Theoretical foundations of Deep Learning and Convolutional Neural Networks

Anabel Gómez Ríos

anabelgrios@decsai.ugr.es

University of Granada

22nd of April, 2021

Outline

Classification of learning algorithms

- Based on the learning task: classification / regression.
- Based on the available information: supervised / unsupervised / semi-supervised.

In this presentation we are going to focus on:

- Supervised learning: we suppose we have the labels associated with the examples of the data set.
- Classification.

Classification of learning algorithms

- Based on the learning task: classification / regression.
- Based on the available information: supervised / unsupervised / semi-supervised.

In this presentation we are going to focus on:

- Supervised learning: we suppose we have the labels associated with the examples of the data set.
- Classification.

Outline

Deep Learning

- Deep Learning (DL) is a particular kind of machine learning.
- It represents the world as a nested hierarchy of concepts, with each concept defined in relation to simpler ones.
- It started in 1943 with the development of the artificial neuron by McCulloch and Pitts.

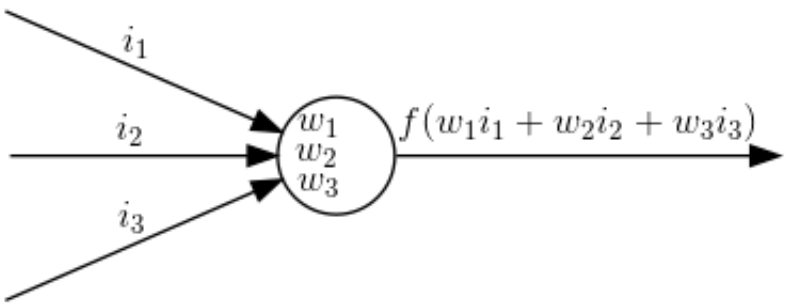


Figure: Artificial neuron.

Evolution of Deep Learning

- 1958: Implementation of the first models (perceptron). It is possible to train a single neuron.
- 1986: Use of back-propagation to train a neural network with one or two hidden layers.
- 1998: The size of the data sets begins to increase. MNIST.
- Early 2000s: Deep networks are believed to be very difficult to train (the algorithms are too computationally costly for the available hardware).
- Late 2000s: Powerful CPUs and GPUs. Better software infrastructure for distributed computing.
- 2012: The CNN AlexNet wins the ILSVRC competition.

Evolution of Deep Learning

- 1958: Implementation of the first models (perceptron). It is possible to train a single neuron.
- 1986: Use of back-propagation to train a neural network with one or two hidden layers.
- 1998: The size of the data sets begins to increase. MNIST.
- Early 2000s: Deep networks are believed to be very difficult to train (the algorithms are too computationally costly for the available hardware).
- Late 2000s: Powerful CPUs and GPUs. Better software infrastructure for distributed computing.
- 2012: The CNN AlexNet wins the ILSVRC competition.

ImageNet and ILSVRC

- The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition which evaluates algorithms for object detection and image classification at large scale. The classification competition uses a ImageNet subset of 1.2 million images to train, 1000 classes and 150.000 images to test the models.
- ImageNet is a large image database containing 14.197.122 images that are organized in a hierarchy.
- The classification competition has been won by CNNs since 2012.

Evolution of Deep Learning

- Deep Learning has become more useful as the amount of available training data has increased.
- DL models have grown in size as computer infrastructure (hardware and software) has improved.
- It has been able to solve increasingly complicated problems with increasing accuracy.

Deep Feedforward Networks

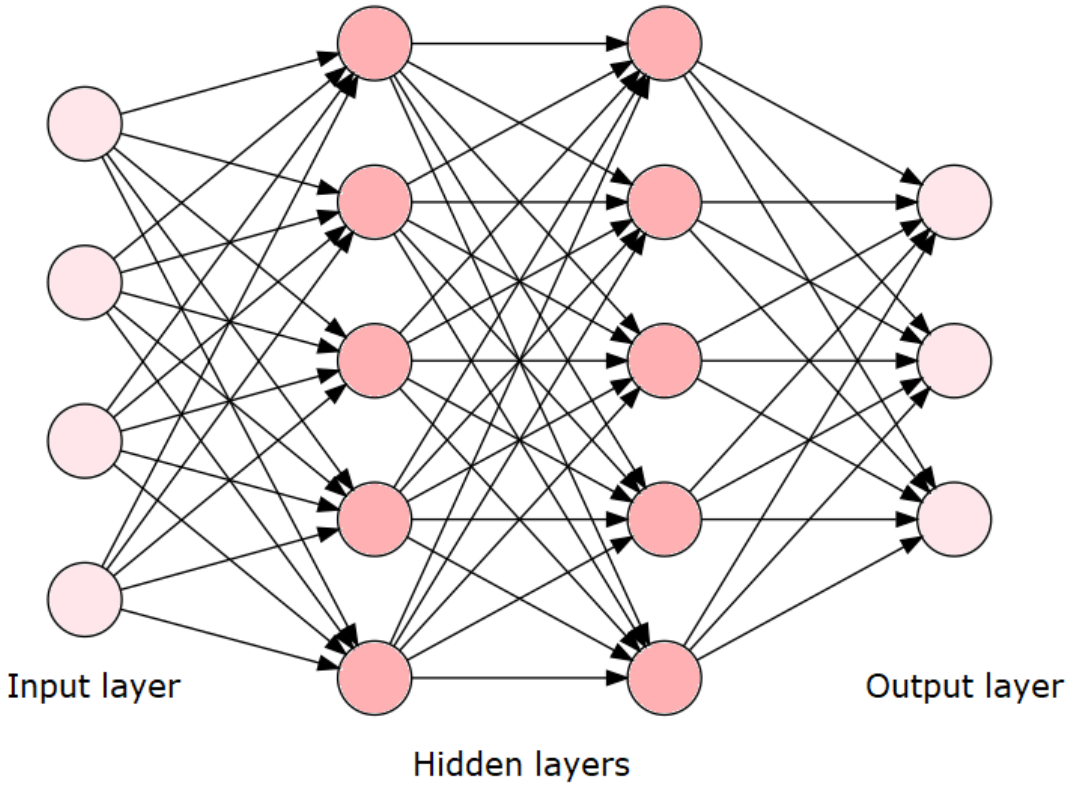


Figure: Feedforward Network.

Deep Feedforward Networks

- In a network with three layers, for an input $\boldsymbol{x}$ we have an output $f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x})))$, where f_1 is the first layer, f_2 the second layer, etc.
- Each layer has a set of parameters that the model will learn, so the model actually provides a function $f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters.
- Deep networks use **nonlinear** functions. Most of them use, in each layer, an affine transformation controlled by learned parameters followed by a fixed nonlinear function (activation) that will be applied element-wise.

Deep Feedforward Networks

- In a network with three layers, for an input $\boldsymbol{x}$ we have an output $f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x})))$, where f_1 is the first layer, f_2 the second layer, etc.
- Each layer has a set of parameters that the model will learn, so the model actually provides a function $f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters.
- Deep networks use **nonlinear** functions. Most of them use, in each layer, an affine transformation controlled by learned parameters followed by a fixed nonlinear function (activation) that will be applied element-wise.

Training a network: the objective function

- The objective function, also called loss function or cost function, is a function over the true value and the prediction of the network that outputs a real number, usually called cost.
- It is the function that the model will optimize during training.
- Some common regression losses are the mean square error or the mean absolute error.
- Some common probabilistic losses for classification are the cross-entropy or the Kullback-Leibler divergence.

Gradient-Based Learning

- The nonlinearity of the DL models causes most used loss functions to become nonconvex, so we lose the global convergence guarantee.
- As a result, DL models are usually trained using iterative, gradient-based optimizers.
- These optimizers drive the objective function to a very low value.
- One of the most used gradient-based algorithms is the Stochastic Gradient Descent (SGD).

Gradient Descent

- The derivative tells us how to change x in order to make a small improvement in y. That way, we can reduce $f(x)$ by moving x in small steps with the opposite sign of the derivate.

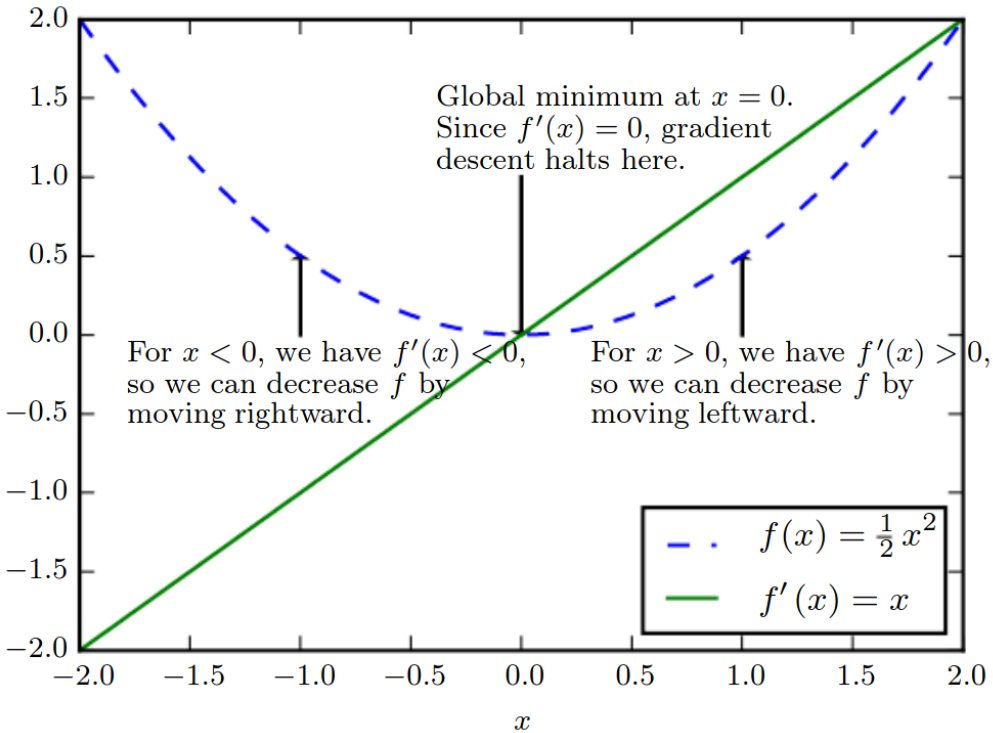


Figure: Gradient Descent. Figure from [1].

Gradient Descent

- When $\boldsymbol{x}$ is a vector, the gradient of f is the vector containing all the partial derivatives, denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.
- In each step, Gradient Descent proposes a new point

$$\boldsymbol{x}' = \boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \ ,$$

where ε is the **learning rate**, a positive scalar determining the size of the step.

Stochastic Gradient Descent

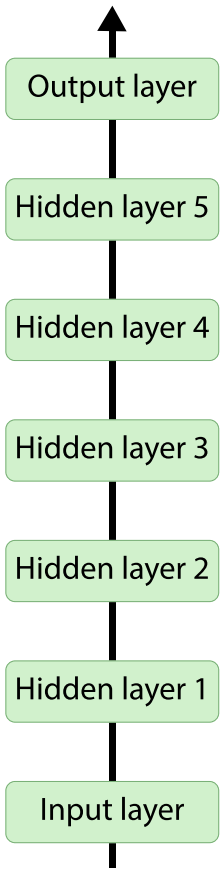
- SGD is an extension of Gradient Descent, where on each step, the algorithm uses a minibatch of the examples, instead of using all of them.
- The minibatch is drawn uniformly from the training set.
- Typically, the size of the minibatch is fixed and chosen to be small.

Training a neural network

- Iterative process.
- In each step, we want to make small changes in the parameters of the network so that the objective function is minimized.
- To do so, we use a gradient-based algorithm with a mini-batch of the training examples.
- How do we calculate the gradient of the objective function?

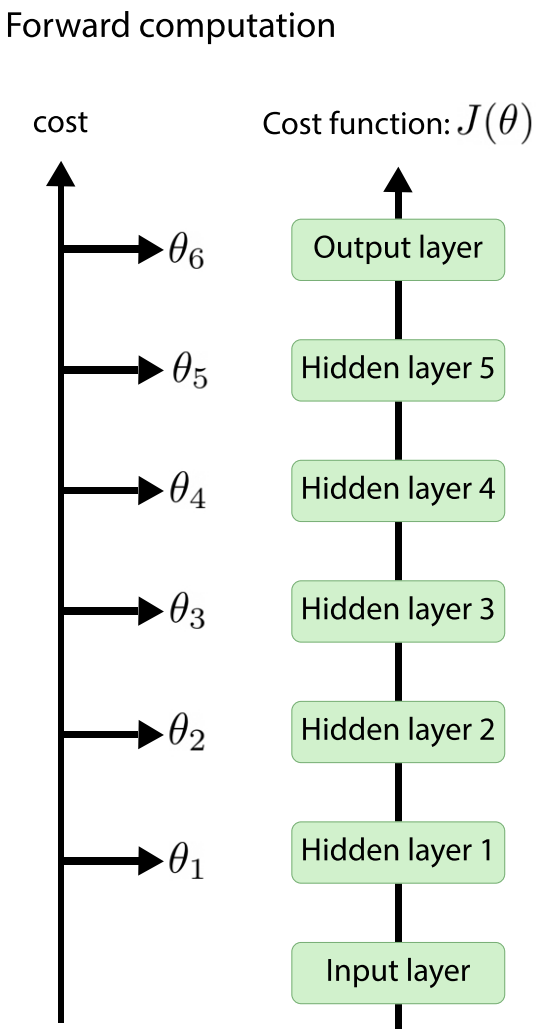
Back-propagation algorithm

- Let $\boldsymbol{x}$ be the input and $\widehat{\boldsymbol{y}}$ the output of the network.



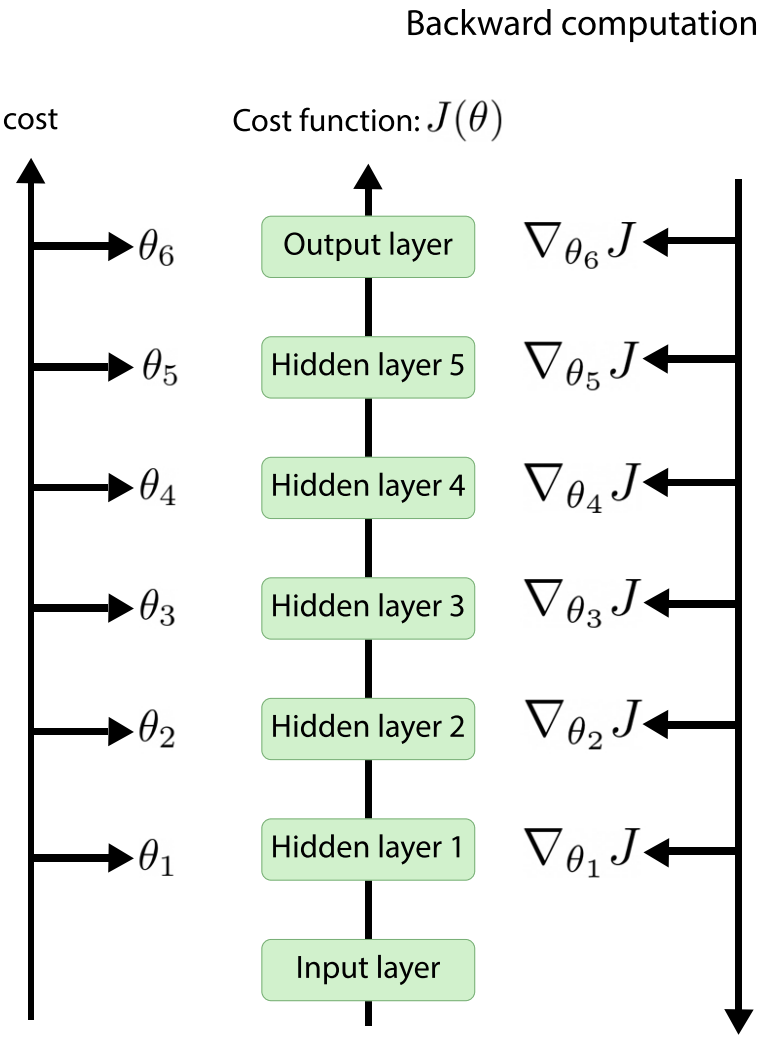
Back-propagation algorithm

- In the forward propagation, $\boldsymbol{x}$ provides the initial information that then propagates through the network, which produces $\widehat{\boldsymbol{y}}$ and a scalar cost (the value of the objective function, $J(\theta)$).



Back-propagation algorithm

- The back-propagation algorithm allows the information from the cost to then flow back backward through the network in order to compute the gradient of the objective function with respect to the parameters.



Back-propagation algorithm

- In each layer k, the computed gradients $\nabla_{\theta_k} J$ are used as part of a gradient-based optimization method, like SGD.
- The gradient-based optimization method then updates the parameters in each layer, θ_k, which will be used in the next iteration of the training.

Softmax function

- For classification, we need to produce a vector $\widehat{\boldsymbol{y}}$ at the end of the network, with $\widehat{y}_i = P(y = i|\boldsymbol{x})$.
- The softmax function transforms the output of the last Fully Connected (FC) layer to these probabilities.
- The last layer of the network has as many neurons as classes we want to classify the images into. Suppose we have K classes. That way, the output of the last FC layer (before softmax) will be a vector $\boldsymbol{z}$ with K components.

Softmax function

- The softmax function computes the probability that the image belongs to class k with $k = 1, \ldots, K$ and K the total number of classes.

$$\text{softmax} : \mathbb{R}^K \to [0, 1]^K$$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_k)}$$

- Then, to choose the class of the image, we choose the class with the highest probability.

Dropout

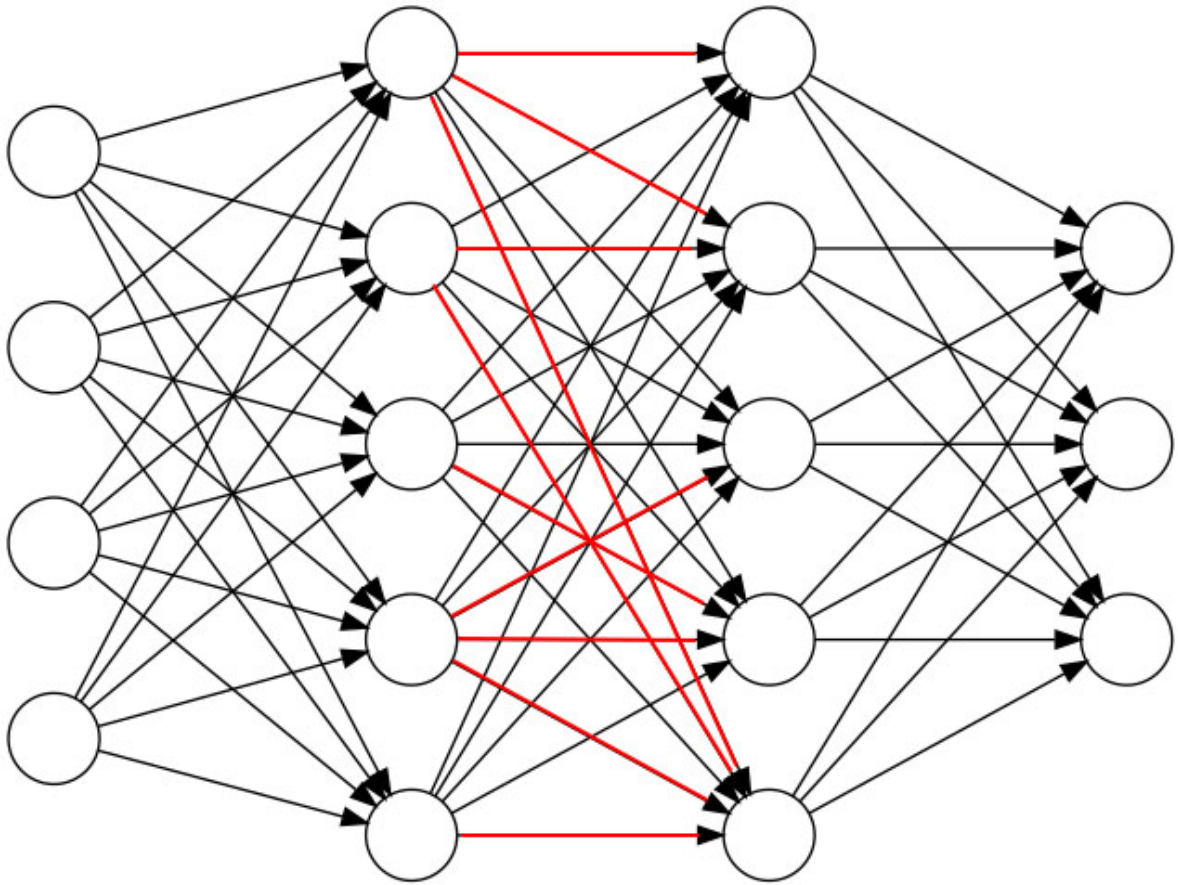


Figure: Example of 40% dropout during a training epoch.

Dropout

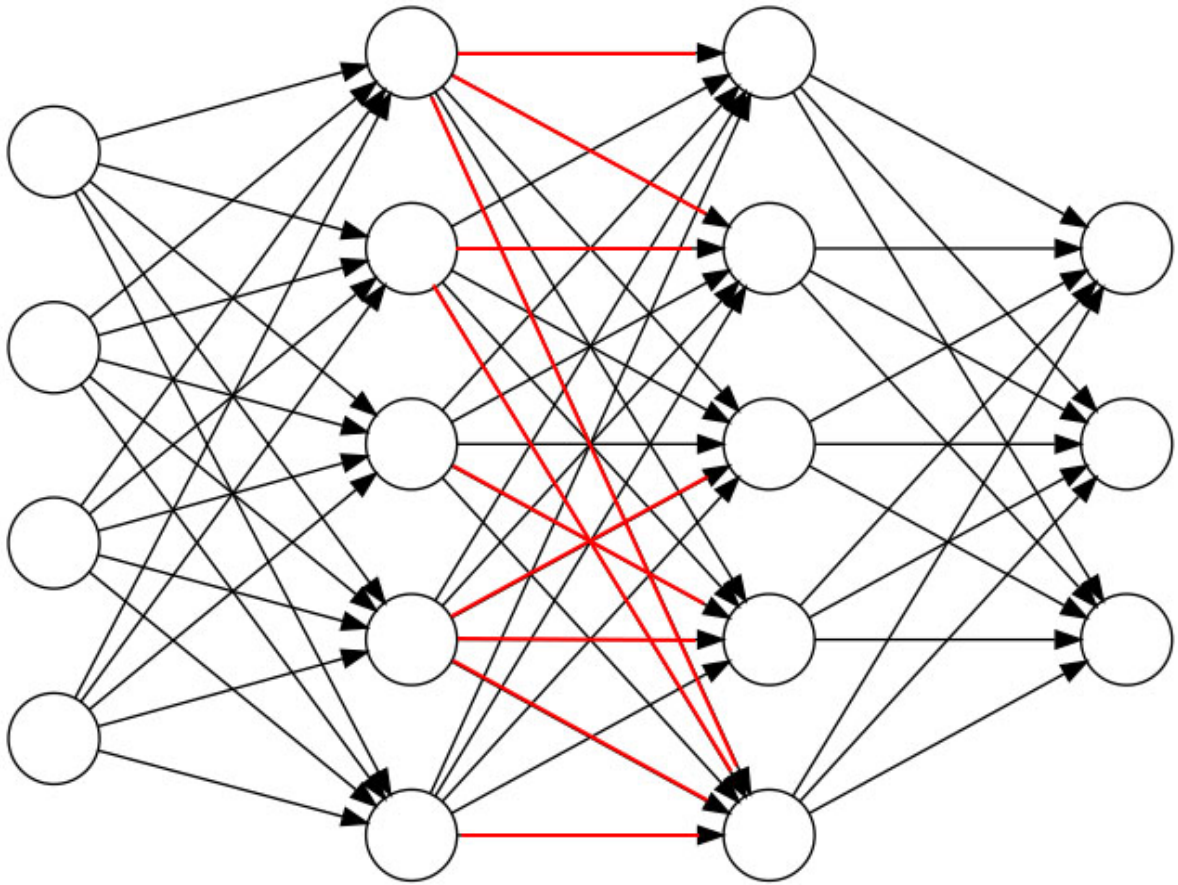


Figure: Example of 40% dropout during a training epoch.

Dropout

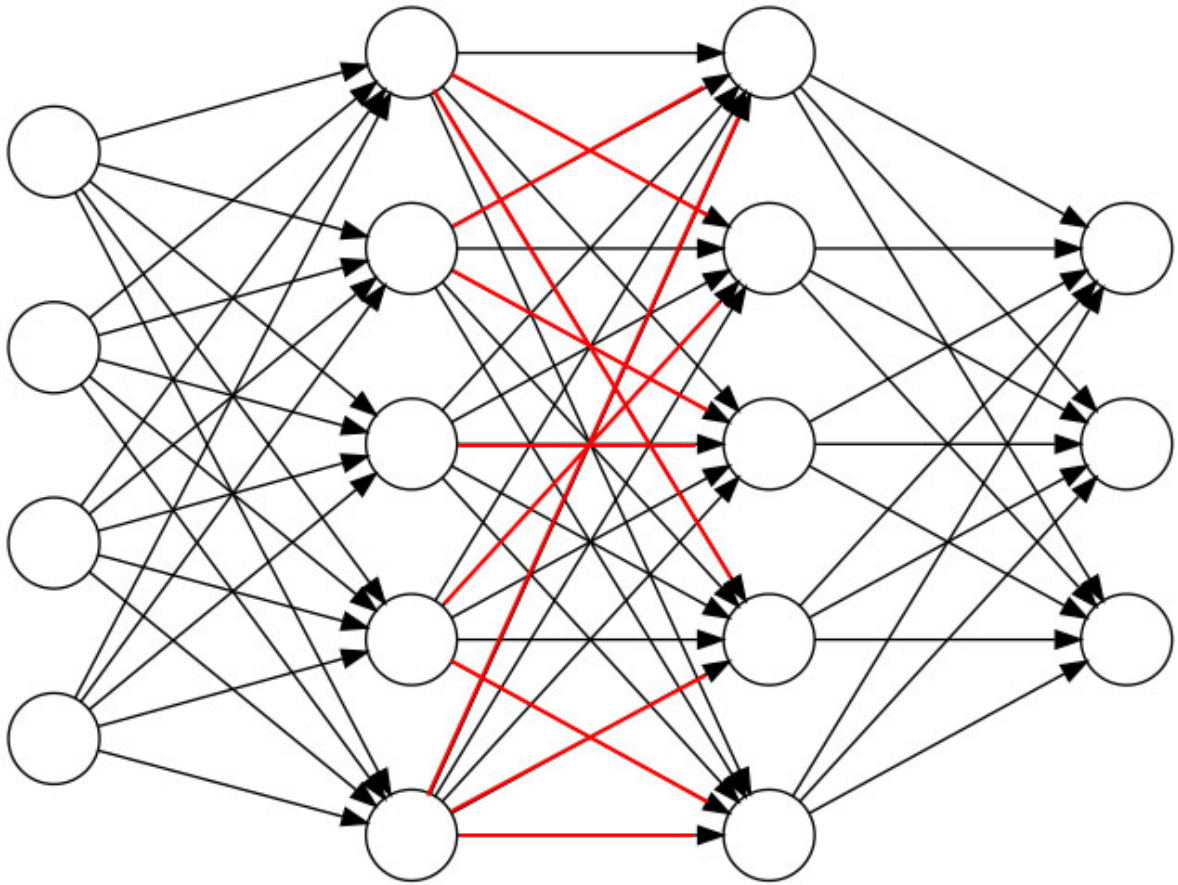


Figure: Example of 40% dropout during another training epoch.

Dropout

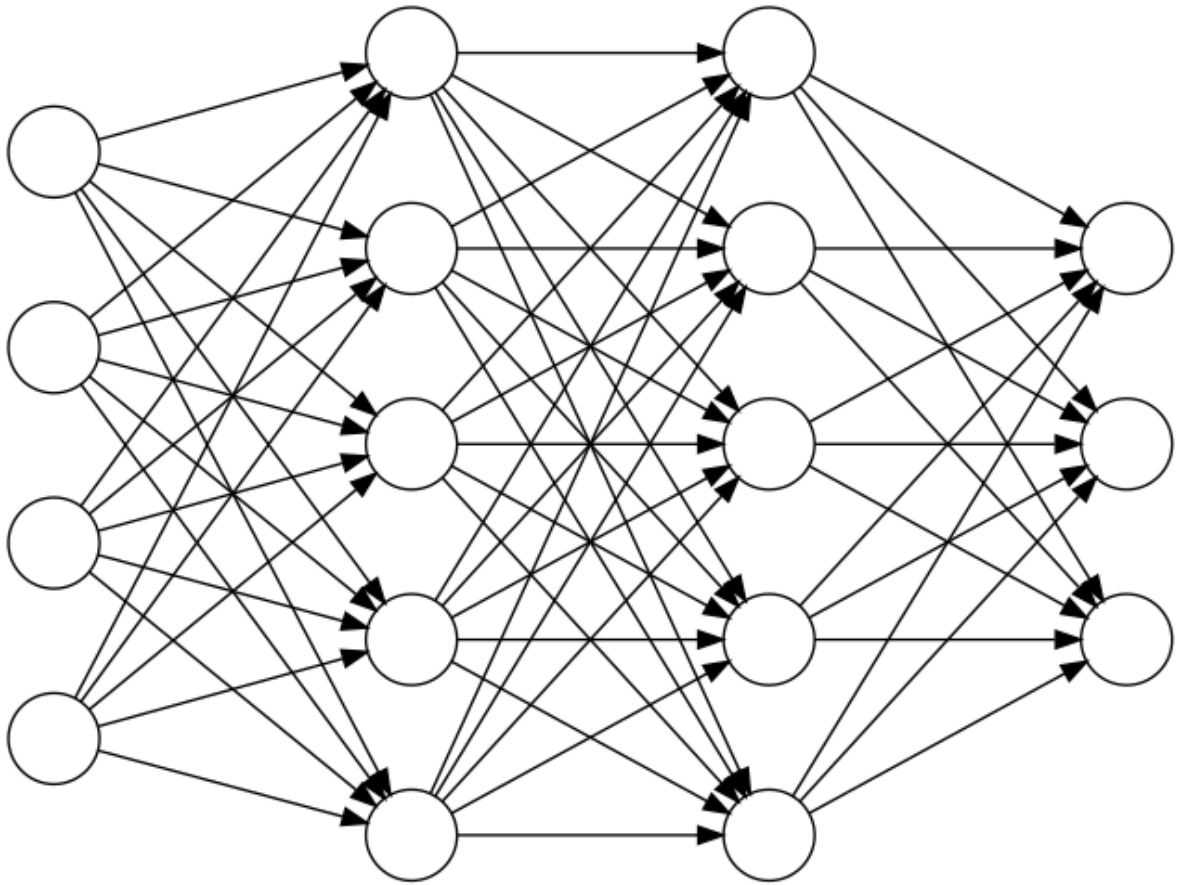


Figure: Example of 40% dropout during the test process.

Outline

Convolutional Neural Network (CNN)

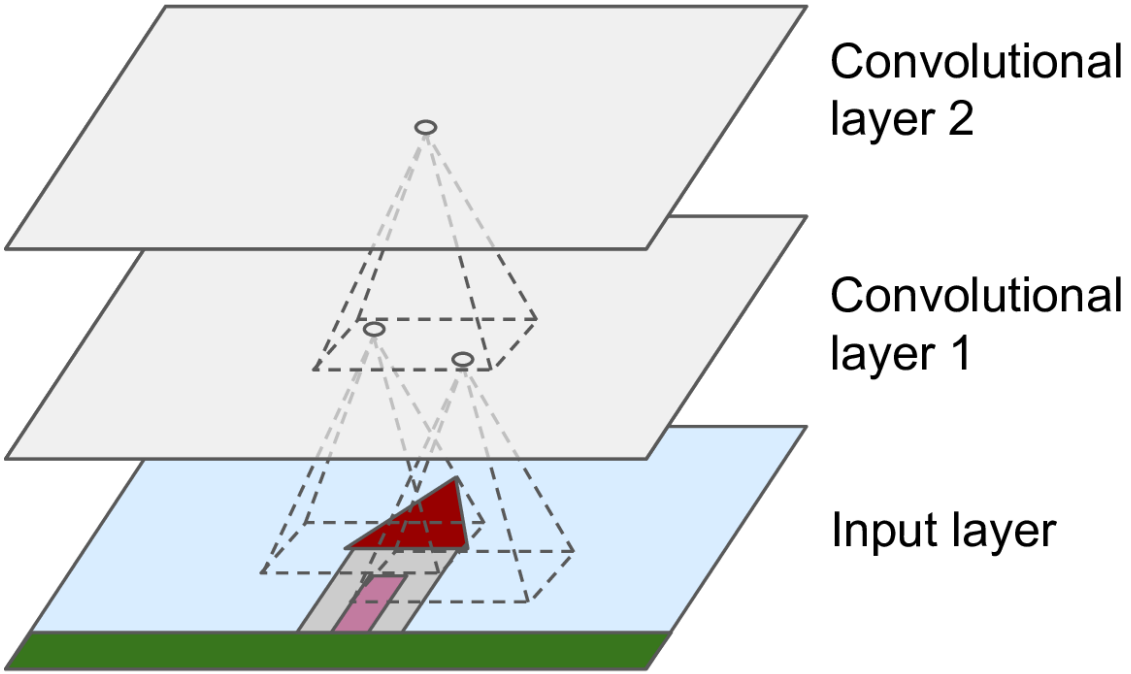


Figure: Convolutional Neural Network. Figure from [2].

The Convolution Operation

- The **convolution** operation is defined as [1]:

$$S(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \ ,$$

where I is the input and K is the kernel of the convolution (both two-dimensional in this case).

- Most libraries implement **cross-correlation** instead of convolution, which is the same but without flipping the kernel:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \ .$$

The Convolution Operation

- The **convolution** operation is defined as [1]:

$$S(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \ ,$$

where I is the input and K is the kernel of the convolution (both two-dimensional in this case).

- Most libraries implement **cross-correlation** instead of convolution, which is the same but without flipping the kernel:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \ .$$

The Convolution Operation

Example:

$$I = \begin{bmatrix} 5 & 9 & 1 & 3 & 2 \\ 0 & 8 & 2 & 4 & 1 \\ 1 & 7 & 3 & 0 & 3 \\ 4 & 0 & 6 & 9 & 0 \\ 5 & 1 & 0 & 2 & 1 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & -2 \\ -1 & 2 \end{bmatrix}.$$

$$S = \begin{bmatrix} 3 & 3 & 1 & -3 \\ -3 & 3 & -9 & 8 \\ -17 & 13 & 15 & -15 \\ 1 & -13 & -8 & 9 \end{bmatrix}.$$

Convolutional Layer

- In practice, a convolutional layer is composed of several different kernels (or filters), applied in parallel, so we can extract several features.
- The output of the convolution with each one of these kernels or filters is called feature map.
- In addition, the input of the convolutional layer is not usually a two-dimensional grid.

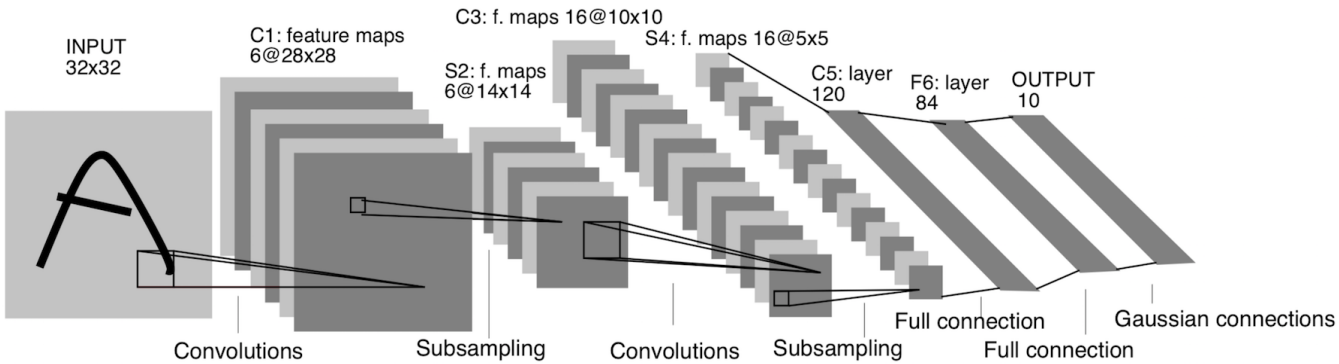


Figure: LeNet5. Figure from [3].

Convolutional Layer

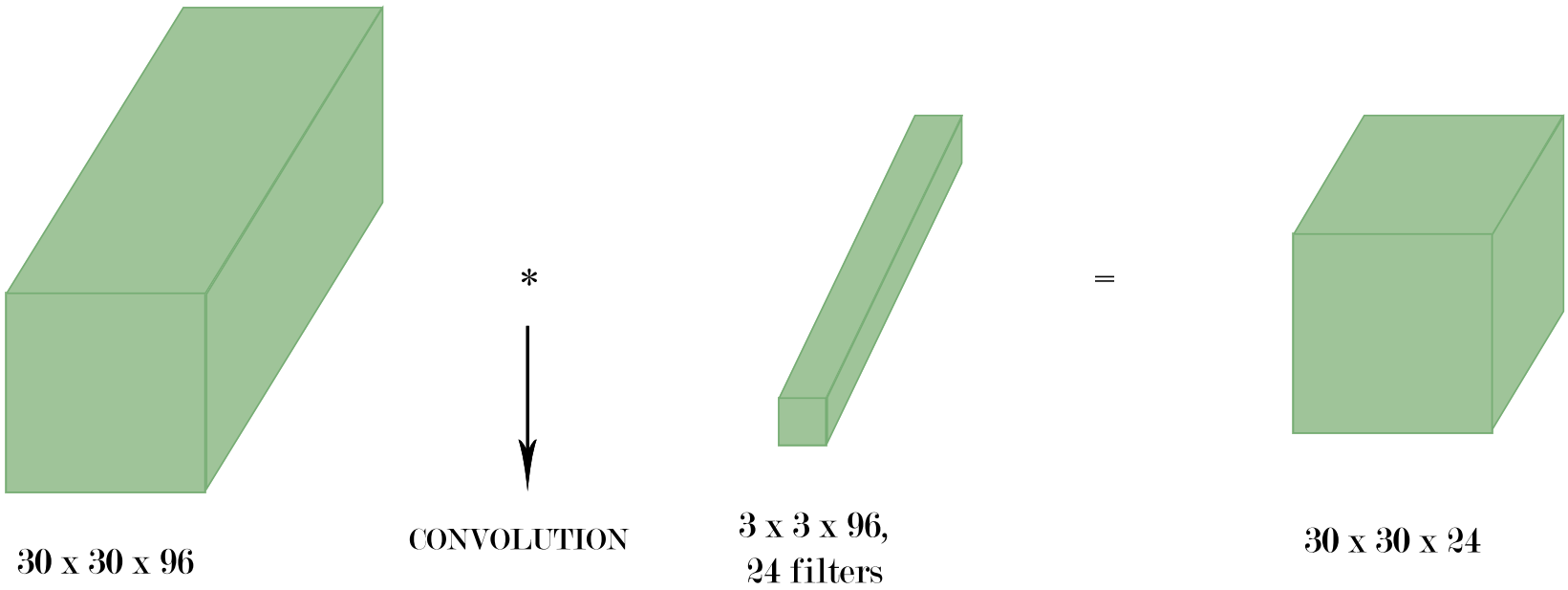


Figure: Example of a convolution performed in a convolutional layer.

Convolutional Layer

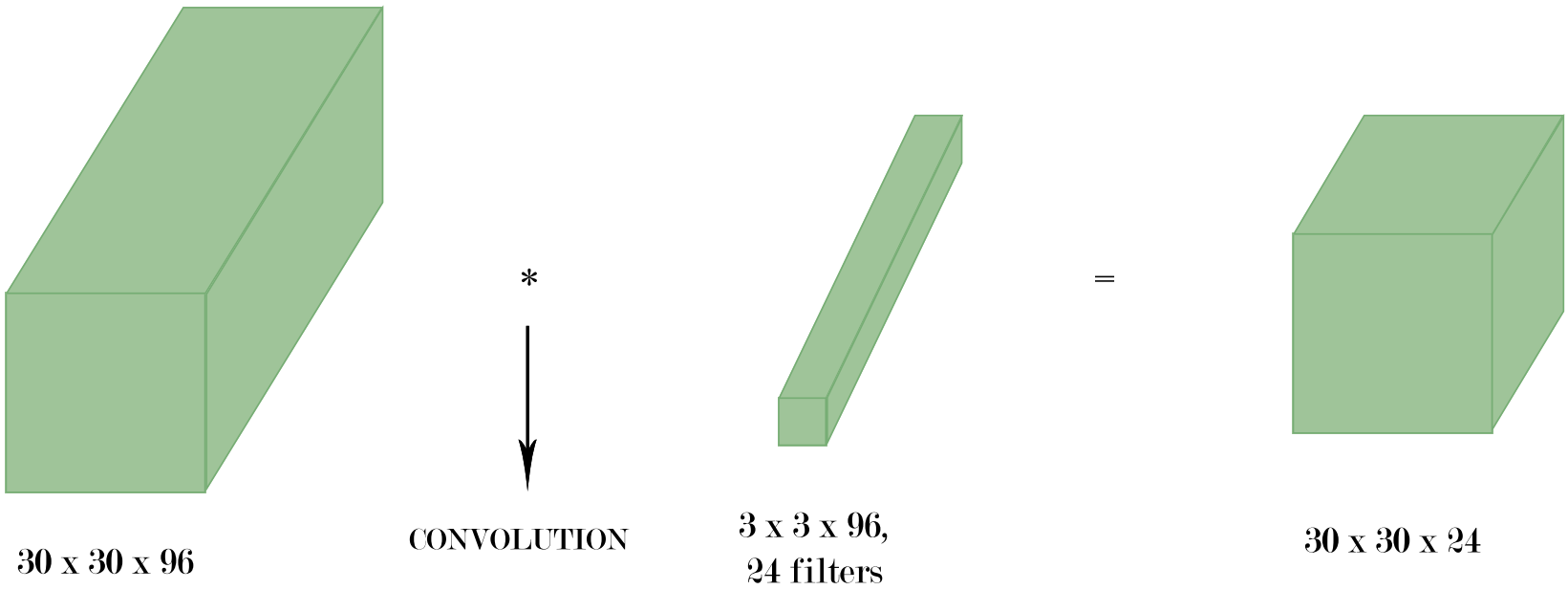


Figure: Example of a convolution performed in a convolutional layer.

Nonlinear activations

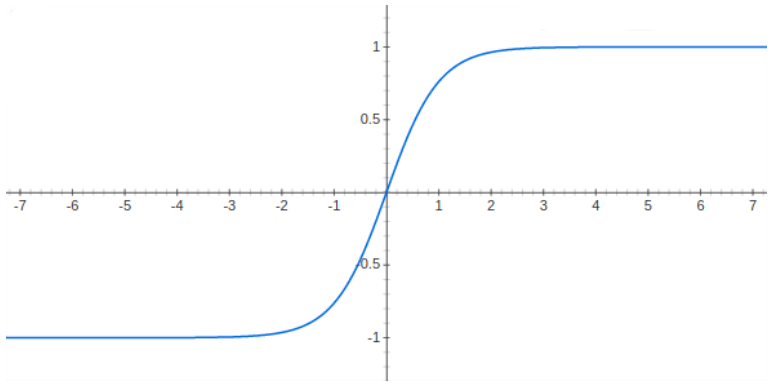


Figure: $\tanh(x)$ function.

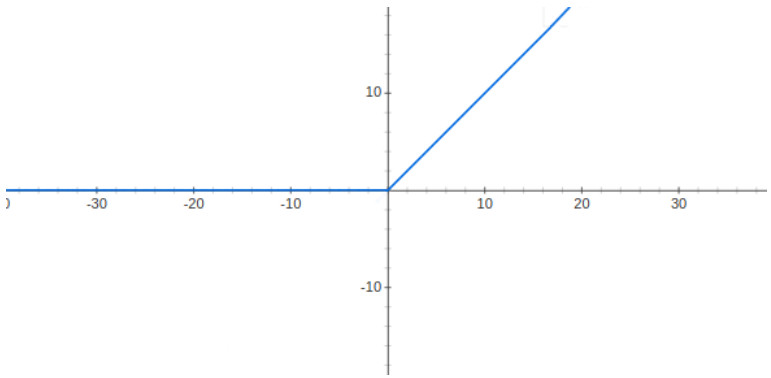


Figure: $ReLU(x)$ function.

ReLU Activation

- $\mathrm{ReLU}(x) = \max(0, x)$.
- Example:

$$I = \begin{bmatrix} 3 & 3 & 1 & -3 \\ -3 & 3 & -9 & 8 \\ -17 & 13 & 15 & -15 \\ 1 & -13 & -8 & 9 \end{bmatrix}.$$

$$\mathrm{ReLU}(I) = \begin{bmatrix} 3 & 3 & 1 & 0 \\ 0 & 3 & 0 & 8 \\ 0 & 13 & 15 & 0 \\ 1 & 0 & 0 & 9 \end{bmatrix}.$$

Pooling

- Replaces the input at a certain location with a summary statistic of its neighbourhood: max pooling, average pooling, L_2 norm...
- It makes the output approximately invariant to small changes in the input.
- It reduces the size of the input (it usually uses a **stride**).
- Example using $2x2$ neighbourhood and stride 2.

$$I = \begin{bmatrix} 3 & 3 & 1 & 0 \\ 0 & 3 & 0 & 8 \\ 0 & 13 & 15 & 0 \\ 1 & 0 & 0 & 9 \end{bmatrix} .$$

$$\text{max pooling}(I) = \begin{bmatrix} 3 & 8 \\ 13 & 15 \end{bmatrix} , \text{ avg pooling}(I) = \begin{bmatrix} 2.25 & 2.25 \\ 3.5 & 6 \end{bmatrix}$$

Batch Normalization

- In a gradient-based algorithm, the gradient tells how to update each parameter, under the assumption that the other layers do not change.
- In practice, all the layers are updated simultaneously. This can cause unexpected results, specially as the depth of the network increases (vanishing/exploding gradients).
- Batch Normalization provides a way of reparametrizing the network and reduces this problem.

Batch Normalization

- First, it normalizes each feature map to have zero mean and a standard deviation of 1. Let B be a minibatch of inputs, μ_B its mean and σ_B its standard deviation. Then, it normalizes all the values in each feature map:

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \delta}} \ ,$$

  where δ is a small positive value to avoid the square root being zero.

- Then, it scales (with a parameter γ) and shifts (with a parameter β) the normalized values:

$$y_i = \gamma \widehat{x_i} + \beta \ .$$

Local Response Normalization (LRN)

- Used to normalize ReLU outputs.
- It is defined as [2]:

$$b_i = a_i \left(k + \alpha \sum_{j=j_{low}}^{j_{high}} a_j^2 \right)^{-\beta},$$

  with $j_{high} = \min(i + r/2, f_n - 1)$ and $j_{low} = \max(0, i - r/2)$.
- b_i is the normalized output of the neuron a_i and f_n is the number of feature maps.
- k, α, β and r are hyperparameters. r is called the *depth radius* and k the *bias*.

Outline

Outline

LeNet-5 (1998)

- Created by Yann LeCun in 1998 [3].

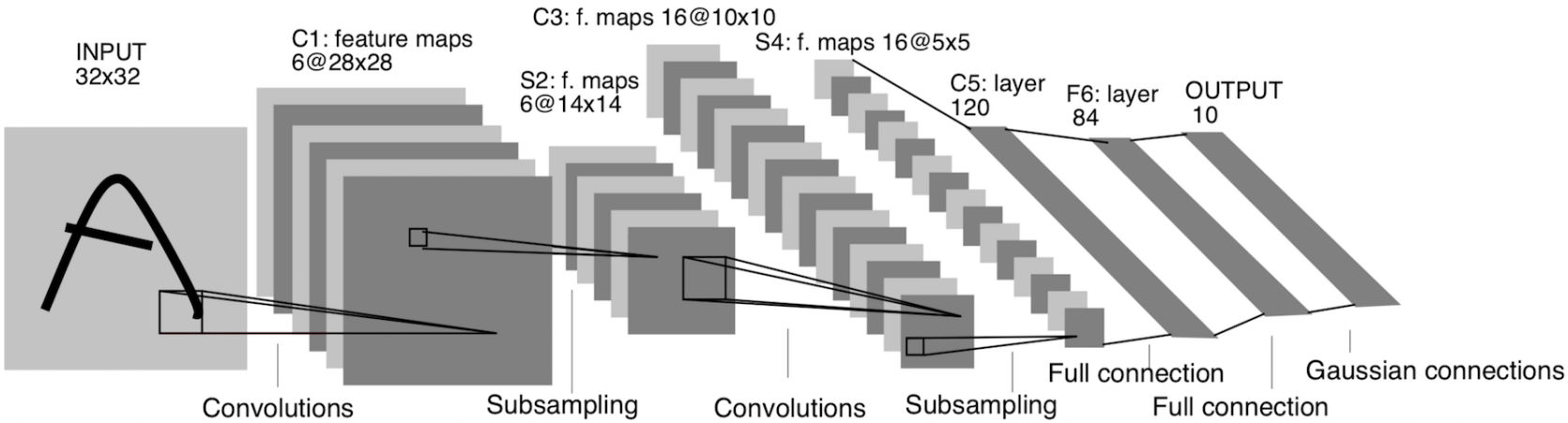


Figure: LeNet5. Figure from [3].

LeNet-5 (1998)

Type	#Filters	Kernel size (stride)	Activation
Convolution	6	5×5 (1)	tanh
Pooling (avg)		2×2 (2)	tanh
Convolution	16	5×5 (1)	tanh
Pooling (avg)		2×2 (2)	tanh
Convolution	120	5×5 (1)	tanh
Fully Connected		84	tanh
Fully Connected		10	RBF

Table: LeNet-5 architecture.

Outline

AlexNet (2012)

- Developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012 [4].

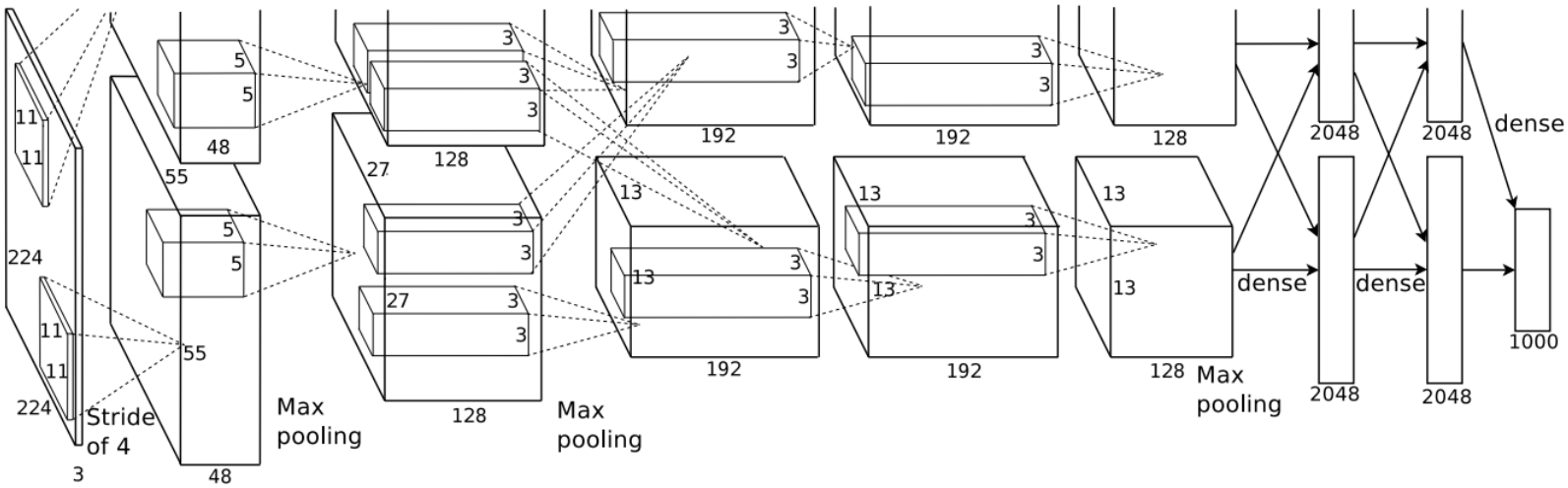


Figure: AlexNet. Figure from [4].

AlexNet (2012)

Type	#Filters	Kernel size (stride)	Activation
Convolution	96	11×11 (4)	ReLU
Pooling (max)		3×3 (2)	
LRN			
Convolution	256	5×5 (1)	ReLU
LRN			
Pooling (max)		3×3 (2)	
Convolution	384	3×3 (1)	ReLU
Convolution	384	3×3 (1)	ReLU
Convolution	256	3×3 (1)	ReLU
Pooling (max)		3×3 (2)	
Fully Connected		4096	ReLU
Fully Connected		4096	ReLU
Fully Connected		1000	Softmax

Table: AlexNet architecture.

Outline

VGGNet (2014)

- Created by Karen Simonyan and Andrew Zisserman [5].

Type	#Filters	Kernel size (stride)	Activation	Repetitions
Convolution	64	3×3 (1)	ReLU	2
Pooling (max)		2×2 (2)		
Convolution	128	3×3 (1)	ReLU	2
Pooling (max)		2×2 (2)		
Convolution	256	3×3 (1)	ReLU	3
Pooling (max)		2×2 (2)		
Convolution	512	3×3 (1)	ReLU	3
Pooling (max)		2×2 (2)		
Convolution	512	3×3 (1)	ReLU	3
Pooling (max)		2×2 (2)		
Fully Connected		4096	ReLU	
Fully Connected		4096	ReLU	
Fully Connected		1000	Softmax	

Table: VGGNet-16 architecture.

Outline

GoogLeNet (2014)

- Developed by Christian Szegedy et al. (Google Research) [6].

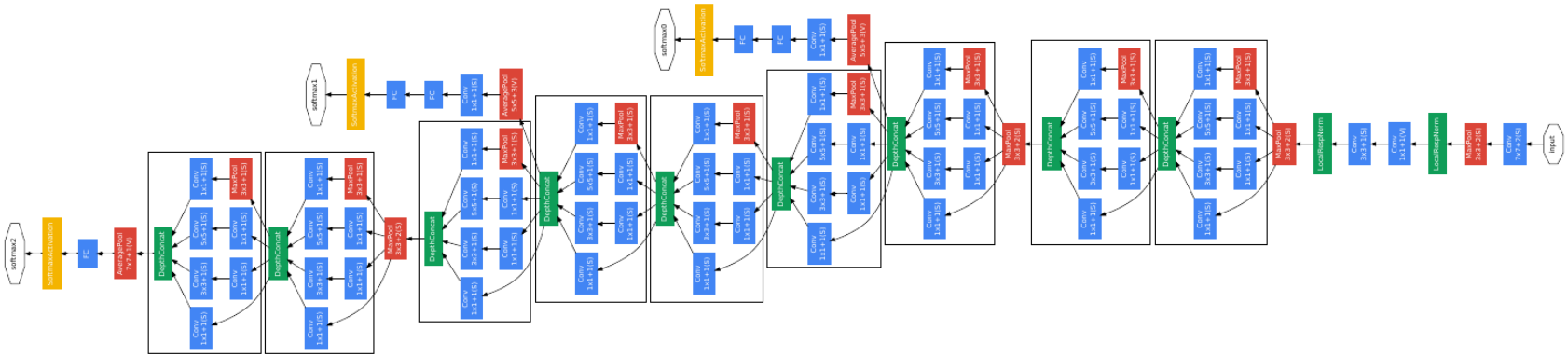


Figure: GoogLeNet. Original figure from [6].

GoogLeNet (2014)

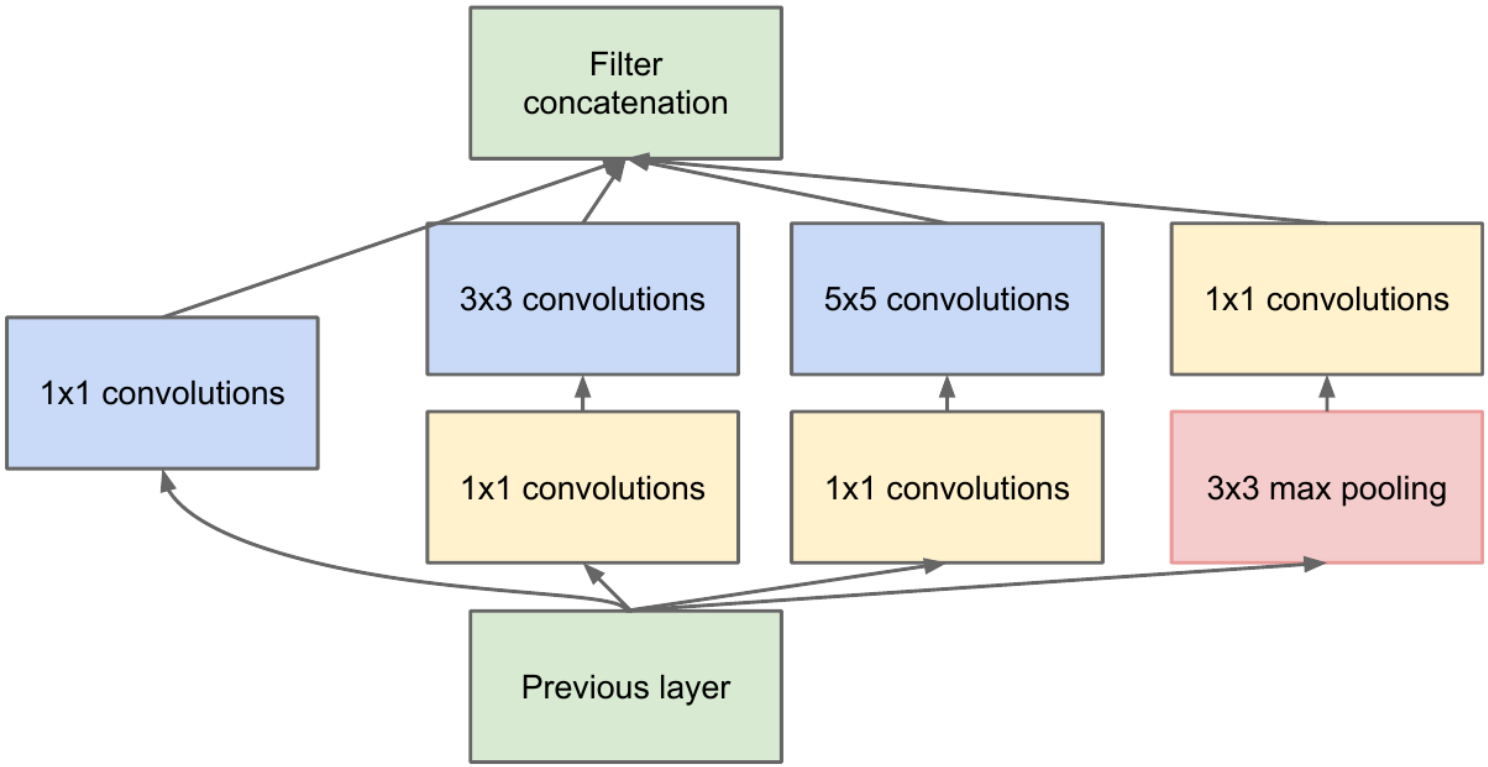


Figure: Inception module. Figure from [6].

Outline

Inception v3 (2015)

- Developed by Christian Szegedy et al. (Google Research) [7].
- It is a modification of GoogLeNet.

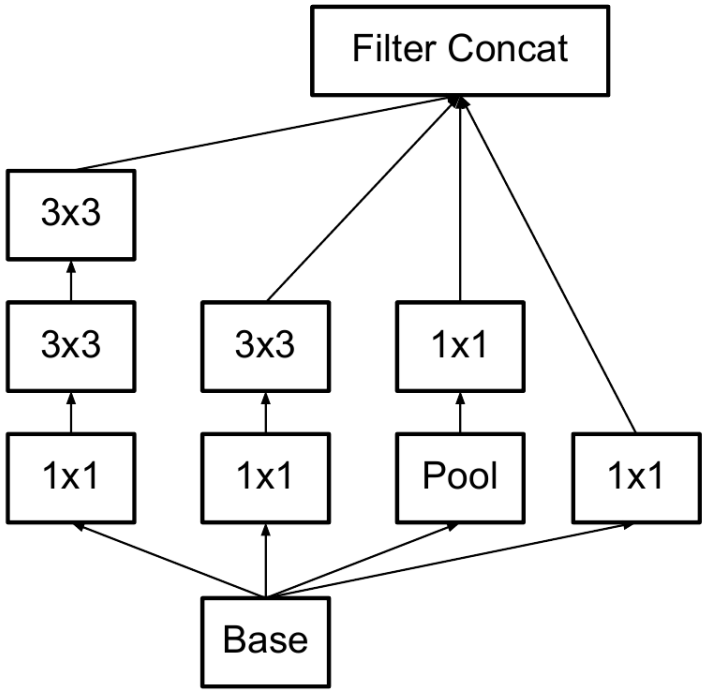


Figure: Inception v3 base module. Figure from [7].

Inception v3 (2015)

- Developed by Christian Szegedy et al. (Google Research) [7].
- It is a modification of GoogLeNet.

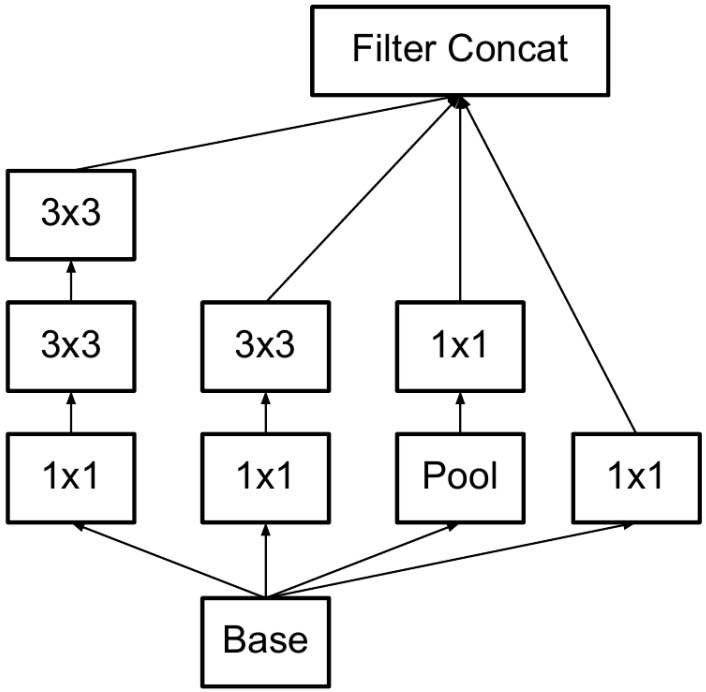


Figure: Inception v3 base module. Figure from [7].

Inception v3 (2015)

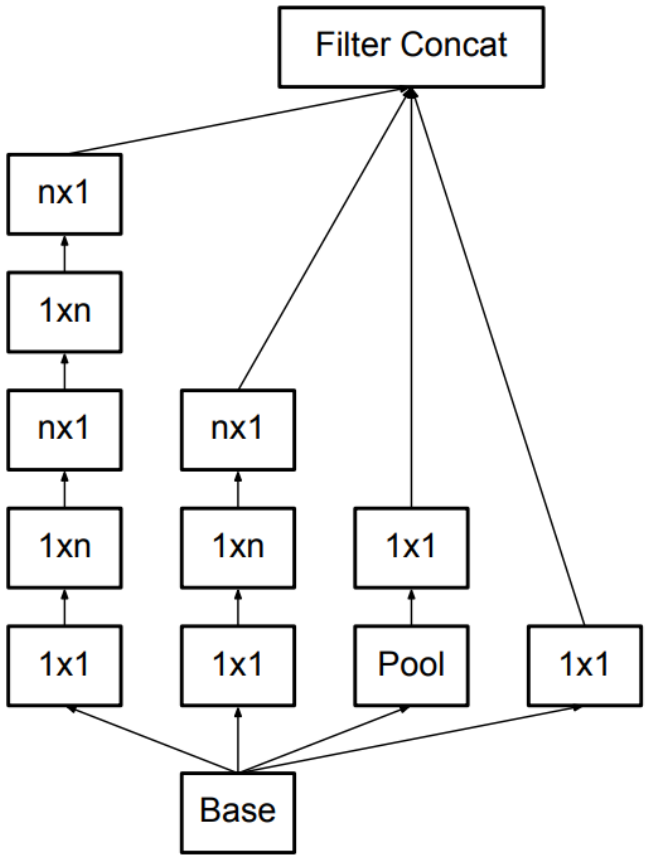


Figure: Second Inception Module. Figure from [7].

Inception v3 (2015)

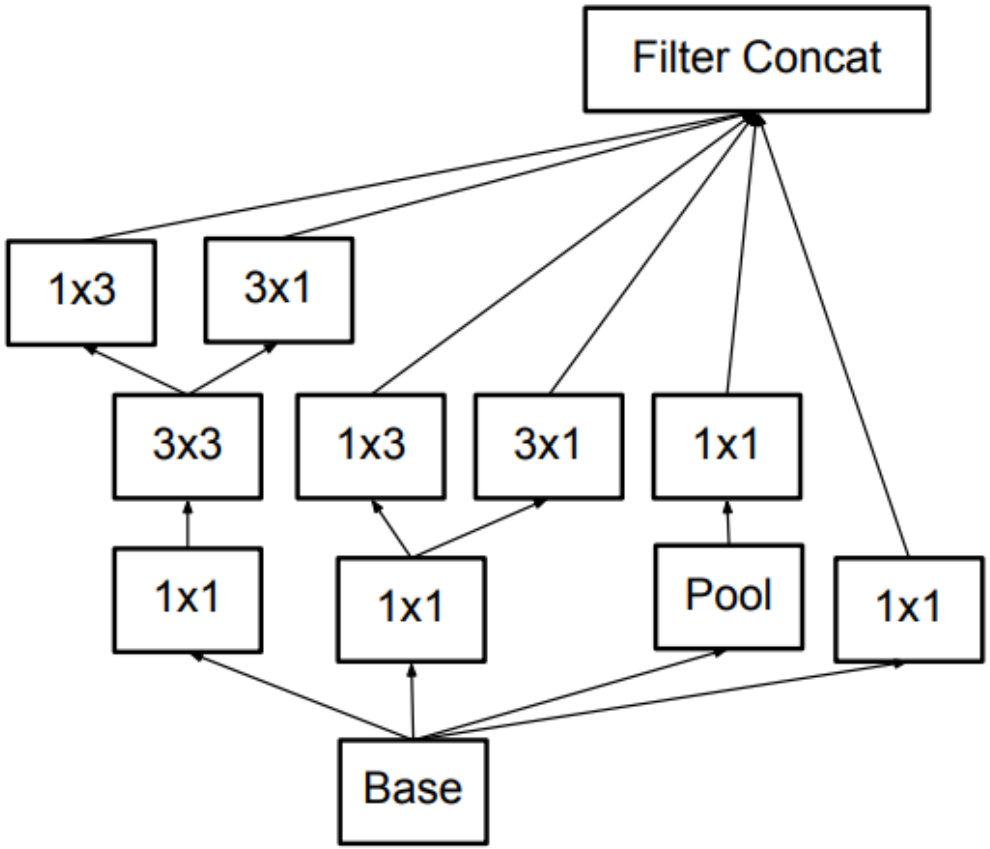


Figure: Third Inception Module. Figure from [7].

Outline

ResNet (2015)

- Created by Kaiming He et al. [8].
- Shortcut connection: $H(x) = F(x) + x$.

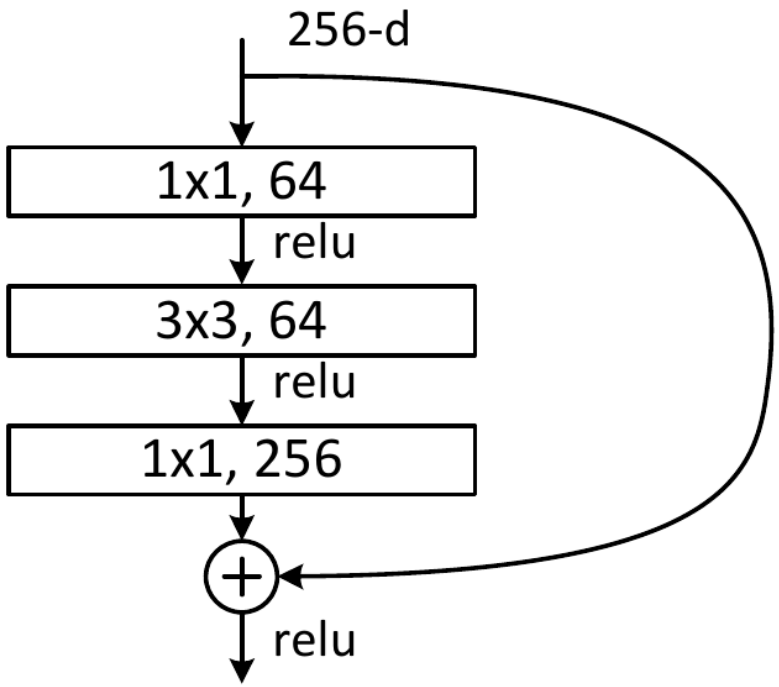


Figure: ResNet module. Figure from [8].

ResNet (2015)

- Created by Kaiming He et al. [8].
- Shortcut connection: $H(x) = F(x) + x$.

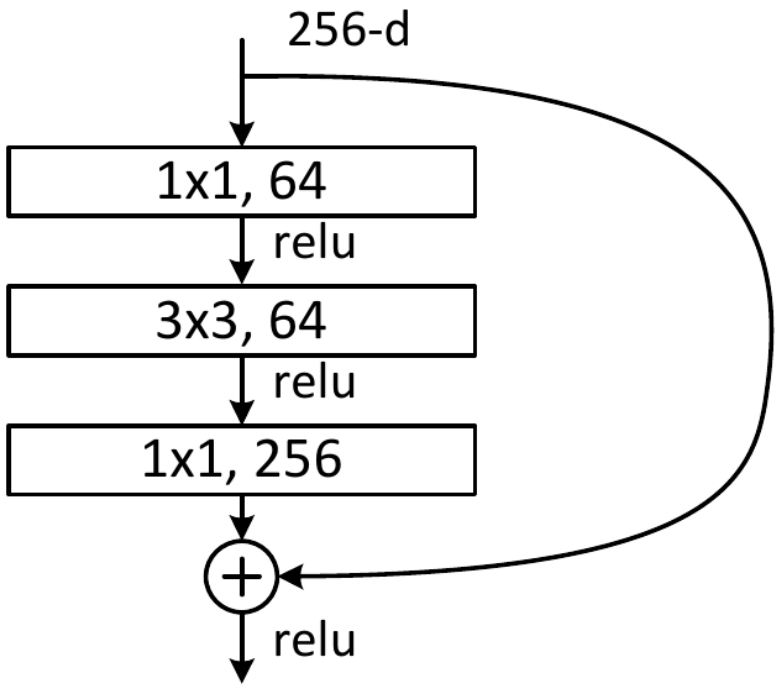


Figure: ResNet module. Figure from [8].

ResNet (2015)

18-layer	34-layer	50-layer	101-layer	152-layer
7×7, 64, stride 2				
3×3 max pool, stride 2				
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
average pool, 1000-d fc, softmax				
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure: ResNet architectures. Figure from [8].

Outline

DenseNet (2016)

- Developed by Gao Huang et al. [9].

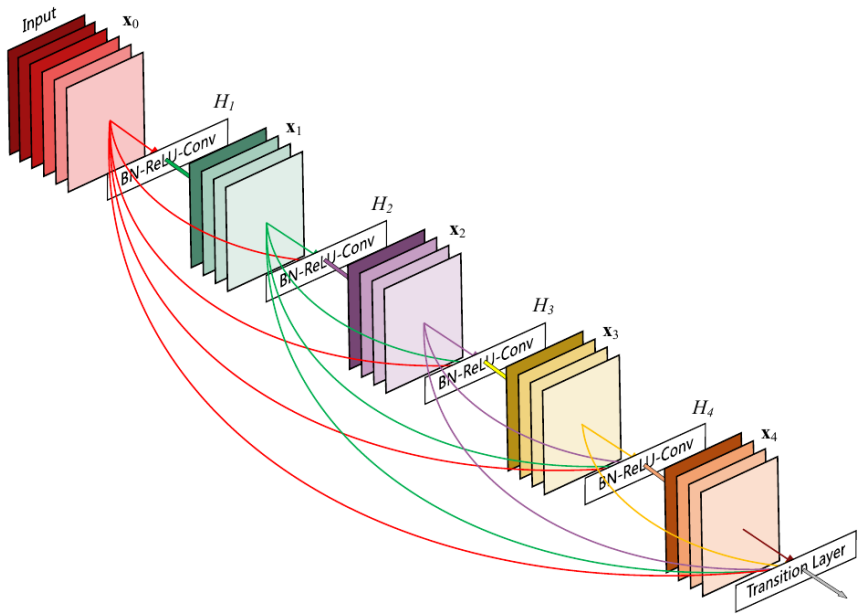


Figure: Example of a DeseNet Module. Figure from [9].

DenseNet (2016)

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure: DenseNet architecture. Figure from [9].

Summary

- CNNs have become increasingly deeper.
- The size of the kernels has become smaller.
- The number of kernels in each convolutional layer have increased.
- The use of ReLU and Batch Normalization has been standardized with respect to $\tanh$ and Local Response Normalization, respectively.

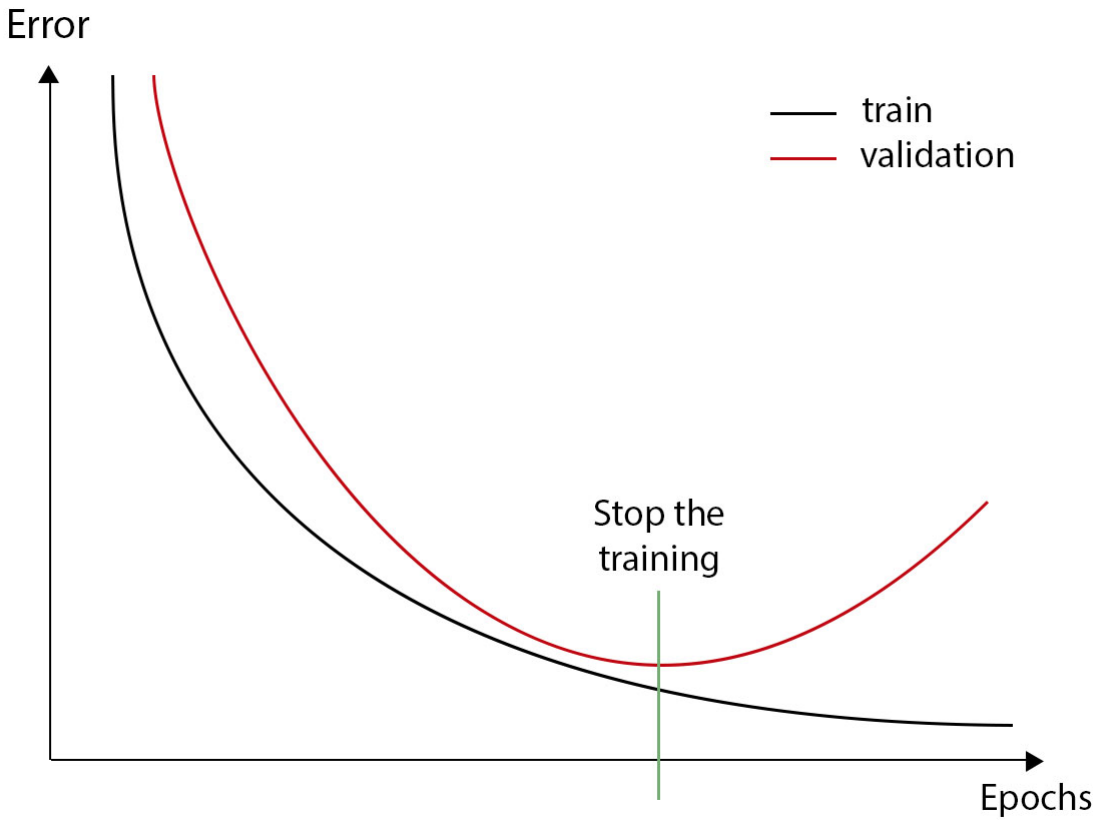
Outline

Regularization Techniques

- They are strategies used to reduce the test error (usually increasing the training error).
- Some of the most used strategies are:
  1. **Parameter norm penalties**. They consist of adding a parameter norm penality to the objective function, to minimize the parameter penalty along with the original objective function. L_2 and L_1 parameter regularization are two of the most used.

Regularization techniques

- Some of the most used strategies are:
  2. **Early stopping**: It consists of returning the parameters obtained when the validation error was minimal.



Regularization Techniques

- Some of the most used strategies are:
  3. **Ensemble methods**: They consist of combining the output of several networks to obtain the final results.
  4. **Dropout**: It consists of randomly cancelling a percentage of the inputs of a certain layer, so they do not have effect. At each step of the training procedure, the inputs that are being cancelled are chosen randomly again.
  5. **Data augmentation**: It consists of adding false training data to the training set.

Outline

Data Augmentation

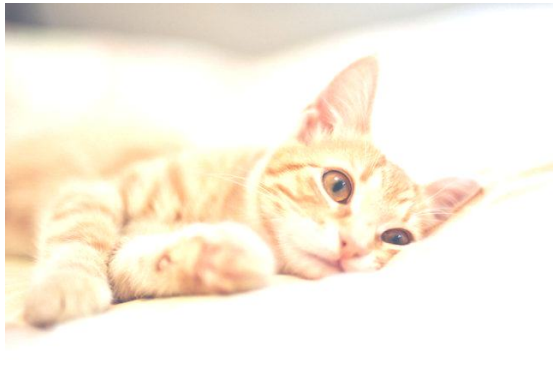
- Data augmentation is a set of techniques that perform some transformations to the original images in order to create slightly different new ones. These new images will be added to the training set, helping the network to generalize better on new data.
- There are several operations that we can perform to the original images to obtain new images. In addition, they can be used alone or in combination.

Data Augmentation Techniques

- Some of the most used transformations are:
  - To shift the image (vertically and/or horizontally): to crop the image (getting rid of parts of the image).
  - To zoom the image.
  - To rotate the image by a specific angle.
  - To flip the image (vertically and/or horizontally): to mirror the image.
  - To change the brightness of the image: to lighten or darken the image, simulating that the images were taken under different lightning conditions.

Data Augmentation Techniques: Examples

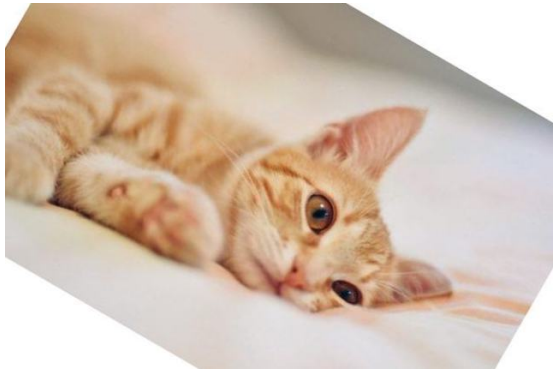
(a) Original



(b) Brightness

(c) Zoom

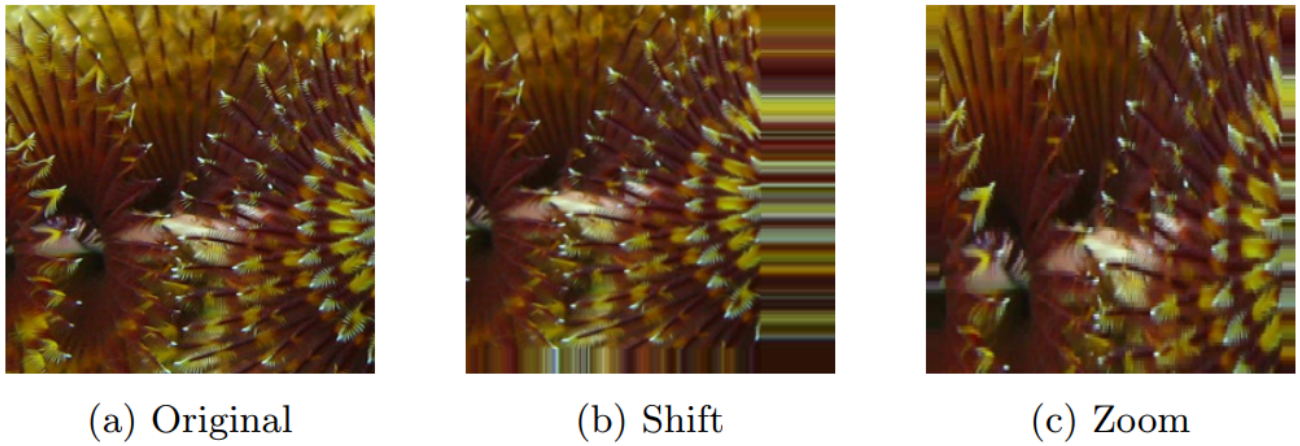
(d) Shift



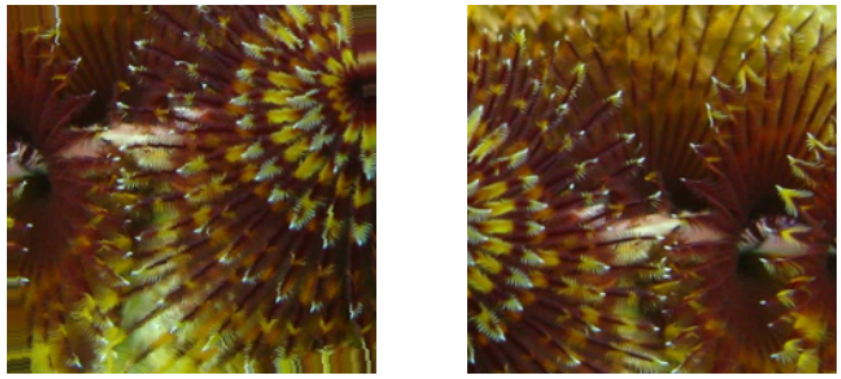
(e) Rotation

(f) Horizontal flip

Data Augmentation Techniques: Examples



(a) Original (b) Shift (c) Zoom



(d) Rotation (e) Flip

Figure: Examples of data augmentation techniques on coral images.

Data Augmentation

- Using these data augmentation techniques, we are making the network invariant to the transformation used. If we use rotation as data augmentation during training, we make the network more invariant to rotation, since the network is used to classify rotated images.
- In this sense, the use of data augmentation techniques makes the models more generalizable to new data.
- However, we need to be careful with:
  1. The specific techniques we are using, depending on the images in our data set, and
  2. The amount of change we are introducing.

Data Augmentation: Warnings

- If we apply transformations to the original images that completely change the original images, we end up with a set of new images that do not represent the problem we wanted to resolve.
- The transformed images need to be realistic enough so that, ideally, an external observer could not distinguish between an original image and an image generated by data augmentation.

Data Augmentation: Warnings

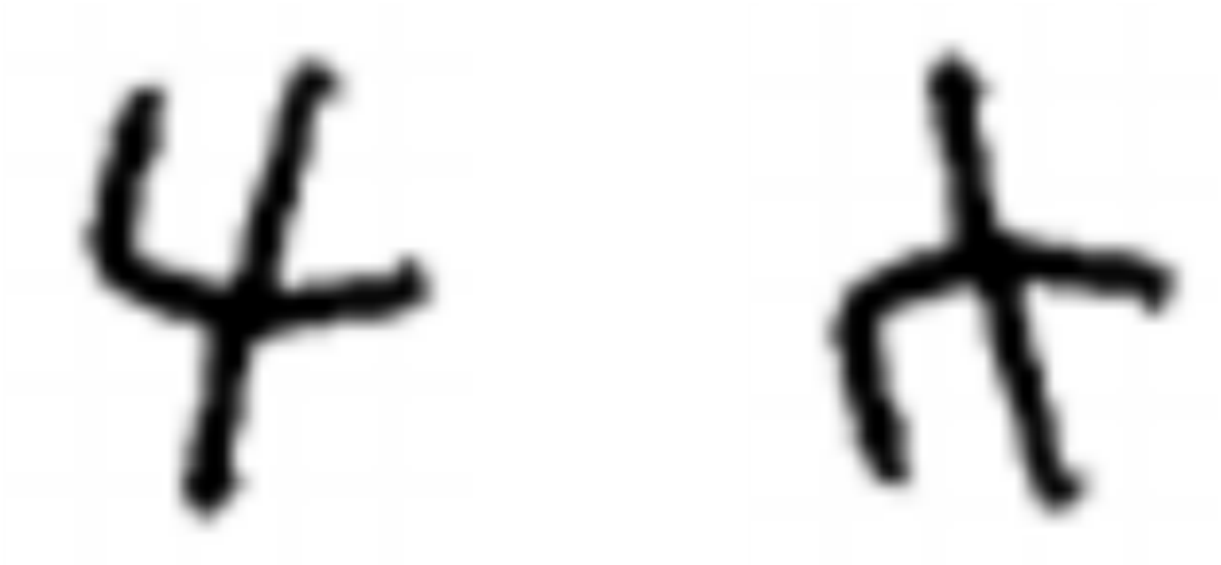
- In general, the new images have to be similar to images that could have been obtained by the same methods as the original ones.
- For example, some of the following problems could happen:
  - To change the class of an image with a specific technique or a combination of techniques.
  - To change (or remove) the meaning of an image.
  - To use so many zoom or shifting (for example) that we loss the main object in the image.

Data Augmentation: Warnings

(a) Original image (b) Vertical flip (c) Vertical and horizontal flip

Figure: Changing the class on an image with a combination of two techniques.

Data Augmentation: Warnings



(a) Original image (b) Vertical flip

Figure: Removing the meaning of an image.

Data Augmentation: Warnings

(a) Original image (b) Zoom

Figure: Loosing the main object in the image.

Data Augmentation at Training Time

All the data augmentation techniques are usually performed at training time. This implies some advantages:

- The images generated by data augmentation do not need to be stored: they are generated, used and discarded.
- The techniques can be done randomly. That is, we can choose the highest degree for, for example, a rotation, x, and at training time a random number from 0 to x will be chosen for each image. Then, each image will be rotated by a different number, obtaining more images.

Data Augmentation at Training Time

All the data augmentation techniques are usually performed at training time. This implies some advantages:

- The images generated by data augmentation do not need to be stored: they are generated, used and discarded.
- The techniques can be done randomly. That is, we can choose the highest degree for, for example, a rotation, x, and at training time a random number from 0 to x will be chosen for each image. Then, each image will be rotated by a different number, obtaining more images.

Data Augmentation

Among the data augmentation techniques that are appropriate for our specific problem, we should:

- Make tests for each of them and find which one, or which combination, works best for our problem.
- Experiment with different hyperparameters for each of them and find the best ones.
- Be careful: there could be some data augmentation techniques that damage the learning, although they are appropriate in terms of non changing the class of the images or non losing their meaning.

Outline

Transfer Learning

- Deep neural networks have, usually, million of parameters. In order to train all of them from scratch, we need big data sets and lots of time.
- When not enough images are available, we can use another network already pre-trained in another different (but similar) problem, and start the training from there. This is called transfer learning.
- This technique also reduces the training time, since the weights are not starting randomly.
- In fact, this technique has been proved to work so well that is common to use it even when the data sets are large enough to train the networks from scratch.

Transfer Learning

- Transfer learning allows to transfer the knowledge learned from one task (classifying a data set D_1) to a different, but related, task (classifying a data set D_2).
- It is important that the two tasks are related. If the images in the classes in D_1 and D_2 share low-level notions of edges and visual shapes, geometric changes, changes in lightning, etc., some of the layers learned with D_1 will help to distinguish between D_2 classes.

Transfer Learning

- That way, the layers will need far fewer changes in their weights than if we had started the training from scratch.
- This implies not only that the training will require less time, but that we will need less images to obtain good weights for our data set D_2.
- The data set D_1 used to pre-train the network, however, has to be large enough to train well enough all the weights in the network.

Transfer Learning

- It is common to use ImageNet to pre-train the networks, as it has more than one million images and 1000 classes of general domain. That is, we have lots of common classes (from breeds of dogs to fruits, furnitures...) that are very likely to relate in some way with a new data set.
- Thanks to the ILSVRC competition, the most known CNN architectures are already trained to classify ImageNet, so we do not have to do this step (which will require days of training).

Transfer Learning

- Since the CNNs are trained to classify the 1000 classes of ImageNet, we need to remove, at least, the last fully connected layer with a softmax activation, which classifies the images into ImageNet classes. In its place, we need to add, at least, a new fully connected layer with a softmax activation but with as many neurons as classes has the new data set D_2 we want to classify.
- In fact, we can remove and add as many layers as we want, but remember that the new layers will have to be trained from scratch.

Transfer Learning: Retraining the network

- At this point, we have two options: we can train only the layers that we have added to the network, or we can retrain the whole network (or parts of it). This second option is called **fine-tuning**.
- This decision is made taking into account the number of images we have in the training set:
  - If it is small, is probably better to just train the new layers.
  - If it is large, is better to first train the new layers, and then try to retrain different parts of the network, even retraining the whole network.

Transfer Learning and Fine-Tuning

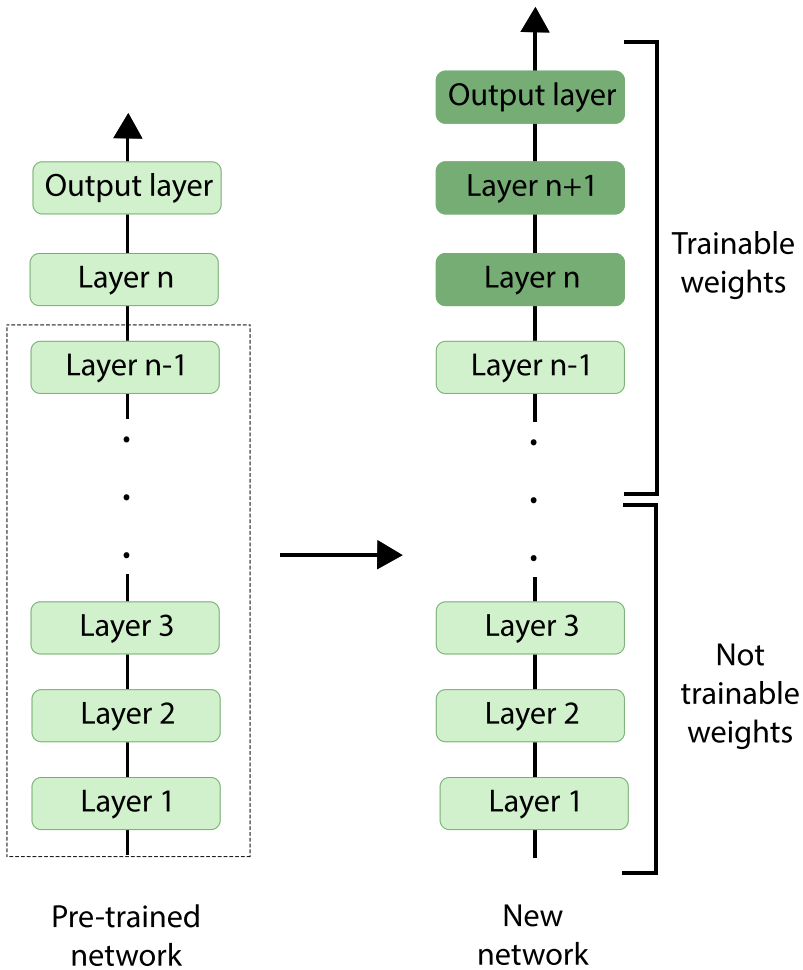


Figure: Transfer Learning and Fine Tuning.

Transfer Learning: Retraining the network

- In general, the smaller the part of the network you retrain, the worse it fits into the new data set.
- It is important to obtain a good tradeoff between the size of the new data set and the part of the network you are retraining:
  - If the data set is too small and we try to retrain a large part of the network, it is likely that we ruin the weights.
  - If the data set is quite large, it is likely that we obtain better results if we retrain larger parts of the network.

Bibliography I

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc, 1 ed., 3 2017.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

Bibliography II

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Bibliography III

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[9] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, p. 3, 2017.

Thank you!
Questions?

Anabel Gómez Ríos

anabelgrios@decsai.ugr.es

University of Granada

22nd of April, 2021