Machine Learning for Computer Vision

Last update: 21 October 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

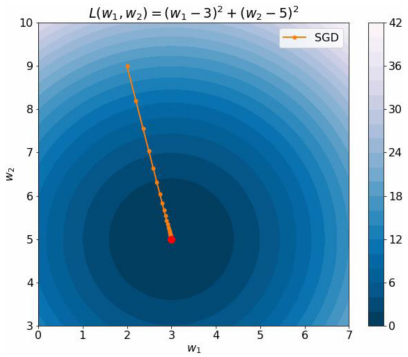
Stochastic gradient descent (SGD) Gradient descent based on a noisy approximation of the gradient computed on mini-batches of B data samples.

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

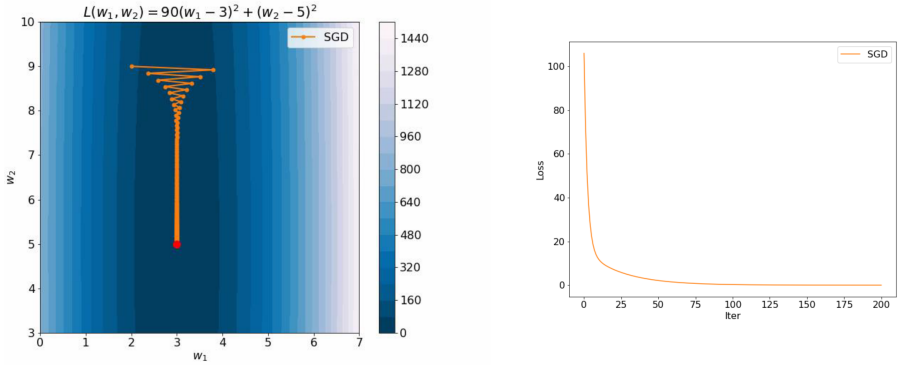
SGD spheres

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.



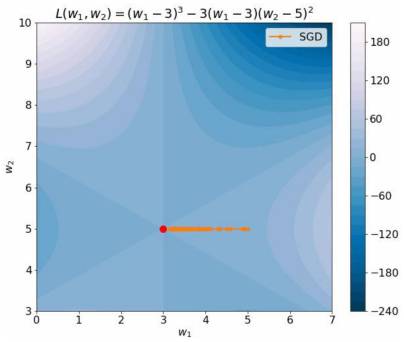
SGD canyons

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.

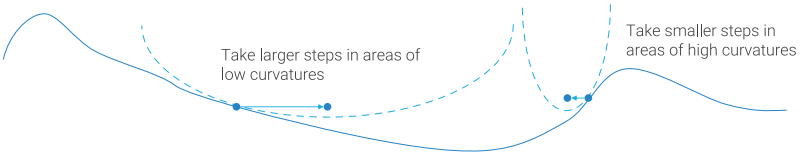
Newton's method Second-order method for the 1D case based on the Taylor expansion:

Newton's method

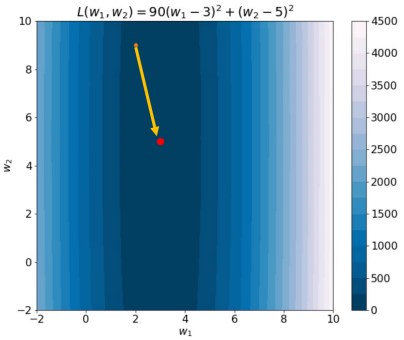
$$f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\Delta x + \frac{1}{2} f''(x_t)\Delta x^2$$

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 updates 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).

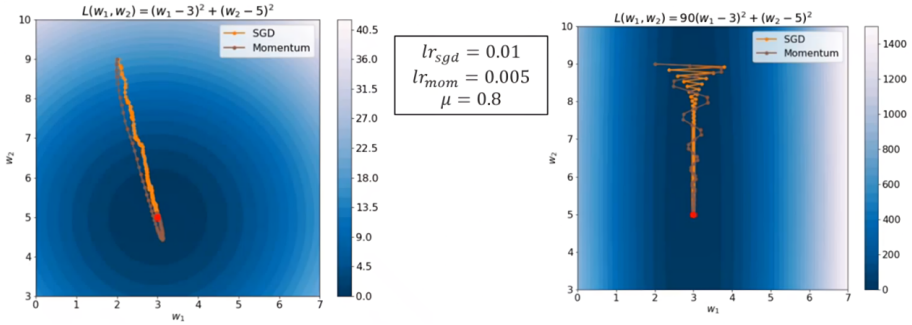


Figure 1.1: Plain SGD vs momentum SGD in a sphere and a canyon. In both cases, momentum converges before SGD.

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 $\boldsymbol{\theta}^{(t)}$ has been partially updated.

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

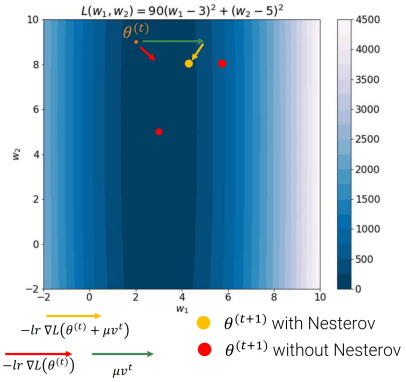


Figure 1.2: Visualization of the step in Nesterov momentum

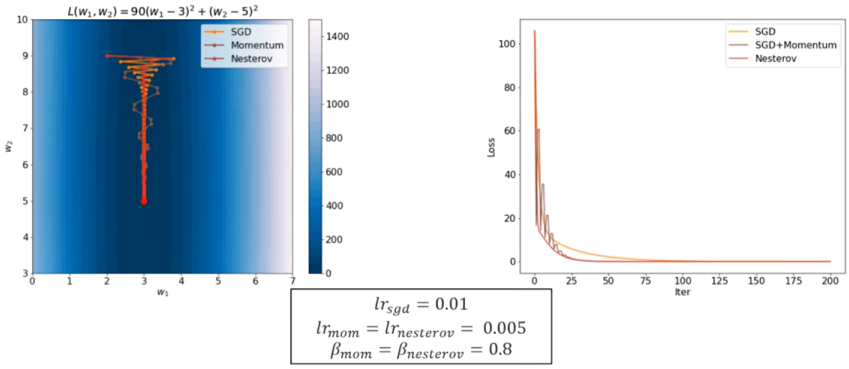


Figure 1.3: Plain SGD vs standard momentum vs Nesterov momentum

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.

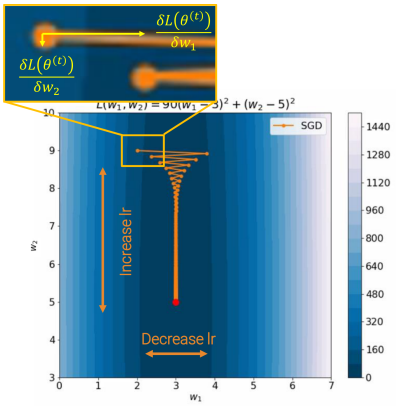


Figure 1.4: Loss where the w_1 parameter has a larger gradient, while w_2 has a smaller gradient

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.

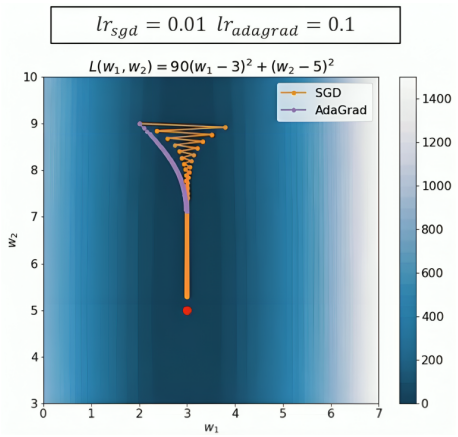


Figure 1.5: SGD vs AdaGrad. AdaGrad stops before getting close to the minimum.

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 and slows down reaching similar performances as SGD.

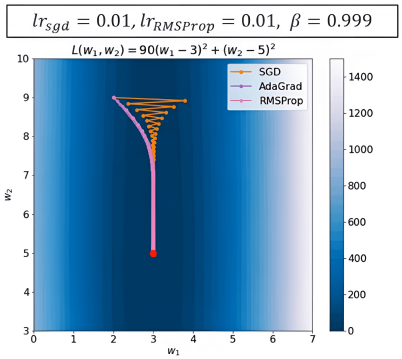


Figure 1.6: SGD vs AdaGrad vs RMSProp

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:

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

where the denominator $(1 - \beta_i^{t+1}) \to 1$ 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.

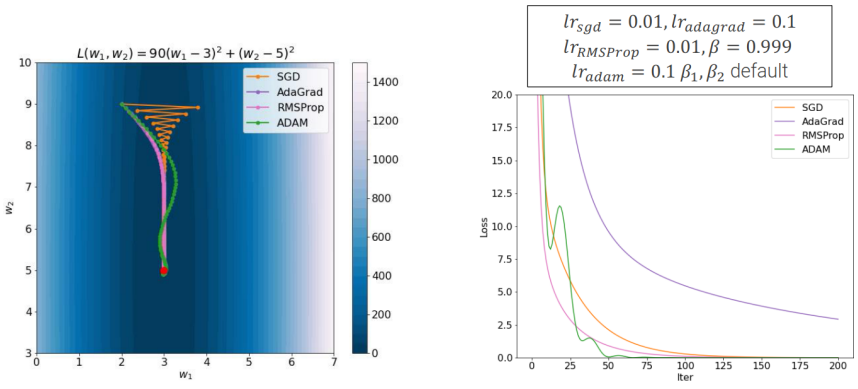


Figure 1.7: SGD vs AdaGrad vs RMSProp vs Adam

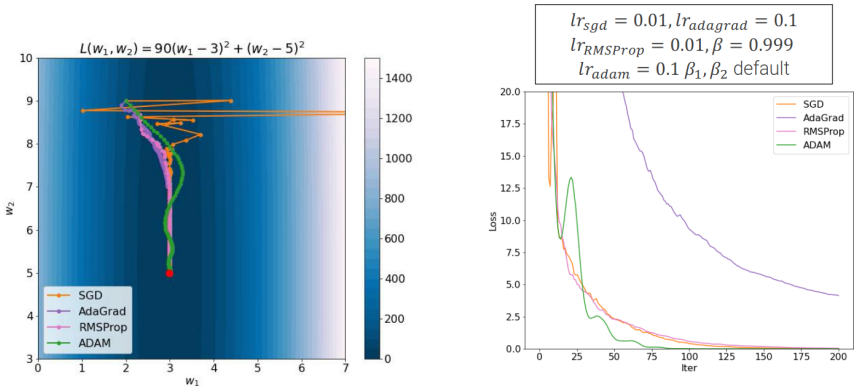
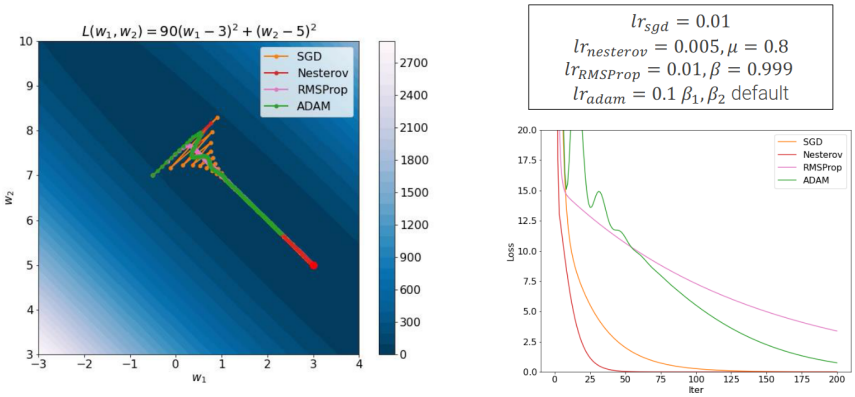


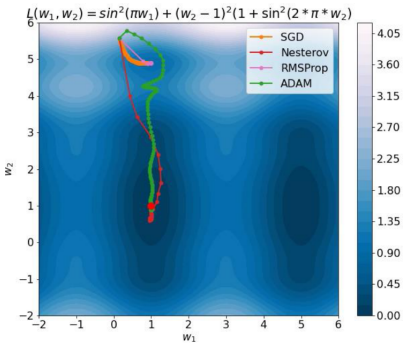
Figure 1.8: SGD vs AdaGrad vs RMSProp vs Adam with a smaller batch size

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" small basins.



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.

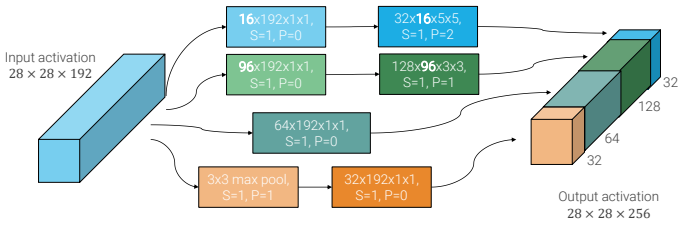


Figure 2.1: Inception module on the output of the stem layers

> **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

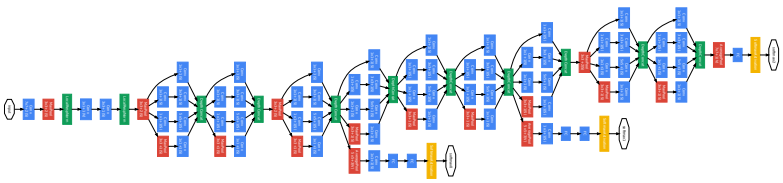


Figure 2.2: Architecture of Inception-v1

[1] Excerpt from IPCV2

2.2 Residual networks[2]

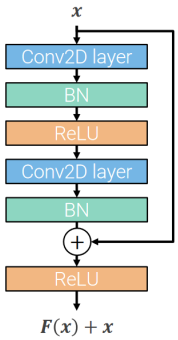
Standard residual block

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:

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

Skip connection

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



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

ResNet-18

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

- 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).

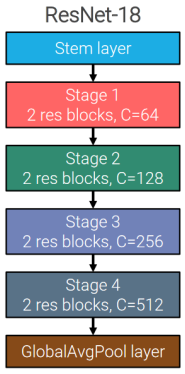


Figure 2.3: Architecture of ResNet-18

Bottleneck residual network

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:

- 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

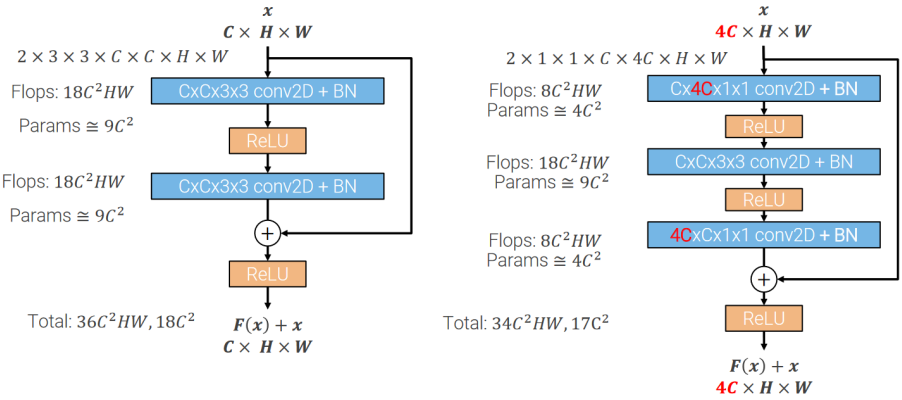


Figure 2.4: Standard residual block (left) and bottleneck block (right)

2.2.2 Inception-ResNet-v4

Network with bottleneck-block-inspired inception modules.

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

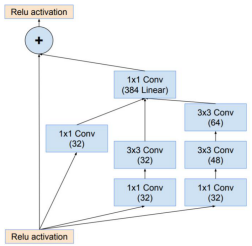
- Directly to the final concatenation.
- To a 3×3 convolution.
- To two 3×3 convolutions (i.e. a factorized 5×5 convolution).

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

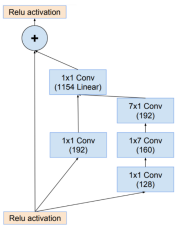
Inception-ResNet-B Three 1×1 convolutions are used to compress the input channels. Each of them leads to: Inception-ResNet-B

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

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



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



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

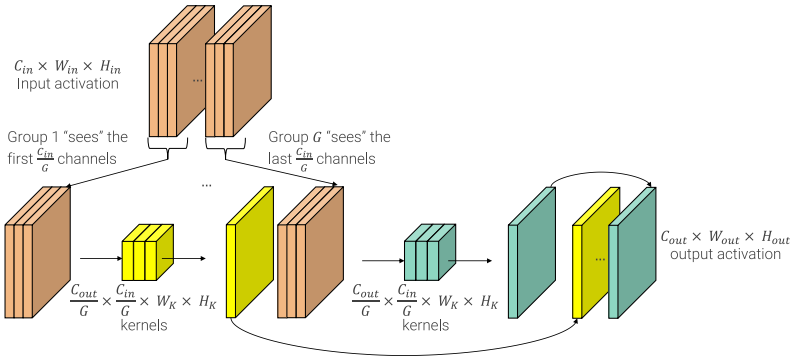
2.3 ResNeXt

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.

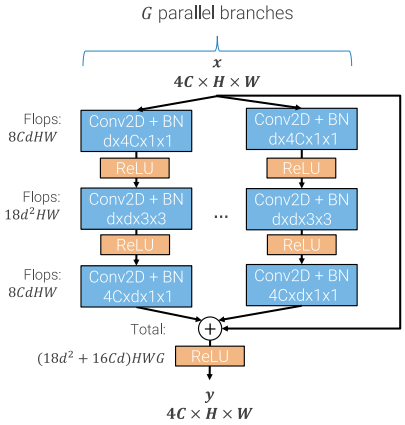


By processing the input in smaller chunks, there are the following gains:

- 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:

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

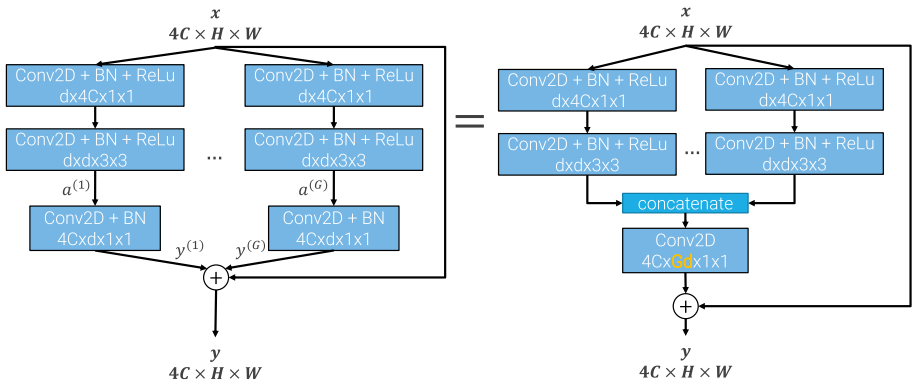
where:

- $*$ 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:

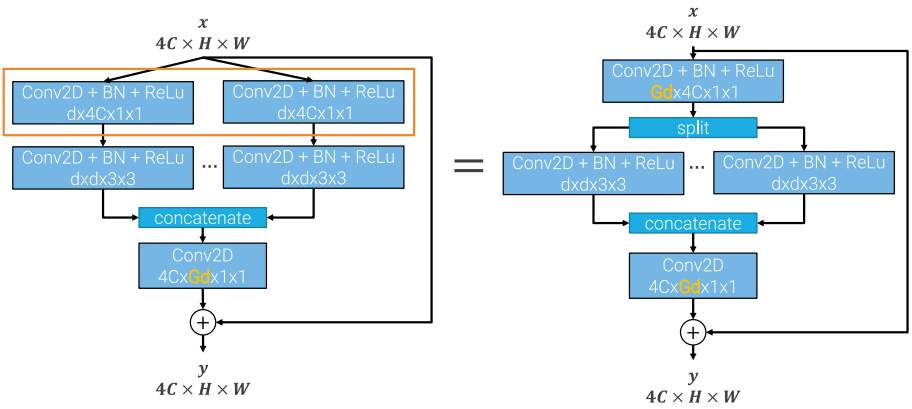
$$\begin{aligned}\mathbf{y}_k(j,i) &= \mathbf{w}_k^{(1)} \cdot \mathbf{a}^{(1)}(j,i) + \mathbf{w}_k^{(2)} \cdot \mathbf{a}^{(2)}(j,i) + \mathbf{x}_k \\ &= \underbrace{\left[\mathbf{w}_k^{(1)} \mathbf{w}_k^{(2)}\right]}_{\text{by stacking, this is a } 1\times 1 \text{ convolution with } 2d \text{ channels}}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

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.

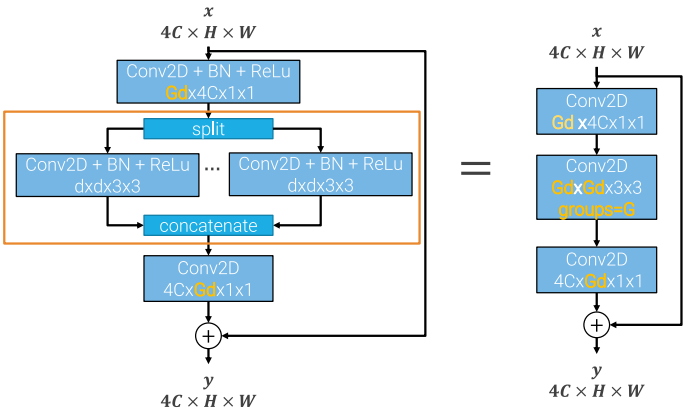


First 1×1 convolution The G 1×1 convolutions at the first layer of ResNeXt all process the same input $\mathbf{x}$. Trivially, this can also be represented using

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 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:

Squeeze-and-excitation module

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

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.

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

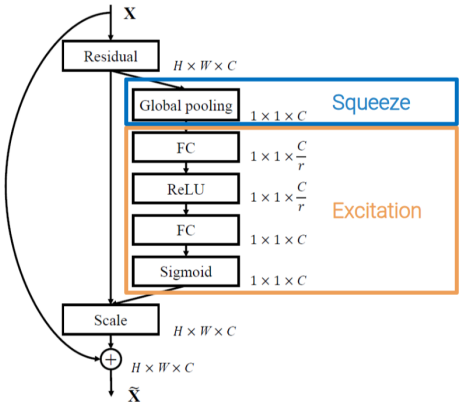


Figure 2.5: SE-ResNet module

2.5 MobileNetV2

Depth-wise separable convolution

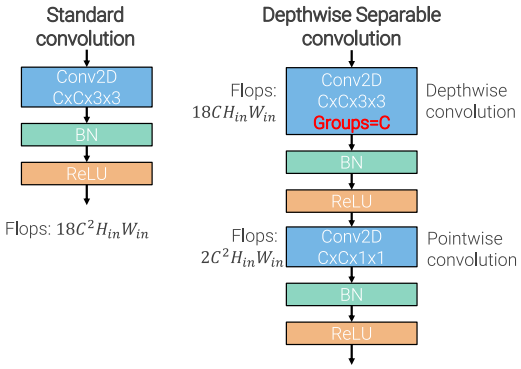
Depth-wise separable convolution Use grouped convolutions to reduce the computational cost of standard convolutions. The operations of filtering and combining features are split:

Depth-wise 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.



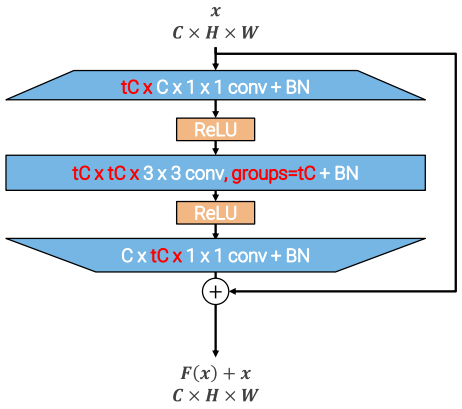
Remark. The 3×3 convolution in bottleneck residual blocks process a compressed version of the input activation, which might cause loss of information when passing through the ReLUs.

Inverted residual block Modified bottleneck block defined as follows:

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
2242×3	`conv2d`	-	32	1	2
1122×32	`bottleneck`	1	16	1	1
1122×16	`bottleneck`	6	24	2	2
562×24	`bottleneck`	6	32	3	2
282×32	`bottleneck`	6	64	4	2
142×64	`bottleneck`	6	96	3	1
142×96	`bottleneck`	6	160	3	2
72×160	`bottleneck`	6	320	1	1
72×320	`conv2d` 1×1	-	1280	1	1
72×1280	`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. It generally always improve 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

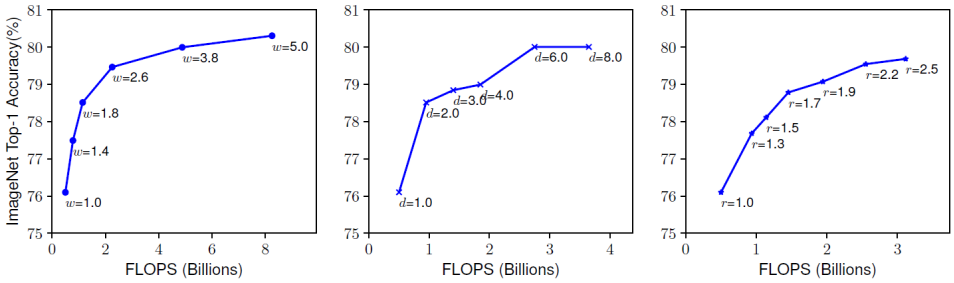


Figure 2.6: Top-1 accuracy variation with width, depth, and resolution scaling on EfficientNet

Compound scaling Scaling across multiple dimensions. — Compound scaling

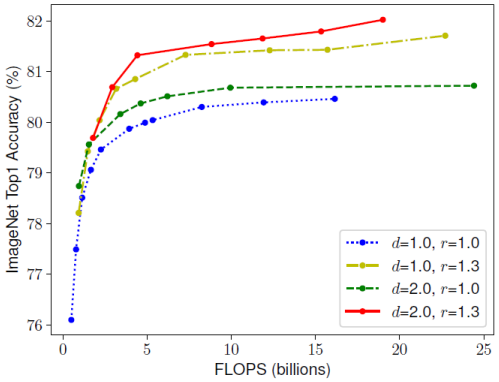


Figure 2.7: Width scaling for different fixed depths and resolutions

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 with depth but quadratically with width and resolution.

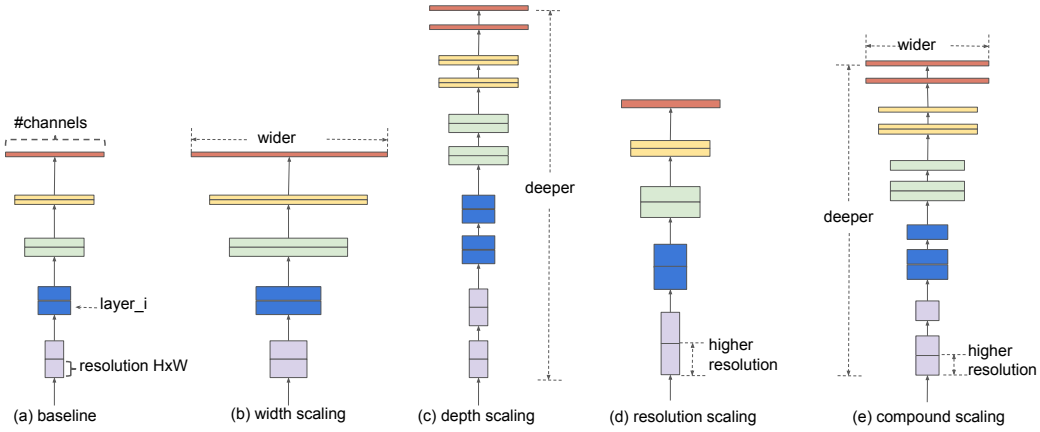
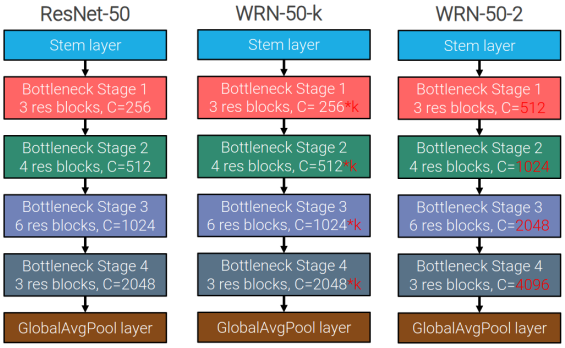


Figure 2.8: Model scaling approaches

2.6.1 Wide ResNet

Wide ResNet (WRN) ResNet scaled width-wise.

Wide ResNet (WRN)

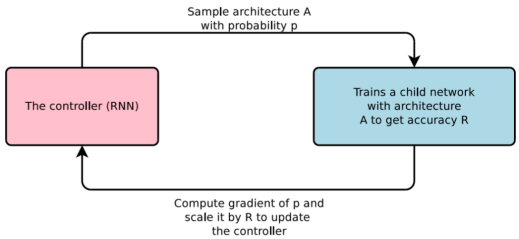


Remark. Wider layers are easier to parallelize on GPUs.

2.7 EfficientNet

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

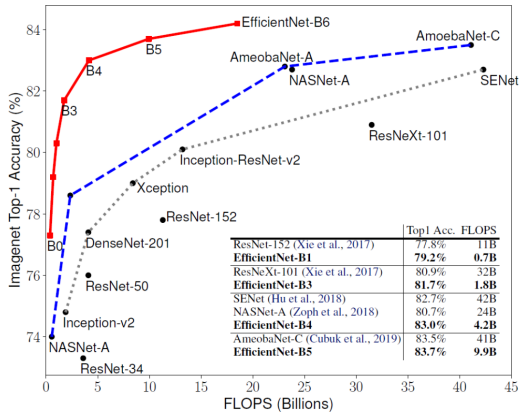
Neural architecture search (NAS)



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**

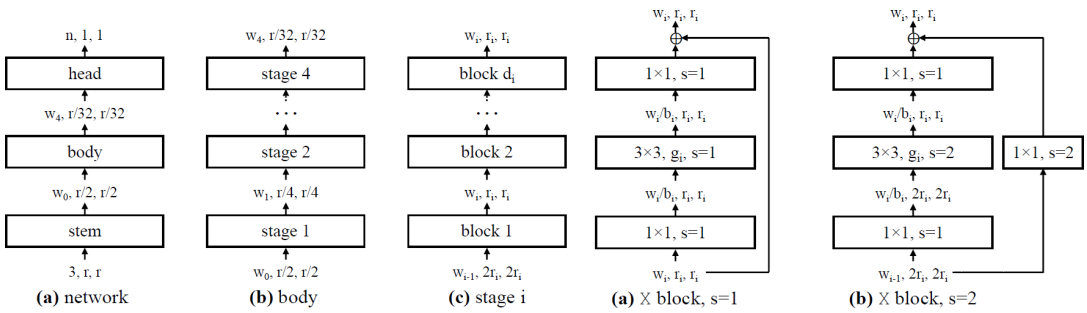
2.8 RegNet

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.

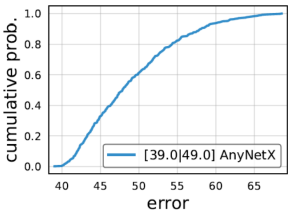


Figure 2.9: Example of cumulative distribution

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.

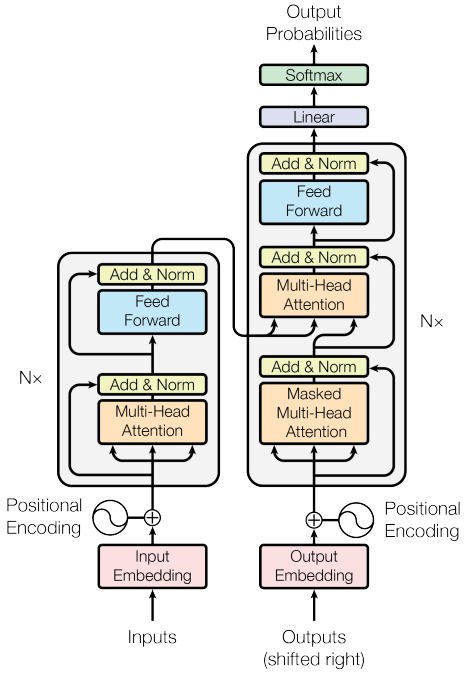
4. Repeat by fixing or finding relationships between parameters (i.e., try to reduce the search space).

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

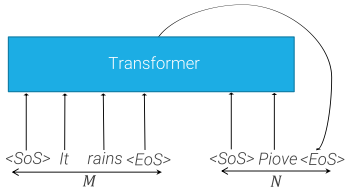


Figure 3.1: Example of autoregressive generation

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

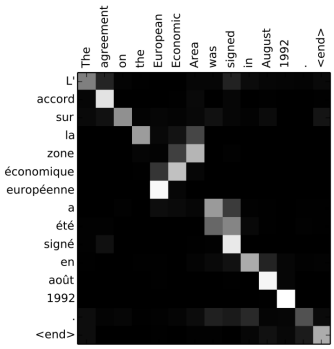


Figure 3.2: Attention weights for machine translation

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.

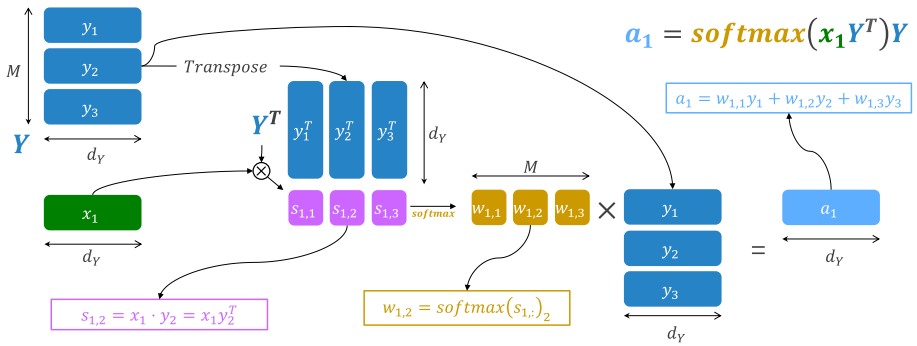
Remark. This is not the same attention of transformers as they do not directly compute attention weights between inputs and outputs.

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

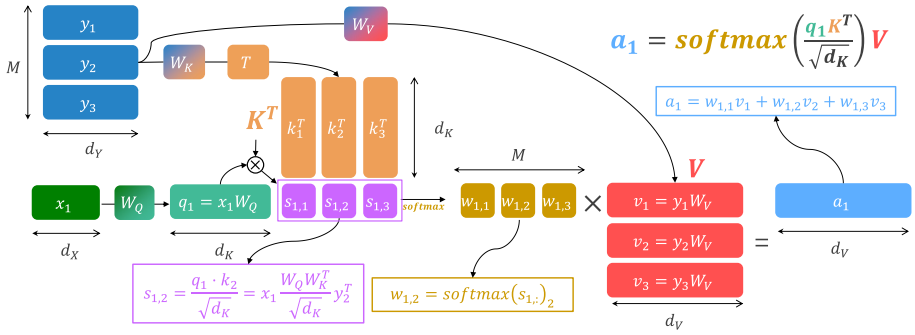
Values With the projection $\boldsymbol{W}_V \in \mathbb{R}^{d_Y \times d_V}$ such that $\mathbb{R}^{M \times d_V} \ni \boldsymbol{V} = \boldsymbol{Y}\boldsymbol{W}_K$, where d_V is the dimension of the values.

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:

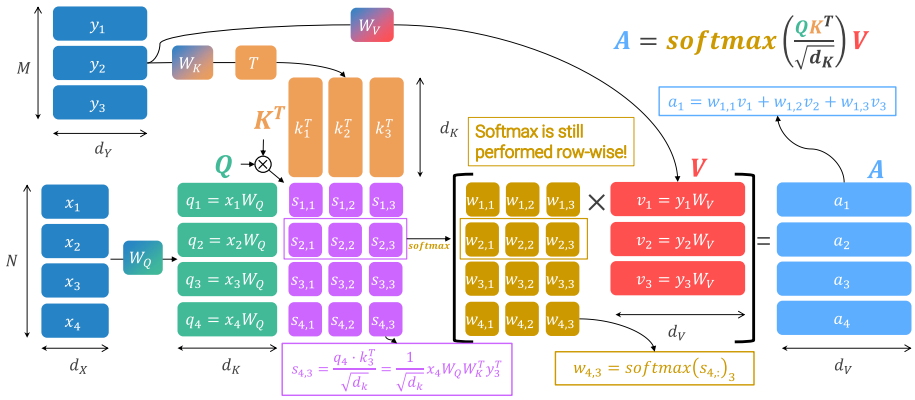
$$\mathbf{a}_1 = \texttt{softmax}\left(\frac{\mathbf{q}_1\boldsymbol{K}^T}{\sqrt{d_K}}\right)\boldsymbol{V}$$



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 $\boldsymbol{A} \in \mathbb{R}^{N \times d_V}$.

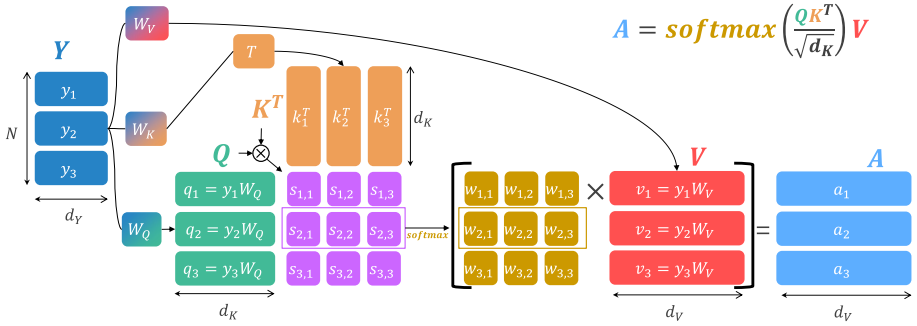
The overall attention mechanism can be defined as:

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



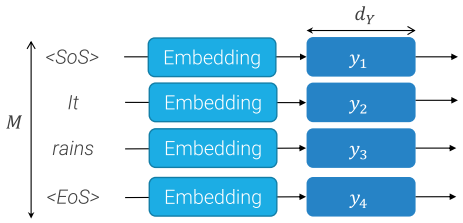
Self-attention Scaled dot-product attention mechanism where the inputs to compute keys, queries, and values are the same. Self-attention

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.

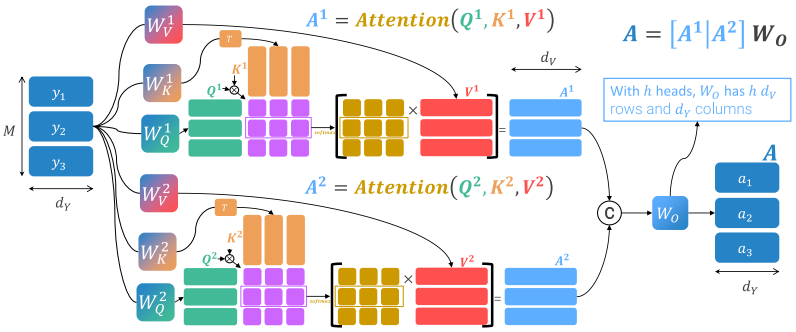


Figure 3.3: `MHSA` with two heads

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 train and inference time.

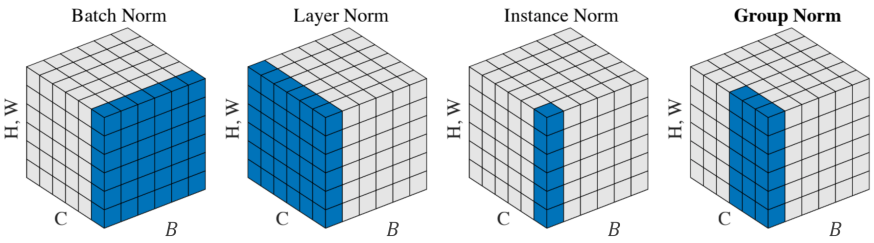


Figure 3.4: Affected axis of normalization methods

Feed-forward network (`FFN`) MLP with one hidden layer applied to each token independently. ReLU or one of its variants are 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:

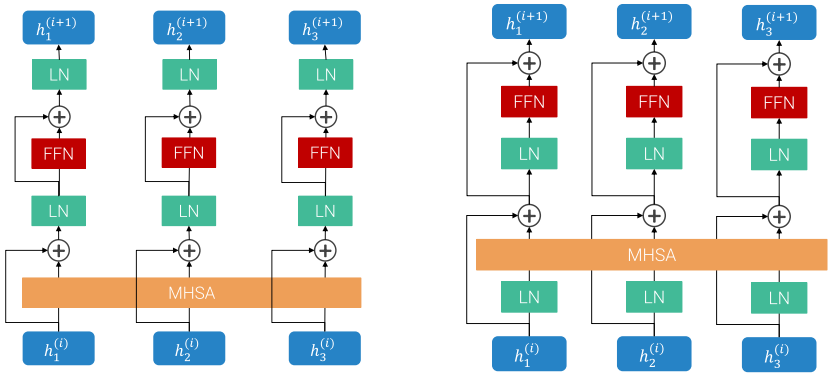
$$\bar{\mathbf{h}}_j^{(i)} = \texttt{LN}\left(\mathbf{h}_j^{(i)} + \texttt{MHSA}_{\boldsymbol{H}^{(i)}}(\mathbf{h}_j^{(i)})\right)$$
$$\mathbf{h}_j^{(i+1)} = \texttt{LN}\left(\bar{\mathbf{h}}_j^{(i)} + \texttt{FNN}(\bar{\mathbf{h}}_j^{(i)})\right)$$

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.

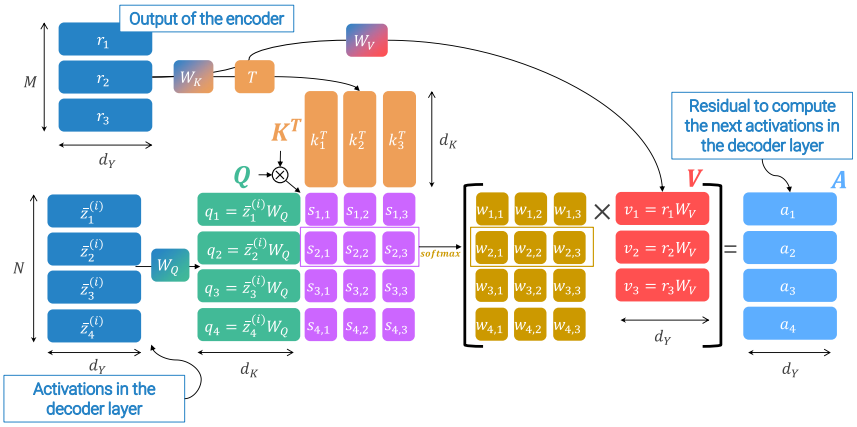


Figure 3.6: Cross-attention data flow

Feed-forward network MLP applied after cross-attention.

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

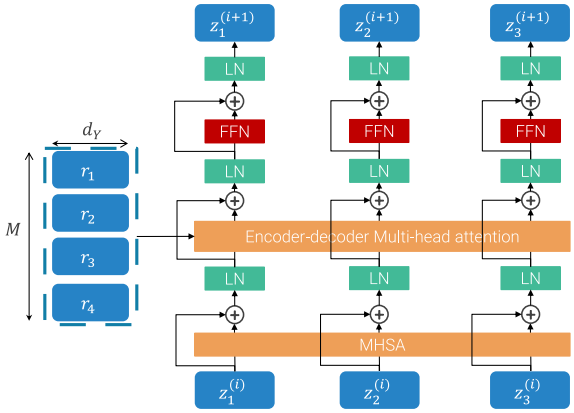


Figure 3.7: Decoder in post-norm transformer

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 $[\texttt{<SoS>}, t_1, \ldots, t_n, \texttt{<EoS>}]$, it is processed by the decoder in the following way:

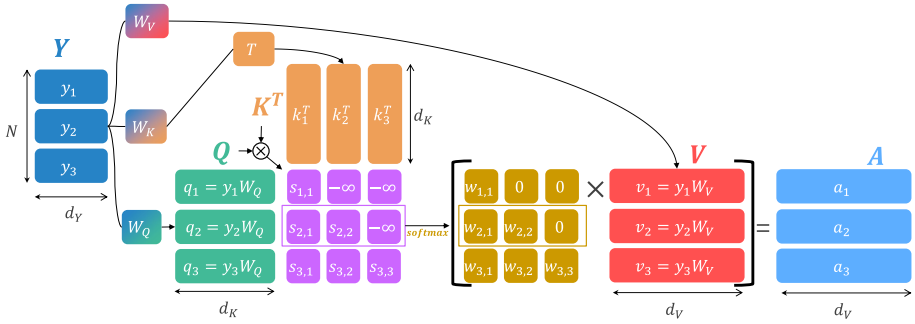
- The input is $[\texttt{<SoS>}, t_1, \ldots, t_n]$ (i.e., without end-of-sequence token).
- The expected output $[t_1, \ldots, t_n, \texttt{<EoS>}]$ (i.e., without start-of-sequence token).

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.

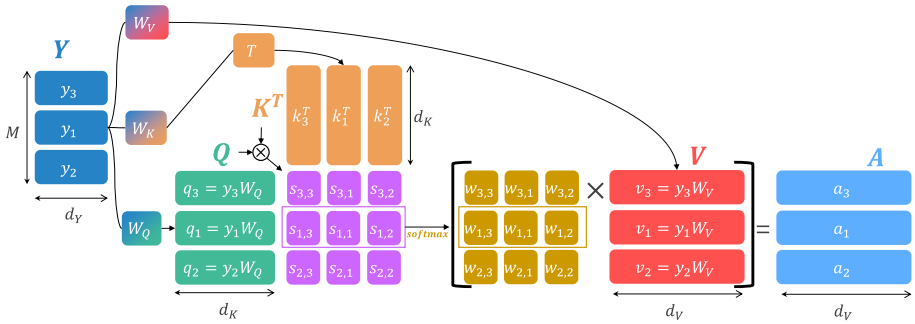
Masked self-attention Modification to self-attention to prevent tokens to attend at future positions (i.e., at their right). This can be done by either setting the similarity scores with future tokens to $-\infty$ or directly setting the corresponding attention weights to 0 (i.e., make the attention weights a triangular matrix).

Masked self-attention



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

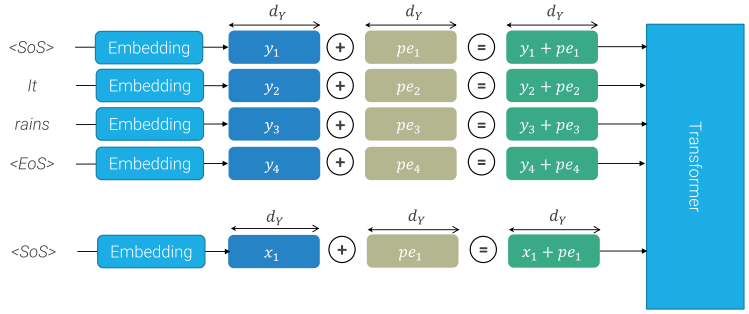
Fixed The vector associated to each position is fixed and known before training.

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

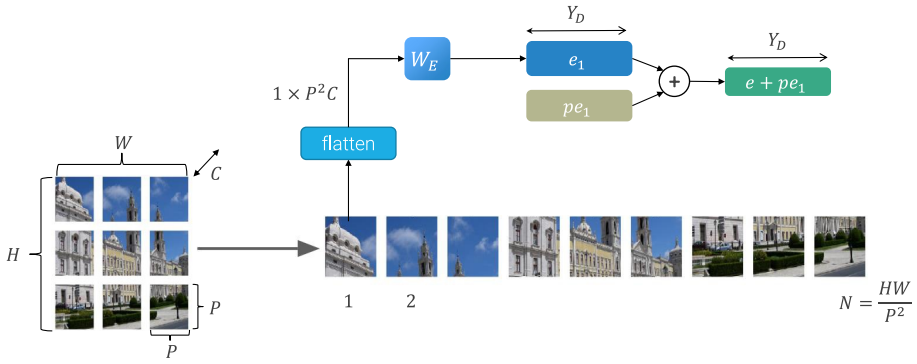
Remark. Using single pixels as tokens is unfeasible due to the complexity scaling of transformers as an $H \times W$ image results in an attention matrix of $(HW)^2$ entries.

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 together is not strictly necessary.

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: Patch

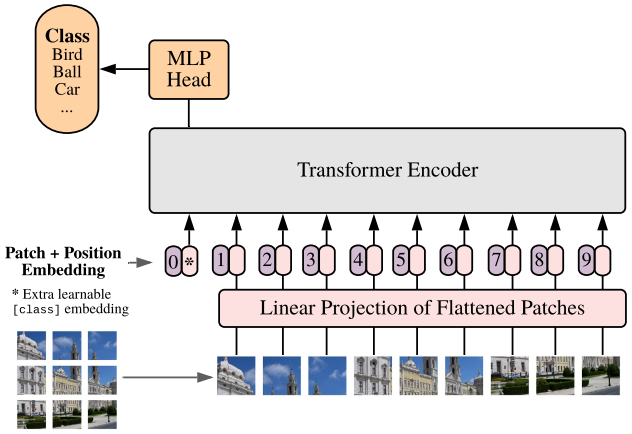
1. Flatten the patch into a P^2C vector.
2. Linearly transform it through a learned projection matrix $W_E \in \mathbb{R}^{P^2C \times Y_D}$.
3. Add positional information.



Vision transformer (ViT) 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.

Vision transformer (ViT)

> **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.

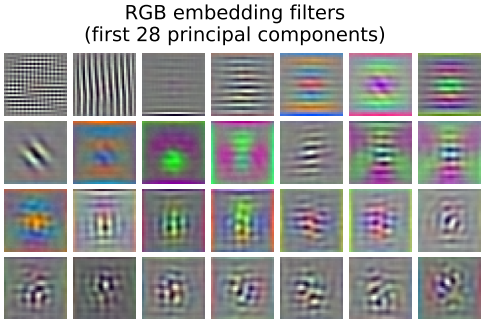


Figure 3.8: Visualization of the columns of the patches linear projection matrix W_E. Each column has shape $3P^2$ and can be reshaped to be a $3 \times P \times P$ image.

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

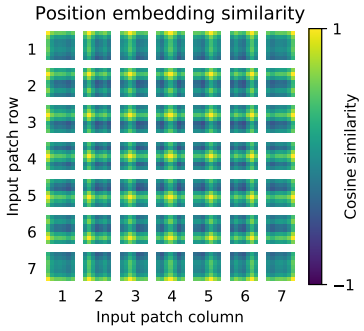


Figure 3.9: Cosine similarity of the positional encoding of each patch compared to all the others

- 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.

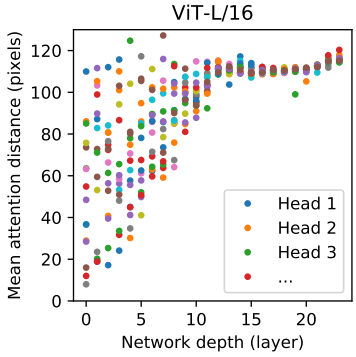


Figure 3.10: Mean attention distance of the heads of ViT-large/16

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

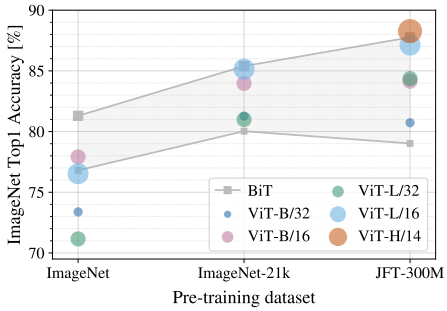


Figure 3.11: ImageNet top-1 accuracy with different pre-training datasets. BiT represents ResNet (two variants).

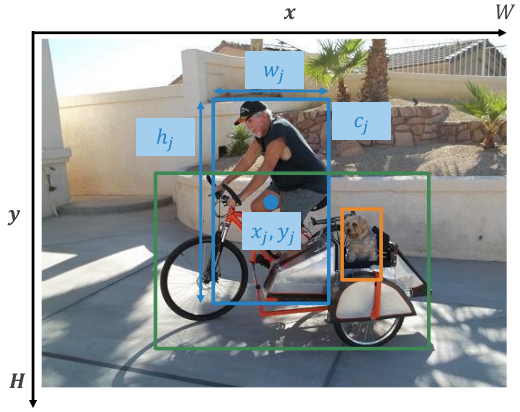
Remark. Comparison between convolutional neural networks and vision transformer is not straightforward.

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:

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

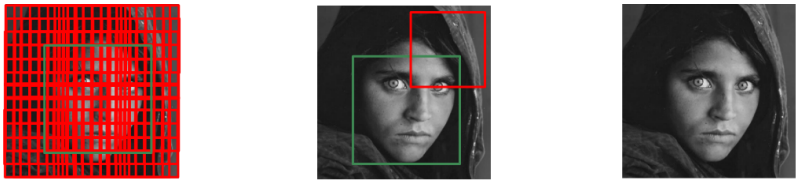


Figure 4.1: Recall and precision in different scenarios

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:

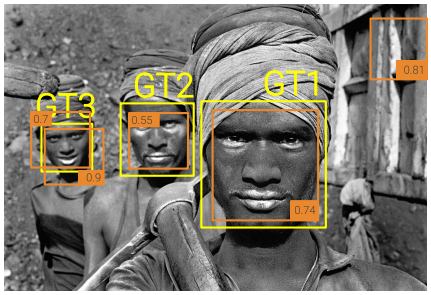
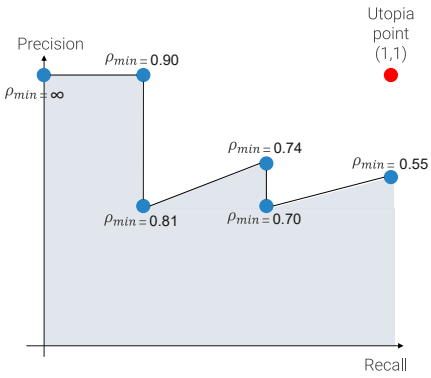


Figure 4.2: Ground-truth (yellow boxes) and predictions (orange boxes) with their confidence score

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:

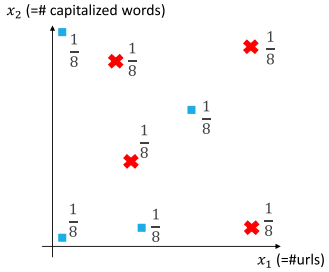
$$w^{(i)} = w^{(i)}\beta_j$$

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

3. Define the strong classifier as:

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

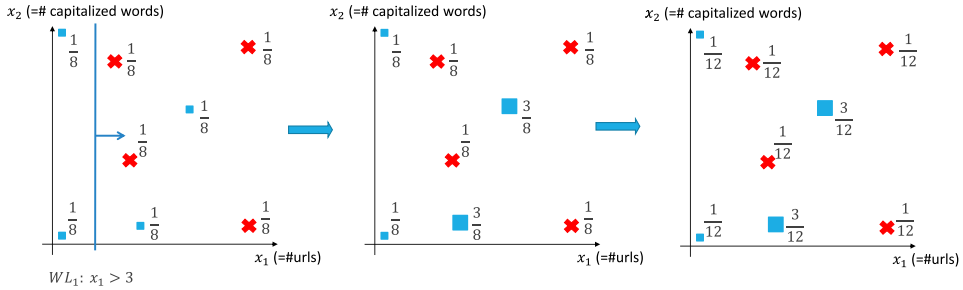
Example. Consider the problem of spam detection with two features x_1 and x_2 (number of URL and capitalized words, respectively). The training samples and their initial weights are the following:



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:

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

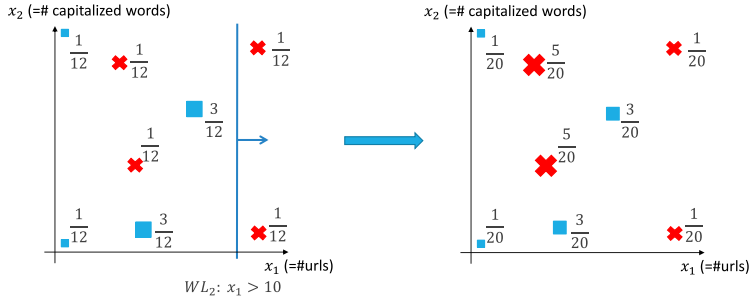
The new reweighed and normalized samples are:



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:

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

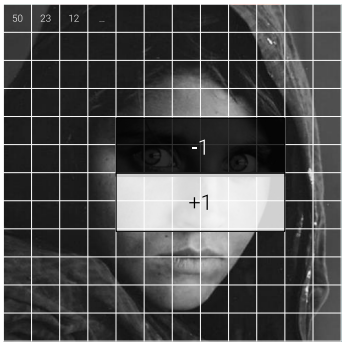
Haar-like features For face detection, a 24×24 patch of the image is considered (for now) and the weak classifiers define rectangular filters composed of 2 to 4 subsections applied at fixed positions of the patch. Haar-like features

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

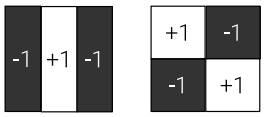
$$\texttt{WL}_j(x) = \begin{cases} 1 & \text{if } s_j f_j \geq s_j \rho_j \\ -1 & \text{otherwise} \end{cases}$$

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).

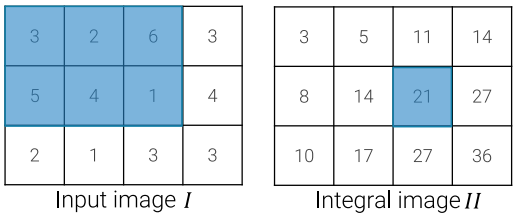


Figure 4.4: Example of integral image

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.

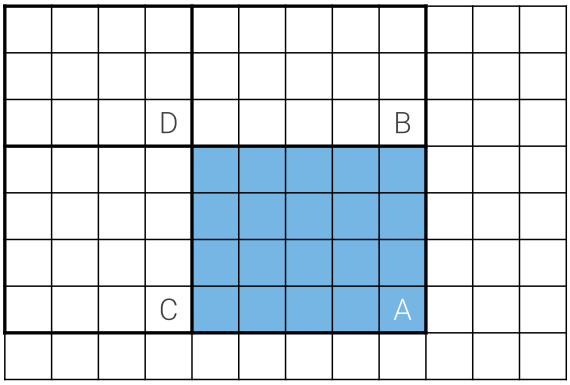


Figure 4.5: Summation of the pixels in the blue area

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.

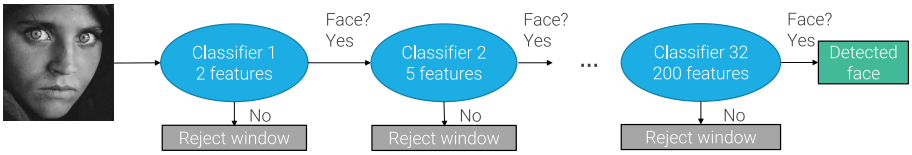
Remark. The integral image allows to compute the features in constant time independently of the patch size.

4.2.3 Cascade

Cascade

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.

|**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)

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:

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

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

CNN for object localization

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

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:

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

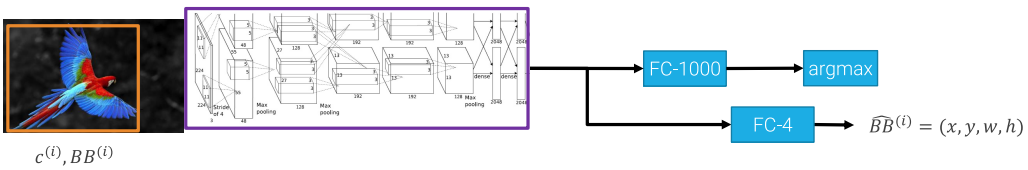


Figure 4.6: Localizer with AlexNet as feature extractor and 1000 classes

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

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.

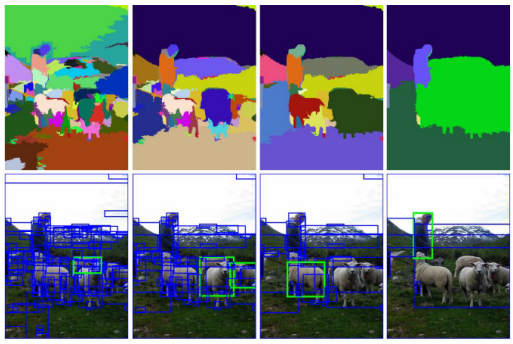


Figure 4.7: Example of some iterations of selective search

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).

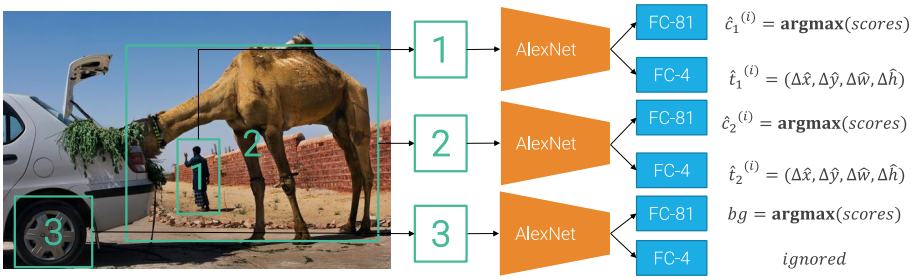


Figure 4.8: Example of R-CNN using AlexNet

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.

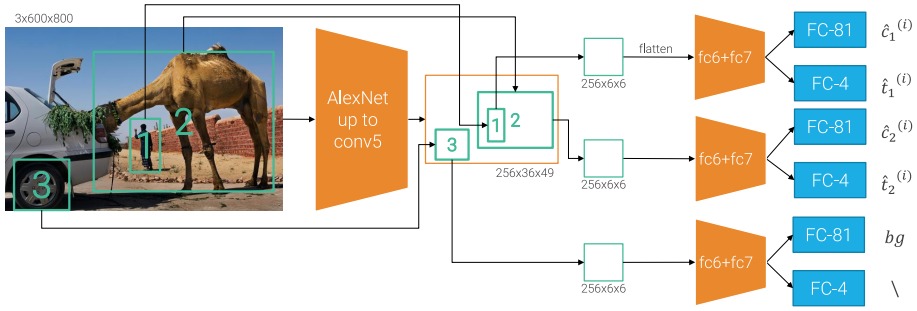


Figure 4.9: Example of fast R-CNN using AlexNet

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).

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

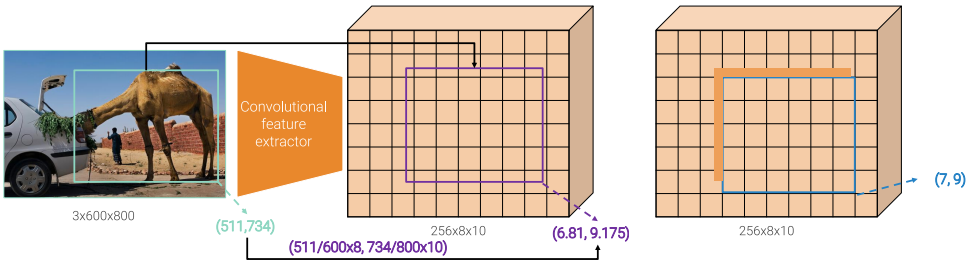


Figure 4.10: Project and snap operations

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.

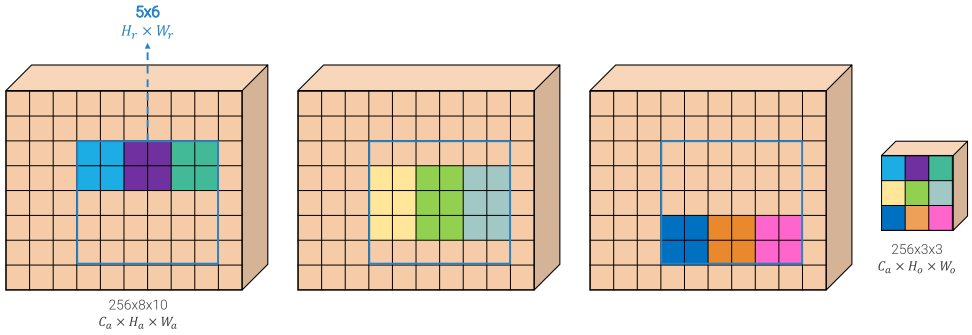


Figure 4.11: Pooling operation with varying kernel size

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.

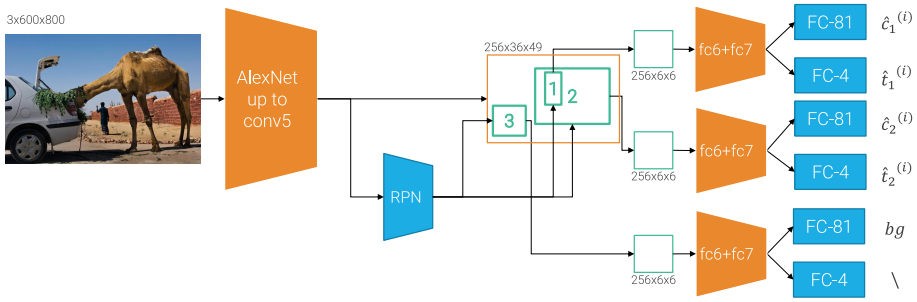


Figure 4.12: Example of faster R-CNN using AlexNet

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

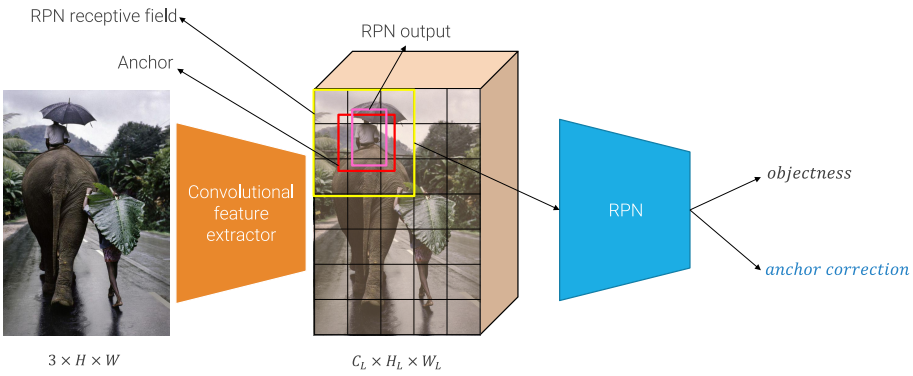


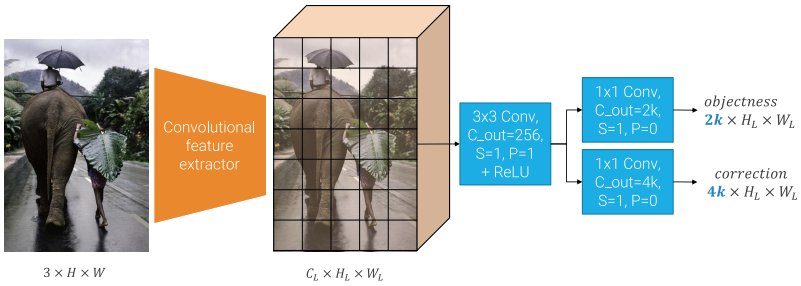
Figure 4.13: Example of an iteration of a 1-anchor RPN

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 0, 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.

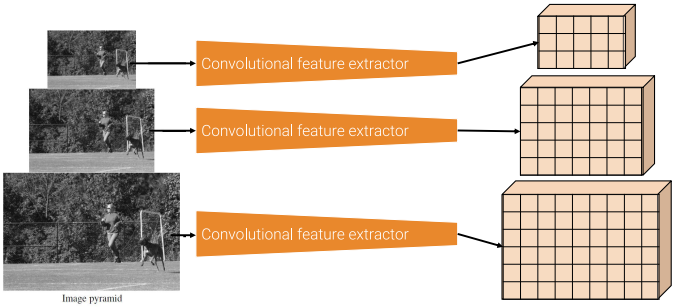
Remark. R-CNN is unable to detect objects smaller than the grid size.

4.3.3 Multi-scale detection

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

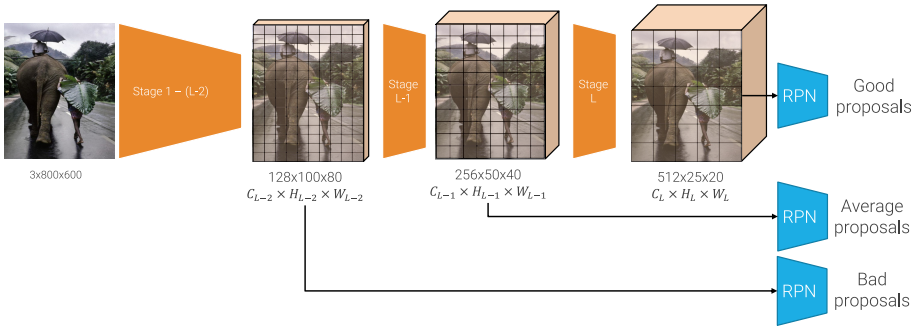
Remark. This approach creates effective features at different scales, but it is computationally expensive.



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)

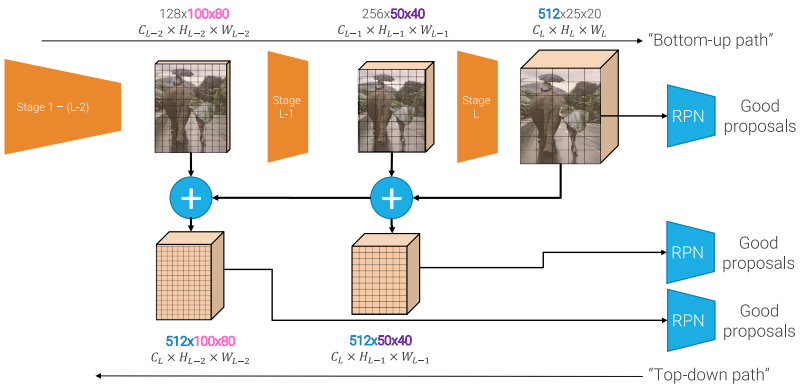


Figure 4.14: General FPN flow

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.

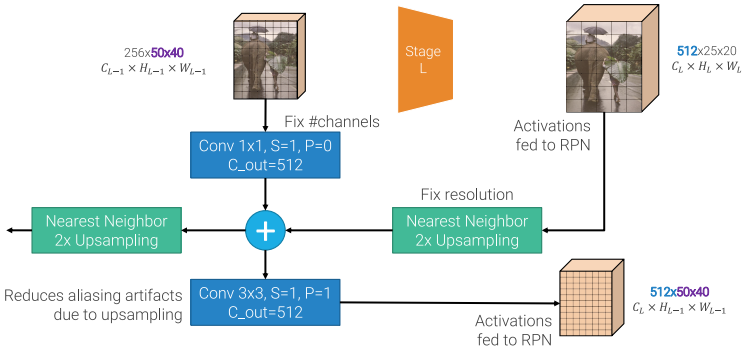


Figure 4.15: FPN top-down flow

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

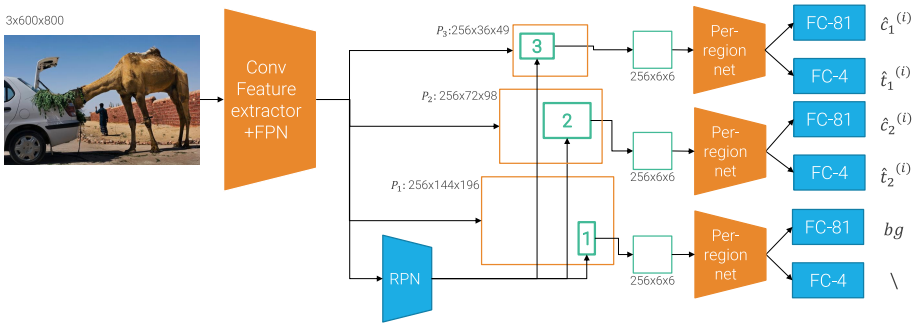


Figure 4.16: Example of faster R-CNN with FPN

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.