Deep Convolutional Neural Networks

Enrique Romero

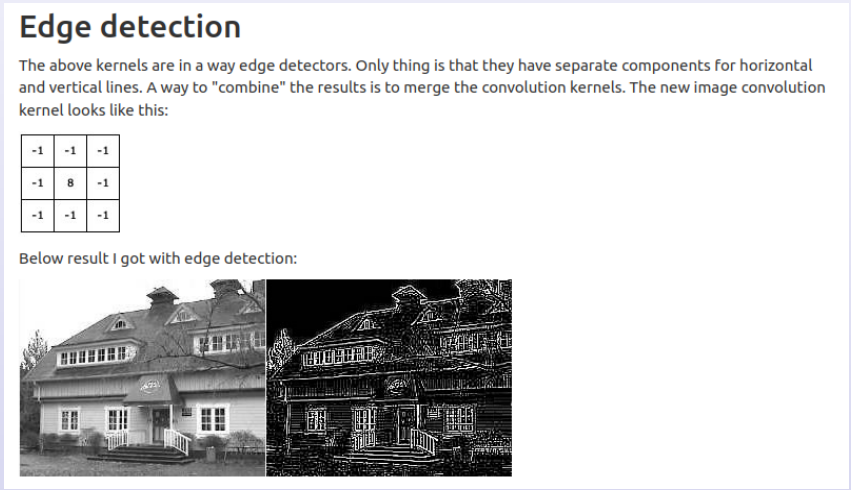
Deep Learning
Master in Artificial Intelligence

Soft Computing Group
Computer Science Department
Universitat Politècnica de Catalunya, Barcelona, Spain

Outline

Convolutions

An example of convolution:



Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to "combine" the results is to merge the convolution kernels. The new image convolution kernel looks like this:

-1	-1	-1
-1	8	-1
-1	-1	-1

Below result I got with edge detection:

More examples at

`https://www.aishack.in/tutorials/image-convolution-examples`

Convolutions

In CNNs we use the **discrete cross-correlation**

For example in 2D CNNs we will define:

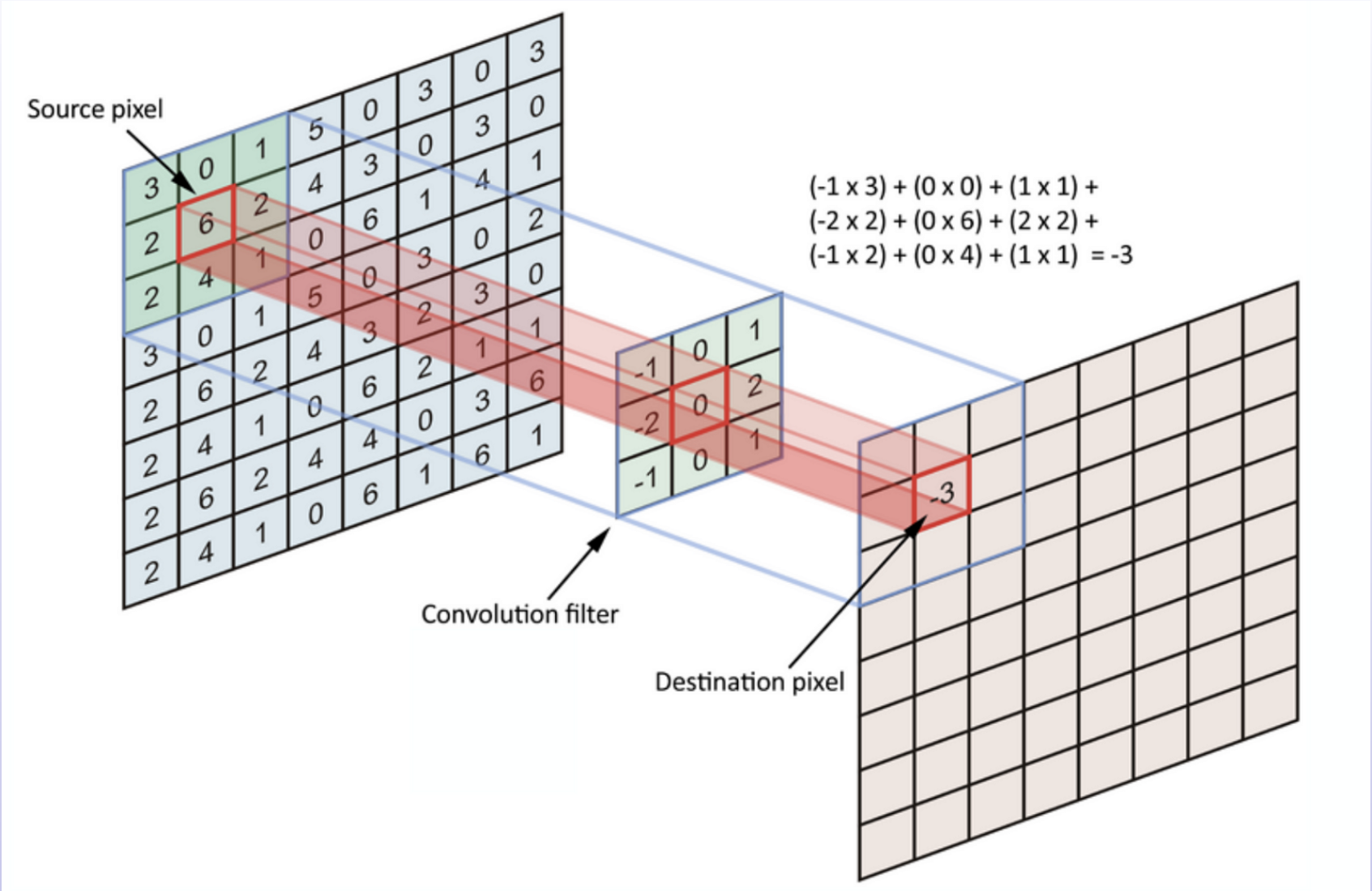
$$C(n,m) = (K * I)(n,m) = \sum_i \sum_j K(i,j) I(n+i, m+j)$$

where

- I is the **input** (an image, for example)
- K is the **kernel**, **feature detector** or **filter** (the weights)
- The sum is defined over all valid values
- C is the **feature map** (the output)

Since the inner product is a measure of similarity, C **will larger in the positions where** K **and** I **are similar**

Convolutions



Convolutions

The aim of the convolution operation is that of obtaining **translation invariance** (be able to find a pattern in any location of the input) and **equivariance** (same response in any location where the pattern is present)

It implicitly leads to (Multi-Layer Perceptrons) MLPs with:

- **Sparse interactions** (sparse connectivity)
- **Weight sharing**

Typically, the convolution operation is combined with other operations, such as **non-linear transformations** and **pooling**

Convolutions

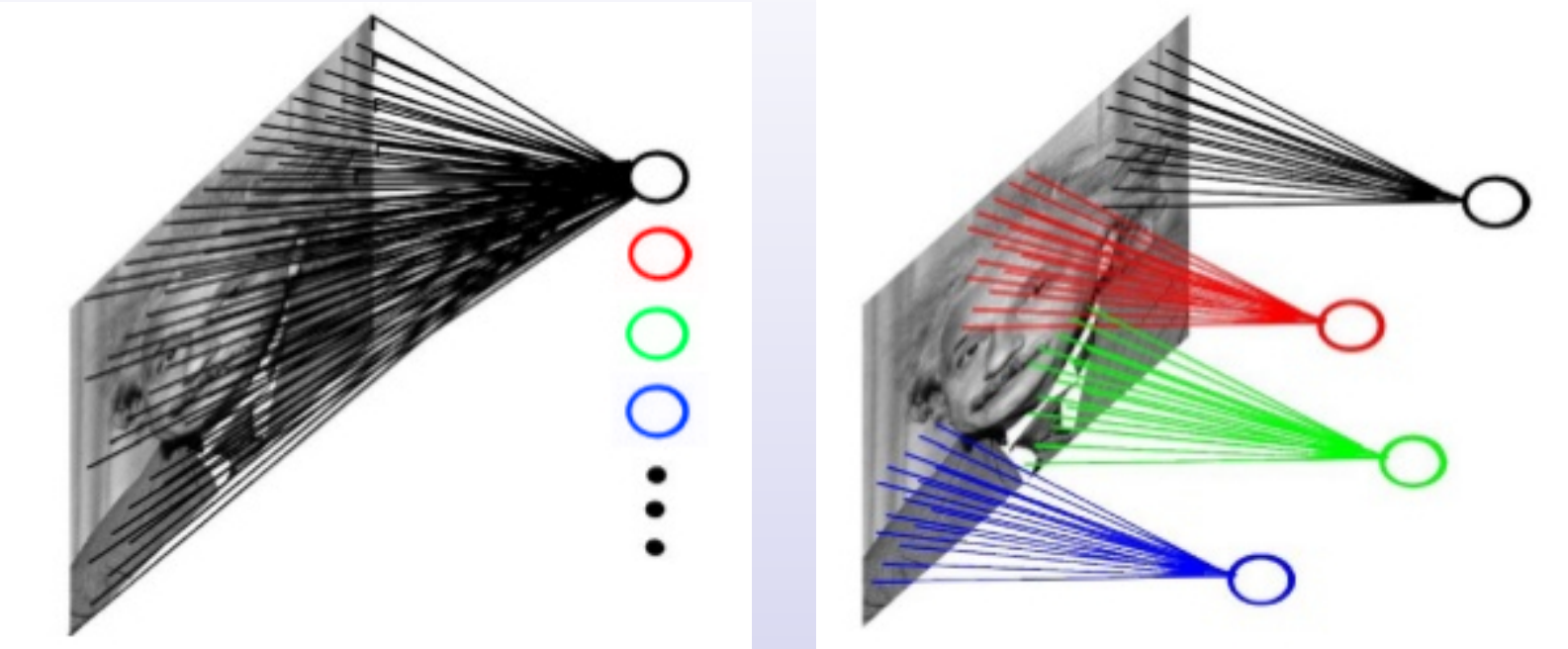


Figure 1: Representation of classical MLP (left) and CNN (right)

With convolutions, the **same weights** are used to compute output values in **different and small parts** of the input: equivalent to replicate the **same hidden unit** at multiple locations across the image

Working with Channels

In real data sets, there are additional dimensions where we do not want to apply the convolution operation:

- 1D: Multi-dimensional time series (for example, animations of "skeletons", where the data is described by the angles of each joint in the skeleton)
- 2D: Color images (each pixel is a rgb vector)
- 3D: Color volumetric data (for example, medical rgb CT scans)

These additional dimensions are usually called **channels** (also referred to as the **depth** of the input)

How does the convolution operate with channels?

Working with Channels

For example, in 2D color images (3 channels), we have:

- The input is a $N \times M \times 3$ tensor
- Every filter K_f is a $A \times B \times 3$ tensor
- The output of the convolution of the input image I and the filter K_f is a $N \times M \times 1$ tensor (a matrix), where each component is (we sum up along the channels):

$$C_{K_f}(n,m) = \sum_{i=1}^{A} \sum_{j=1}^{B} \sum_{k=1}^{3} K_f(i,j,k) I(n+i, m+j, k)$$

- Every filter "outputs an image of just one channel": If we have F filters, the output of the convolutional layer is a $N \times M \times F$ tensor, where the third component has the convolutions of the image with every filter:

$$C(n,m,f) = C_{K_f}(n,m)$$

Working with Channels

Therefore, **the outputs of a convolutional layer, in turn, are also reshaped in a structure with channels**, so that it can be used as the input of a new convolutional layer:

- The number of dimensions of the original image ($N \times M$) is constant (approximately, see below) between adjacent convolutional layers
- The number of output channels of a convolutional layer is the number of filters of that layer (F)

This **allows to construct CNNs with many layers in a natural way, similar to MLPs**

Working with Bias

We can optionally add some bias terms to the convolution

How do they operate?

Typically, in every filter we will have one bias per input channel, shared across all locations in the convolution:

$$C_{K_f}(n,m) = \sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{3}[K_f(i,j,k)I(n+i,m+j,k) + b_f(k)]$$

Parameters of Convolutions

The first parameter is, obviously, **the size of the filter** $A \times B$

With the previous definition of convolution, the size of the output of the convolution of a $N \times M$ image with an $A \times B$ filter is exactly $(N - A + 1) \times (M - B + 1)$, since only valid positions can be used to compute the convolution

We may want:

- On the one hand, to have the possibility to obtain outputs with the same size than the original one by adding extra values at the borders of the image (**Padding**)
- On the other hand, reduce the computational cost by removing (or not computing) some of the outputs of the convolution (**Stride**)

Parameters of Convolutions

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8

Figure 2: Zero-padding (left) and Reflection-padding (right)

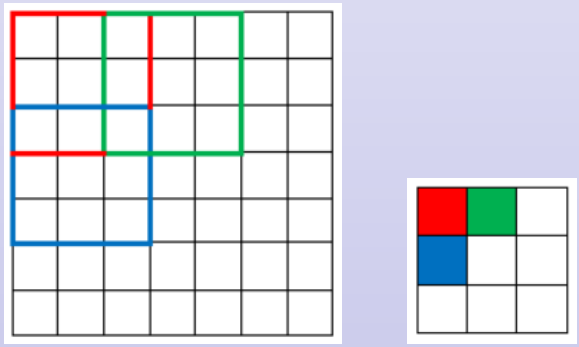


Figure 3: Convolution with Stride 2

Pooling layers

Pooling layers **replace the output of the network at a certain rectangle with a summary statistic of its contents**

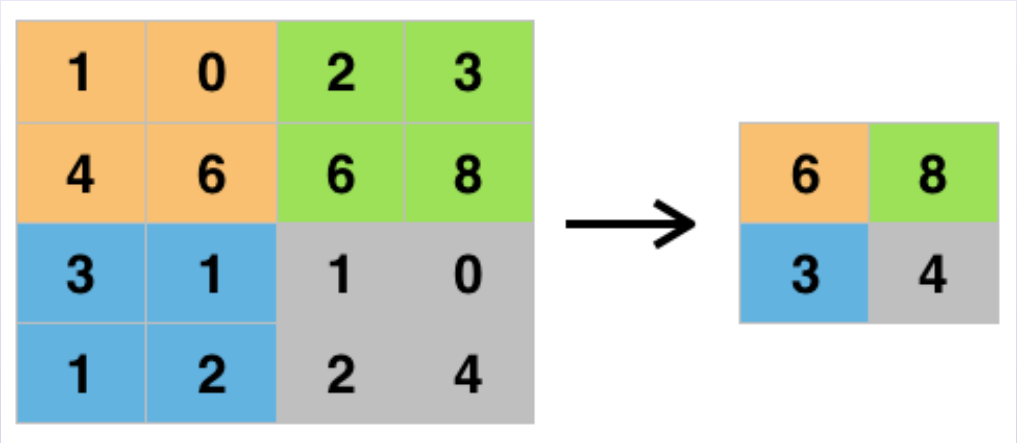


Figure 4: Max-Pooling of size 2×2 and stride 2

Additional Operations and Layers

Note that CNNs can handle **inputs of variable size**: applying each filter at different positions depending on the size of the input (similar to the stride parameter)

CNNs may have other types of layers:

- Non-linearities: apply a non-linear transformation to the output of the convolution, such as
  - logistic, hyperbolic tangent
  - ReLU and its variations: leaky ReLU, PReLU, ELU, Swish, Mish,... (see [Li et al., 2022])
- Fully connected layers: standard layers (typically on top of the convolutional layers)

Summary

In summary, we may have:

- Convolutional layers, which take as input a $N \times M \times C$ tensor and (assuming padding = *same* and stride = 1) convert it into a $N \times M \times F$ tensor
- Pooling layers, which take as input a $N \times M \times C$ tensor and (assuming size = $K \times K$ and stride = K) convert it into a $N/K \times M/K \times C$ tensor
- Non-linearities, that do not change the size of the input
- Fully connected layers, that work as in MLPs, converting arbitrarily vectors into vectors

Training CNNs

As for MLPs, training a CNN consists of **finding a set of weights** that optimizes the cost function (architecture set *a priori*)

For CNNs the most common settings are similar to MLPs:

- **Type of problem**: regression, classification, object detection, image segmentation,...
- **Model representation**: the selected architecture of the CNN
- **Cost function**: combinations of
  - Sum-of-squares error
  - Cross-entropy
  - Intersection over Union
  - ...
  - With or without regularization terms
  - ...
- **Optimization technique**: based on gradient descent with backpropagation

Outline

A Bit of History

Convolutional Neural Networks (CNNs) are very old models:

- Neocognitron [Fukushima, 1980] incorporated most of the elements in modern CNNs, but with a different approach
- Time-Delay Neural Networks
  [Lang and Hinton, 1988, Lang et al., 1990] are
  one-dimensional versions of CNNs applied to time series,
  trained with backpropagation
- Convolutional (Neural) Networks are described for the first time in [LeCun et al., 1989]

More recently, CNNs have attracted much attention because the winner models of several important contests were based on CNNs (the first one was the 2012 ImageNet object recognition challenge: `http://www.image-net.org/challenges/LSVRC/2012/results.html`)

A Bit of History

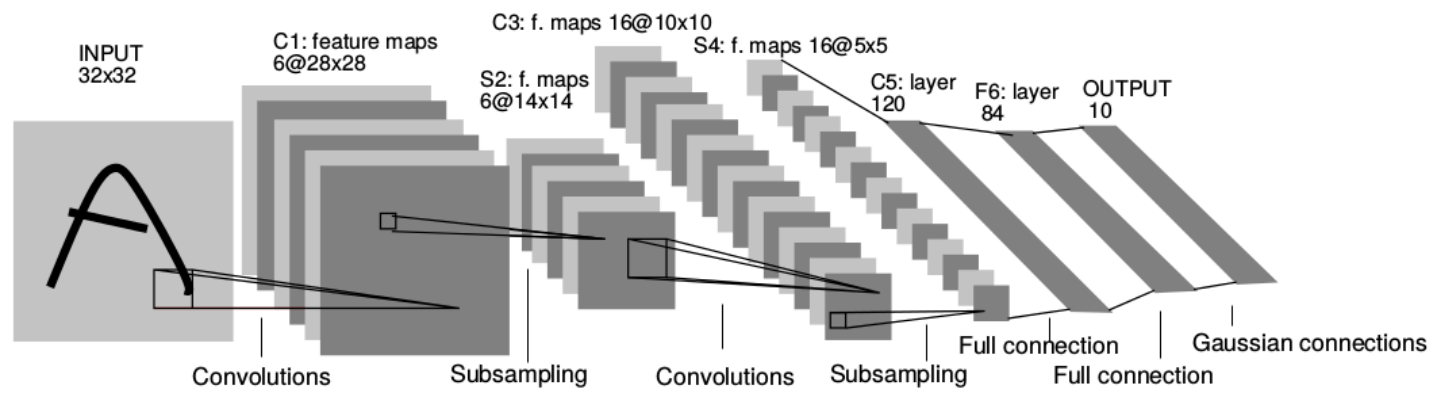


Figure 5: LeNet-5 architecture [LeCun et al., 1998]

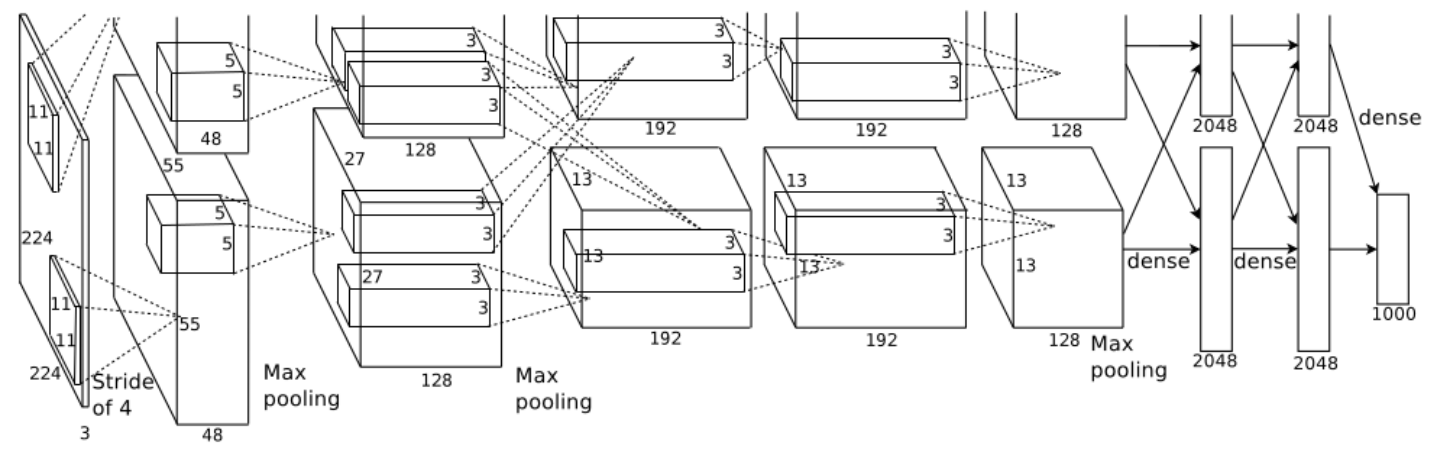


Figure 6: AlexNet architecture [Krizhevsky et al., 2012]

A Bit of History

Number of papers per year with the strings "Convolutional Neural Networks" or "Convolutional Networks" in the Title, Abstract or Keywords (`https://www.scopus.com/`)

1989-1995 (7y)	12
1996-2000 (5y)	17
2001-2005 (5y)	55
2006-2010 (5y)	174
2011-2015 (5y)	1,636
2016-2018 (3y)	24,137
2019-2021 (3y)	89,799
2022-2024 (3y)	150,653

Outline

Outline

Main Concepts of Deep CNNs and Deep Learning Schemes

Apart from **translation invariance and equivariance**, there are several important concepts to take into account when working with Deep CNNs:

- **In theory, deep neural networks are not necessary**: MLPs with two layers of weights are universal approximators
- In Computer Vision applications, we typically need to learn a **(complex) hierarchical structure** present in images, for example:
  - Several layers for low-level features: corners, balls, lines, etc
  - A second sequence of layers which combines these low-level features, to get eyes, noses, mouths, ears, fingers, etc
  - Higer-order layers for faces, hands, legs, bodies, etc
- **INTUITIVELY, a deep architecture may help the system to learn this hierarchy of features** (shallow networks may get stuck in very poor local minima)

- In most Deep Learning schemes (such as Deep CNNs), almost always it is possible to identify an **encoder-decoder** scheme
  - The **encoder** is the responsible for transforming the input into a (usually high-dimensional) vector which is assumed to be a context-sensitive representation
    - In Deep CNNs, the encoder is the set of convolutional layers (together with the non-linear transformations a pooling layers)
    - In general, it can be seen as a form of inductive bias, result of some prior knowledge of the domain [Goyal and Bengio, 2022]
    - The encoders can be reused in similar tasks: Transfer Learning, Fine-tuning, Feature Extraction, ...
  - The **decoder** reads the vector constructed by the encoder to obtain the prediction
    - In Deep CNNs, the decoder is usually a MLP
    - In general, they are less task-dependent than encoders

What Make CNN Work?

What make CNN work? The combination of several ingredients:

- Technological advances
  - High performance resources (GPUs, supercomputers)
  - Availability of large set of labeled examples
  - Software tools that offer efficient symbolic differentiation
- Scientific advances, aimed to
  - Speed up the training process
    - Non-saturated activation functions, such as Rectified Linear Units (ReLUs) $f(x) = max(0, x)$ (or any of its variants)
    - Adaptive gradient schemes (Adagrad, RMSProp, Adam, ...)
  - Improve generalization
    - Strong regularization techniques, such as Dropout, batch normalization, etc
  - **Design and adapt different architectures and so as to construct (learn) the encoder in many different ways $\rightarrow$ WE ARE GOING TO FOCUS ON THIS POINT**

Outline

Outline

LeNet-5 [LeCun et al., 1998]

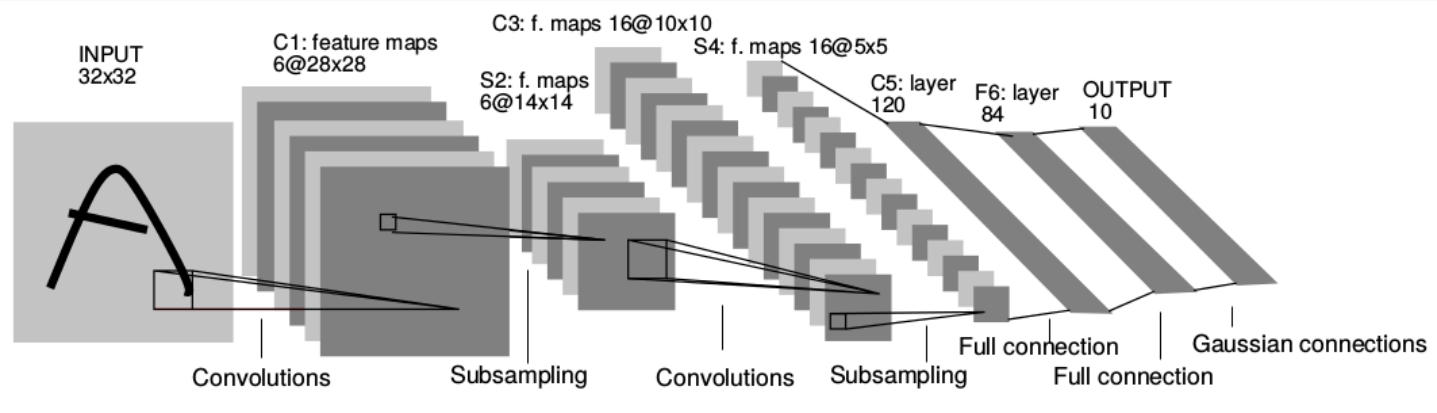


Figure 7: LeNet-5 architecture

Architecture (5 layers with weights):

- Two blocks of (C1, S2, C3, S4)
  - Convolution (sizesK: 5x5; stride: 1; numK: 6, 16)
  - Hyperbolic Tangent
  - Average-Pooling (size: 2; stride: 2)
- A convolutional layer (C5) of size 5x5 and 120 kernels
- A Fully Connected block: A MLP of one hidden layer with 84 hyperbolic tangent (F6) and 10 softmax units
- Total number of parameters (approx.): 60,000

Outline

AlexNet [Krizhevsky et al., 2012]

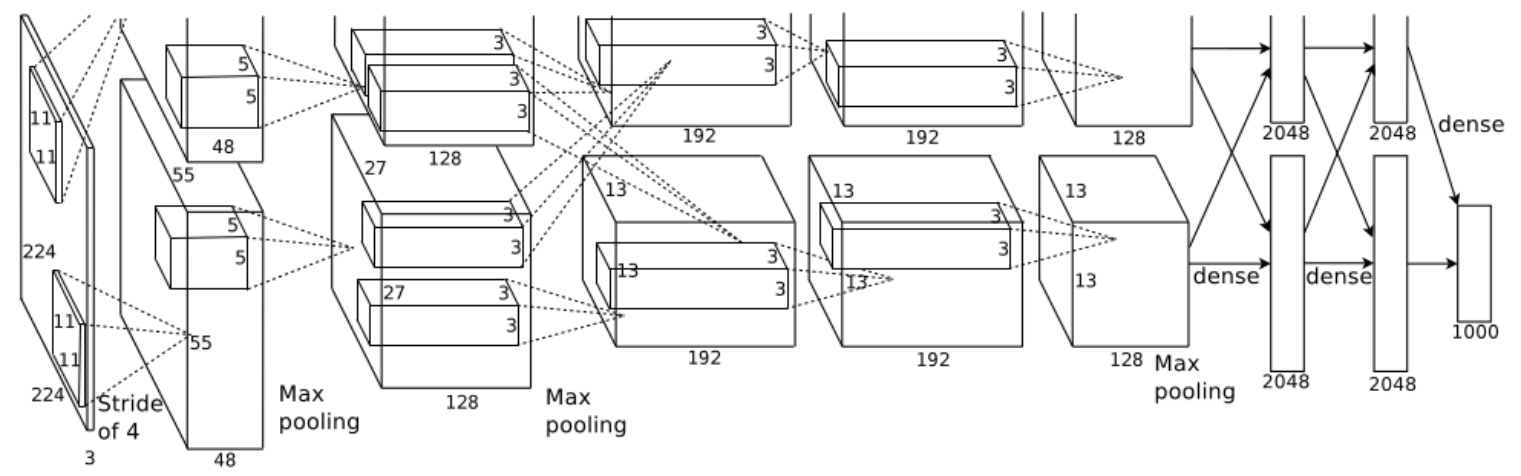


Figure 8: AlexNet architecture

AlexNet [Krizhevsky et al., 2012]

Architecture (8 layers with weights):

- Two blocks of
  - Convolution (sizesK: 11x11, 5x5; strides: 4,1; numK; 96, 256)
  - ReLU
  - Local Response Normalization (see original paper for details)
  - (overlapping) Max-Pooling (size: 3; stride: 2)
- Three blocks of
  - Convolution (sizesK: 3x3; stride: 1; numK: 384, 256, 256)
  - ReLU
  - (overlapping) Max-Pooling (size: 3; stride: 2)
- Several convolutional layers are connected to the feature maps on the same GPU (see figure and original paper for details)
- A Fully Connected block: A MLP of three layers of (4096, 4096, 1000) units with Dropout and ReLU (hidden layers) and softmax outputs
- Total number of parameters (torchvision): 57,044,810

AlexNet [Krizhevsky et al., 2012]

Dataset:

- ILSVRC ImageNet-1K (2010/2012) [Russakovsky et al. 2015]
- 1,261,406 (2010) / 1,229,413 (2011) / 1,281,167 (2012) training examples
- 668-3047 (2010) / 384-1300 (2011) / 732-1300 (2012) images per class
- 50,000 validation examples
- 100,000 test examples
- 1,000 classes
- Images with different resolutions:
  - For example, in the "fish" category resolution ranges from 75x56 to 4288x2848
  - Mean resolution: 469x387

AlexNet [Krizhevsky et al., 2012]

Training (1):

- Stochastic Gradient Descent (SGD) with Batch size: 128
- Weight decay: 0.0005
- Momentum: 0.9
- Same learning rate for all layers, adjusted manually throughout training: initialized at 0.01, and divided by 10 when the validation error rate stopped improving with the current value (it was reduced three times during training)
- Dropout (p=0.5) used in the first two fully-connected layers [“**Without Dropout, there is substantial overfitting**”]
- Weights initialization:
  - Biases in the second, fourth, fifth convolutional layers, and fully-connected hidden layers: 1.0 (rest of biases: 0.0) [“**This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs**”]
  - Rest of weights: Zero-mean Gaussian distribution with standard deviation 0.01

AlexNet [Krizhevsky et al., 2012]

Training (2):

- Network divided between two GPUs, and GPUs communicate only in certain layers (see figure)
- Number of epochs (early stopping): 90, which took five to six days on two NVIDIA GTX 580 3GB GPUs
- Images are rescaled such that the shortest side was of length 256, cropped out the central 256x256 patch, and normalized
- Data augmentation:
  1) Translation and horizontal reflections: extract random 224x224 patches (and their horizontal reflections) from the 256x256 images (discarding 23.44% of the pixels) to train the network [“**This increases the size of the training by a factor of 2048; Without it, there is substantial overfitting**”]
  2) Alter the intensities of the RGB channels, adding multiples of the principal components, with magnitudes proportional to the corresponding eigenvalues times a Gaussian random variable [“**This scheme reduces the top-1 error rate by over 1%**”]

AlexNet [Krizhevsky et al., 2012]

Test:

- At test time, the network makes a prediction by extracting five 224x224 patches (the four corner patches and the center patch) as well as their horizontal reflections (ten patches in all), and averaging the predictions
- Results for ILSVRC-2010: 37.5% (top-1) and 17.0% (top-5) test set error rates respectively (winner: 47.1% and 28.2%)
- Without averaging, results were 39.0% and 18.3%

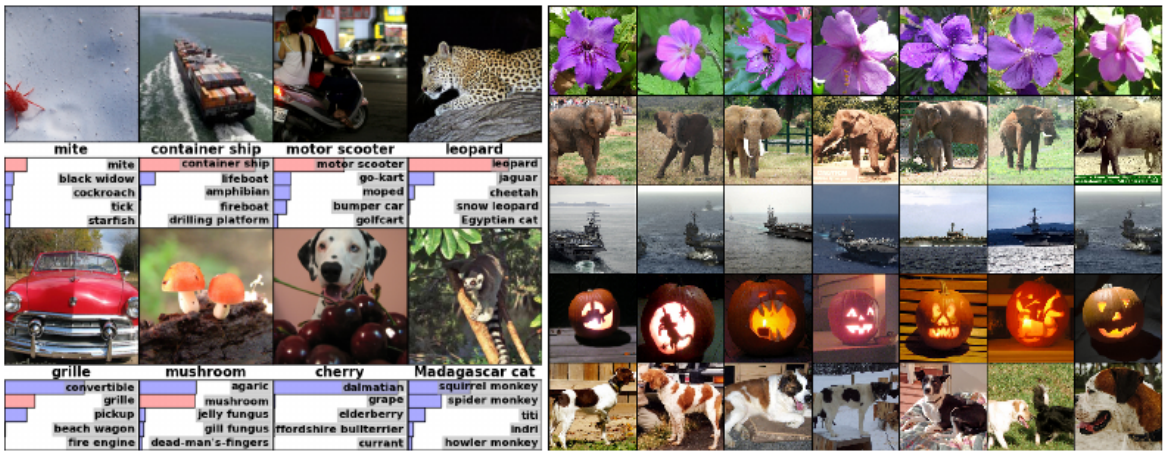


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

AlexNet - Summary

The main contributions of AlexNet was a successful combination of

- A deep CNN architecture with several innovations, such as the **use of ReLU non-linearities**
- Strong Regularization techniques:
  - Data Augmentation
  - **Dropout**
- The **use of GPUs for the training process**

An ensemble of five AlexNet models (Supervision) was the **winner of the ILSVRC-2012 competition**:

`http://www.image-net.org/challenges/LSVRC/2012/results.html`

(the top-5 error was 16.4%, the second-best entry obtained 26.2%)

AlexNet was not just the winner of a competition and a significant model in image classification: **It revolutionized Machine Learning** in general, leading to a shift from traditional techniques towards Neural Networks in general and Deep Learning in particular

AlexNet vs [Cireșan et al., 2011]

Previously, in [Cireșan et al., 2011], very similar architectures to AlexNet had been trained using

- Number of layers varying between 2 and 8
- Data Augmentation (as in [Cireșan et al., 2010] for MLPs)
- GPUs

These networks **obtained state-of-the-art results in several benchmark datasets, as NORB, CIFAR-10 and MNIST**

The use of GPUs in the context of CNNs is even older:
[Chellapilla et al., 2006, Jarrett et al., 2009,
Uetz and Behnke, 2009, Strigl et al., 2010]

Outline

ZfNet [Zeiler and Fergus, 2014]

In [Zeiler and Fergus, 2014], a deep analysis of AlexNet was performed, together with several innovations:

- First, they propose DeconvNet, a visualization/interpretation technique of the activities in the intermediate layers
- Second, after reproducing the results, an analysis of AlexNet was performed, from which several modifications were proposed to improve the model
- Third, feature extraction and transfer learning was successfully applied to other datasets

ZfNet [Zeiler and Fergus, 2014]

Training:

- Architecture similar to that used in AlexNet [Krizhevsky et al., 2012]:
  - The network is not divided between two GPUs
  - Rest of changes motivated by the analysis (see below)
- Stochastic Gradient Descent with Batch size: 128
- Momentum: 0.9
- Weights initialization: Biases to 0.0, rest of weights to 0.01
- Equal learning rate for all layers, adjusted manually throughout training: initialized at 0.01, and reduced when the validation error plateaus
- Dropout (p=0.5) used in the first two fully-connected layers
- Dataset: ImageNet-1K (2012)
- Images are rescaled such that the shortest side was of length 256, cropped out the central 256x256 patch, and normalized
- Data augmentation: 10 different sub-crops of size 224x224: corners and center with/out horizontal flips (ten crops in all)

ZfNet [Zeiler and Fergus, 2014]

The main features of DeconvNet are:

- Maps the activities back to the input pixel space
- Shows input patterns that (approximately) cause the activities
- Is based on Deconvolutional (not Neural) Networks [Zeiler et al., 2011, Zeiler et al., 2010]

(In the next slides) **The projections showed a hierarchy in the features learned by the networks, from general features (first layers) to class-specific ones (last layers)**

ZfNet [Zeiler and Fergus, 2014]

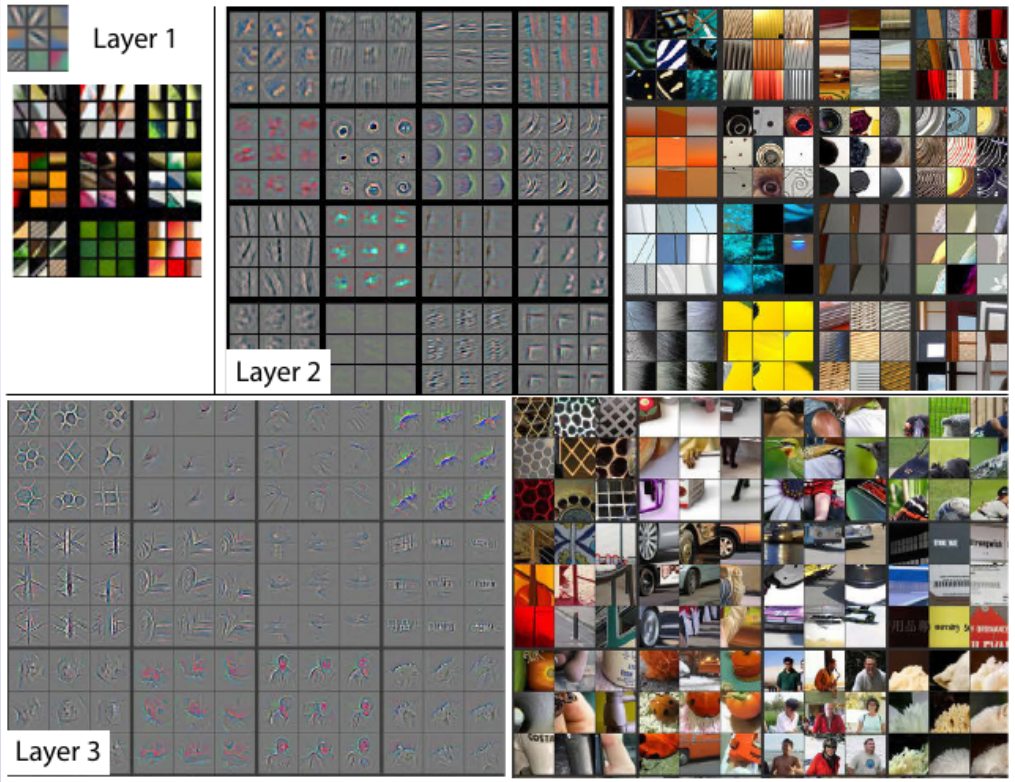


Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1).

ZfNet [Zeiler and Fergus, 2014]

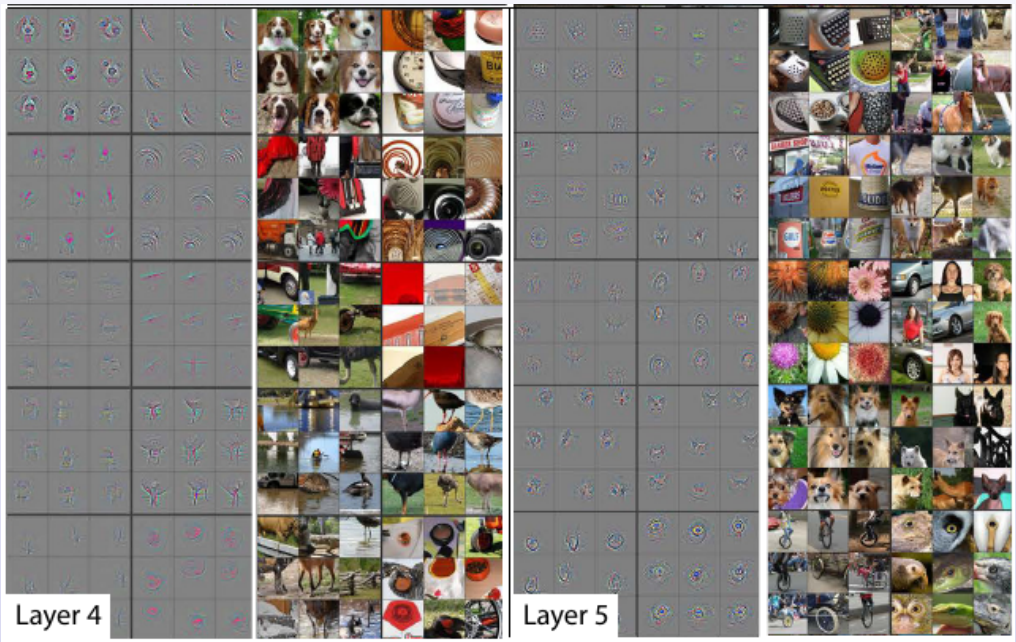


Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1).

ZfNet [Zeiler and Fergus, 2014]

Regarding the analysis of AlexNet, they found that

- The first layer filters were a mix of filters with different granularity
- The stride value used in the first layer (4) was found too large, causing aliasing artifacts

Therefore,

- The sizes of the filters were reduced from 11x11 to 7x7
- The stride value was set to 2

Only with these modifications, test top-5 accuracy improved by 1.7% in the ImageNet dataset from ILSVRC-2012

Note that the experimental setting was slightly different

ZfNet [Zeiler and Fergus, 2014]

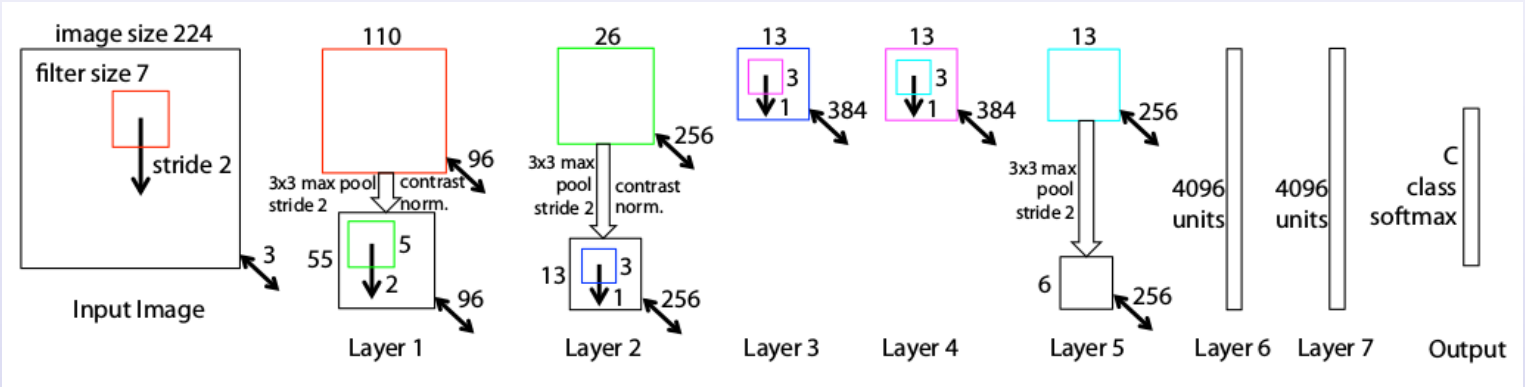


Figure 9: ZfNet architecture

ZfNet [Zeiler and Fergus, 2014]

In addition, a number of experiments were performed by varying the size of the layers or removing them entirely in the AlexNet architecture. The main conclusions were:

- Removing the first two fully connected layers or the middle convolutional layers only gives a slight increase in error, but removing both middle convolutional layers and the first two fully connected layers is dramatically worse
  This would suggest that the overall depth is crucial
- Changing the size of the fully connected layers does not substantially change the performance, but increasing the size of the middle convolutional layers improves the top-1 and top-5 accuracy by 2.5% and 2.1% respectively
  This would suggest that the width of the convolutional layers is also crucial

ZfNet [Zeiler and Fergus, 2014]

Table 2. ImageNet 2012 classification error rates with various architectural changes to the model of Krizhevsky *et al.* [18] and our model (see Fig. 3)

Error %	Train Top-1	Val Top-1	Val Top-5
Our replication of Krizhevsky *et al.* [18], 1 convnet	35.1	40.5	18.1
Removed layers 3,4	41.8	45.4	22.1
Removed layer 7	27.4	40.0	18.4
Removed layers 6,7	27.4	44.8	22.4
Removed layer 3,4,6,7	71.1	71.3	50.1
Adjust layers 6,7: 2048 units	40.3	41.7	18.8
Adjust layers 6,7: 8192 units	26.8	40.0	18.1
Our Model (as per Fig. 3)	33.1	38.4	16.5
Adjust layers 6,7: 2048 units	38.2	40.2	17.6
Adjust layers 6,7: 8192 units	22.0	38.8	17.0
Adjust layers 3,4,5: 512,1024,512 maps	18.8	**37.5**	**16.0**
Adjust layers 6,7: 8192 units and Layers 3,4,5: 512,1024,512 maps	**10.0**	38.3	16.9

Figure 10: ZfNet results

ZfNet [Zeiler and Fergus, 2014]

Finally, they used feature extraction and transfer learning to other problems, obtaining very good results

All the layers except the output one (the softmax fully connected layer) were frozen, and a new classifier was trained on top

When training from scratch, results were significantly worse

By the way, the first paper on transfer learning in neural networks dates back to 1976 [Bozinovski and Fulgosi, 1976] (see also [Bozinovski, 2020])

ZfNet - Summary

The main contributions of ZfNet were:

- Point out that **features learned by the CNNs follow a certain hierarchy, from simple (general) to complex (class-specific) ones**
- Suggest that one can
  - **Reduce the size of the filters**
  - **Increase the depth of the network**
  - **Increasing the width of the network**

  and still improve performance
- Apply **feature extraction and transfer learning** successfully

ZfNet (Clarifai) **won the Classification task at ILSVRC-2013**:

`https://image-net.org/challenges/LSVRC/2013/results.php`

Outline

VGG [Simonyan and Zisserman 2014]

Motivated by the results in [Zeiler and Fergus, 2014] respect to the reduction of the size of the filters and the importance of the overall depth, [Simonyan and Zisserman 2014] performed a systematic analysis of CNNs with **small 3x3 filters and increasing depth**

To that end, they constructed several architectures named VGG (from Visual Geometry Group, Department of Engineering Science, University of Oxford) with different number of layers:

- VGG11 (two versions)
- VGG13
- VGG16 (two versions)
- VGG19

Github: `https://github.com/deep-diver/VGG`

VGG [Simonyan and Zisserman 2014]

Justification for using small 3x3 filters:

- It is very easy to see that a stack of two/three 3x3 convolutional layers has an **effective receptive field** of 5x5/7x7 (they are not equivalent, but similar in some sense!)
- In a stack of three 3x3 convolutional layers there are, in addition, three non-linear transformations, making the whole function more flexible
- The number of parameters is lower: The number of parameters of a stack of three 3x3 convolutional layers and a 7x7 convolution layer are $27C^2$ $(= 3(3^2C^2))$ and $49C^2$ $(= 7^2C^2)$ respectively

VGG [Simonyan and Zisserman 2014]

The architectures have a modular structure, and most of the parameters are fixed, with "reasonable" (according to the state-of-the-art) values:

- Kernels size: 3x3; stride: 1, padding: *same*
- A ReLU layer is added after every convolutional layer
- Convolutions structured by blocks:
  - The convolutions of every block have the same number of filters, with increasing values: 64, 128, 256, 512
  - Every block may have 2, 3 or 4 convolutional layers, also with increasing values
  - After every block, a Max-Pooling layer (size: 2; stride: 2) is added
- After all convolutional blocks, a Fully Connected block is inserted: A MLP of three layers of weights (sizes: 4096, 4096, #Classes) with Dropout and ReLU in the hidden layers and softmax output units

VGG [Simonyan and Zisserman 2014]

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

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 11: VGG architectures

VGG [Simonyan and Zisserman 2014]

Several comments:

- **LRN** stands for "Local Response Normalization", as in AlexNet, but the authors argue that does not help to improve the results
- **conv1** layers are "1x1 convolutions" followed by a non-linearity, which performs worse than the same architecture with **conv3** ("3x3 convolutions") in their experiments

  (we will go back to "1x1 convolutions" in the GoogleNet subsection, where they are more relevant)

VGG16 [Simonyan and Zisserman 2014]

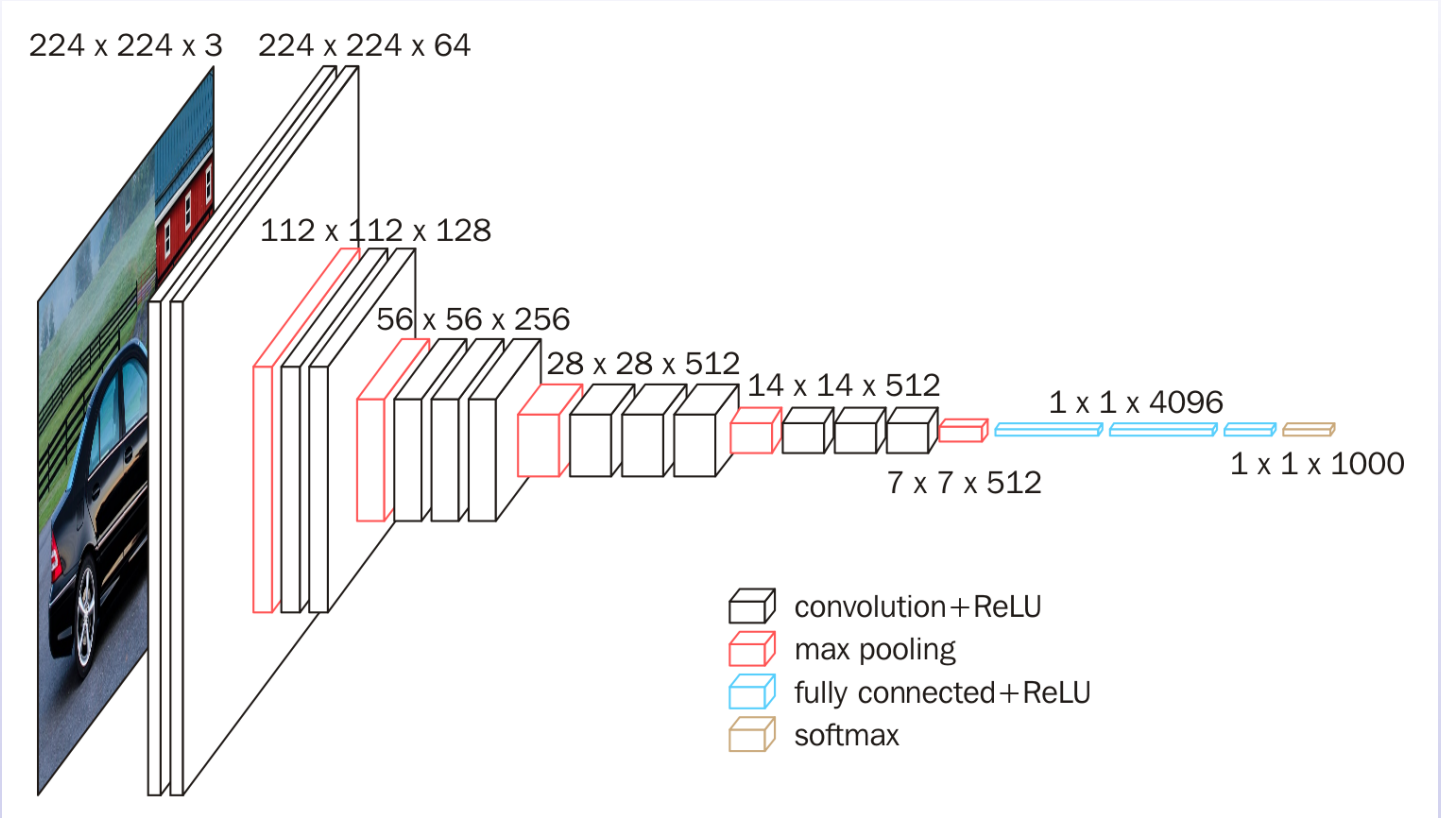


Figure 12: VGG16 architecture

VGG16 [Simonyan and Zisserman 2014]

Architecture of VGG16:

- Two blocks of
  - Convolution (sizeK: 3x3; stride; 1; numK: 64) + ReLU
  - Convolution (sizeK: 3x3; stride; 1; numK: 128) + ReLU
  - Max-Pooling (size: 2; stride: 2)
- Three blocks of
  - Convolution (sizeK: 3x3; stride: 1; numK: 256) + ReLU
  - Convolution (sizeK: 3x3; stride: 1; numK: 512) + ReLU
  - Convolution (sizeK: 3x3; stride: 1; numK: 512) + ReLU
  - Max-Pooling (size: 2; stride: 2)
- A Fully Connected block: A MLP of three layers of weights with Dropout and ReLU in the hidden layers and softmax output units
- Total number of parameters: 138,357,544

VGG [Simonyan and Zisserman 2014]

Training, similar to AlexNet (1):

- Stochastic Gradient Descent with Batch size: 256
- Weight decay: 0.0005
- Momentum: 0.9
- Same learning rate scheme than for AlexNet (initialized at 0.01, adjusted manually throughout training, ...)
- Dropout (p=0.5) used in the first two fully-connected layers
- Weights initialization:
  - First, configuration A was trained with random initialization
  - For deeper architectures, the first four convolutional layers and the fully connected layers were initialized with the layers of A, while the intermediate layers were initialized randomly
    It can be done because of their modular structure
  - Random weights were sampled from a Gaussian distribution with zero mean and variance 0.01; Biases were initialized to 0.0

VGG [Simonyan and Zisserman 2014]

Training (2):

- Number of iterations/epochs (early stopping): 370K/74
- Dataset: ImageNet-1K (2012)
- Images are randomly cropped (size 224x224) from rescaled training images (one crop per image and SGD iteration)
- Different types of rescaling:
  - Fixed S: the image is rescaled such that the shortest side has length S, for $S = 256, 384$ (for AlexNet and ZFNet, S=256)
  - Variable: S is randomly selected in $[256, 512]$
- Data augmentation similar to AlexNet (after rescaling): random horizontal flipping and random RGB colour shift

VGG [Simonyan and Zisserman 2014]

The rescaling procedure is an important point, since:

- Images contain useful **predictive features at different scales**
- Rescaling followed by cropping at a fixed size can be seen as a form of **zoom in / zoom out** for a patch of the image
- It can be done in the training phase, in the test phase or in both phases, as in this model
- In the test phase, it can also be done in several ways:
  - Using multiple crops for the images at different scales, as in [Howard 2013, Szegedy et al., 2014]
  - In a more sophisticated way, as in [Sermanet et al. 2014]

The model presented in [Howard 2013] **was in the top-5 of the Classification task at ILSVRC-2013**, and [Sermanet et al. 2014] (OverFeat) **won the Localization task at ILSVRC-2013**:

`https://image-net.org/challenges/LSVRC/2013/results.php`

VGG [Simonyan and Zisserman 2014]

Several test procedures are presented:

- The main one, based on [Sermanet et al. 2014]:

> At test time, given a trained ConvNet and an input image, it is classified in the following way. First, it is isotropically rescaled to a pre-defined smallest image side, denoted as Q (we also refer to it as the test scale). We note that Q is not necessarily equal to the training scale S (as we will show in Sect. 4, using several values of Q for each S leads to improved performance). Then, the network is applied densely over the rescaled test image in a way similar to (Sermanet et al., 2014). Namely, the fully-connected layers are first converted to convolutional layers (the first FC layer to a 7×7 conv. layer, the last two FC layers to 1×1 conv. layers). The resulting fully-convolutional net is then applied to the whole (uncropped) image. The result is a class score map with the number of channels equal to the number of classes, and a variable spatial resolution, dependent on the input image size. Finally, to obtain a fixed-size vector of class scores for the image, the class score map is spatially averaged (sum-pooled). We also augment the test set by horizontal flipping of the images; the soft-max class posteriors of the original and flipped images are averaged to obtain the final scores for the image.

- Multiple crops (only for comparison with [Szegedy et al., 2014], see below):

> is captured. While we believe that in practice the increased computation time of multiple crops does not justify the potential gains in accuracy, for reference we also evaluate our networks using 50 crops per scale (5×5 regular grid with 2 flips), for a total of 150 crops over 3 scales, which is comparable to 144 crops over 4 scales used by Szegedy et al. (2014).

Results (test with one scaling value Q):

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Figure 13: VGG results

VGG [Simonyan and Zisserman 2014]

Test results:

- Results with fixed scaling Q:
  - Errors decrease as long as depth increases
  - A-LRN performs worse than A: **LRN** AlexNet "Local Response Normalization" does not help in these configurations
  - C performs worse than D: **conv1** layers do not help to improve
  - **Variable rescaling at training time leads to significantly better results than training images at a fixed scale**
- Results with multiple scaling values are consistently better for all configurations, achieving 24.8% (top-1) and 7.5% (top-5) error for D and E configurations
- Multiple crops results are slightly better: 24.4% and 7.1%
- An ensemble of the best two multi-scale models for configurations D and E obtained a top-5 error of 7.0%

VGG - Summary

The main contribution of VGG was to **set a new trend in CNNs to work with**:

- **Very small size filters**
- **Increasing depth**, that can be designed in a **modular fashion**
- **Variable rescaling** of the images

VGG obtained the (overall) **second place in the ILSVRC-2014 competition**: `https://image-net.org/challenges/LSVRC/2014/results.php`

Its **main limitation** was the use of a **very large number of parameters** (138-144 millions), which make it **computationally expensive**

Outline

General Scheme of Standard CNNs Architectures

Architecture:

- Similar scheme of [Convolution + Non-linearity] + Pooling
- ReLU non-linearities

Training:

- Stochastic Gradient Descent
- Image rescaling
- Data augmentation
- Dropout in the fully-connected layers
- Use of heuristics (initialization, learning rate decaying scheme, etc), in general without strong theoretical justification

Test:

- Use of heuristics (average of predictions for different crops, in different ways,...), difficult to justify and sometimes replicate

Developed in an academic environment

Standard Architectures of CNNs

In a standard CNN architecture:

- A pooling layer is inserted after several convolutional and non-linearity layers (**encoder**)
- The output layers (**decoder**) are fully connected (MLP)

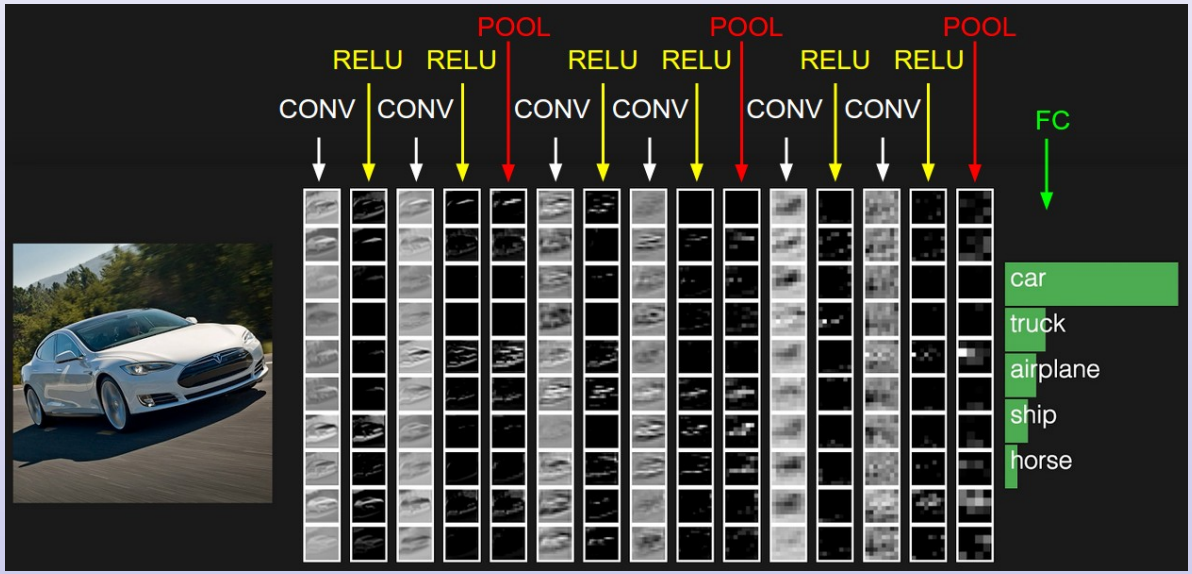


Figure 14: Typical architecture of a CNN (layers)

Standard Architectures of CNNs

What about the sizes? A common practice is to divide the size of the inputs by a factor and multiply the depth by another factor, forming a pyramid

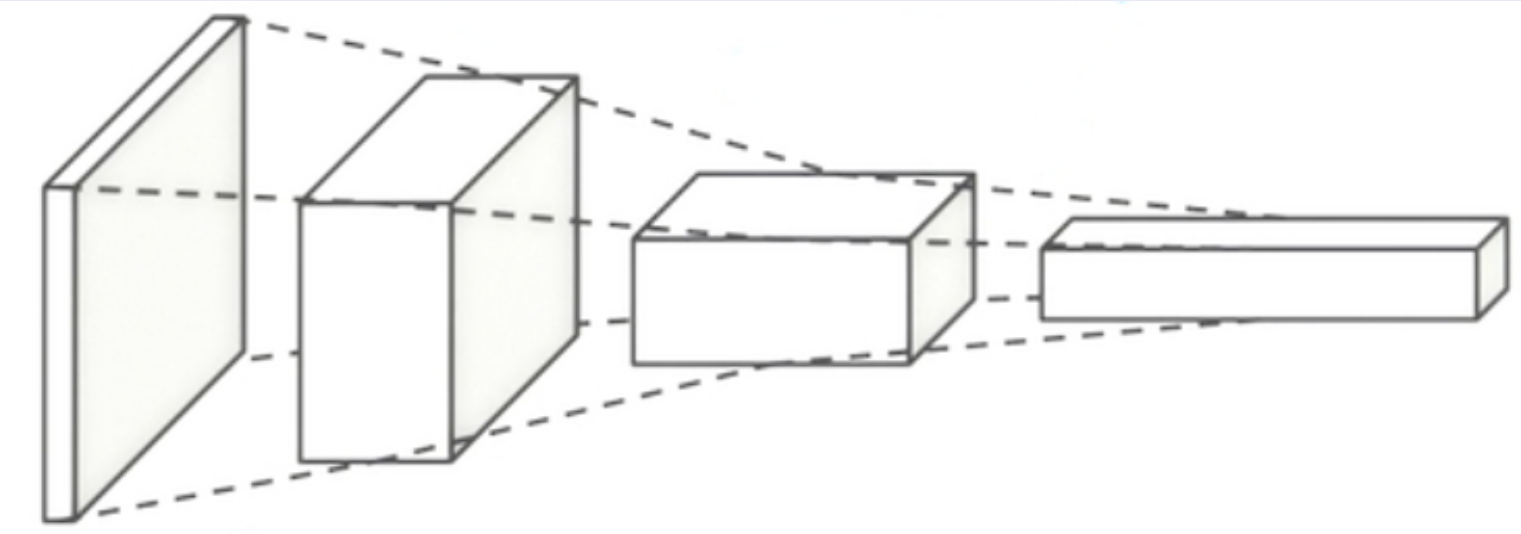


Figure 15: Typical architecture of a CNN (sizes)

How are the parameters distributed? In the lab session...

Standard Architectures of CNNs

Main findings (compared to LeNet-5):

- **Network depth** can be increased (AlexNet, ZfNet, VGG), and it seems that the features learned at each level have different complexity
- **Non-saturated activation functions** (ReLU) are necessary (AlexNet, ZfNet, VGG)
- **Regularization** (data augmentation, dropout) is crucial (AlexNEt, ZfNet, VGG)
- **Kernel size** can be reduced (ZFNet, VGG)
- **Network width** can be increased (ZfNet)
- **Transfer learning** is a promising idea (ZfNet)
- **Variable rescaling** of the images helps to improve (VGG), probably related to the fixed size of the kernel
- **GPUs** accelerates everything (AlexNet, ZfNet, VGG)

Outline

Beyond Standard Architectures

The architectures reviewed contain the basic elements for posterior developments

In this way, we can classify new architectures into seven categories by the variations made in the basic elements [Khan et al., 2020]:

- Spatial exploitation
- Depth exploitation
- Multi-path exploitation
- Width exploitation
- Feature-map exploitation
- Channel boosting
- Attention-based architectures

Since the variations may depend on different purposes, the same model may belong to more than one category, as we will see

Outline

Outline

Spatial Exploitation

CNNs are based on the convolution operation, and the most important parameter of the convolution operation is the size of the filter, which encapsulates the level of granularity at which we want to look at the data (small- and large-size filters for extracting fine- and coarse-grained information respectively)

Spatial exploitation refers to the research made in this direction: how can performance be improved by **varying the size of the filters**

Spatial Exploitation

We can consider that the standard architectures belong to this group, since the size of the filter (together with the depth of the network), was one of the first issues explored

Several models in this category:

- LeNet-5 [LeCun et al., 1998]
- The model in [Cireşan et al., 2011]
- AlexNet [Krizhevsky et al., 2012]
- ZfNet [Zeiler and Fergus, 2014]
- VGG [Simonyan and Zisserman 2014]
- GoogleNet (Inception-V1) [Szegedy et al., 2014]
- ...

Outline

GoogleNet (Inception-V1) [Szegedy et al., 2014]

The main objective of GoogleNet was to **achieve good accuracy with a reduced computational cost**

It was inspired by theoretical results in [Arora et al., 2014] and based on the "Network in Network" model [Lin et al., 2013]

Regarding [Arora et al., 2014]:

- [*"A layer-by layer construction where one should analyze the correlation statistics of the previous layer and cluster them into groups of units with high correlation"*] is suggested
- In the lower layers (close to the input) correlated units would concentrate in local regions, but as we move to higher layers, their spatial concentration is expected to decrease
- Therefore, **the size of the region where look for correlations may/should change along layers**

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Regarding "Network in Network" [Lin et al., 2013]

- The standard "AxB Convolution + Non-linearity" operation is replaced by a **MLPConvolution layer**:
  - Since a standard "AxB Convolution" is linear in nature, the "AxB Convolution + Non-linearity" operation would be similar to a Single-Layer Perceptron (**SLPConvolution layer**)
  - SLPs have limited capacities and can be replaced by MLPs, which are **stacks of SLPs** and are **universal approximators**
  - A simple way to replace the **SLPConvolution layer** while keeping the underlying ideas of both Convolutions and "stacks of SLPs" is by means of the **MLPConvolution layer**, which is "a stack of [1x1 Convolutions + Non-linearity]":
    A **1x1 convolution** can be seen as the application of the same SLP at every position: $C(n,m) = \sum_k [K_f(k) I(n,m,k) + b_f(k)]$
- Instead of using fully connected layers after the convolutional blocks, use **Global Average-Pooling**: compute the mean of each channel (the output is a vector of dimension C)

GoogleNet (Inception-V1) [Szegedy et al., 2014]

In this way, GoogleNet:

- Inspired by [Arora et al., 2014], introduces the **Inception block**, that incorporates **multi-scale convolutions** implemented as a sequence **split, transform** and **merge** operations
- Similar to [Lin et al., 2013], uses 1x1 convolutions, but with the aim of reducing the computational cost (see below): **1x1 filters are applied before larger kernels**
- Similar to [Lin et al., 2013], **replaces the fully connected layers by a Global Average-Pooling layer** (again for computational reasons)

GoogleNet (Inception-V1) [Szegedy et al., 2014]

The **multi-scale convolutions** include:

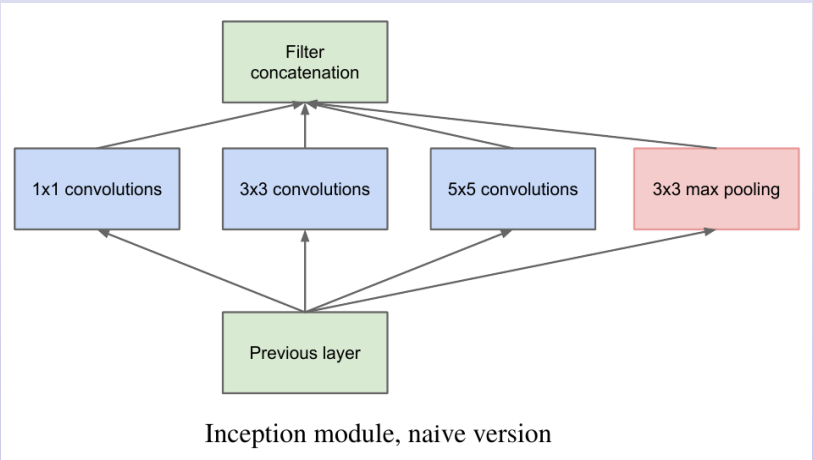
- 1x1 convolutions, as suggested in [Lin et al., 2013]
- 3x3 and 5x5 convolutions (see justification below)
- 3x3 Max-Pooling (see justification below)

The **split, transform** and **merge** operations can be implemented in an **Inception block (naive version)**:

Merge $\longrightarrow$

Transform $\longrightarrow$

Split $\longrightarrow$

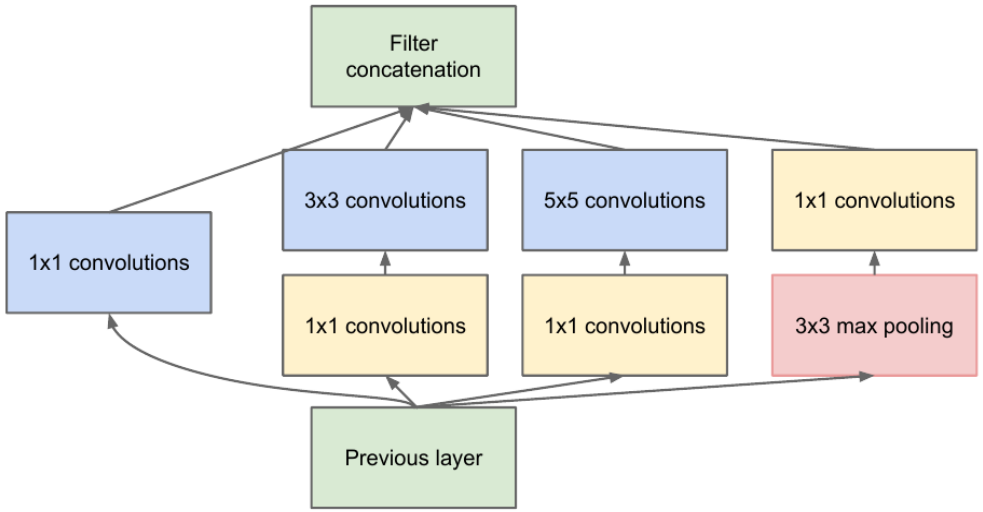


Inception module, naive version

GoogleNet (Inception-V1) [Szegedy et al., 2014]

The naive version of the Inception block is not efficient, since the number of parameters, the output size and the computational cost is larger than in a standard layer (it computes several convolutions in the same layer)

Since one of the goals was to have a reduced computational cost, a **dimensionality reduction (as a bottleneck)** is performed **using 1x1 convolutions**, leading to the final **Inception block**:



Inception module with dimensionality reduction

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Comparing a standard convolutional layer and a Inception block, suppose that the size of previous layer is $N \times M \times C$ (C is the number of channels), and we want to apply F filters of size $S \times S \times C$ (stride 1, padding = 'same'):

- In a standard convolutional layer:
  - The output size is $N \times M \times F$
  - The computational cost is $N \cdot M \cdot C \cdot S^2 \cdot F$
- In an Inception block with a dimensionality reduction with factor K (i.e., number of 1x1 convolutions = F/K):
  - The output size of the 1x1 convolutions is $N \times M \times F/K$
  - The computational cost of the 1x1 convolutions is $N \cdot M \cdot C \cdot F/K$
  - The output size of the $S \times S \times F/K$ convolutions is $N \times M \times F$
  - The computational cost of the $S \times S \times F/K$ convolutions is $N \cdot M \cdot F/K \cdot S^2 \cdot F$ (the number of channels is reduced to F/K)

Therefore, we can choose K so as to control the computational cost of the block: if $K \geq 1/S^2 + F/C$ the computational cost of the block "1x1 + SxS" convolution will be smaller

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Several comments on the Inception block:

- **Inside the Inception block there are also non-linearities (ReLU in this case)**
  [*"That is, 1x1 convolutions are used to compute reductions before the expensive 3x3 and 5x5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation making them dual-purpose... All the convolutions, including those inside the Inception modules, use rectified linear activation as well"*]
- **The Inception block can be seen as a way to find a local optimal configuration at a particular layer (i.e., degree of complexity) of the network**, that can be repeated at different levels (layers)

GoogleNet (Inception-V1) [Szegedy et al., 2014]

"Justifications" of the choices into the Inception block:

- Regarding the convolutions included in the Inception block:
  [*(sic) "In order to avoid patch-alignment issues, current incarnations of the Inception architecture are restricted to filter sizes 1x1, 3x3 and 5x5;* **this decision was based more on convenience rather than necessity**. *It also means that the suggested architecture is a combination of all those layers with their output filter banks concatenated into a single output vector forming the input of the next stage"*]
- Regarding the inclusion of the Max-Pooling operation in the Inception block:
  [*(sic) "... since pooling operations have been essential for the success of current convolutional networks,* **it suggests that adding an alternative parallel pooling path in each stage should have additional beneficial effect, too**"]

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Architecture:

- In GoogleNet, **conventional convolutional layers are replaced by the Inception block**, aimed to capture spatial information at different scales
- [*(sic) "For technical reasons (memory efficiency during training),* **it seemed beneficial to start using Inception modules only at higher layers** *while keeping the lower layers in traditional convolutional fashion;* **This is not strictly necessary, simply reflecting some infrastructural inefficiencies in our current implementation**"]
- Besides the 1x1 convolutions in the Inception block, the computational cost is additionally controlled by **replacing the fully connected layers by a Global Average-Pooling layer**, as in [Lin et al., 2013]
- **Auxiliary learners, connected to intermediate layers, are included** to speed up convergence

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Standard convolutions

Inception blocks

Auxiliary Classifiers

Average Pooling

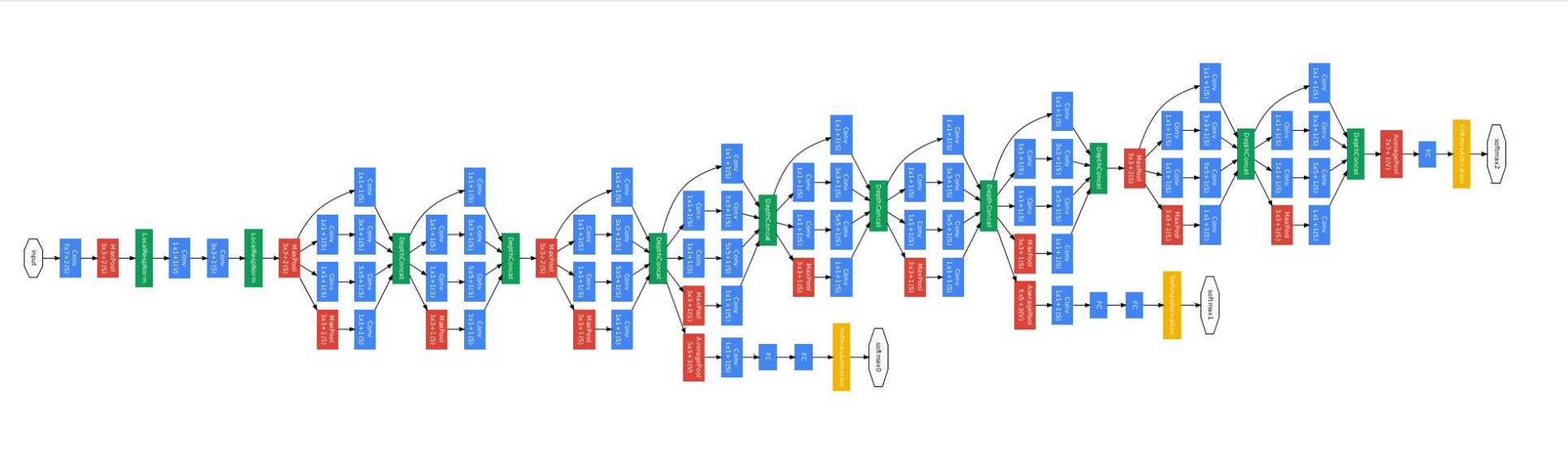


Figure 16: GoogleNet architecture

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Sizes:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Training procedure somewhat difficult to replicate:

GoogLeNet networks were trained using the DistBelief [4] distributed machine learning system using modest amount of model and data-parallelism. Although we used a CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage. Our training used asynchronous stochastic gradient descent with 0.9 momentum [17], fixed learning rate schedule (decreasing the learning rate by 4% every 8 epochs). Polyak averaging [13] was used to create the final model used at inference time.

Image sampling methods have changed substantially over the months leading to the competition, and already converged models were trained on with other options, sometimes in conjunction with changed hyperparameters, such as dropout and the learning rate. Therefore, it is hard to give a definitive guidance to the most effective single way to train these networks. To complicate matters further, some of the models were mainly trained on smaller relative crops, others on larger ones, inspired by [8]. Still, one prescription that was verified to work very well after the competition, includes sampling of various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area with aspect ratio constrained to the interval $[\frac{3}{4}, \frac{4}{3}]$. Also, we found that the photometric distortions of Andrew Howard [8] were useful to combat overfitting to the imaging conditions of training data.

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Other comments and justifications related to the training process:

- [*"however* **the use of dropout remained essential even after removing the fully connected layers**"]
- [*(sic) "By adding auxiliary classifiers connected to these intermediate layers,* **discrimination in the lower stages** *was expected; This was* **thought to combat the vanishing gradient problem while providing regularization**"]
  [*"These classifiers take the form of smaller convolutional networks put on top of the output of the Inception (4a) and (4d) modules"*]
  [*"***During training, their loss gets added to the total loss of the network with a discount weight (weighted by 0.3); At inference time, the auxiliary networks are discarded***"*]
  [*(sic) "Later control experiments have shown that the effect of the auxiliary networks is relatively minor (around 0.5%) and that it required only one of them to achieve the same effect"*]

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Test procedure:

used for training. In addition to the training techniques aforementioned in this paper, we adopted a set of techniques during testing to obtain a higher performance, which we describe next.

1. We independently trained 7 versions of the same GoogLeNet model (including one wider version), and performed ensemble prediction with them. These models were trained with the same initialization (even with the same initial weights, due to an oversight) and learning rate policies. They differed only in sampling methodologies and the randomized input image order.

2. During testing, we adopted a more aggressive cropping approach than that of Krizhevsky et al. [9]. Specifically, we resized the image to 4 scales where the shorter dimension (height or width) is 256, 288, 320 and 352 respectively, take the left, center and right square of these resized images (in the case of portrait images, we take the top, center and bottom squares). For each square, we then take the 4 corners and the center 224×224 crop as well as the square resized to 224×224, and their mirrored versions. This leads to $4 \times 3 \times 6 \times 2 = 144$ crops per image. A similar approach was used by Andrew Howard [8] in the previous year's entry, which we empirically verified to perform slightly worse than the proposed scheme. We note that such aggressive cropping may not be necessary in real applications, as the benefit of more crops becomes marginal after a reasonable number of crops are present (as we will show later on).

3. The softmax probabilities are averaged over multiple crops and over all the individual classifiers to obtain the final prediction. In our experiments we analyzed alternative approaches on the validation data, such as max pooling over crops and averaging over classifiers, but they lead to inferior performance than the simple averaging.

GoogleNet (Inception-V1) [Szegedy et al., 2014]

Results: ILSVRC comparison (top) and test strategies (bottom)

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance.

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Table 3: GoogLeNet classification performance break down.

GoogleNet (Inception-V1) - Summary

GoogleNet obtained the (overall) **first place in the ILSVRC-2014 competition**: `https://image-net.org/challenges/LSVRC/2014/results.php`

The **main contribution** of GoogleNet was the **Inception block**, that allows multi-resolution search in an efficient way

Its **main limitations** are (see [Khan et al., 2020]):

- **The topology, that needs customization from module to module**, as the authors point out: [*"... are 3-10x faster than similarly performing networks with non-Inception architecture, however this requires careful manual design at this point"*]
- The use of 1x1 convolutions for dimensionality reduction implies a representation bottleneck **may lead to loss of useful information**
- Training and test procedures are **difficult to replicate**

Outline

The Inception Family

Subsequently, different versions of the Inception family have been described, since, as the authors themselves say:

- [Szegedy et al., 2016] [*(sic) "Still,* **the complexity of the Inception architecture makes it more difficult to make changes to the network**... *Also, [Szegedy et al., 2014]* **does not provide a clear description about the contributing factors** *that lead to the various design decisions of the GoogLeNet architecture. This makes it* **much harder to adapt it** *to new use-cases while maintaining its efficiency"*]
- [Szegedy et al., 2017] [*(sic) "... we have also studied whether Inception without residual connections can be made more efcient by making it deeper and wider. For that purpose, we evaluated a new version named* **Inception-V4 which has a more uniform simplified architecture** *and more inception modules than Inception-V3. Historically, Inception-V3* **had inherited a lot of the baggage** *of the earlier incarnations"*]

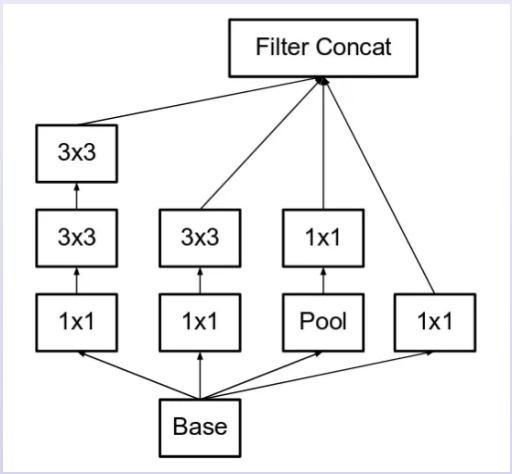
The Inception Family: Inception-V2 and -V3

The main idea in Inception-V2 and Inception-V3 is to **factorize the convolutions** in different ways [Szegedy et al., 2016]:

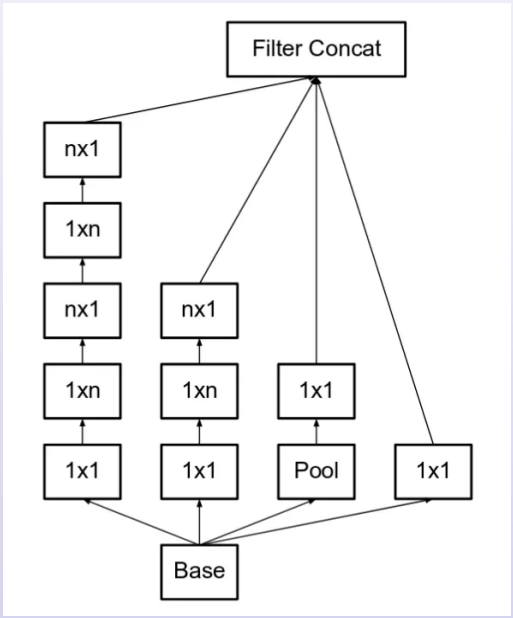
- (**Block A**) Factorize convolutions 5x5 by a stack of two convolutions 3x3 (reducing the computational cost by a factor of $25/(9+9) = 1.39$)
- (**Block B**) Instead of include a convolution NxN, factorize it and replace it by a stack of two convolutions 1xN and Nx1 (it is not equivalent, but similar in some sense!)
- (**Block C**) Replace (making wider) some of the 3x3 convolutions by a concatenation of two 1x3 and 3x1 convolutions

Additionally, Batch Normalization [Ioffe and Szegedy, 2015] was use after the convolutions

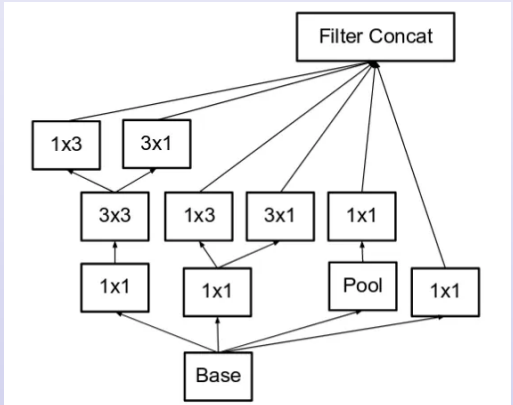
The Inception Family



Block A



Block B



Block C

The Inception Family: Inception-V4

The main idea in Inception-V4 is to **simplify and make more uniform** the architecture [Szegedy et al., 2017]:

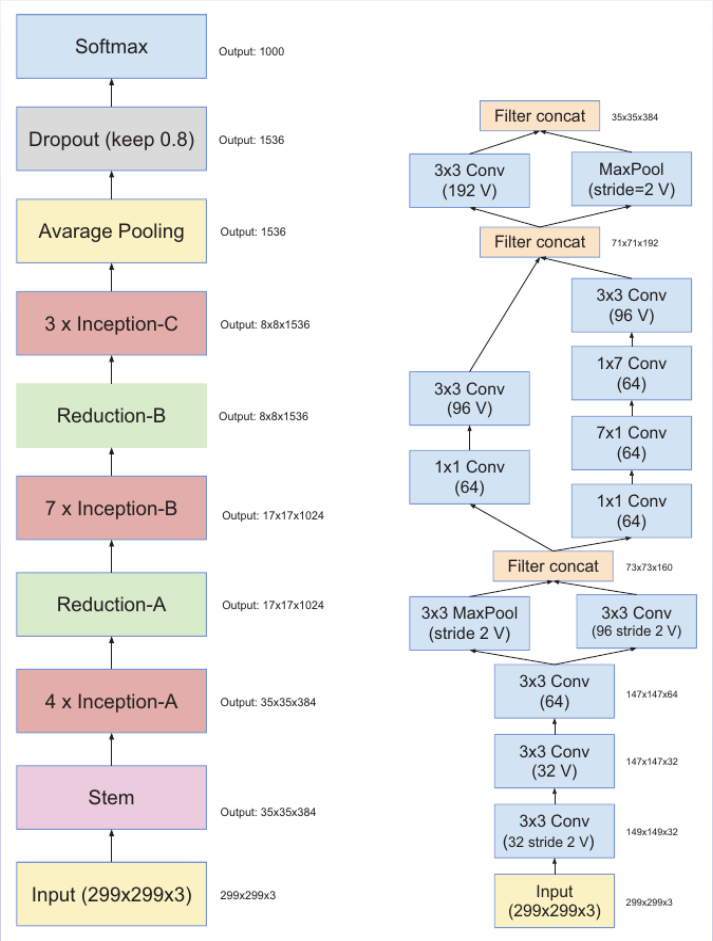
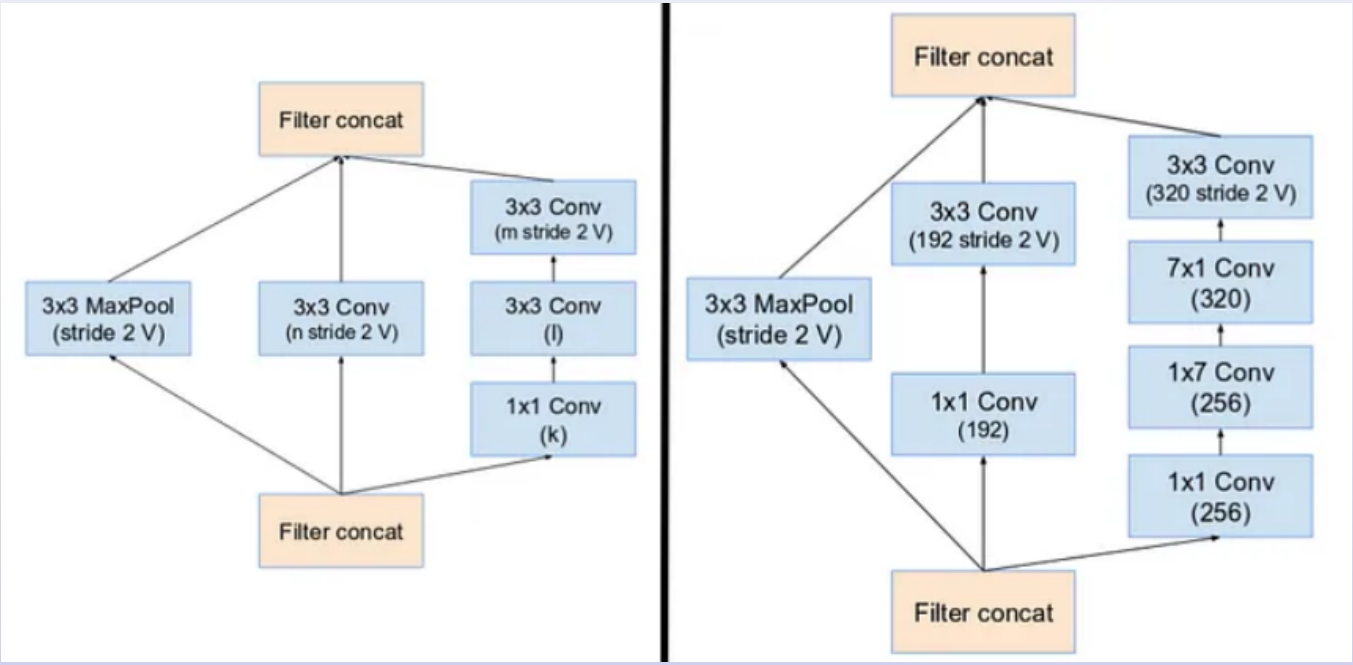


Figure 17: Architecture (left) and stem (right) of inception-V4

The Inception Family

The reduction blocks Reduction-A and -B are used to change the width and height of the grid:



The Inception Family: Inception-ResNet

The main idea in Inception-ResNet consists of the **addition of residual connections to the Inception architecture** [Szegedy et al., 2017]

We will explain residual connections below (see ResNet section)

The effect of including residual connections was that of **accelerating the convergence while maintaining the accuracy**

Outline

Outline

Depth Exploitation

Deep CNNs are based on the hypothesis that the depth of the network is related with the complexity of the learned features [Bengio, 2009]

Several theoretical results also support this idea, such as:

- In [Csáji, 2001] it was proved that a single hidden layer is sufficient to approximate any function, but with exponentially many neurons, making it computationally non-realistic
- In [Delalleau and Bengio, 2011] it is proved that there exist families of functions that can be represented with substantially fewer hidden units with a deep network than with a shallow one
- [Montufar et al. 2014] show that the number of regions that a neural network with L layers and n hidden units in each layer can construct grows much faster than a network with nL units in one hidden layer (see also [Bianchini and Scarselli, 2014])

Depth Exploitation

In practice, results in ILSVRC seem to confirm these claims

Therefore, **the depth of the network is probably the most important item to explore**: What happens when increasing the depth? Can we increase it without limit?

Several models in this category:

- VGG [Simonyan and Zisserman 2014] (it included a systematic analysis of networks with different depths)
- Highway Networks [Srivastava et al., 2015]
- Residual Networks [He et al. 2016]
- The Inception family (Inception-V2, -V3 -V4, -ResNet) [Szegedy et al., 2016, Szegedy et al., 2017]
- ...

Outline

ResNet [He et al. 2016]

The starting points of ResNet are:

- On the one hand, **the evidence that the depth of the network is of crucial importance** in practical applications
- On the other, **the fact that increasing depth makes more difficult to train the network**: it seems that training deeper networks is not as straightforward as simply adding layers

The reasons for the second point are not clear (going back 25 years?), and although a number of improvements such as:

- New initialization schemes ([Glorot and Bengio, 2010], ...)
- Training networks in multiple stages (VGG, see above)
- Auxiliary classifiers in internal layers (GoogleNet, see above)
- Batch Normalization [Ioffe and Szegedy, 2015]

only networks with **no more than tens of layers** can be trained (it is not clear the **vanishing gradient** can explain everything)

ResNet [He et al. 2016]

The main idea of ResNet is that of **Deep Residual Learning** framework: **"Instead of hoping that a group of stacked layers fit a desired underlying mapping, we explicitly let these layers fit a RESIDUAL mapping"**

Formally, suppose we have groups of stacked layers $F_1(\cdot)$ and $F_2(\cdot)$ and we want to approximate a desired underlying mapping $H(x)$:

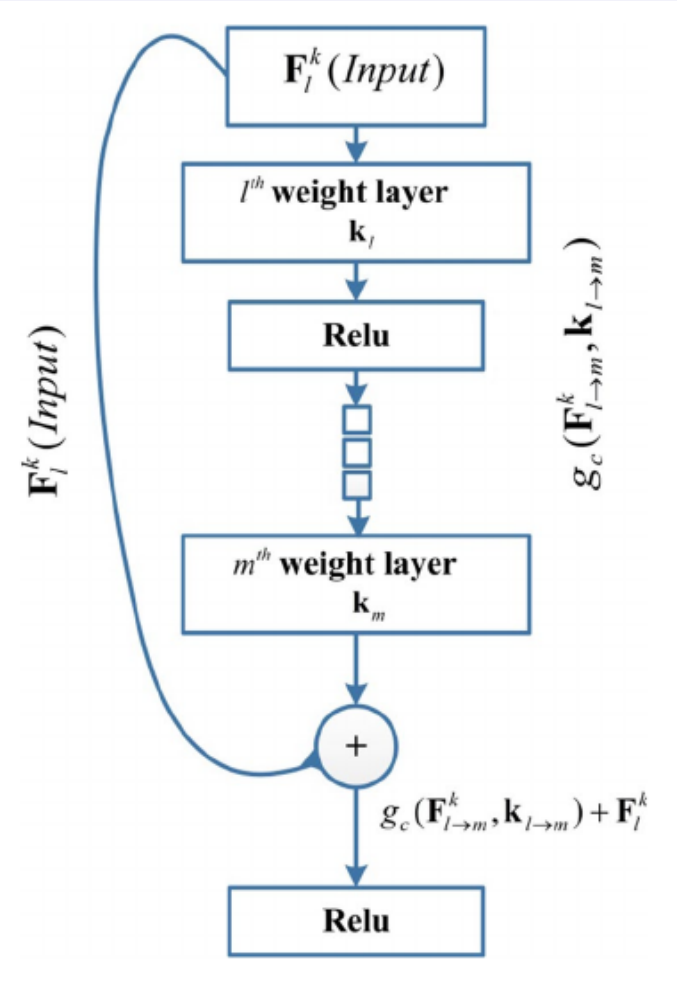
- For standard networks, we directly want $F_2(F_1(x)) \approx H(x)$
- For residual networks, we want $F_2(F_1(x))$ approximate the residue: $F_2(F_1(x)) \approx H(x) - F_1(x)$, which is equivalent to

$$F_2(F_1(x)) + F_1(x) \approx H(x)$$

Therefore, in ResNet we want to substitute $F(z)$ by $F(z) + z$, which can be easily implemented with **shortcut connections** (in CNNs and Feed-forward Networks in general)

ResNet [He et al. 2016]

The architecture of the **Residual block** of ResNet looks like:



ResNet [He et al. 2016]

Several comments:

- This scheme can also be applied to any Feed-forward Network, although in ResNet is applied to convolutional layers
- Note that there is a non-linearity after every convolution
- When $F(z)$ and z do not have the same dimension, a linear projection $z \rightarrow L(z)$ is added to make them equal, so that $F(z) + L(z)$ is computed instead of $F(z) + z$:
  - A) Identity + Zero-padding shortcuts for additional dimensions
  - B) 1x1 convolutions shortcuts for additional dimensions (slightly better results wrt option A)
- Shortcut connections do not add significant computational complexity
- The gradient can be still computed with backpropagation

ResNet [He et al. 2016]

Shortcut (or skip) connections are a very old idea, widely used for Recurrent Neural Networks in the form of gating units (see [Hochreiter and Schmidhuber, 1997], for example)

In fact, **Highway Networks** [Srivastava et al., 2015], developed at the same time as ResNet, also present shortcut connections implemented with gating functions, in a more flexible (and complex) formulation: In Highway Networks, the gated shortcuts can be closed (representing standard networks), completely open (similar to ResNet) or in intermediate situations

ResNet [He et al. 2016]

A **Residual Network (ResNet)** is a "standard network" with shortcut connections between **ResNet blocks**

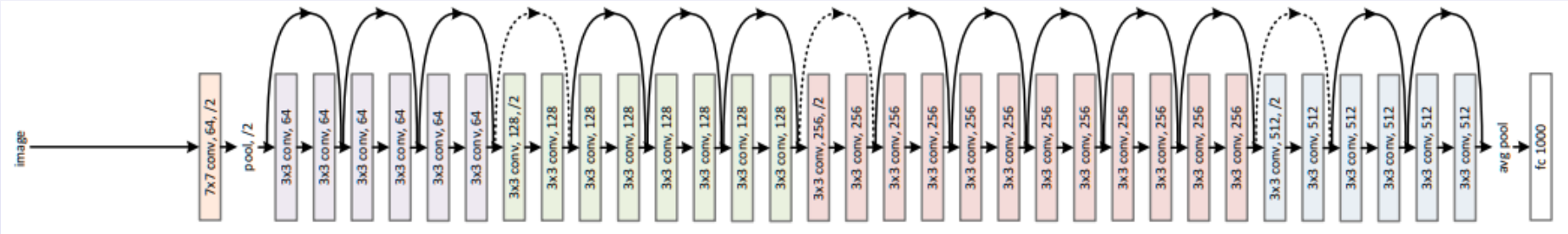


Figure 18: ResNet architecture (34 layers)

- The **first RestNet block** contains (BN: Batch Normalization)
  "7x7 Convolution + BN + ReLU + Max-Pooling"
- Each **intermediate ResNet block** contains
  "3x3 Convolution + BN + ReLU + 3x3 Convolution + BN"
- The **last RestNet block** contains
  "Average-Pooling + Fully Connected"

ResNet [He et al. 2016]

Additional comments:

- Dotted connections indicate a change in any of the dimensions
- Note that there are only two pooling layers and one FC layer (no Dropout), but many BN layers [Ioffe and Szegedy, 2015]

ResNet [He et al. 2016]

Training, similar to AlexNet and VGG:

- Stochastic Gradient Descent with Batch size: 256
- Weight decay: 0.0001
- Momentum: 0.9
- Learning rate scheme: initialized at 0.1, adjusted manually (divided by 10 when the validation error plateaus)
- Weights initialization as in their previous works
- Dataset: ImageNet-1K (2012)
- 224x224 crops randomly sampled from rescaled images
- Variable rescaling randomly selected in $[256, 480]$, as in VGG
- Data augmentation (not explained in detail)

The Test procedure is the same as in AlexNet

ResNet [He et al. 2016]

In several exploratory experiments:

- They compared a 18-layers and a 34-layers networks without shortcut connections and observed that the former has **higher training and validation error** than the latter
- Therefore, there exists a difficulty in training deeper networks which seems something implicit as a consequence of increasing the network depth
- **When the same experiment is repeated with networks with shortcut connections, the situation is reversed**
- Moreover, **residual networks converge faster**
- ResNet with 50/101/152 layers obtained less error on image classification task than 34-layers "plain" networks

Therefore, the authors **suggest that ResNet are easy to optimize and still can improve when increasing depth**

ResNet [He et al. 2016]

For computational reasons, in ResNet models with 50/101/152 layers the **intermediate ResNet blocks**

"3x3 Convolution + BN + ReLU + 3x3 Convolution + BN"

are replaced by **Bottleneck ResNet blocks**

"1x1 Conv + BN + 3x3 Conv + BN + 1x1 Conv + BN + ReLU"

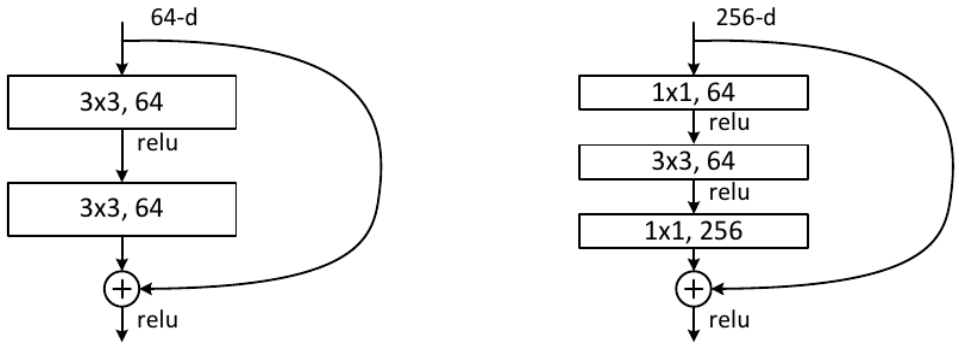


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

ResNet - Summary

ResNet revolutionized CNNs by introducing the concept of Residual Learning in CNNs and devised an efficient methodology for the training of deep networks

ResNet is 20 and 8 times deeper than AlexNet and VGG, respectively, showing less computational cost
(note that ResNet uses Batch Normalization, developed after AlexNet and VGG)

ResNet (MSRA) was **the winner of the the ILSVRC-2015 Object Detection and Localization tasks** with a 152-layers CNN: `https://image-net.org/challenges/LSVRC/2015/results.php`

ResNet also obtained **good performance on object detection and localization tasks**

Outline

Outline

Multi-path Exploitation

As already explained, Deep CNNs may suffer from problems directly caused by the increase in the depth, such as the vanishing gradient problem or other (not very well-known) causes

Similar to ResNet, other models have exploited and refined the idea of **using multiple paths (or shortcut connections)**, such as:

- Highway Networks [Srivastava et al., 2015]
- ResNet [He et al. 2016]
- DenseNet [Huang et al. 2017]
- ...

where different ways of implementing the shortcut connections are proposed

Outline

Outline

Width Exploitation

Back to the origins of MLPs, the next element explored was **the width of the network (i.e., the number of kernels)**:

- For MLPs, the width of the network was a very relevant parameter, as it was well-known for many years
- In [lu et al., 2017, Hanin and Sellke 2017] it was shown that Neural Networks with the ReLU activation function have to be wide enough to hold universal approximation property (along with an increase in depth)
- In [Kawaguchi et al., 2019], it was both theoretically and empirically shown that the quality of local minima tends to improve as depth and width increase
- Additionally, a deep and narrow network is slower to train than a wider (and not so deep) network with the same number of parameters (parallel computations in the same layer)

Width Exploitation

In this way, several researchers started to study the behaviour of "wide and not so deep" CNN architectures, and compared them with the (up to this moment) "very deep and narrow" architectures

Several models in this category:

- All the architectures in the "Inception family": Inception-V2 and -V3 [Szegedy et al., 2016], Inception-V4 and -ResNet [Szegedy et al., 2017]
- Xception [Chollet 2017]
- Wide Residual Network [Zagoruyko and Komodakis, 2017]
- Pyramidal Residual Network [Han et al. 2017]
- ResNext [xie et al., 2017]
- ...

Outline

WideResNet [Zagoruyko and Komodakis, 2017]

In [Zagoruyko and Komodakis, 2017], a number of experiments of the ResNet architecture is presented by varying:

- The structure of the convolutions in every ResNet block (including 1x1 and 3x3 convolutions in different ways)
- The number of Convolutional layers in every ResNet block
- The width of the ResNet blocks, including a widening factor k which multiplies the number of kernels
- The number of ResNet blocks (i.e. the number of residual connections)

keeping (as much as possible) the same number of parameters in the compared architectures

The rest of the architecture is similar to ResNet, except for the **addition of a Dropout layer between convolutions and after ReLU** in order to reduce the need of data augmentation

WideResNet [Zagoruyko and Komodakis, 2017]

Without using Dropout in the ResNet blocks, results in [Zagoruyko and Komodakis, 2017] can be summarized as follows:

- Blocks with comparable number of parameters perform similar, so **blocks with 3x3 only convolutions** are selected
- The **optimal number of Convolutional layers in every ResNet block is two** (out of two, three, four)
- Regarding the widening factor k:
  - Usually, **keeping the number of layers fixed, performance improves as width is increased**
  - Usually, **keeping the widening factor fixed, performance improves as depth is increased**
  - **With the same number of parameters**
    - WideResNet ($k = 4, 8, 10$) **outperform original ResNet**
    - The number of layers is reduced by a factor between 25 and 35
    - **Training is several times faster**

Using Dropout in the ResNet blocks results in slight improvements

WideResNet - Summary

Training procedure:

For experiments we chose well-known CIFAR-10, CIFAR-100, SVHN and ImageNet image classification datasets. CIFAR-10 and CIFAR-100 datasets [17] consist of 32×32 color images drawn from 10 and 100 classes split into 50,000 train and 10,000 test images. For data augmentation we do horizontal flips and take random crops from image padded by 4 pixels on each side, filling missing pixels with reflections of original image. We don't use heavy data augmentation as proposed in [9]. SVHN is a dataset of Google's Street View House Numbers images and contains about 600,000 digit images, coming from a significantly harder real world problem. For experiments on SVHN we don't do any image preprocessing, except dividing images by 255 to provide them in [0,1] range as input. All of our experiments except ImageNet are based on [13] architecture with pre-activation residual blocks and we use it as baseline. For ImageNet, we find that using pre-activation in networks with less than 100 layers does not make any significant difference and so we decide to use the original ResNet architecture in this case. Unless mentioned otherwise, for CIFAR we follow the image preprocessing of [8] with ZCA whitening. However, for some CIFAR experiments we instead use simple mean/std normalization such that we can directly compare with [13] and other ResNet related works that make use of this type of preprocessing.

In all our experiments we use SGD with Nesterov momentum and cross-entropy loss. The initial learning rate is set to 0.1, weight decay to 0.0005, dampening to 0, momentum to 0.9 and minibatch size to 128. On CIFAR learning rate dropped by 0.2 at 60, 120 and 160 epochs and we train for total 200 epochs. On SVHN initial learning rate is set to 0.01 and we drop it at 80 and 120 epochs by 0.1, training for total 160 epochs. Our implementation is based on Torch [6]. We use [21] to reduce memory footprints of all our networks. For ImageNet experiments we used `fb.resnet.torch` implementation [10]. Our code and models are available at https://github.com/szagoruyko/wide-residual-networks.

WideResNet - Summary

In summary, results of WideResNet indicate that

- **The main power of ResNet lays in the residual blocks, not in the depth**
- **Increasing width is as important as increasing depth**

Outline

Outline

Feature-map Exploitation

Remember that:

- **Feature Selection** aims the select the **input** features that are most relevant to the problem (i.e. remove the less important)
- A flexible version of Feature Selection is to assign a **Feature Saliency (or relative importance)** to every input feature

Back to the view of a CNN as a MLP:

- In MLPs, every layer can be seen as a **transformation of a set of features into another ones**
- These features are, except for the first layer, **learned from data** during the training process
- This property is **shared between MLPs and CNNs** (in fact, this is **one of the main properties of Neural Networks)**

Feature-map Exploitation

Imagine that we join the ideas of Feature Learning and Feature Saliency:

- We want to **learn the transformations among features**
- We also want to **assign a relative importance to every feature** learned

Then,

- In MLPs, it means to assign a saliency to every hidden unit
- In CNNs, it means **to assign a saliency to every kernel (equivalently, to every channel of the convolution or Feature Map)**

Feature-map Exploitation

Feature-map exploitation methods aim to **EXPLICITLY model a mechanism to selectively emphasize more informative features and mitigate less useful ones**

The word **EXPLICITLY** is important in this context, because it could be thought that the network can do it automatically (assigning different ranges of values to the kernels or including linear transformations, for example), but in practice it is much better to force it explicitly (as in the Feature Selection case)

Several models in this category:

- Squeeze-and-Excitation Networks [Hu et al. 2018a]
- Competitive Squeeze-and-Excitation Networks [Hu et al. 2018b]
- ...

Outline

Squeeze-and-Excitation Networks [Hu et al. 2018a]

The main idea of the Squeeze-and-Excitation Networks is to include a **Squeeze-and-Excitation block** after any transformation $\mathbf{F_{tr}}$ in the network

$$\mathbf{F_{tr}} : \mathbb{R}^{H' \times W' \times C'} \rightarrow \mathbb{R}^{H \times W \times C}$$

Typically, $\mathbf{F_{tr}}$ **will be a stack of convolutions**, but it may include any combination of layers

Suppose that $\mathbf{F_{tr}}(X) = U \in \mathbb{R}^{H \times W \times C}$, where $U = [U_1, U_2, ..., U_C]$, and each U_k is a $H \times W$ matrix (the feature map for channel k)

The application of a Squeeze-and-Excitation block $\mathbf{F_{se}}$ after $\mathbf{F_{tr}}$ will compute a vector of scalars $\boldsymbol{s}$ that will weight every element in U:

$$\mathbf{F_{se}}(\mathbf{F_{tr}}(X)) = \mathbf{F_{se}}(U) = [s_1 U_1, s_2 U_2, ..., s_C U_C]$$

Squeeze-and-Excitation Networks [Hu et al. 2018a]

The scheme of the **Squeeze-and-Excitation block** is:

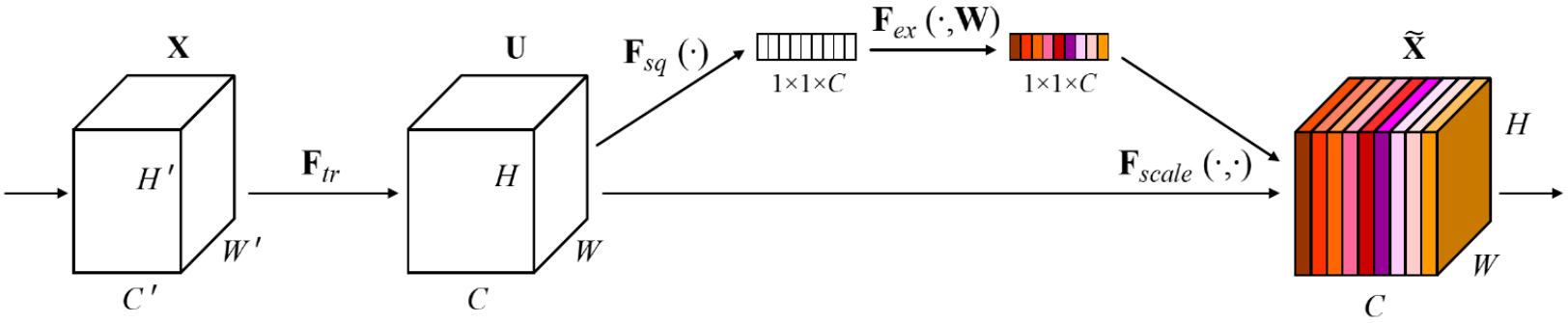


Figure 1: A Squeeze-and-Excitation block.

The Squeeze-and-Excitation block contains three steps:

- **Squeeze**, which aggregates the values of the feature maps
- **Excitation**, where the saliency (excitation) of each channel is computed
- **Reweighting** of the values in U with the saliency values

Squeeze-and-Excitation Networks [Hu et al. 2018a]

- The squeeze function $\mathbf{F_{sq}}$ computes the average of channel k:

$$\mathbf{F_{sq}}(U_k) = \frac{1}{H \cdot W} \sum_{1}^{H} \sum_{i}^{W} U_k(i,j)$$

  so that $\mathbf{F_{sq}}(U)$ is a vector of dimension C

- The excitation function $\mathbf{F_{ex}}$ is a one-hidden-layer MLP with ReLU (δ) hidden units, sigmoid (σ) output units and a reduction factor $\mathbf{r}$:

$$\mathbf{F_{ex}}(z) = \sigma(W_2 \cdot \delta(W_1 \cdot z)), \quad W_1 \in \mathbb{R}^{\frac{C}{\mathbf{r}} \times C} \; W_2 \in \mathbb{R}^{C \times \frac{C}{\mathbf{r}}}$$

  so that $\mathbf{F_{ex}}(z)$ is a vector $\{s_1, s_2..., s_C\}$ in $[0,1]^C$

- The output after reweighting is

$$\mathbf{F_{se}}(X) = [s_1 U_1, s_2 U_2, ..., s_C U_C]$$

  where $U = \mathbf{F_{tr}}(X)$ and $\{s_1, s_2..., s_C\} = \mathbf{F_{ex}}(\mathbf{F_{sq}}(U))$, $s_i \in [0,1]$

Squeeze-and-Excitation Networks [Hu et al. 2018a]

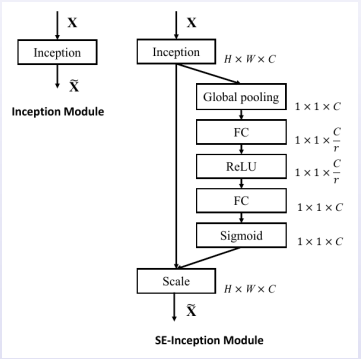
Note that:

- More sophisticated strategies can be used in the squeeze and excitation functions
- The sigmoid in the output layer of the excitation function guarantees that the weights are in $[0, 1]$
- The reduction factor $\mathbf{r}$ is a hyperparameter to be set *a priori*
- Squeeze-and-Excitation blocks only add a slight increase in the number of parameters and computational cost

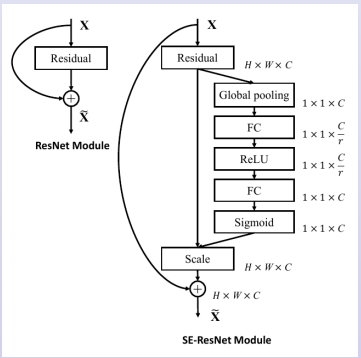
Squeeze-and-Excitation Networks [Hu et al. 2018a]

Squeeze-and-Excitation blocks can be added to any architecture:

- Straightforward for standard architectures (AlexNet, VGG, ...)
- For Inception networks, it is added to an entire Inception block



- For ResNet, it is added in the non-residual branch



- Similar to Inception networks or ResNet, it can be added to other architectures (Inception-ResNet, ResNext, MobileNet, ShuffleNet,...)

Squeeze-and-Excitation Networks [Hu et al. 2018a]

The training and testing procedures do not seem difficult to replicate but has several particularities for every model:

Each plain network and its corresponding SE counterpart are trained with identical optimisation schemes. During training on ImageNet, we follow standard practice and perform data augmentation with random-size cropping [43] to 224×224 pixels (299×299 for Inception-ResNet-v2 [42] and SE-Inception-ResNet-v2) and random horizontal flipping. Input images are normalised through mean channel subtraction. In addition, we adopt the data balancing strategy described in [36] for mini-batch sampling. The networks are trained on our distributed learning system "ROCS" which is designed to handle efficient parallel training of large networks. Optimisation is performed using synchronous SGD with momentum 0.9 and a mini-batch size of 1024. The initial learning rate is set to 0.6 and decreased by a factor of 10 every 30 epochs. All models are trained for 100 epochs from scratch, using the weight initialisation strategy described in [9].

When testing, we apply a centre crop evaluation on the validation set, where 224×224 pixels are cropped from each image whose shorter edge is first resized to 256 (299×299 from each image whose shorter edge is first resized to 352 for Inception-ResNet-v2 and SE-Inception-ResNet-v2).

Squeeze-and-Excitation Networks - Summary

In the experiments described:

- A reduction factor value of $r = 16$ shows a good trade-off between accuracy and complexity
- Regarding the role of the excitation values:
  - The distribution across labels is nearly identical in lower layers but becomes more specific in deeper layers (consistent with the idea that learned features are general for lower layer features and specific for higher ones)
  - However, at the end of the network they have no relevance and can be removed
- State-of-the art results on several datasets are obtained with different Squeeze-and-Excitation architectures

A small ensemble of Squeeze-and-Excitation Networks (WMW) was **the winner of the the ILSVRC-2017 Classification task**: `https://image-net.org/challenges/LSVRC/2017/results.php`, obtaining a 2.25% error on the test set (more details in [Hu et al. 2018a])

Outline

Outline

Channel Boosting

In the same line of Feature-map Exploitation, but in a different direction, **Channel Boosting** methods try to **boost the representational power of the network by adding extra channels** at some point of the architecture

These extra channels are **generated by some auxiliary systems**, already trained with the data (Auto-encoders, for example), and typically **inserted at the initial block of the network**

Several models in this category:

- Channel Boosted CNNs [Khan et al., 2018]
- ...

Outline

Outline

Attention-based Architectures

In Cognitive Sciences, **Attention** is the ability to choose and concentrate on relevant stimuli, and is essential for human behaviour

The different levels of abstraction learned by CNNs, that have already been shown very effective for many problems, but they **typically apply to individual entities**

In contrast, an attention mechanism should **selectively pay attention to the context of these individual entities**: This process may infer different interpretations of the objects at a particular location and thus help to capture the global structure in a better way

Attention-based Architectures

Similar to Long Short-term Memory Networks [Hochreiter and Schmidhuber, 1997] or Transformers [Vasvani et al., 2017], where attention modules are implemented for sequential data, **Attention-based** CNN architectures implement attention mechanisms for working with images

Several models in this category:

- Residual Attention Networks [wang et al., 2017]
- Convolutional Block Attention Module [woo et al., 2018]
- Concurrent Spatial and Channel Squeeze and Excitation Networks [Roy et al., 2018]
- Convolutional Vision Transformer Networks [wu et al., 2021]
- ...

Outline

And much more...

Apart from changes in the architecture, there are still many things that you can modify:

- The convolution operation
- The optimization scheme
- The non-linear activation functions
- The pooling schemes
- The data augmentation procedures
- The cost function
- ...

They can be general or specific for the task at hand

You can find more details at [Li et al., 2022, Younesi et al., 2044]

Outline

Other Types of Convolutions

Apart from the standard convolution already explained, we can find other types of convolutions:

- Depthwise separable convolutions, which decouple spatial and channel operations
- Deformable convolution, more robust to geometric deformations
- Steerable convolution, more robust to linear transformations
- Dilated (aka atrous) convolutions, that increase the receptive field without increasing the number of parameters
- Transposed convolutions (aka deconvolutions), used for upsampling (i.e., increase the feature map size) for the reconstruction of higher-resolution representations from lower-resolution ones
- ...

Outline

Optimization Schemes

Beyond the standard implementation of SGD, different adaptive gradient schemes can be found, with the aim of accelerating the training process:

- RMSProp [Tieleman and Hinton, 2012]
- Adam [Kingma and Ba, 2015]
- Adagrad [Duchi et al., 2011]
- ...

Outline

Non-linear Activation Functions

A number of variations have been defined for the ReLU, the most commonly used non-linear activation function [Li et al., 2022]:

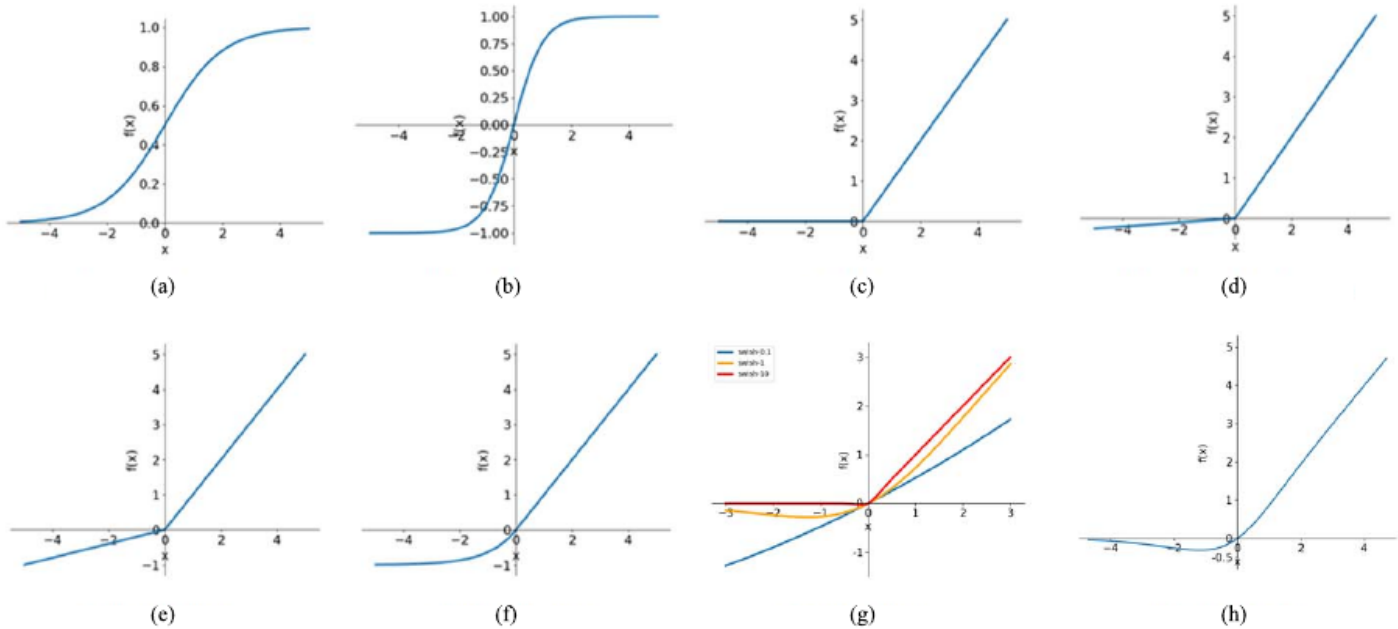


Fig. 13. Diagrams of sigmoid, tanh, ReLU, leaky ReLU, PReLU, ELU, swish, and mish. (a) Sigmoid function. (b) Tanh function. (c) ReLU function. (d) Leaky ReLU function. (e) PReLU function. (f) ELU function. (g) Swish functions. (h) Mish function.

More recently, [Bingham and Miikkulainen, 2022] presented an evolutionary algorithm for the automatic construction of customized activation functions

Outline

Pooling Schemes

You can also find different pooling schemes:

- Global Average Pooling (as in GoogleNet)
- Adaptive Average-Pooling
- Pyramid Pooling
- Spatial Pyramid Pooling
- ...

Outline

Data Augmentation and Dropout Schemes

Since data augmentation scheme is very important to reduce overfitting, different variants have been proposed:

- MixUp (combine two images)
- CutOut (remove a patch)
- CutMix (remove and combine)
- ...

In the same way, dropout can be applied in different ways:

- Spatial dropout
- Cutout dropout
- ...

Outline

Cost Functions

As we already know, the loss function is related to the type of the problem we want to solve

For classification problems, the most usual loss function is the well-known cross-entropy function

$$J(\boldsymbol{\theta}, \boldsymbol{D}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{c} -1_{(y^{(i)}=c)} \cdot \log h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right)_c$$

which is essentially equivalent to **negative log-likelihood** when $h_{\boldsymbol{\theta}}(\cdot)_c$ is a softmax function

Cost Functions

From a Physics point of view, we can see the learning process as obtaining the optimal θ so as to **minimize the global energy of the system** (in $\boldsymbol{D}$ or the whole space): $-1_{(y^{(i)}=c)} \log h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)}\right)_c$ is the **energy** of the pair $(\boldsymbol{x}^{(i)}, y^{(i)})$ when assigned to class c

The cross-entropy **drives the energy of the desired output (assign it to the correct class) to be lower than the energies of all the other possible outputs** [LeCun and Huang, 2005]: Minimizing the loss function should result in "holes" at locations near the training samples, and "hills" at undesired locations

In tasks different from classification, where the minimization of the cross-entropy makes no sense (for example, construct a similarity metric between images), we still can apply the same principles to construct well-designed loss functions

Cost Functions

One of the most widely used loss functions following this principle is the **contrastive loss** [Chopra et al., 2005], which

- Assigns an energy $E_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ for a pair of input vectors
- Defines the individual loss as the sum of two terms:

$$J(\boldsymbol{\theta}, Y_{ij}, \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (1 - Y_{ij}) L_S \left(E_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \right) + Y_{ij} L_D \left(E_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \right)$$

  where
  - $Y_{ij} \in \{0, 1\}$ specifies if $\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}$ are similar ($Y_{ij} = 0$) or not
  - L_S and L_D are typically monotonically increasing/decreasing
- The global loss function is obtained summing up over the whole data

Cost Functions

In this setting:

- When the loss function is minimum, similar pairs have low energy and dissimilar pairs have high energy: it can be seen, again, as a measure of the energy of the system
- The Y_{ij}) values have come from our **prior knowledge** of the problem (in other words, we are free to construct different models if we change the set of similar inputs, as in a standard classification problem.
- Constrative loss can be used instead of the cross-entropy loss function [Hadsell et al., 2006] or as a **complement to obtain specific properties of the output function**: minimize the intra-class distances of the **internal representations**, maximize the inter-class distances, try to obtain more uniform distributions, etc

Cost Functions

Different variations of the loss function (most of them including ideas from contrastive loss) can be found in the literature:

- Large-margin softmax loss
- Center loss
- Contrastive Center loss
- Constrained Center Loss
- ...

A different approach can be found in **style transfer**, where the loss function is the sum of two terms, one for the content and other one for the style [Gatys et al., 2016]

Outline

More Architectures

- FractalNet
- PyramidalNet
- EfficientNet
- ConvNext
- U-Net
- ShuffleNet
- GhostNet
- MobileNet and other architectures specially designed for devices with limited computing power
- ...

That's it!

That's it!

Bibliography

- Arora, S., Bhaskara, A., Ge, R., and Ma, T. (2014). Provable Bounds for Learning Some Deep Representations. In *International Conference on Machine Learning*, pages 584–592).
- Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.
- Bianchini, M., and Scarselli, F. (2014). On the Complexity of Neural Network Classifiers: A Comparison between Shallow and Deep Architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565.
- Bingham, G., and Miikkulainen, R. (2022). Discovering Parametric Activation Functions. *Neural Networks*, 148:48–65.
- Bozinovski, S. and Fulgosi, A. (1976). The Influence of Pattern Similarity and Transfer Learning on the Base Perceptron Training (original in Croatian). *Proceedings of Symposium Informatica*, 3–121–5, Bled.
- Bozinovski, S. and (12020). Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, 44:291–302.
- Chellapilla, K., Puri, S., and Simard, P. (2006). High Performance Convolutional Neural Networks for Document Processing. In *International Workshop on Frontiers in Handwriting Recognition*.
- Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1251–1258.
- Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 539–546.
- Cireșan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. *Neural Computation*, 22(12), 3207–3220.
- Cireșan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. In *International Joint Conference on Artificial Intelligence*.
- Csáji, B.C. (2001). Approximation with Artificial Neural Networks. MSc Thesis, *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48), 7.

Bibliography

- Delalleau, O., and Bengio, Y. (2011). Shallow vs. Deep Sum-product Networks. In *Advances in Neural Information Processing Systems 24*.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.
- Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36:193–202.
- Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image Style Transfer using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2414–2423.
- Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256.
- Goyal, A. and Bengio, Y. (2022). Inductive Biases for Deep Learning of Higher-level Cognition. *Proceding of the Royal Society. A*, 478:20210068.
- Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1735–1742.
- Han, D., Kim, J., and Kim, J. (2017). Deep Pyramidal Residual Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 5927–5935.
- Hanin, B. and Sellke, M. (2017). Approximating Continuous Functions by ReLU Nets of Minimal Width. *arXiv preprint* arXiv:1710.11278.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-term Memory. *Neural Computation*, 9(8):1735-1780.
- Howard, A. G. (2013). Some Improvements on Deep Convolutional Neural Network Based Image Classification. *arXiv preprint* arXiv:1312.5402.
- Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-Excitation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 7132–7141.

Bibliography

- Hu, Y., Wen, G., Luo, M., Dai, D., Ma, J., and Yu, Z. (2018). Competitive Inner-imaging Squeeze and Excitation for Residual Network. *arXiv preprint* arXiv:1807.08920.
- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 4700–4708.
- Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32th International Conference on Machine Learning*.
- Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., and LeCun, Y. (2009). What is the Best Multi-stage Architecture for Object Recognition? In *International Conference on Computer Vision*.
- Kawaguchi, K., Huang, J., and Kaelbling, L. P. (2019). Effect of Depth and Width on Local Minima in Deep Learning. *Neural Computation*, 31(7), 1462–1498.
- Khan, A., Sohail, A., and Ali, A. (2018). A New Channel Boosted Convolutional Neural Network using Transfer Learning. *arXiv preprint* arXiv:1804.08528.
- Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2020). A Survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artificial Intelligence Review*, 53:5455-5516.
- Kingma, D. P. and Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*.
- Lang, K. J. and Hinton, G. E. (1988). The Development of the Time-delay Neural Network Architecture for Speech Recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University.
- Lang, K. J., Waibel, A. H., and Hinton, G. E. (1990). A Time-delay Neural Network Architecture for Isolated Word Recognition.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

Bibliography

- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- LeCun, Y., and Huang, F. J. (2005). Loss Functions for Discriminative Training of Energy-based Models. In *International Workshop on Artificial Intelligence and Statistics (PMLR)*, pages 206–213.
- Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33:6999–7019.
- Lin, M., Chen, Q., and Yan, S. (2013). Network in Network. *arXiv preprint* arXiv:1312.4400.
- Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The Expressive Power of Neural Networks: A View from the Width. In *Advances in Neural Information Processing Systems 30*.
- Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the Number of Linear Regions of Deep Neural Networks. In *Advances in Neural Information Processing Systems 27*.
- Roy, A. G., Navab, N., and Wachinger, C. (2018). Concurrent Spatial and Channel Squeeze & Excitation in Fully Convolutional Networks. *Lecture Notes in Computer Science* 11070 LNCS:421429.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. Berg, A. C. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 211-252.
- Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv preprint* arXiv:1312.6229.
- Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint* arXiv:1409.1556.
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training Very Deep Networks. In *Advances in Neural Information Processing Systems 28*.
- Strigl, D., Kofler, K., and Podlipnig, S. (2010). Performance and Scalability of GPU-based Convolutional Neural Networks. In *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*.

Bibliography

- Szegedy, C., Wei, L., Yangqing, J., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv preprint* arXiv:1409.4842.
- Szegedy C., Vanhoucke V., Ioffe S., Shlens, J., and Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2818-2826.
- Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *AAAI Conference on Artificial Intelligence*
- Tieleman, T. and Hinton, G. E. (2012). Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*.
- Uetz, R. and Behnke, S. (2009). Large-scale Object Recognition with CUDA-accelerated Hierarchical Neural Networks. In *IEEE International Conference on Intelligent Computing and Intelligent Systems*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit. J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. In *Advances in Neural Information Processing Systems 30*.
- Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., and Tang, X. (2017). Residual Attention Network for Image Classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 3156–3164.
- Woo, S., Park, J., Lee, J. Y., and Kweon, I. S. (2018). CBAM: Convolutional Block Attention Module. In *European Conference on Computer Vision*, 3–19.
- Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., and Zhang, L. (2021). Cvt: Introducing Convolutions to Vision Transformers. In *International Conference on Computer Vision*, 22–31.
- Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017) Aggregated Residual Transformations for Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1492–1500.
- Younesi, A., Ansari, M., Fazli, M., Ejlali, A., Shafique, M., and Henkel, J. (2024). A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends. *IEEE Access*, 12:41180–41218.
- Zagoruyko, S. and Komodakis, N. (2017). Wide Residual Networks. *arXiv preprint* arXiv:1605.07146.

Bibliography

- Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2528–2535.
- Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In *2011 IEEE International Conference on Computer Vision*, pages 2018–2025.
- Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision (Lecture Notes in Computer Science 8689)*, pages 818–833.