplet loss

$$\mathcal{L}_{\text{angular\_triplet}}(A,P,N) = \max\left\{0, \tilde{f}(A)^T\tilde{f}(N) - \tilde{f}(A)^T\tilde{f}(P)\right\}$$

To avoid the need of sampling negatives, it is possible to consider all of them in the optimization process as follows:

$$\begin{aligned}&\mathcal{L}_{n+1\text{-tuples}}(A,P,N_1,\dots,N_n)\\&\quad= \max\left\{0, \left(\tilde{f}(A)^T\tilde{f}(N_1) - \tilde{f}(A)^T\tilde{f}(P)\right),\dots,\left(\tilde{f}(A)^T\tilde{f}(N_n) - \tilde{f}(A)^T\tilde{f}(P)\right)\right\}\end{aligned}$$

Remark. Backpropagating `max` considers the gradient of the maximally violating triplet only (i.e., the hardest one which, as seen, is not ideal).

Remark (Recall: cross-entropy with softmax)**.** When minimizing cross-entroy with softmax, the loss for a single sample is computed as:

$$-\log\left(\frac{\exp(\mathbf{s}_{y^{(i)}})}{\sum_{k=1}^{C}\exp(\mathbf{s}_k)}\right) = -\mathbf{s}_{y^{(i)}} + \log\left(\sum_{k=1}^{C}\exp(\mathbf{s}_k)\right)$$

The term $\log(\sum_{k=1}^{C}\exp(\mathbf{s}_k))$ is known as the `logsumexp` and it is usually approximated as the `max` of the logit:

$$\log\left(\sum_{k=1}^{C}\exp(\mathbf{s}_k)\right) \approx \max_k \mathbf{s}_k$$

Or, seen on the other side, `logsumexp` is an approximation of `max`.

N-pairs/InfoNCE loss Use the `logsumexp` as an approximation of `max` so that all negatives contribute to the gradient step: N-pairs/InfoNCE loss

$$\begin{aligned}\mathcal{L}_{\text{InfoNCE}}(A, P, N_1, \dots, N_n) &= \texttt{logsumexp}\left(0, \left(\tilde{f}(A)^T\tilde{f}(N_1) - \tilde{f}(A)^T\tilde{f}(P)\right), \dots, \left(\tilde{f}(A)^T\tilde{f}(N_n) - \tilde{f}(A)^T\tilde{f}(P)\right)\right) \\ &= \log\left(e^0 + \sum_{i=1}^{n}\exp\left(\tilde{f}(A)^T\tilde{f}(N_i) - \tilde{f}(A)^T\tilde{f}(P)\right)\right) \\ &= \log\left(1 + \exp\left(-\tilde{f}(A)^T\tilde{f}(P)\right)\sum_{i=1}^{n}\exp\left(\tilde{f}(A)^T\tilde{f}(N_i)\right)\right) \\ &= \log\left(\frac{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right)}{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right)}\left[1 + \exp\left(-\tilde{f}(A)^T\tilde{f}(P)\right)\sum_{i=1}^{n}\exp\left(\tilde{f}(A)^T\tilde{f}(N_i)\right)\right]\right) \\ &= \log\left(\frac{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right) + \sum_{i=1}^{n}\exp\left(\tilde{f}(A)^T\tilde{f}(N_i)\right)}{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right)}\right) \\ &= -\log\left(\frac{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right)}{\exp\left(\tilde{f}(A)^T\tilde{f}(P)\right) + \sum_{i=1}^{n}\exp\left(\tilde{f}(A)^T\tilde{f}(N_i)\right)}\right)\end{aligned}$$

Remark. In the literature, this loss is known as N-pairs (2016) or InfoNCE (2018).

NT-Xent loss Add a temperature τ to N-pairs/InfoNCE loss so that more negatives are relevant when backpropagating: NT-Xent loss

$$\mathcal{L}_{\text{NT-Xent}}(A, P, N_1, \dots, N_n) = -\log\left(\frac{\exp\left(\frac{\tilde{f}(A)^T\tilde{f}(P)}{\tau}\right)}{\exp\left(\frac{\tilde{f}(A)^T\tilde{f}(P)}{\tau}\right) + \sum_{i=1}^{n}\exp\left(\frac{\tilde{f}(A)^T\tilde{f}(N_i)}{\tau}\right)}\right)$$

Remark. This loss is sometimes identified as the contrastive loss.

Remark. Empirical studies found out that the different metric learning losses all perform similarly with fixed hyperparameters while avoiding test set feedback (i.e., avoid leaking test data into training).

7.3 Zero-shot classification

Zero-shot classification Classify images from the test set of a dataset without training on its training set. Zero-shot classification

Remark. Natural language supervision works well for zero-shot classification by connecting the representation of images and texts. An easy way to obtain image-text pairs is to use the `alt` tag of HTML images (i.e., the description of the image used by screen readers).

7.3.1 Contrastive language-image pre-training (CLIP)

Contrastive language-image pre-training (CLIP) Network composed of: Contrastive language-image pre-training (CLIP)

Text encoder Transformer encoder where the `[EOS]` token is used as the representation of the sequence.

Image encoder ResNet with global average pooling or ViT where the `[CLS]` token is used as representation.

A linear projection is used to match the shapes of the two encoders.

Training Given a batch of text-image pairs $(t_1, i_1), \dots, (t_N, i_N)$, texts and images are processed by their respective encoders. The embeddings T_j, I_j are then compared pairwise: text and image embeddings corresponding to the same entity are considered a positive class, otherwise, they represent a negative class. NT-Xent loss is computed across the images by fixing a text or vice versa.

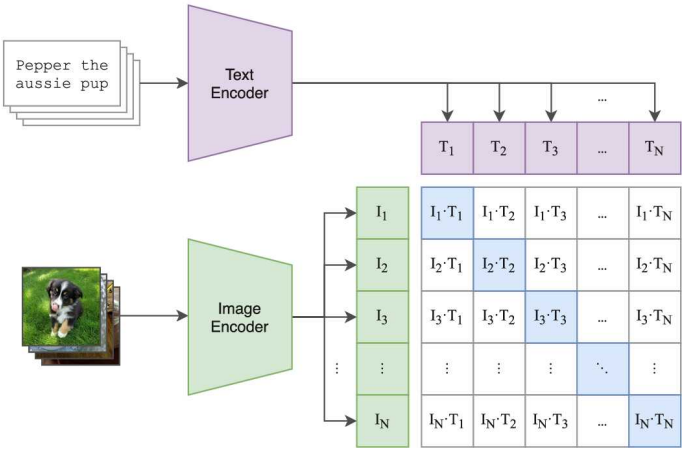


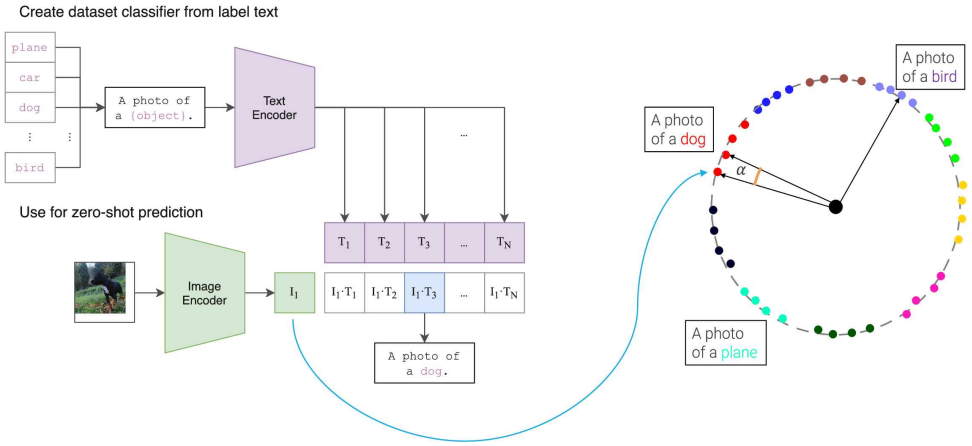
Figure 7.6: CLIP training flow. NT-Xent loss is applied column or row-wise in the dot product matrix.

Remark. It has been seen that a bag-of-words approach for the text encoder is better than using transformers.

Remark. As the NT-Xent loss is used, a large batch size is used.

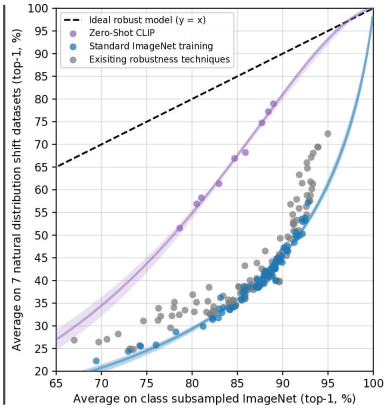
Remark. CLIP with ViT-L/14 is the "standard" version. It has been pre-trained for an additional epoch on images with a higher resolution, similarly to FixRes (i.e., deal with different train and test resolutions).

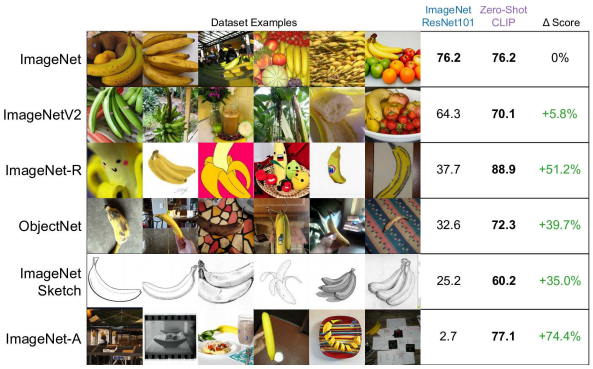
Inference Given an image to classify, it is embedded and compared with the embeddings of prompts referencing the classes (e.g., `a photo of a [object]`). The closest one is considered as the predicted class.



Remark. CLIP features are robust to distribution shifts (i.e., the network does not take "shortcuts" when classifying).

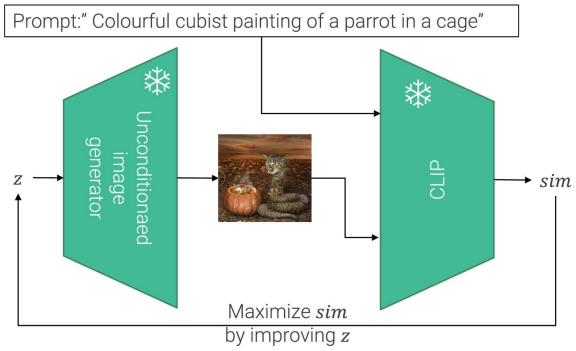
Example. As lesions in x-ray images are marked by the doctors, a network trained to detect lesions might actually learn to predict the mark put by the doctors.





Remark (CLIP for text-conditioned generative models)**.** CLIP can be used as a loss for a generative model to condition generation based on a prompt. Given a desired prompt, the steps are the following:

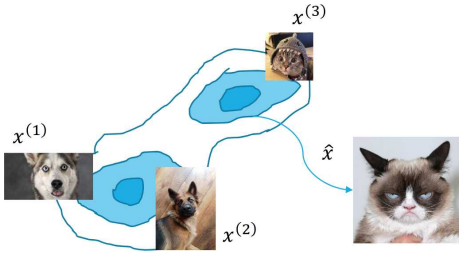
1. Freeze both the generator and CLIP.
2. Feed the generator a random input z to generate an image.
3. Embed the generated image with CLIP and compare it to the embedding of the prompt.
4. Gradient ascent is used to improve z aiming to maximize the similarity.



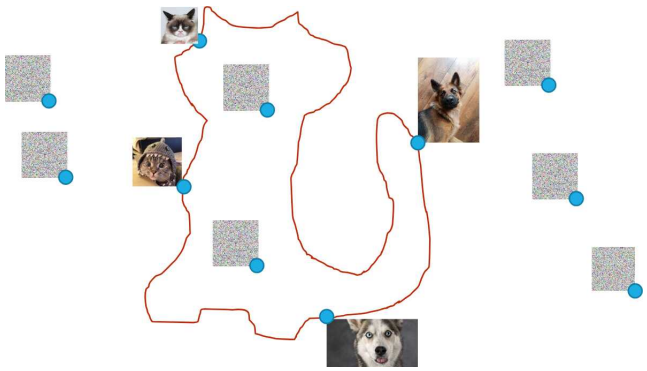
8 Generative models

Generative task Given the training data $\{x^{(i)}\}$, learn its distribution so that a model can sample new examples: Generative task

$$\hat{x}^{(i)} \sim p_{\text{gen}}(x; \boldsymbol{\theta})$$



Remark. Generative tasks are hard as natural images lay on a low dimensional subspace (i.e., only a tiny subset of all the possible RGB images makes sense).

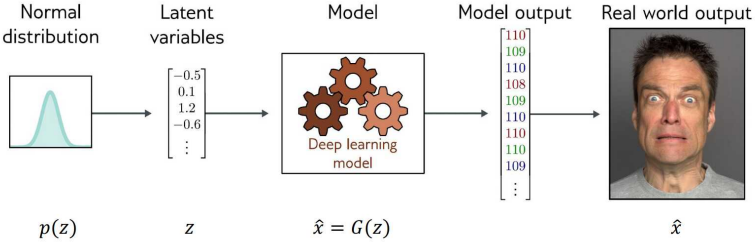


Latent vector Low-dimensional representation to encode an image. Latent vector

Example. Face expression depends on 42 muscles. A latent representation for different poses of a face can be represented with a 42-dimensional vector.

It is assumed that the factors of a latent vector are independent or mildly correlated, and can be sampled from a known distribution.

Generative model Model that takes as input a latent representation and maps it into an output image. Generative model



Remark. An ideal generative model should have the following properties:

- Be computationally efficient when sampling.
- Produce high-quality samples.
- Represent the entire training distribution.
- Produce a plausible output from any latent input. Smooth changes to the input should be reflected on the output.
- Have a disentangled latent space (i.e., changing a dimension of the latent space corresponds to interpretable changes in the output image).
- Have the possibility to calculate the probability of the produced images (when the model is probabilistic).

8.1 Metrics

Expectation/Expected value

Expectation/Expected value Informally, it is the generalization of the weighted average:

$$\mathbb{E}_{x\sim p}[f(\cdot)] = \sum_{x\in\mathbb{X}} p(x)f(x) \qquad \mathbb{E}_{x\sim p}[f(\cdot)] = \int_{x\in\mathbb{X}} p(x)f(x)\,dx$$

Monte Carlo approximation

Monte Carlo approximation Approximation for expectation using N i.i.d. samples drawn from $p(x)$:

$$\mathbb{E}_{x\sim p}[f(\cdot)] \approx \frac{1}{N}\sum_{x_i\sim p(x)} f(x_i)$$

Self-information

Self-information Given a probability mass function of an event, the self-information of an event x is defined as:

$$I(x) = -\log_b(p(x)) = \log_b\left(\frac{1}{p(x)}\right)$$

Intuitively, it can be seen as a measure of surprise.

Remark. If $b = 2$, self-information is measured in bits. If $b = e$, it is measured in natural unit of information (*nat*).

Example. Consider the toss of a fair coin. The self-information for the outcomes are:

$$I(\texttt{heads}) = I(\texttt{tails}) = \log_2\left(\frac{1}{0.5}\right) = 1 \text{ bit}$$

If the coin is loaded toward tails with probability:

$$\mathcal{P}(\texttt{heads}) = 0.05 \qquad \mathcal{P}(\texttt{heads}) = 0.95$$

The self-information is:

$$I(\texttt{heads}) = \log_2\left(\frac{1}{0.05}\right) = 4.31 \text{ bits} \qquad I(\texttt{tails}) = \log_2\left(\frac{1}{0.95}\right) = 0.07 \text{ bits}$$

Entropy

Entropy Expected value of the self-information of a probability mass function:

$$H(p(\cdot)) = \mathbb{E}_{x\sim p}\left[-\log(p(x))\right] \approx -\sum_{x\in\mathbb{X}} p(x)\log(p(x))$$

Intuitively, it measures the average surprise of a distribution.

Example. Consider the distribution of a fair, loaded, and constant coin. We have that:

Maximum entropy $H(p_{\text{fair}}(\cdot)) = 0.5\log(2) + 0.5\log(2) = \log(2)$

Low entropy $H(p_{\text{loaded}}(\cdot)) = 0.05 \cdot 4.32 + 0.95 \cdot 0.07 = 0.28$

Minimum entropy $H(p_{\text{constant}}(\cdot)) = 0 + 1 \cdot 0 = 0$

Kullback-Leibler (KL) divergence Distance between probability distributions:

Kullback-Leibler (KL) divergence

$$D_{\text{KL}}(p||q) = \mathbb{E}_{x\sim p(x)}\left[\log(p(x)) - \log(q(x))\right] = \mathbb{E}_{x\sim p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right]$$
$$\approx \sum_{x\in\mathbb{X}}\left(p(x)\log\left(\frac{p(x)}{q(x)}\right)\right)$$

Remark. KL divergence only samples from one of the distributions. Therefore, it is not symmetric (i.e., it is not a metric).

Remark. KL divergence goes to infinity if $q(x) = 0$ and $p(x) \neq 0$.

Example. Consider the following cases of a coin toss:

	Other distribution is loaded	Other distribution is constant
p is fair	$0.5\log\left(\frac{0.5}{0.05}\right) + 0.5\log\left(\frac{0.5}{0.95}\right) = 0.83$	$0.5\log\left(\frac{0.5}{0}\right) + 0.5\log\left(\frac{0.5}{1}\right) = +\infty$
q is fair	$0.05\log\left(\frac{0.05}{0.5}\right) + 0.95\log\left(\frac{0.95}{0.5}\right) = 0.49$	$0\log\left(\frac{0}{0.5}\right) + 1\log\left(\frac{1}{0.5}\right) = 0.69$

Jensen-Shannon (JS) divergence Variation of KL divergence to make it symmetric and bounded:

Jensen-Shannon (JS) divergence

$$[0, \log_b(2)] \ni D_{\text{JS}}(p||q) = \frac{1}{2}D_{\text{KL}}\left(p||\frac{p+q}{2}\right) + \frac{1}{2}D_{\text{KL}}\left(q||\frac{p+q}{2}\right)$$

Remark. JS divergence has the highest value when p and q are disjoint (i.e., $p(x) > 0 \Rightarrow q(x) = 0$ or vice versa).

Remark. JS divergence is symmetric and respects the triangle inequality.

8.1.1 Inception score

Inception score (IS) Given a generator and a pre-trained image classifier, Inception score assesses that:

Inception score (IS)

- The generated images are classified by the classifier with a high confidence (i.e., $p_{\text{cls}}(y|x)$ should have low entropy).
- The generated images cover all the classes handled by the classifier (i.e., $p_{\text{cls}}(y)$ should have high entropy).

Inception score is computed as:

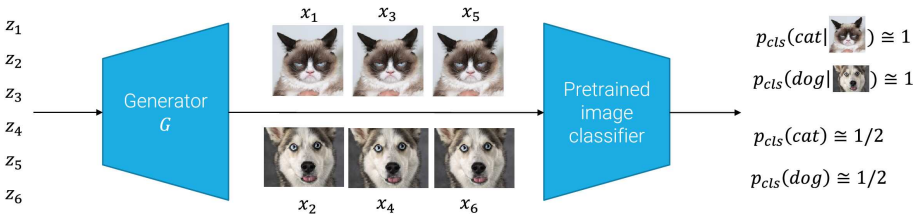
$$[1, C] \ni IS = \exp\left(\mathbb{E}_{x_i\sim p_{\text{gen}}(x)}\Big[D_{\text{KL}}\big(p_{\text{cls}}(y|x_i)\,||\,p_{\text{cls}}(y)\big)\Big]\right)$$
$$= \exp\left(H(p_{\text{cls}}(y)) - \mathbb{E}_{x_i\sim p_{\text{gen}}(x)}\Big[H(p_{\text{cls}}(y|x_i))\Big]\right)$$

where $p_{\text{cls}}(y)$ is computed through Monte Carlo approximation:

$$p_{\text{cls}}(y) = \mathbb{E}_{x \sim p_{\text{gen}}} \left[p_{\text{cls}}(y|x) \right] \approx \frac{1}{N} \sum_{x_i \sim p_{\text{gen}}(x)} p_{\text{cls}}(y|x_i)$$

Remark. Inception score has the following problems:

- It uses the Monte Carlo approximation and requires a large sample size.
- It is sensitive to the performance of the classifier.
- It does not measure generation diversity within a class.



8.1.2 Fréchet Inception distance

Earth mover's distance (EMD)

Earth mover's distance (EMD) Given two discrete distributions p and q, EMD measures the "work" needed to match p and q by considering the amount of mass to move and the distance between bins.

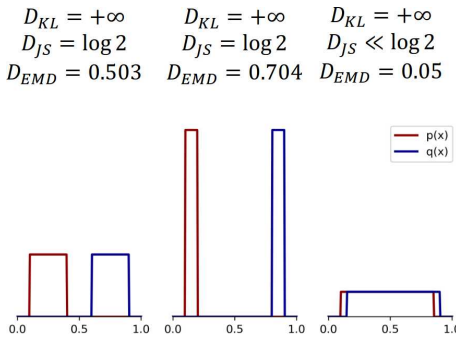
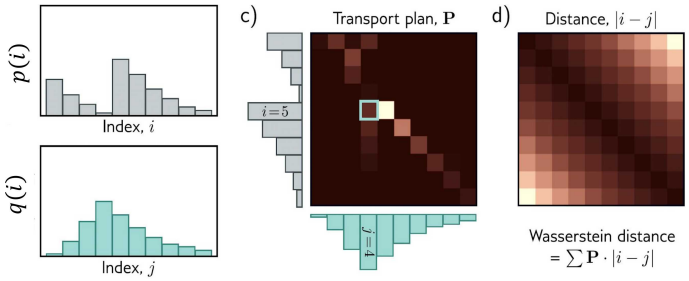


Figure 8.1: Three cases of density functions distance. The distributions in the first case are closer than the second one. In the third case, they are mostly overlapping.

Earth mover's distance is formulated as a linear programming problem:

$$\begin{aligned} D_{\text{EMD}}(p||q) = \min_{\boldsymbol{P}} & \left[\sum_{i,j} \boldsymbol{P}_{i,j} |i-j| \right] \\ \text{subject to} & \sum_j \boldsymbol{P}_{i,j} = p(i) \wedge \\ & \sum_i \boldsymbol{P}_{i,j} = q(j) \wedge \\ & \boldsymbol{P}_{i,j} \geq 0 \end{aligned}$$

where $\boldsymbol{P}$ is the transport plan such that $\boldsymbol{P}_{i,j}$ indicates the amount of mass to move from bin i (of p) to bin j (of q).



Fréchet Inception distance (FID)

Fréchet Inception distance (FID)

Remark. In the continuous case, EMD is computed using the Fréchet distance. Applying it as-is has two problems:

- Comparing pixels is not effective.
- The distribution of the generated images cannot be analytically determined.

Fréchet Inception distance is based on two ideas:

- Use the embedding (Inception-v3) of an image instead of pixels.
- Assume that the embedding space is a multi-variate Gaussian.

Given the Gaussians fitted on the embeddings of the real images $\mathcal{N}(x; \mu_r, \Sigma_r)$ and the embeddings of the generated images $\mathcal{N}(x; \mu_g, \Sigma_g)$, FID is computed as:

$$D^2_{\text{FR}}\left[\mathcal{N}(x; \mu_r, \Sigma_r)||\mathcal{N}(x; \mu_g, \Sigma_g)\right] = \|\mu_g - \mu_r\|_2^2 + \text{tr}\left(\Sigma_r + \Sigma_g - 2\sqrt{\Sigma_r\Sigma_g}\right)$$

Remark. FID accounts for both realism and diversity (FID grows if one of the two distribution is more diverse than the other) both across and within classes. However:

- It is sensitive to the pre-trained feature extractor (but it is more robust than Inception score).
- It is not possible to distinguish in the final score whether realism or diversity is the problem.

8.1.3 Manifold precision/recall

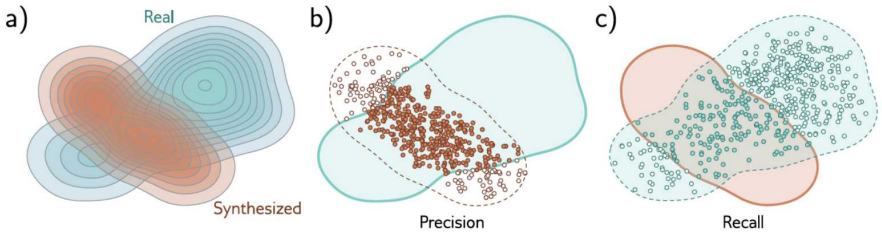
Manifold precision Fraction of generated images that fall into the real data manifold (i.e., realism).

Manifold precision

Manifold recall Fraction of real images that fall into the generator manifold (i.e., diversity).

Manifold recall

Remark. These metrics are computed in the embedding space of the images. The manifold is approximated with a hypersphere centered on each embedding with a radius proportional to the distance to the k-th nearest neighbor.



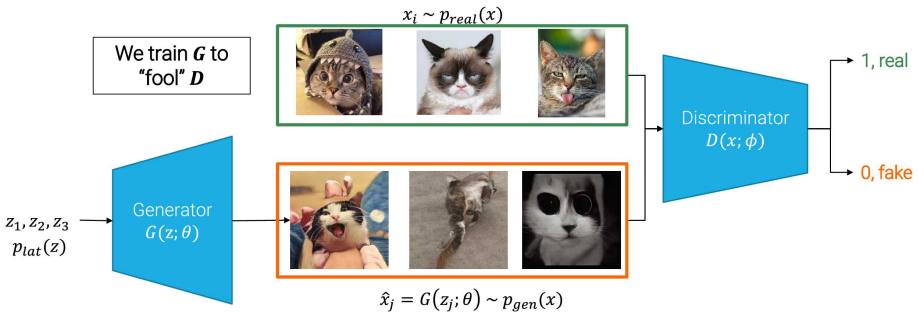
8.2 Generative adversarial networks

Generative adversarial network (GAN) Given:

Generative adversarial network (GAN)

- A generator $G(z;\theta)$ that takes an input latent vector $z_i \sim p_{\text{lat}}(z)$ and produces an image $\hat{x}_j \sim p_{\text{gen}}(x)$,
- A discriminator $D(x;\phi)$ that determines whether x_i is a real image from $p_{\text{real}}(x)$.

A generative adversarial network trains both D and G with the aim of making p_{gen} converge to p_{real}.



Discriminator loss The discriminator solves a binary classification task using binary cross-entropy:

Discriminator loss

$$\mathcal{L}_{\text{BCE}}(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

As real images always have label 1 and generated ones always 0, the optimal parameters are given by the problem:

$$\begin{aligned}\phi^* &= \arg\min_{\phi}\left\{\frac{1}{I}\sum_{i=1}^{I}\mathcal{L}_{\text{BCE}}(D(x_i;\phi),1) + \frac{1}{J}\sum_{j=1}^{J}\mathcal{L}_{\text{BCE}}(D(\hat{x}_j;\phi),0)\right\} \\ &= \arg\min_{\phi}\left\{-\frac{1}{I}\sum_{i=1}^{I}\log\left(D(x_i;\phi)\right) - \frac{1}{J}\sum_{j=1}^{J}\log\left(1-D(\hat{x}_j;\phi)\right)\right\}\end{aligned}$$

Therefore, the discriminator loss can be formulated as:

$$\mathcal{L}_D(\phi) = -\sum_{i=1}^{I}\log(D(x_i;\phi)) - \sum_{j=1}^{J}\log(1-D(\hat{x}_i;\phi))$$

Generator loss The generator aims to fool the discriminator. Its objective is therefore to maximize the loss the discriminator is minimizing: Generator loss

$$\theta^* = \arg\max_\theta \left\{ \min_\phi \left\{ -\frac{1}{I}\sum_{i=1}^{I} \log\left(D(x_i;\phi)\right) - \frac{1}{J}\sum_{j=1}^{J} \log\left(1 - D(G(z_j;\theta);\phi)\right) \right\} \right\}$$

As the generator only influences the second term, the overall generator loss is:

$$\mathcal{L}_G(\theta) = \sum_{j=1}^{J} \log\left(1 - D(G(z_j;\theta);\phi)\right)$$

Training Generator and discriminator are trained together in a minimax game as follows:

1. Sample a batch of latent vectors $z_1, \ldots, z_j$ to generate $\hat{x}_1, \ldots, \hat{x}_j$.
2. Sample a batch of real images $x_1, \ldots, x_i$.
3. Merge the two batches and optimize for $\mathcal{L}_D(\phi)$ and $\mathcal{L}_G(\theta)$.

Remark (Optimal discriminator)**.** The discriminator loss can be seen as a sum of two expectations (that uses Monte Carlo approximation):

$$\begin{aligned}
\mathcal{L}_D(\phi) &= -\frac{1}{I}\sum_{i=1}^{I} \log(D(x_i;\phi)) - \frac{1}{J}\sum_{j=1}^{J} \log(1 - D(\hat{x}_i;\phi)) \\
&\approx -\mathbb{E}_{x_i \sim p_{\text{real}}(x)}\left[\log(D(x_i;\phi))\right] - \mathbb{E}_{\hat{x}_j \sim p_{\text{gen}}(x)}\left[\log(1 - D(\hat{x}_j;\phi))\right] \\
&= -\int p_{\text{real}}(x)\log(D(x;\phi)) + p_{\text{gen}}(x)\log(1 - D(x;\phi))\, dx
\end{aligned}$$

By calling $a = p_{\text{real}}(x)$, $b = p_{\text{gen}}(x)$, and $y = D(x;\phi)$, we have that:

$$\int a\log(y) + b\log(1-y)\, dx$$

To obtain the minimum w.r.t. $y = D(x;\phi)$, we have that:

$$\begin{aligned}
\frac{\partial}{\partial y}\left[a\log(y) + b\log(1-y)\right] = 0 \Rightarrow \frac{a}{y} - \frac{b}{1-y} &= 0 \\
\Rightarrow y &= \frac{a}{a-b}
\end{aligned}$$

Therefore, the optimal theoretical discriminator is:

$$D^*(x) = \frac{p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)}$$

This makes sense as:

- With a real image ($p_{\text{real}}(x)$ high) and a bad generator ($p_{\text{gen}}(x)$ low), the discriminator is able to detect the real image ($D^*(x) \to 1$).
- With a bad generator ($p_{\text{real}}(x)$ low and $p_{\text{gen}}(x)$ high), the discriminator is able to detect the generator ($D^*(x) \to 0$).
- With a perfect generator ($p_{\text{real}}(x)$ and $p_{\text{gen}}(x)$ are both high), the discriminator is undecided ($D^*(x) \to 0.5$).

Remark (Optimal generator)**.** By inserting the optimal discriminator into the (full) generator loss, we have that:

$$
\begin{aligned}
\mathcal{L}_G(x) &= -\int p_{\text{real}}(x)\log\left(\frac{p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right) + p_{\text{gen}}(x)\log\left(1-\frac{p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx \\
&= -\int p_{\text{real}}(x)\log\left(\frac{p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx - \int p_{\text{gen}}(x)\log\left(\frac{p_{\text{gen}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx \\
&= -\int p_{\text{real}}(x)\log\left(\frac{2}{2}\frac{p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx - \int p_{\text{gen}}(x)\log\left(\frac{2}{2}\frac{p_{\text{gen}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx \\
&= -\int p_{\text{real}}(x)\log\left(\frac{2p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right) + \log\left(\frac{1}{2}\right)dx - \int p_{\text{gen}}(x)\log\left(\frac{2p_{\text{gen}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right) + \log\left(\frac{1}{2}\right)dx \\
&= -\int p_{\text{real}}(x)\log\left(\frac{2p_{\text{real}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx - \int p_{\text{gen}}(x)\log\left(\frac{2p_{\text{gen}}(x)}{p_{\text{real}}(x)+p_{\text{gen}}(x)}\right)dx + \log(4) \\
&= -\int p_{\text{real}}(x)\log\left(\frac{p_{\text{real}}(x)}{\frac{p_{\text{real}}(x)+p_{\text{gen}}(x)}{2}}\right)dx - \int p_{\text{gen}}(x)\log\left(\frac{p_{\text{gen}}(x)}{\frac{p_{\text{real}}(x)+p_{\text{gen}}(x)}{2}}\right)dx + \log(4) \\
&= -D_{\text{KL}}\left(p_{\text{real}}(x)||\frac{p_{\text{real}}(x)+p_{\text{gen}}(x)}{2}\right) - D_{\text{KL}}\left(p_{\text{gen}}(x)||\frac{p_{\text{real}}(x)+p_{\text{gen}}(x)}{2}\right) + \log(4) \\
&= -2D_{\text{JS}}\left(p_{\text{real}}(x)||p_{\text{gen}}(x)\right) + \log(4)
\end{aligned}
$$

Therefore, the generator aims to maximize $-2D_{\text{JS}}\left(p_{\text{real}}(x)||p_{\text{gen}}(x)\right) + \log(4)$. As $D_{\text{JS}}(\cdot) \geq 0$, the optimal generator aims to achieve $D_{\text{JS}}(\cdot) = 0$ which happens when:

$$p_{\text{real}}(x) = p_{\text{gen}}(x)$$

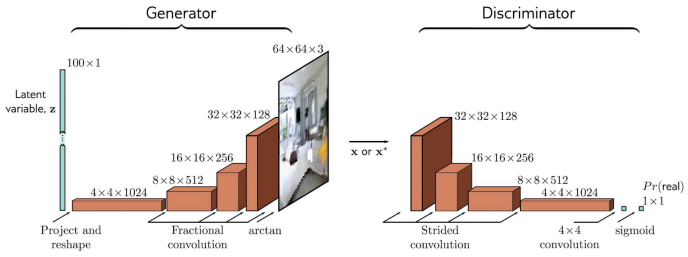
Remark. Even though on a theoretical level a perfect discriminator and generator exist, in practice neural networks are finite and it is not clear if the alternating minimization steps converge.

Remark. The original GAN only uses fully-connected layers.

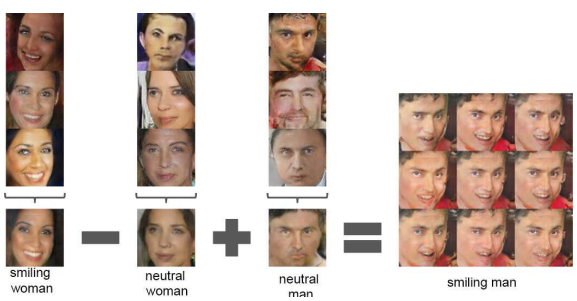
8.2.1 Deep convolutional GAN

Deep convolutional GAN (DCGAN) GAN with a fully-convolutional architecture.

Deep convolutional GAN (DCGAN)



Remark. The latent space of GANs is well-behaved: directions in the latent space have a meaning.

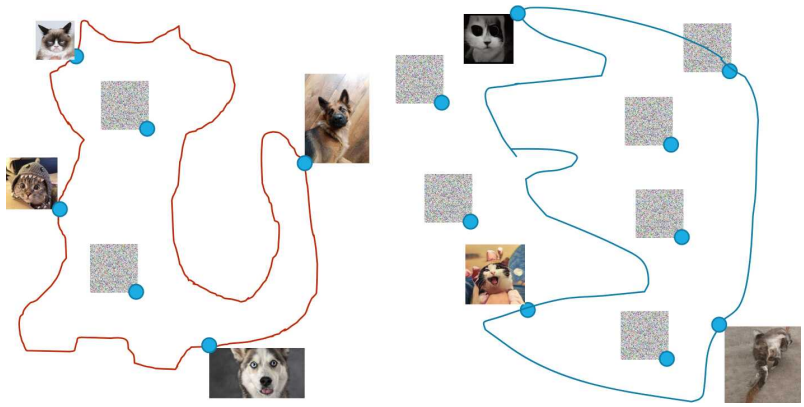


Remark (Mode dropping/collapse)**.** Only some modes of the distribution of the real data are represented by the mass of the generator.
Consider the training objective of the optimal generator. The two terms model coverage and quality, respectively:

$$-\frac{1}{I}\sum_{i=1}^{I}\log\left(D(x_i;\phi)\right)-\frac{1}{J}\sum_{j=1}^{J}\log\left(1-D(G(z_j;\theta);\phi)\right)$$
$$=\underbrace{-D_{\mathrm{KL}}\left(p_{\mathrm{real}}(x)||\frac{p_{\mathrm{real}}(x)+p_{\mathrm{gen}}(x)}{2}\right)}_{\text{Coverage}}\underbrace{-D_{\mathrm{KL}}\left(p_{\mathrm{gen}}(x)||\frac{p_{\mathrm{real}}(x)+p_{\mathrm{gen}}(x)}{2}\right)}_{\text{Quality}}+\log(4)$$

The coverage term is high for regions with real images and no generated ones. The quality term is high for regions with generated images and no real ones.
As the generator only affects the second term (quality), this might be a reason that causes mode collapse.

Remark (Disjoint distributions)**.** As real and generated images lie in low-dimensional subspaces, it is likely that their distributions are disjoint. The training objective of GANs aims to minimize the JS divergence, which is maximum if the distributions do not overlap causing no significant signal for gradient updates.
In other words, whenever the generator or discriminator becomes too performant (i.e., the distributions become too different), updates to the other component carry no signal.



Example. By freezing the generator while training the discriminator, it takes few epochs for it to achieve near perfect accuracy.

8.2.2 Wasserstein GAN

Wasserstein GAN (WGAN) Train a GAN using the Wasserstein distance (i.e., EMD) to compare distributions. Wasserstein GAN (WGAN)

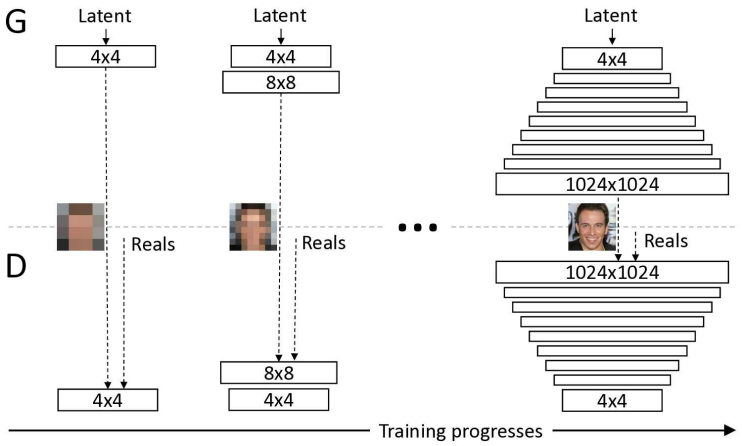
Remark. The Wasserstein distance is meaningful even with disjoint distributions.

Remark. It can be shown that the objective is easier to optimize in the dual form of the linear programming formulation.

8.2.3 Progressive GAN

Progressive GAN (ProGAN) GAN for high-resolution generation with a coarse-to-fine approach. Progressive GAN (ProGAN)

Training The network starts as a generator for 4×4 images and its resolution is incrementally increased.



Layer fade-in When moving from an $n \times n$ to $2n \times 2n$ resolution, the following happens:

- The generator outputs a linear combination between the $n \times n$ image up-sampled (with weight $1 - \alpha$) and the $n \times n$ image passed through a transpose convolution (with weight α).
- The discriminator uses a linear combination between the $2n \times 2n$ image down-sampled (with weight $1-\alpha$) and the $2n \times 2n$ image passed through a convolution.

Where α grows linearly from 0 to 1 during training. This allows the network to use old information when the resolution changes and gradually adapt.

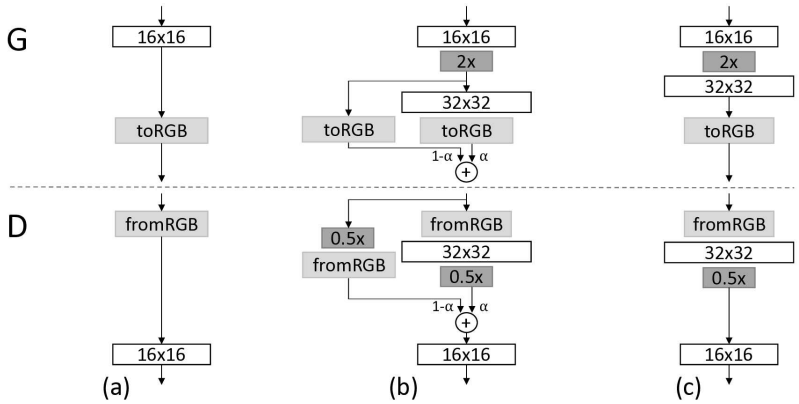


Figure 8.2: ProGAN fade-in. (a) is the starting resolution. (b) depicts the fade-in process. (c) represents the network at the end of the training process for this resolution (i.e., with $\alpha = 1$)

8.2.4 StyleGAN

Instance normalization Normalize along spatial dimensions only (i.e., there is a mean and variance for every channel of the image and every element of the batch).

Instance normalization

Adaptive instance normalization (AdaIN) Given a content $c \in \mathbb{R}^{B \times C \times H \times W}$ and a style $s \in \mathbb{R}^{B \times C \times H \times W}$, AdaIN normalizes c while injecting information from s

Adaptive instance normalization (AdaIN)

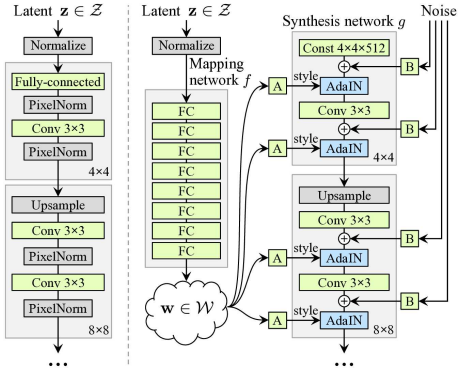
as follows:

$$\sqrt{v_s}\frac{c-\mu_c}{\sqrt{v_c+\varepsilon}}+\mu_s$$

where $\mu_c \in \mathbb{R}^{B\times C\times 1\times 1}$ and $v_c \in \mathbb{R}^{B\times C\times 1\times 1}$ are mean and variance of c and $\mu_s \in \mathbb{R}^{B\times C\times 1\times 1}$ and $v_s \in \mathbb{R}^{B\times C\times 1\times 1}$ are mean and variance of s.

StyleGAN Network that does the following: StyleGAN

1. Preprocess the input latent vector with fully-connected layers.
2. Add to the starting constant image (learned during training) some noise.
3. Pass the current image as the content of AdaIN and use the refined latent vector as the style.
4. Continue as normal GANs, with AdaIN using the latent vector as style after every convolution.



Remark. It has been observed that the refined latent representation is more disentangled.

Remark. In normal GANs, the latent vector encodes both information on the image to generate and some noise for variability. In StyleGAN, these two aspects are ideally separated.

Remark. Adversarial losses can also be used in supervised problems (e.g., generate a colored version of a black-and-white image).

Remark (BigGAN)**.** To improve realism (with loss in coverage), a latent can be sampled from only the high-probability areas of the distribution. BigGAN

8.3 Diffusion models

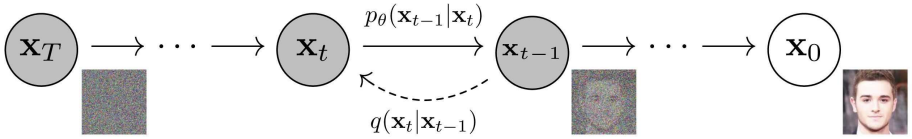
Diffusion model Architecture that generates an image by iteratively denoising the input latent vector. Diffusion model

Remark. Empirical results show that the generation quality is generally better than other models. However, inference is slow.

Training Given an image $\boldsymbol{x}_0$, training is done in two steps:

Forward process The original image $\boldsymbol{x}_0$ is iteratively transformed into a latent image $\boldsymbol{x}_T$ by adding noise (i.e., transform the complex distribution $q(\boldsymbol{x}_0)$ of the original image into a simpler one $q(\boldsymbol{x}_T)$).

Reverse process The latent image $\boldsymbol{x}_T$ is iteratively denoised to reconstruct the original image $\boldsymbol{x}_0$.



8.3.1 Forward process

Forward process Given an image $\boldsymbol{x}_{t-1}$, produce a noisier version of it as: Forward process

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_t$$
$$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}; \beta_t \boldsymbol{I})$$

where:

- $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(0; \boldsymbol{I})$ is the noise
- $\beta_t \in [0, 1)$ is a hyperparameter (noise schedule) and represents the variance.
- $\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}$ is the mean.

Remark. $\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}$ and β_t are the mean and variance due to the fact that sampling a vector $\mathbf{x}$ from a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is equivalent to:

$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{\frac{1}{2}}\mathbf{y} \qquad \text{where } \mathbf{y} \sim \mathcal{N}(0; \boldsymbol{I})$$

If $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, it holds that $\boldsymbol{\Sigma}^{\frac{1}{2}} = \sigma \boldsymbol{I}$ and we have that:

$$\mathbf{x} = \boldsymbol{\mu} + (\sigma \boldsymbol{I})\mathbf{y}$$

Remark. This step does not have learnable parameters.

Diffusion kernel It is possible to generate the latent vector $\boldsymbol{x}_t$ at time t directly from $\boldsymbol{x}_0$ as: Diffusion kernel

$$\boldsymbol{x}_t = \sqrt{\prod_{i=1}^{t}(1-\beta_i)} \cdot \boldsymbol{x}_0 + \sqrt{1 - \prod_{i=1}^{t}(1-\beta_i)} \cdot \boldsymbol{\varepsilon} \qquad \text{where } \boldsymbol{\varepsilon} \sim \mathcal{N}(0; \boldsymbol{I})$$

By setting the intermediate constant $\alpha_t = \prod_{i=1}^{t}(1-\beta_i)$, we have that:

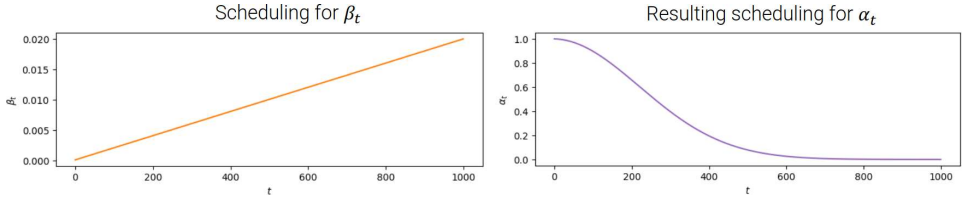
$$\boldsymbol{x}_t = \sqrt{\alpha_t}\boldsymbol{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\varepsilon}$$
$$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathcal{N}(\sqrt{\alpha_t}\boldsymbol{x}_0; (1-\alpha_t)\boldsymbol{I})$$

Remark. As $\beta_t < 1$, it holds that $\lim_{t \to +\infty} \alpha_t = 0$. In other words, for large $t = T$, only noise remains in the latent vector:

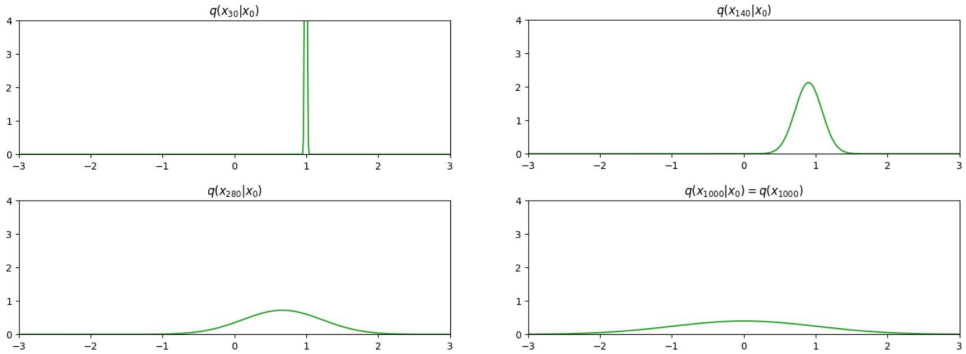
$$q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) = q(\boldsymbol{x}_T) = \mathcal{N}(0; \boldsymbol{I})$$

Which achieves the goal of transforming a complex distribution $q(\boldsymbol{x}_0)$ into a simpler one (i.e., Gaussian).

Example. Consider the 1D case where x represents a pixel. By using a linear scheduling for β_t as follows:



We obtain that some diffusion kernels for varying t with $x_0 = 1$ are the following (note that the signal converges to $\mathcal{N}(0; 1)$):



Remark. As the forward process is stochastic, the same starting pixel can produce a different resulting pixel. Therefore, diffusion models work with trajectories in latent space.

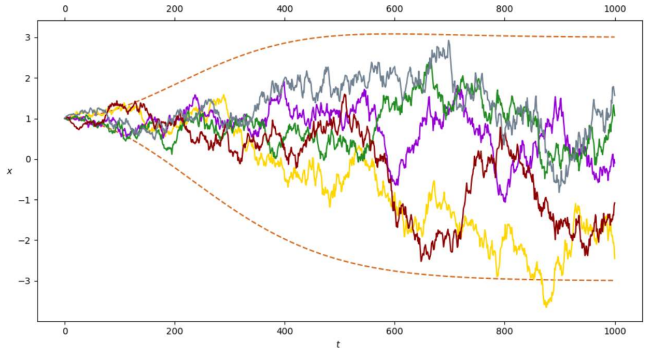


Figure 8.3: Trajectories starting from $x_0 = 1$. The dashed lines mark the $\mu_t \pm 3\sigma_t$ area.

8.3.2 Reverse process

Remark. In principle, one could invert the forward process by applying Bayes' rule:

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) \frac{q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

However, closed-form expressions for $q(\boldsymbol{x}_{t-1})$ and $q(\boldsymbol{x}_t)$ are not available.
By exploiting the Markov chain properties, it is possible to compute the conditional distribution w.r.t. $\boldsymbol{x}_0$, which is available at training time, as:

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) = q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0) \frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)} = \underbrace{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})}_{\text{Forward process}} \underbrace{\frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)}}_{\text{Diffusion kernels}}$$

It can be shown that this is equivalent to:

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\boldsymbol{x}_t + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t}\beta_t \boldsymbol{x}_0; \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\boldsymbol{I}\right)$$

However, this formulation requires knowing $\boldsymbol{x}_0$, which is only available at training time, making inference impossible.

Learned reverse process (mean)

Learned reverse process (mean) Learn a Markov chain of probabilistic mappings to reconstruct the original image $\boldsymbol{x}_0$ starting from the latent vector $\boldsymbol{x}_T$:

$$p(\boldsymbol{x}_T) = \mathcal{N}(0; \boldsymbol{I}) = q(\boldsymbol{x}_T)$$
$$p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = \mathcal{N}(\mu_t(\boldsymbol{x}_t; \boldsymbol{\theta}_t); \sigma_t \boldsymbol{I})$$

where:

- $\mu_t(\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ is a neural network to estimate the mean of $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$.
- σ_t is, for the case of simple diffusion models, predetermined.

Remark. In general, $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ does not necessarily follow a Gaussian distribution as this is only true for $\beta_t \to 0$. However, by using small β_t and large T, it can be approximately considered Gaussian.

Loss The loss function for a set of images $\{\boldsymbol{x}_0^{(i)}\}_{i=1}^I$ is based on the MSE of the predicted means:

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T) \\
&= \sum_{i=1}^{I}\left(-\log\left(\mathcal{N}(\boldsymbol{x}_0^{(i)}; \mu_1(\boldsymbol{x}_1^{(i)}; \boldsymbol{\theta}_1), \sigma_1 \boldsymbol{I})\right) + \sum_{t=2}^{T}\frac{1}{2\sigma_t}\left\|\boldsymbol{\mu}_{q(x_{t-1}|x_t,x_0)} - \mu_t(\boldsymbol{x}_t^{(i)}; \boldsymbol{\theta}_t)\right\|^2\right) \\
&= \sum_{i=1}^{I}\left(\underbrace{-\log\left(\mathcal{N}(\boldsymbol{x}_0^{(i)}; \mu_1(\boldsymbol{x}_1^{(i)}; \boldsymbol{\theta}_1), \sigma_1 \boldsymbol{I})\right)}_{\text{Reconstruction of } x_0 \text{ from } x_1} + \sum_{t=2}^{T}\frac{1}{2\sigma_t}\left\|\underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\boldsymbol{x}_t^{(i)} + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t}\beta_t \boldsymbol{x}_0^{(i)}}_{\text{Ground-truth mean of } q(x_{t-1} \mid x_t, x_0)} - \underbrace{\mu_t(\boldsymbol{x}_t^{(i)}; \boldsymbol{\theta}_t)}_{\text{Prediction}}\right\|^2\right)
\end{aligned}$$

Remark. As T is usually large, the MSE term has more relevance.

Proof. The overall training objective for a set of real images $\{\boldsymbol{x}_0^{(i)}\}_{i=1}^I$ is to maximize the likelihood of the reconstructed image:

$$\boldsymbol{\theta}_1^*, \dots, \boldsymbol{\theta}_T^* = \arg\max_{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T} \sum_{i=1}^{I} \log\left(p(\boldsymbol{x}_0^{(i)} \mid \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T)\right)$$

Lemma 8.3.1 (Latents joint probabilites)**.** As each image is obtained as a sequence of latents, we have that:

$$\begin{aligned}
&p(\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T) \\
&\quad = p(\boldsymbol{x}_0 \mid \boldsymbol{x}_1, \dots, \boldsymbol{x}_T, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T) p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T) && p(x, y|z) = p(x|y, z)p(y|z) \\
&\quad = p(\boldsymbol{x}_0 \mid \boldsymbol{x}_1, \boldsymbol{\theta}_1) p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_T) && \text{Markov chain} \\
&\quad = \dots && \text{Repeat} \\
&\quad = p(\boldsymbol{x}_0 \mid \boldsymbol{x}_1, \boldsymbol{\theta}_1)\left(\prod_{t=2}^{T} p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{\theta}_t)\right) p(\boldsymbol{x}_T)
\end{aligned} \tag{8.1}$$

By using Lemma 8.3.1, the likelihood of $\boldsymbol{x}_0$ can be computed through marginal-

ization over the latent images as follows:

$$p(\boldsymbol{x}_0^{(i)} \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T) = \int p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)\, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_T$$

However, in practice this approach is computationally intractable due to the high number and high dimensionality of the latent variables.

Lemma 8.3.2 (Jensen's inequality)**.** Given a concave function $f(\cdot)$ and the expectation of the data x. It holds that:

$$f(\mathbb{E}_{x\sim p(x)}[x]) \geq \mathbb{E}_{x\sim p(x)}[f(x)]$$

Example. Consider the logarithm function and a discrete random variable. It holds that:

$$\log\left(\mathbb{E}_{x\sim p(x)}[x]\right) \geq \mathbb{E}_{x\sim p(x)}[\log(x)] \Rightarrow \log\left(\sum_{x\in\mathbb{X}} p(x)x\right) \geq \sum_{x\in\mathbb{X}}(p(x)\log(x))$$

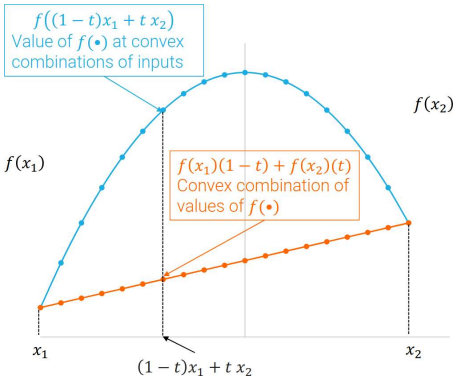


Figure 8.4: Visualization of Jensen's inequality

Lemma 8.3.3 (Evidence lower bound (ELBO))**.** Method to compute a lower-bound of the log-likelihood. It holds that:

Evidence lower bound (ELBO)

$$\begin{aligned}
&\log(p(\boldsymbol{x}_0 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)) \\
&\quad = \log\left(\int p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)\, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_T\right) \\
&\quad = \log\left(\int \frac{q(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0)} p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)\, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_T\right) \\
&\quad = \log\left(\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T|\boldsymbol{x}_0)}\left[\frac{p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)}{q(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0)}\right]\right)
\end{aligned}$$

By applying Jensen's inequality, ELBO is computed as:

$$\begin{aligned}
&\log\left(\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T|\boldsymbol{x}_0)}\left[\frac{p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)}{q(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0)}\right]\right) \geq \\
&\quad \mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T|\boldsymbol{x}_0)}\left[\log\left(\frac{p(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)}{q(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0)}\right)\right] = \texttt{ELBO}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)
\end{aligned}$$

During training, we aim to maximize ELBO as a proxy to maximize the likelihood. By applying Lemma 8.3.1 to the argument of the logarithm in ELBO,

we have that:

$$\log\left(\frac{p(\boldsymbol{x}_0,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_T)}{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\right)$$
$$=\log\left(\frac{p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\left(\prod_{t=2}^{T}p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)\right)p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)\prod_{t=2}^{T}q(\boldsymbol{x}_t\mid\boldsymbol{x}_{t-1},\boldsymbol{x}_0)}\right)$$
$$=\log\left(\frac{p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)}{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\right)+\log\left(\frac{\prod_{t=2}^{T}p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{\prod_{t=2}^{T}q(\boldsymbol{x}_t\mid\boldsymbol{x}_{t-1},\boldsymbol{x}_0)}\right)+\log(p(\boldsymbol{x}_T))$$
$$=\log\left(\frac{p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)}{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\right)+\log\left(\prod_{t=2}^{T}\left(\frac{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_0)}{q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}\right)\right)+\log(p(\boldsymbol{x}_T))\quad\text{Bayes on denom.}$$

The second term introduced by Bayes' rule can be simplified as follows:

$$\begin{aligned}\prod_{t=2}^{T}\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_0)}{q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}&=\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{\cancel{q(\boldsymbol{x}_2\mid\boldsymbol{x}_0)}}\frac{\cancel{q(\boldsymbol{x}_2\mid\boldsymbol{x}_0)}}{\cancel{q(\boldsymbol{x}_3\mid\boldsymbol{x}_0)}}\cdots\frac{\cancel{q(\boldsymbol{x}_{T-1}\mid\boldsymbol{x}_0)}}{q(\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\\&=\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{q(\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\\&=\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{q(\boldsymbol{x}_T)}&&\text{Time } T \text{ is known to be } \mathcal{N}(0;\boldsymbol{I})\end{aligned}$$

Therefore, we have that:

$$\log\left(\frac{p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)}{\cancel{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}}\right)+\log\left(\frac{\cancel{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}}{q(\boldsymbol{x}_T)}\prod_{t=2}^{T}\frac{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\right)+\log(p(\boldsymbol{x}_T))$$
$$=\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)+\log\left(\prod_{t=2}^{T}\frac{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\right)+\log\left(\frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T)}\right)\qquad\frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T)}\approx 1\text{ as they are both }\mathcal{N}(0;\boldsymbol{I})$$
$$=\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)+\sum_{t=2}^{T}\log\left(\frac{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\right)$$

By going back to ELBO, we have that:

$$\texttt{ELBO}(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_T)=\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\left[\log\left(\frac{p(\boldsymbol{x}_0,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_T)}{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\right)\right]$$
$$\approx\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\left[\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)+\sum_{t=2}^{T}\log\left(\frac{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\right)\right]$$
$$=\mathbb{E}_{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\left[\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)\right]-\sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)}\left[\log\left(\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}\right)\right]$$

$$\mathbb{E}_{q(x,y)}=\mathbb{E}_{q(y)}\mathbb{E}_{q(x\mid y)}$$

$$=\mathbb{E}_{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\left[\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)\right]-\sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}\mathbb{E}_{q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{t-1},\boldsymbol{x}_{t+1},\ldots,\boldsymbol{x}_T\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\left[\log\left(\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}\right)\right]$$
$$=\mathbb{E}_{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\left[\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)\right]-\sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}\mathbb{E}_{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}\left[\log\left(\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}{p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)}\right)\right]$$
$$=\mathbb{E}_{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}\left[\log\left(p(\boldsymbol{x}_0\mid\boldsymbol{x}_1,\boldsymbol{\theta}_1)\right)\right]-\sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}\Big[D_{\mathrm{KL}}\big(q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)\|p(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{\theta}_t)\big)\Big]$$

To make ELBO a computable loss function, we have to:

- Approximate expectations with Monte Carlo.
- Expand p and q with their definition.
- Expand the KL divergence. As it is between two Gaussians with constant covariance matrices, it can be computed in closed form as:

$$\begin{aligned} & D_{\mathrm{KL}}\big(q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \| p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{\theta}_t)\big) \\ & \quad = \frac{1}{2\sigma_t} \left\| \boldsymbol{\mu}_{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)} - \boldsymbol{\mu}_{p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{\theta}_t)} \right\|^2 + c \\ & \quad = \frac{1}{2\sigma_t} \left\| \left(\frac{1-\alpha_{t-1}}{1-\alpha_t} \sqrt{1-\beta_t} \boldsymbol{x}_t + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t} \beta_t \boldsymbol{x}_0 \right) - \mu_t(\boldsymbol{x}_t; \boldsymbol{\theta}_t) \right\|^2 + c \end{aligned}$$

  where c is a constant.

Finally, the loss is defined from ELBO as:

$$-\sum_{i=1}^{I} \left(\log\left(\mathcal{N}(\boldsymbol{x}_0^{(i)}; \mu_1(\boldsymbol{x}_1^{(i)}; \boldsymbol{\theta}_1), \sigma_1 \boldsymbol{I})\right) - \sum_{t=2}^{T} \frac{1}{2\sigma_t} \left\| \frac{1-\alpha_{t-1}}{1-\alpha_t} \sqrt{1-\beta_t} \boldsymbol{x}_t^{(i)} + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t} \beta_t \boldsymbol{x}_0^{(i)} - \mu_t(\boldsymbol{x}_t^{(i)}; \boldsymbol{\theta}_t) \right\|^2 \right)$$

□

Learned reverse process (noise) Learn a network $\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ to predict the noise at time t instead of the mean.

Learned reverse process (noise)

Loss The loss function is the MSE between noises:

$$\mathcal{L}(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T) = \sum_{i=1}^{I} \left(\sum_{t=1}^{T} \frac{\beta_t^2}{(\alpha_{t-1})(1-\beta_t)} \left\| \varepsilon_t\left(\sqrt{\alpha_t} \boldsymbol{x}_0^{(i)} + \sqrt{1-\alpha_t} \boldsymbol{\varepsilon}_t; \boldsymbol{\theta}_t\right) - \boldsymbol{\varepsilon}_t \right\|^2 \right)$$

Remark. In practice, this approach works better.

Theorem 8.3.4. Predicting the noise is equivalent to predicting the mean.

Proof. Consider the diffusion kernel:

$$\boldsymbol{x}_t = \sqrt{\alpha_t} \boldsymbol{x}_0 + \sqrt{1-\alpha_t} \boldsymbol{\varepsilon}_t \iff \boldsymbol{x}_0 = \frac{1}{\sqrt{\alpha_t}} \boldsymbol{x}_t + \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \boldsymbol{\varepsilon}_t$$

By substituting $\boldsymbol{x}_0$ in the definition of the mean of $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$, we have that:

$$\begin{aligned} \boldsymbol{\mu}_{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)} &= \frac{1-\alpha_{t-1}}{1-\alpha_t} \sqrt{1-\beta_t} \boldsymbol{x}_t + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t} \beta_t \boldsymbol{x}_0 \\ &= \frac{1-\alpha_{t-1}}{1-\alpha_t} \sqrt{1-\beta_t} \boldsymbol{x}_t + \frac{\sqrt{\alpha_{t-1}}}{1-\alpha_t} \beta_t \left(\frac{1}{\sqrt{\alpha_t}} \boldsymbol{x}_t + \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \boldsymbol{\varepsilon}_t \right) \\ &= \dots \\ &= \frac{1}{\sqrt{1-\beta_t}} \boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_{t-1}} \sqrt{1-\beta_t}} \boldsymbol{\varepsilon}_t \end{aligned}$$

Therefore, with $\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ it is possible to obtain the mean.

Moreover, the MSE term of the loss becomes:

$$
\begin{aligned}
&\left\|\boldsymbol{\mu}_{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)} - \mu(\boldsymbol{x}_t;\boldsymbol{\theta}_t)\right\|^2 \\
&= \left\| \left(\frac{1}{\sqrt{1-\beta_t}}\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_{t-1}}\sqrt{1-\beta_t}}\boldsymbol{\varepsilon}_t \right) - \left(\frac{1}{\sqrt{1-\beta_t}}\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_{t-1}}\sqrt{1-\beta_t}}\varepsilon(\boldsymbol{x}_t;\boldsymbol{\theta}_t) \right) \right\|^2 \\
&= \frac{\beta_t^2}{\alpha_{t-1}(1-\beta_t)} \|\varepsilon(\boldsymbol{x}_t;\boldsymbol{\theta}_t) - \boldsymbol{\varepsilon}_t\|^2 \\
&= \frac{\beta_t^2}{\alpha_{t-1}(1-\beta_t)} \|\varepsilon(\sqrt{\alpha_t}\boldsymbol{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\varepsilon}_t;\boldsymbol{\theta}_t) - \boldsymbol{\varepsilon}_t\|^2
\end{aligned}
$$

Therefore, the loss that only uses MSE computed on I images is:

$$
\sum_{i=1}^{I} \left(\sum_{t=1}^{T} \frac{\beta_t^2}{(\alpha_{t-1})(1-\beta_t)} \left\| \varepsilon_t \left(\sqrt{\alpha_t}\boldsymbol{x}_0^{(i)} + \sqrt{1-\alpha_t}\boldsymbol{\varepsilon}_t;\boldsymbol{\theta}_t \right) - \boldsymbol{\varepsilon}_t \right\|^2 \right)
$$

□

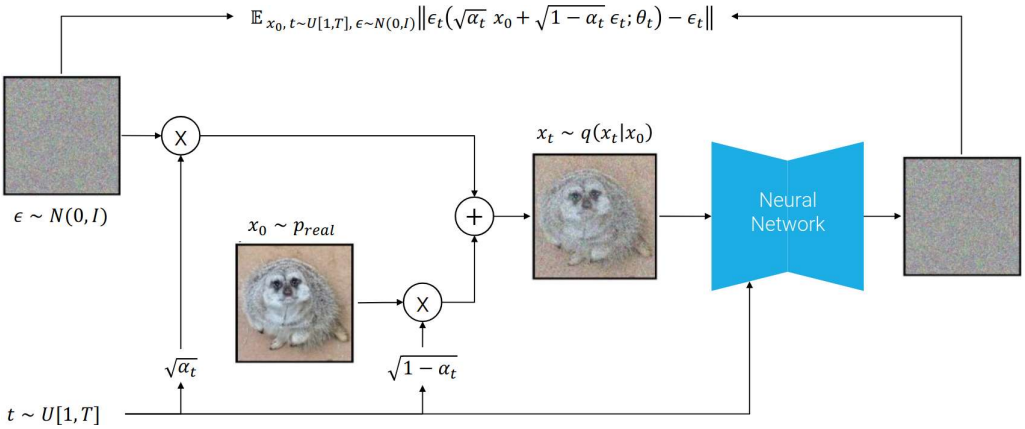


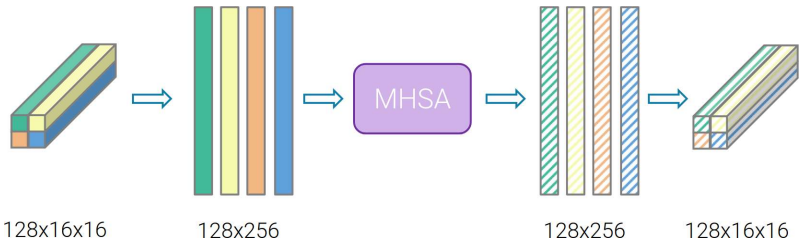
Figure 8.5: Diffusion models training flow

8.3.3 Architecture

Generation architecture Standard U-Net or transformer to predict the noise.

U-Net with self-attention Add global self-attention at the layers of the backbone where the resolution of the image is sufficiently small. It is applied as follows:

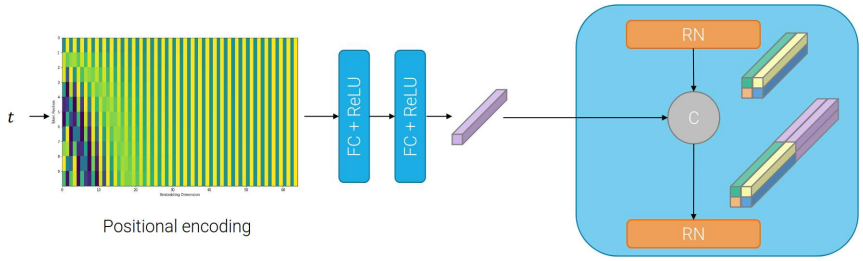
1. Flatten the spatial dimension to obtain C 1D activations.
2. Pass the flattened activations through the self-attention layer.
3. Reshape the output to match the original activation.



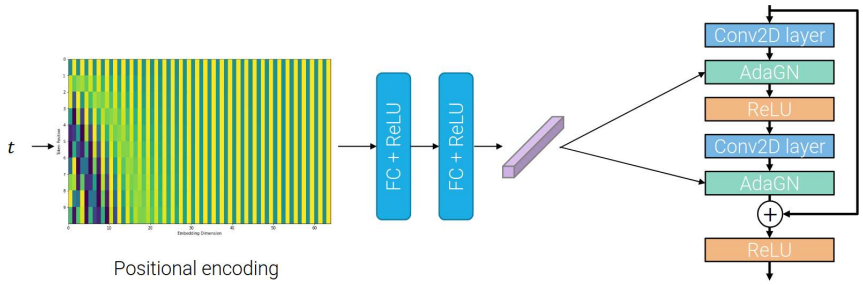
Time conditioning In practice, the same network with the same set of weights is used to process each time step. Therefore, some time information has to be injected. Time conditioning

Use transformer positional encoding, refined through some fully-connected layers to obtain an activation encoding time information. Then, two approaches are possible:

Concatenation The time activation is concatenated along every spatial dimension of the image activations.



Adaptive group normalization The time activation is used as the modulator for adaptive group normalization (similar mechanism to AdaIN).



8.3.4 Inference

Denoising diffusion probabilistic model (DDPM) Given a random latent $\boldsymbol{x}_T \sim \mathcal{N}(0; \boldsymbol{I})$, generation is done as follows: Denoising diffusion probabilistic model (DDPM)

1. For $t = T, \dots, 2$:
   a) Compute the mean of $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ by predicting the noise:

$$\boldsymbol{\mu}_t = \frac{1}{\sqrt{1-\beta_t}}\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_{t-1}}\sqrt{1-\beta_t}}\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta})$$

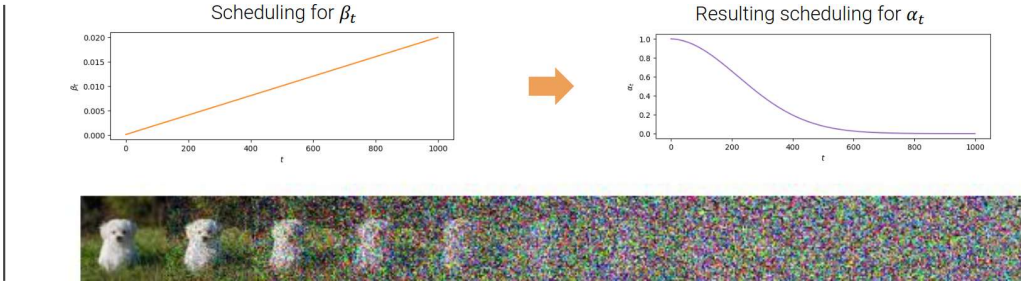
   b) Sample the next less noisy image from $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$:

$$\boldsymbol{x}_{t-1} = \boldsymbol{\mu}_t + \sigma_t \boldsymbol{\varepsilon}_t \qquad \text{with } \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0; \boldsymbol{I})$$

2. Use the mean of $p(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)$ as the output image:

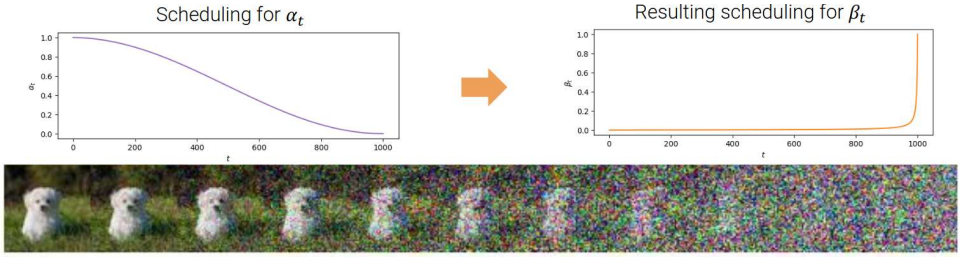
$$\boldsymbol{x}_0 = \frac{1}{\sqrt{1-\beta_t}}\boldsymbol{x}_1 - \frac{\beta_1}{\sqrt{1-\alpha_0}\sqrt{1-\beta_1}}\varepsilon_1(\boldsymbol{x}_1; \boldsymbol{\theta})$$

Remark. In the original paper, a linear schedule for β_t has been used. This results in a schedule for α_t that made the image mostly noise very quickly.



Improved DDPM (IDDPM) Use a cosine schedule for α_t (with $\beta_t = 1 - \frac{\alpha_t}{\alpha_{t-1}}$) so that the trajectory does not destroy the image too quickly.

Improved DDPM (IDDPM)



Remark. The loss of diffusion models (with DDPM) only considers the marginal $q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$:

$$\sum_{i=1}^{I}\left(\sum_{t=1}^{T}\frac{\beta_t^2}{(\alpha_{t-1})(1-\beta_t)}\left\|\varepsilon_t(\underbrace{\sqrt{\alpha_t}\boldsymbol{x}_0^{(i)}+\sqrt{1-\alpha_t}\boldsymbol{\varepsilon}_t;\boldsymbol{\theta}_t}_{\text{Sampled from } q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)})-\boldsymbol{\varepsilon}_t\right\|^2\right)$$

Therefore, any new family of forward processes that use this same diffusion kernel (i.e., able to sample $\boldsymbol{x}_t$ conditioned to only $\boldsymbol{x}_0$) can reuse a pre-trained DDPM model.

Denoising diffusion implicit model (DDIM)

Denoising diffusion implicit model (DDIM)

Forward process Use a family of non-Markovian forward distributions conditioned on the real image $\boldsymbol{x}_0$ and parametrized by a positive standard deviation $\boldsymbol{\sigma}$ defined as:

$$q_{\boldsymbol{\sigma}}(\boldsymbol{x}_1,\dots,\boldsymbol{x}_T\mid\boldsymbol{x}_0)=q_{\sigma_T}(\boldsymbol{x}_T\mid\boldsymbol{x}_0)\prod_{t=2}^{T}q_{\sigma_t}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)$$

where:

$$q_{\sigma_T}(\boldsymbol{x}_T\mid\boldsymbol{x}_0)=\mathcal{N}(0,\boldsymbol{I})$$
$$q_{\sigma_t}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)=\mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0+\sqrt{1-\alpha_{t-1}-\alpha_t^2}\frac{\boldsymbol{x}_t-\sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}};\sigma_t^2\boldsymbol{I}\right)$$

With this definition, it can be shown that:

$$q_{\sigma_t}(\boldsymbol{x}_t\mid\boldsymbol{x}_0)=\mathcal{N}(\sqrt{\alpha_t}\boldsymbol{x}_0;(1-\alpha_t)\boldsymbol{I})$$

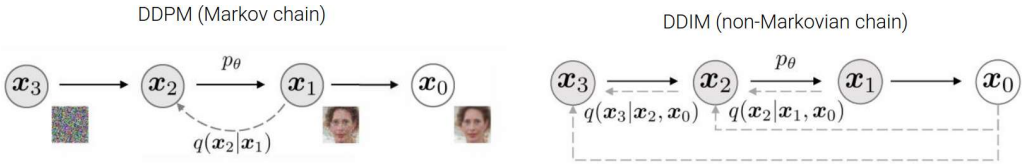
Remark. With a specific choice for $\boldsymbol{\sigma}$ ($\sigma_t=\sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t}}\sqrt{1-\frac{\alpha_t}{\alpha_{t-1}}}$), it is possible to obtain DDPM (i.e., DDIM is a generalization of DDPM).
In practice, instead of tuning σ_t directly, a proxy hyperparameter η is used as

follows:

$$\sigma_t(\eta) = \eta\sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t}}\sqrt{1-\frac{\alpha_t}{\alpha_{t-1}}}$$

In other words, η controls σ_t using the DDPM model as reference (with $\eta = 1$ resulting in DDPM).

Remark. With $\sigma_t \to 0$, the generation process becomes more deterministic. With $\sigma_t = 0$ ($\eta = 0$), the mean is always sampled (i.e., fully deterministic).



Reverse process Given a latent $\boldsymbol{x}_t$ and a DDPM model $\varepsilon_t(\cdot;\boldsymbol{\theta})$, generation at time step t is done as follows:

1. Compute an estimate of the real image for the current time step t:

$$\hat{\boldsymbol{x}}_0 = \frac{\boldsymbol{x}_t - \sqrt{\alpha_{t-1}}\varepsilon_t(\boldsymbol{x}_t;\boldsymbol{\theta})}{\sqrt{\alpha_t}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$$

   Note that the formula comes from the usual $\boldsymbol{x}_t = \sqrt{\alpha_t}\boldsymbol{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\varepsilon}_t$.

2. Sample the next image from:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = q_{\boldsymbol{\sigma}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, f_{\boldsymbol{\theta}}(\boldsymbol{x}_t))$$

   (i.e., $\boldsymbol{x}_0$ in $q_{\boldsymbol{\sigma}}$ has been replaced with an estimation of it). The image is obtained as:

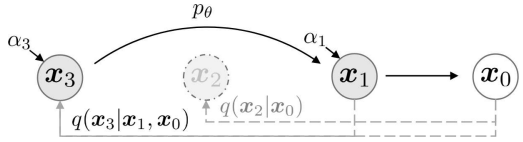
$$\boldsymbol{x}_{t-1} = \boldsymbol{\mu}_{q_{\boldsymbol{\sigma}}} + \boldsymbol{\Sigma}_{q_{\boldsymbol{\sigma}}}\boldsymbol{\varepsilon}$$

Accelerate sampling Use a forward process that only considers a subset of time steps. This allows to skip k steps in the reverse process as: Accelerate sampling

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-k} \mid \boldsymbol{x}_t) &= q_{\boldsymbol{\sigma}}(\boldsymbol{x}_{t-k} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \\ &= \mathcal{N}\left(\sqrt{\alpha_{t-k}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-k}-\sigma_t^2}\frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}; \sigma_t^2\boldsymbol{I}\right) \end{aligned}$$

Remark. Skipped steps are actually present in the forward process as during training all steps are considered. Therefore, it is only possible to skip steps during inference.

Remark. It has been observed that determinism ($\sigma_t = 0/\eta = 0$) with accelerated generation has the best performance.

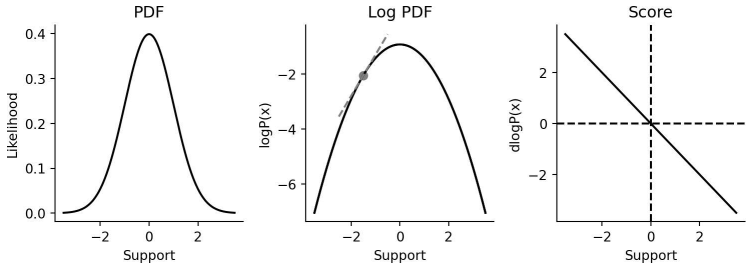


8.3.5 Interpretation of diffusion models as score estimators

Score function Given a probability density function $p(x)$, its score function is defined as: Score function

$$s(x) = \nabla_x\left[\log(p(x))\right]$$

Remark. Differently from a probability distribution, the score function s does not have to be normalized to sum 1. Therefore, it is easier to approximate with a neural network $s(x; \theta)$.

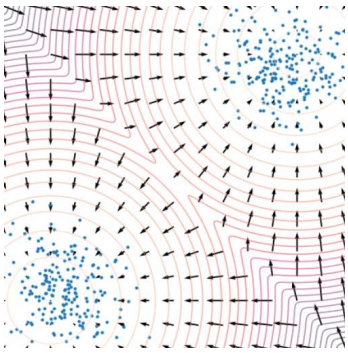


The score function indicates which direction maximally increases $p(x)$. In other words, it defines a vector field that points towards the modes of $p(x)$.

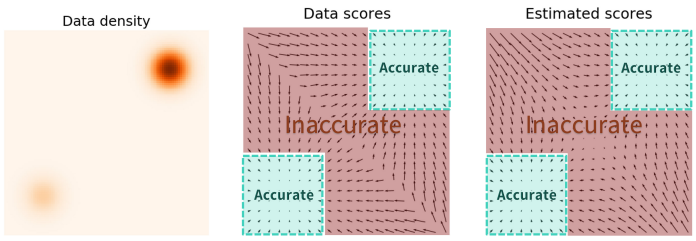
Langevin dynamics Method to sample from a score function as: Langevin dynamics

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t + c\nabla_{\boldsymbol{x}} \left[\log(p(x))\right] + \sqrt{2c}\boldsymbol{\varepsilon}$$

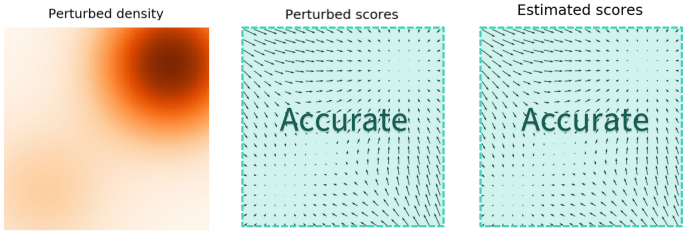
where c is a hyperparameter.



Remark. Score functions are inaccurate in low density regions. Therefore, sampling is inaccurate in areas with fewer data points.



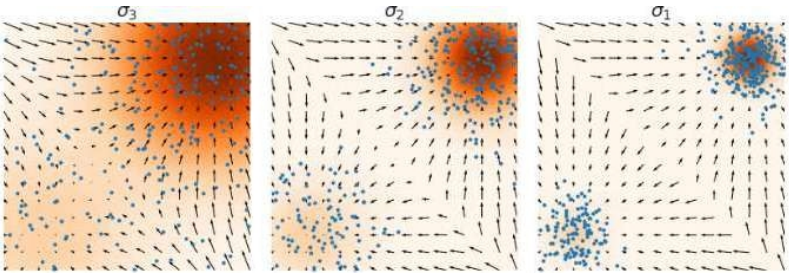
Langevin dynamics with noise Add noise to the original data to make the trained score function more robust. Langevin dynamics with noise



Remark. Larger scales of noise significantly alter the original distribution. Smaller scales of noise do not cover enough low density regions.

Annealed Langevin dynamics Use multiple scales of noise to estimate a family of score functions $s_t(\boldsymbol{x}_t; \boldsymbol{\theta})$. Then, run a few steps of Langevin dynamics for $t = T, \dots, 1$, each time restarting from $t - 1$.

Annealed Langevin dynamics



Diffusion model as score estimator The score function of an isotropic Gaussian distribution is:

$$\begin{aligned} s(\boldsymbol{x}) &= \nabla_{\boldsymbol{x}} \left[\log(p(\boldsymbol{x}))\right] \qquad \text{with } \boldsymbol{x} \sim p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}; \sigma^2 \boldsymbol{I}) \\ &= \nabla_{\boldsymbol{x}} \left[\log\left(\frac{1}{c} e^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^2}{2\sigma^2}}\right)\right] \\ &= \nabla_{\boldsymbol{x}} \left[\log\left(\frac{1}{c}\right)\right] + \nabla_{\boldsymbol{x}} \left[-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^2}{2\sigma^2}\right] \\ &= -\frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sigma^2} \end{aligned}$$

As it holds that:

$$\boldsymbol{x} = \boldsymbol{\mu} + \sigma\boldsymbol{\varepsilon} \iff \boldsymbol{\varepsilon} = \frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sigma} \qquad \text{with } \boldsymbol{\varepsilon} \sim \mathcal{N}(0; \boldsymbol{I})$$

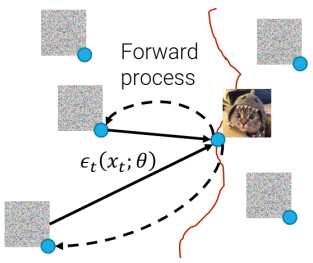
The score function can be rewritten as an estimator of the Gaussian noise:

$$s(\boldsymbol{x}) = -\frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sigma^2} = -\frac{\boldsymbol{\varepsilon}}{\sigma}$$

Therefore, as diffusion models learn to predict $\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta})$ from a Gaussian with $\sigma = \sqrt{1-\alpha_t}$, they can be seen as score functions with a scaling factor $-\sigma$:

$$s(\boldsymbol{x}) = -\frac{\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta})}{\sqrt{1-\alpha_t}} \iff \varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) = -\sqrt{1-\alpha_t} s(\boldsymbol{x})$$

As a result, diffusion models implicitly perform annealed Langevin dynamics when generating an image.



8.3.6 Generation conditioning

One-hot class conditioning Condition generation based on a class c. The model predicting noise becomes:

One-hot class conditioning

$$\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta})$$

Architecturally, similarly to time conditioning, the one-hot encoding of the class is refined through fully-connected layers to create an embedding that is appended to the image activations.

> **Remark.** This works as conditioning the likelihood with a class c does not change the previous proofs.

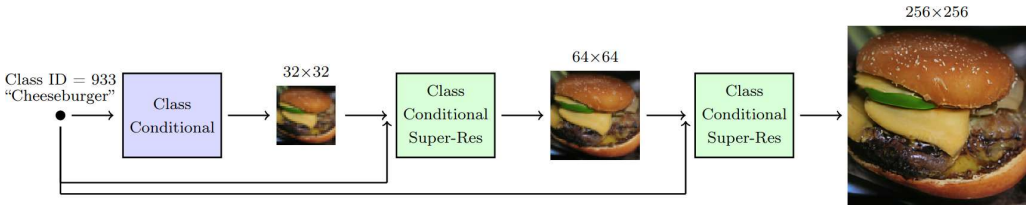
Cascaded diffusion models Approach to generate high resolution images starting from some class conditioning.

Cascaded diffusion models

Given a standard diffusion model d_1 and a series of super-resolution diffusion models $d_2, \dots, d_n$ with increasing resolution, the generation of an image of class c is done as follows:

1. Use the first diffusion model d_1 to generate a starting low-resolution image $\boldsymbol{I}_1$ from a latent and the class c.
2. Iterate over the super-resolution diffusion models $i = 2, \dots, n$:
   a) Up-sample the previously generated image $\boldsymbol{I}_{i-1}$ to match the shape of the current diffusion model d_i.
   b) Generate a higher resolution image $\boldsymbol{I}_i$ using the diffusion model d_i from a latent conditioned on the class c and the previous image $\boldsymbol{I}_{i-1}$ (which is concatenated along the spatial dimension of the latent).

> **Remark.** Higher-resolution models in the pipeline can be seen as detail generators.



Classifier guidance Use a classifier to compute $p_{\text{cls}}(c \mid \boldsymbol{x}_t, t)$ and guide generation to a class c. With the interpretation of diffusion models as score estimators, the classifier is used to steer the trajectories of Langevin dynamics given the latent $\boldsymbol{x}_t$ at time t.

Classifier guidance

Given a latent classifier, the class-guided noise can be predicted as follows:

$$\begin{aligned}
\varepsilon_t^{\text{cls}}(\boldsymbol{x}_t, c; \boldsymbol{\theta}) &= -\sqrt{1-\alpha_t}\nabla_{x_t}\Big[\log(p(\boldsymbol{x}_t, c))\Big] \\
&= -\sqrt{1-\alpha_t}\nabla_{x_t}\Big[\log\big(p(\boldsymbol{x}_t)p_{\text{cls}}(c \mid \boldsymbol{x}_t, t)\big)\Big] \\
&= -\sqrt{1-\alpha_t}\nabla_{x_t}[\log(p(\boldsymbol{x}_t))] - \sqrt{1-\alpha_t}\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))] \\
&= -\sqrt{1-\alpha_t}s(\boldsymbol{x}_t) - \sqrt{1-\alpha_t}\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))] \\
&= \varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) - \sqrt{1-\alpha_t}\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))]
\end{aligned}$$

In practice, a weight w is used to control the strength of guidance:

$$\varepsilon_t^{\text{cls}}(\boldsymbol{x}_t, c; \boldsymbol{\theta}) = \varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) - w\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))]$$

Remark. Guidance allows to balance the trade-off between realism and coverage (in a better way than one-hot conditioning).

Remark. The best results have been obtained by using an already conditional diffusion model. Therefore, $\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta})$ can be substituted in place of $\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta})$.

Remark. The classifier usually has to be trained on latents from scratch and it is domain specific.

Classifier-free guidance

Classifier-free guidance Generation guidance method that does not require a classifier for latents.

Consider the formulation of classifier guidance starting with a conditional generator:

$$\varepsilon_t^{\text{cls}}(\boldsymbol{x}_t, c; \boldsymbol{\theta}) = \varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta}) - w\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))]$$

By applying Bayes' rule on the second term, we have that:

$$\begin{aligned} w\nabla_{x_t}[\log(p_{\text{cls}}(c \mid \boldsymbol{x}_t, t))] &= w\nabla_{x_t}\left[\log\left(p(\boldsymbol{x}_t \mid c)\frac{p(c)}{p(\boldsymbol{x}_t)}\right)\right] \\ &= w\nabla_{x_t}\left[\log(p(\boldsymbol{x}_t \mid c))\right] + w\nabla_{x_t}\left[\log(p(c))\right] - w\nabla_{x_t}\left[\log(p(\boldsymbol{x}_t))\right] \\ &= w\nabla_{x_t}\left[\log(p(\boldsymbol{x}_t \mid c))\right] + 0 - w\nabla_{x_t}\left[\log(p(\boldsymbol{x}_t))\right] \\ &\approx -w\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta}) + w\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) \end{aligned}$$

Therefore, for guidance without a classifier, two models are required:

- A conditional generative model (i.e., $\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta})$).
- An unconditional generative model (i.e., $\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta})$).

The overall class guided noise is computed as:

$$\begin{aligned} \varepsilon_t^{\text{cls}}(\boldsymbol{x}_t, c; \boldsymbol{\theta}) &= \varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta}) + w\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta}) - w\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) \\ &= (1 + w)\varepsilon_t(\boldsymbol{x}_t, c; \boldsymbol{\theta}) - w\varepsilon_t(\boldsymbol{x}_t; \boldsymbol{\theta}) \end{aligned}$$

Remark. In practice, a single model is used for both conditional and unconditional generation.

Training The model is trained as a one-hot class conditioned model. In addition, with probability p_{uncond} (e.g., 0.1), training is done unconditioned (i.e., the one-hot vector is zeroed).

Remark. During inference, the model has to be run twice on the latent to compute the conditioned and unconditioned noise.

Text conditioning

Text conditioning Embed text using an encoder and use the outputs at each token as keys and values of the cross-attentions in U-Net while the queries come from the image.

Training Similarly to classifier-free guidance, the model is trained both with and without a conditioning prompt.

Remark. The training procedure can also be generalized to negative prompts.

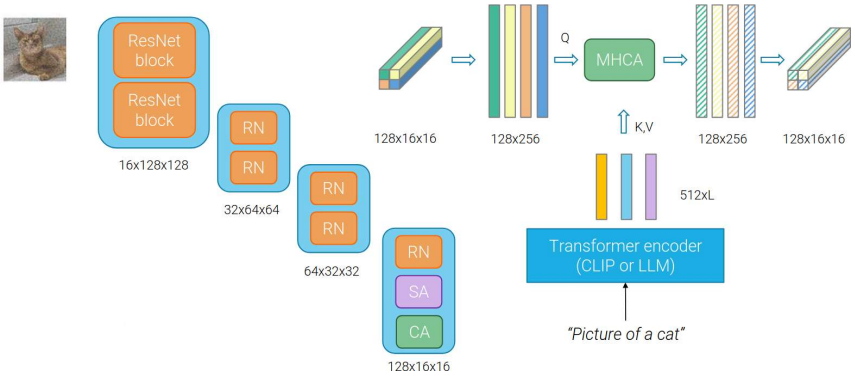
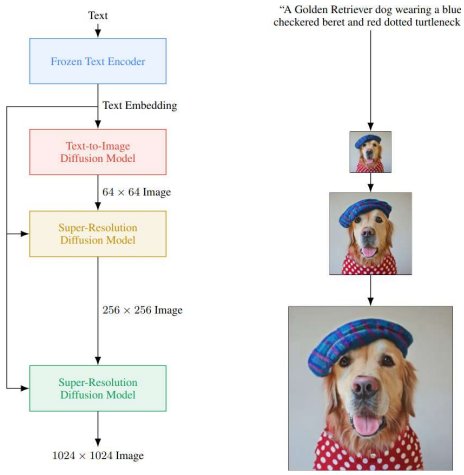


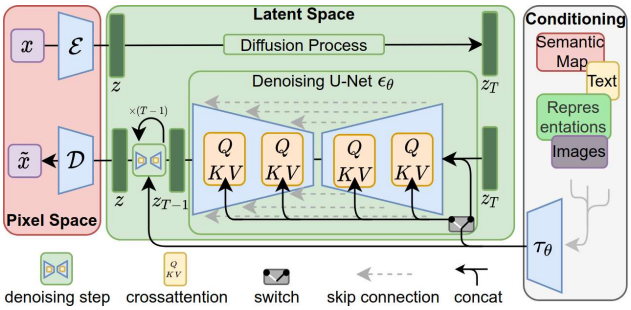
Imagen Architecture based on the following steps: Imagen

1. Embed a prompt using a frozen text encoder.
2. Generate an initial low-resolution image using a diffusion model that takes as input only the prompt.
3. Pass the low-resolution image and the prompt embeddings through a series of super-resolution diffusion models.



8.3.7 Latent diffusion models

Latent diffusion model Use an autoencoder to generate a compressed latent image to pass through the diffusion model and decode it at the end of generation. Latent diffusion model



Stable diffusion Model based on latent diffusion with text conditioning. Stable diffusion

<end of course>