Convolutional Neural Networks Ingredients

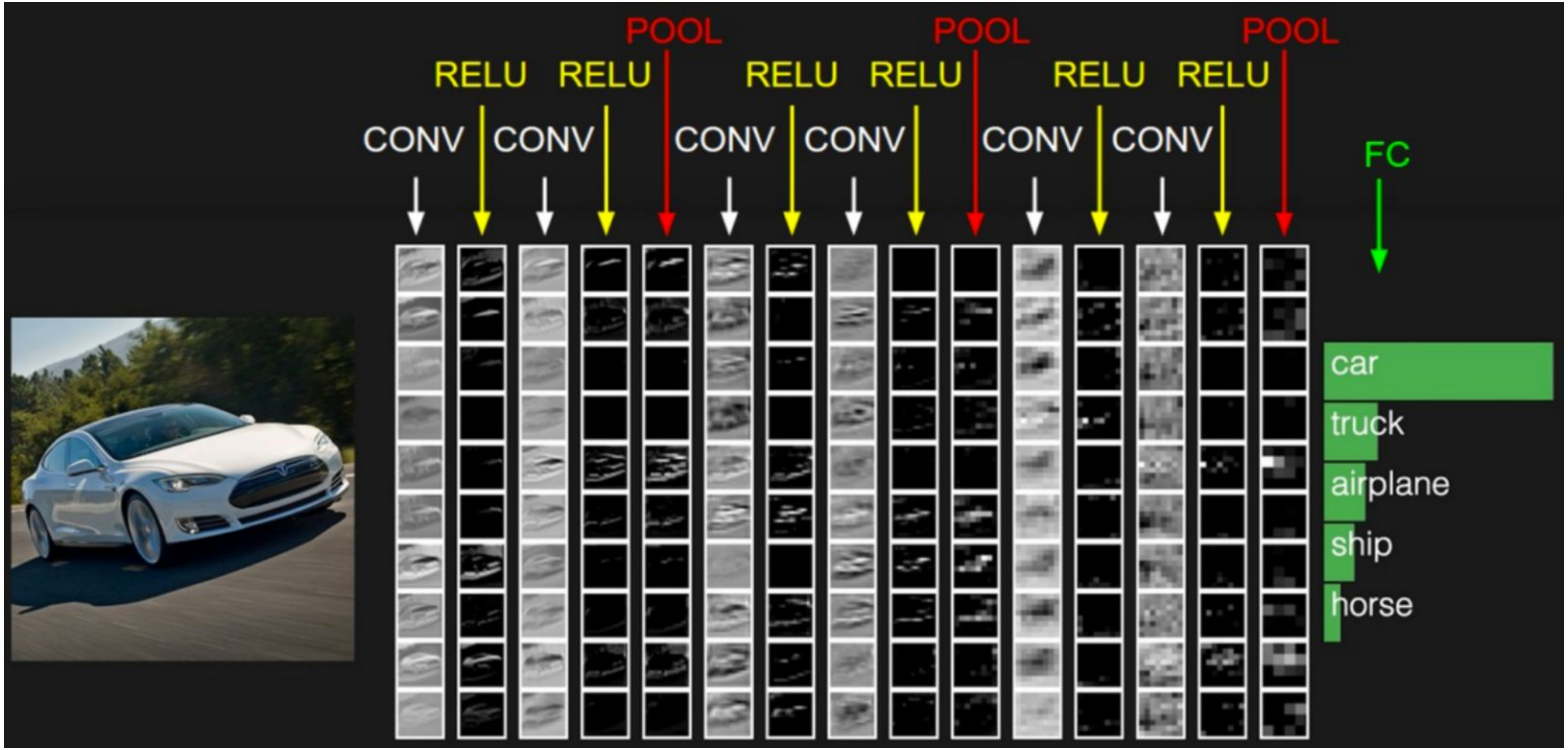
A Visual Guide

Natalia Díaz Rodríguez

ROB313: **Perception pour les Systèmes Autonomes**

http://perso.ensta-paristech.fr/~manzaner/Cours/ROB313/

Image classification
Objective: mapping an image to a score

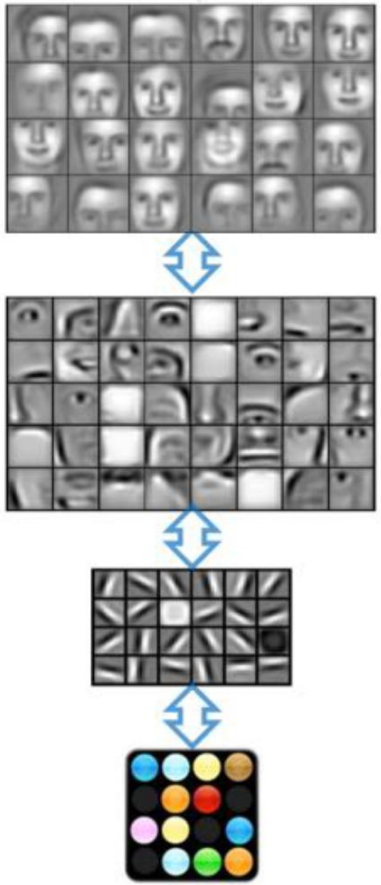


http://cs231n.github.io/convolutional-networks/

ConvNets main operators

- Convolution layers
- Non Linearity (Rectifying Linear Unit: ReLU)
- Pooling (or sub-sampling) layers
- Classification (Fully Connected: FC)

linear layer



A general architecture: *LeNet* (LeCun 1998)

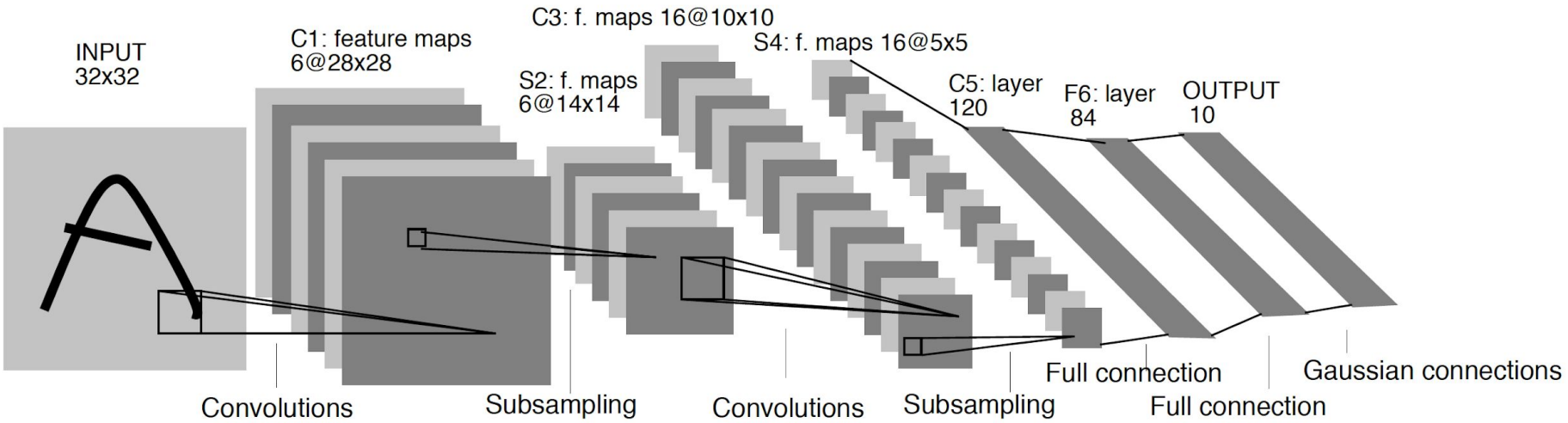


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolution kernel: Visualization and Effect

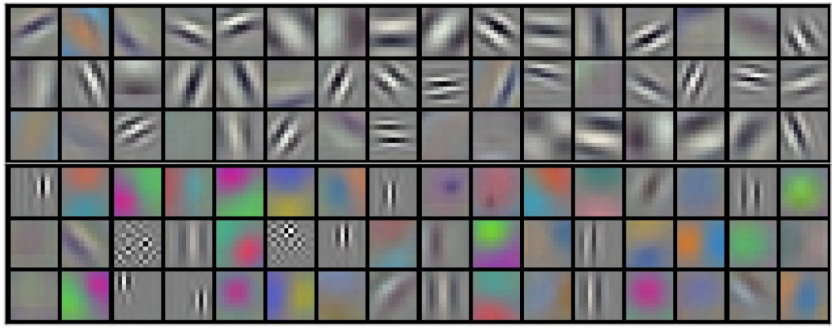
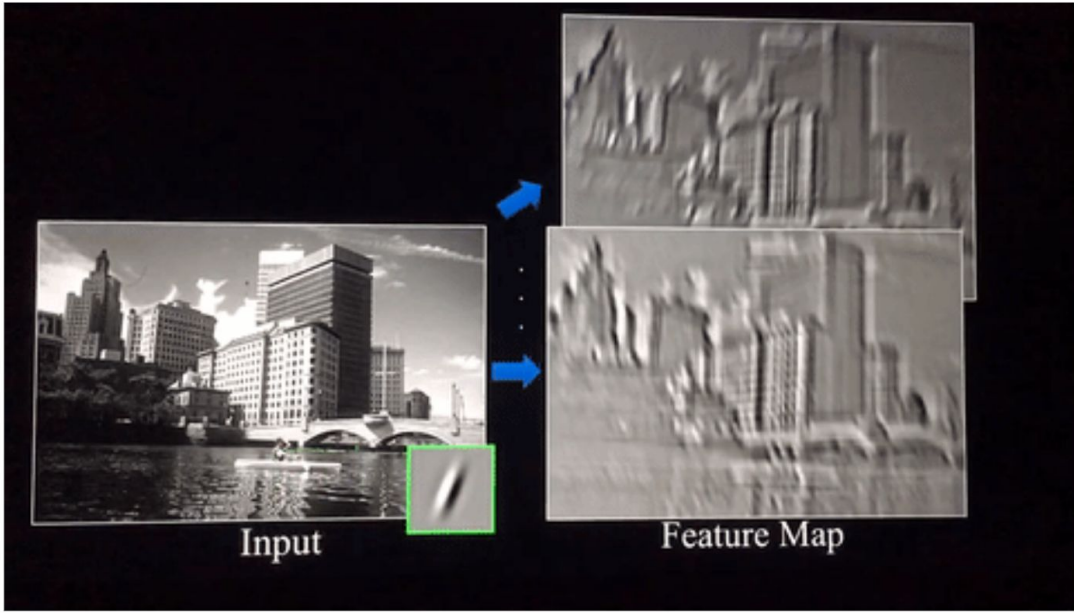
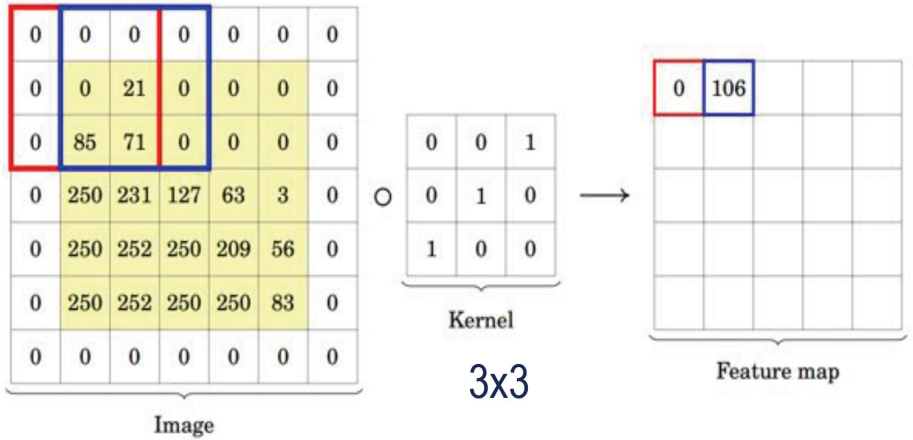


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU



[AlexNet 2012]

Architecture Ingredients: Convolution Layer



Convolution Formulas:

The formulas relating the output shape (C_2, H_2, W_2) of the convolution to the input shape (C_1, H_1, W_1) is:

$$H_2 = \lfloor \frac{H_1 - kernel\_size + 2 \times padding}{stride} \rfloor + 1$$

$$W_2 = \lfloor \frac{W_1 - kernel\_size + 2 \times padding}{stride} \rfloor + 1$$

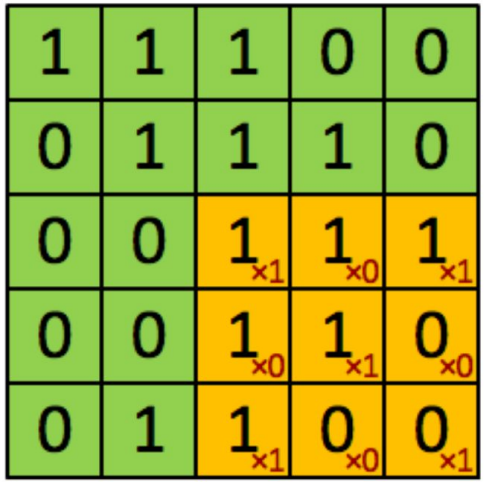
$$C_2 = \text{number of filters used in the convolution}$$

NOTE: $C_2 = C_1$ in the case of max pooling

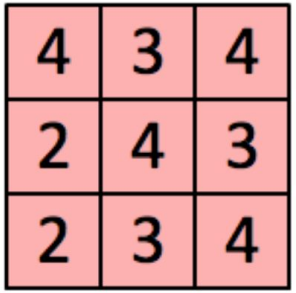
where:

- H_2: height of the output volume
- W_2: width of the output volume
- C_1: in_channels, number of channels in the input volume
- C_2: out_channels

Architecture Ingredients: Convolution Layer



Image



Convolved Feature

(result feature map)

Convolutional Kernel:

1 0 1
0 1 0
1 0 1

Convolution Formulas:

The formulas relating the output shape (C_2, H_2, W_2) of the convolution to the input shape (C_1, H_1, W_1) is:

$$H_2 = \lfloor \frac{H_1 - kernel\_size + 2 \times padding}{stride} \rfloor + 1$$

$$W_2 = \lfloor \frac{W_1 - kernel\_size + 2 \times padding}{stride} \rfloor + 1$$

$C_2 =$ number of filters used in the convolution

NOTE: $C_2 = C_1$ in the case of max pooling

where:

- H_2: height of the output volume
- W_2: width of the output volume
- C_1: in_channels, number of channels in the input volume
- C_2: out_channels

Architecture Ingredients: Convolution Layer

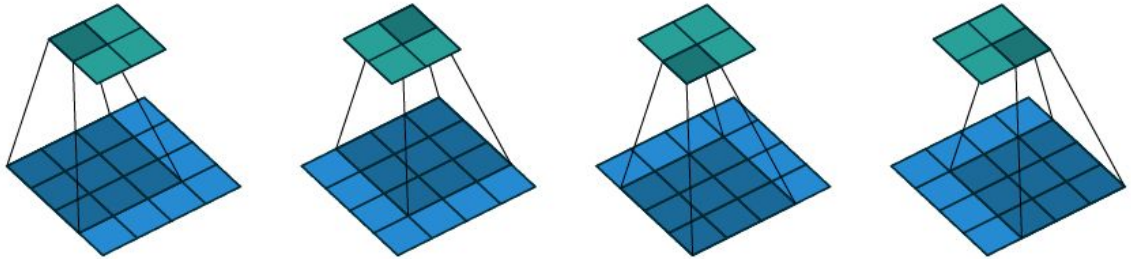


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

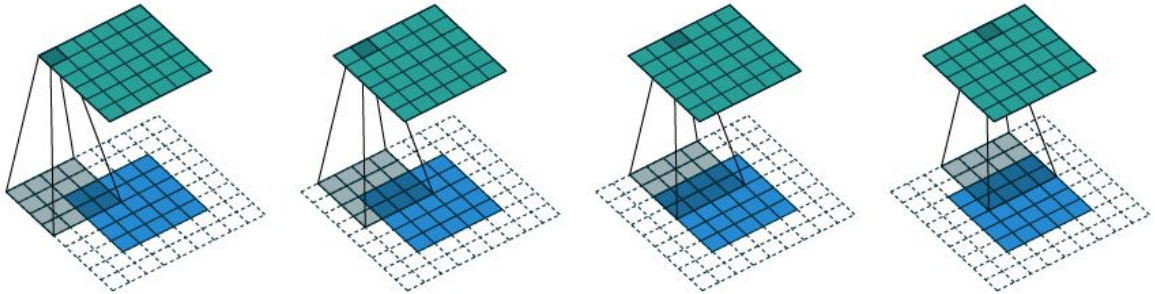


Figure 2.2: (Arbitrary padding, unit strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

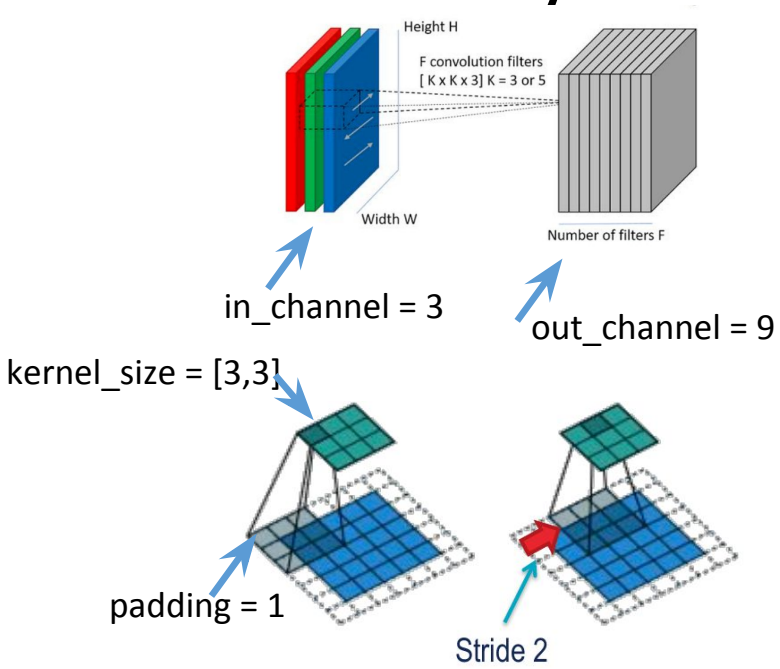
https://github.com/vdumoulin/conv_arithmetic

Architecture Ingredients: Convolution Layer

PyTorch class:

***Class* `torch.nn.`Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True*)**

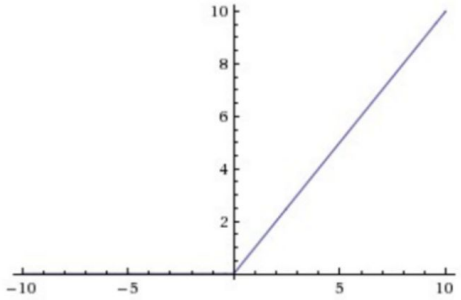
- **in_channels**: Nr of channels in the input image (if input layer = dimension of image colormap. E.g.: grey scale = 1; RGB = 3 channels)
- **out_channels**: Nr of channels produced by the convolution. If no grouping = number of kernels
- **kernel_size** = Size of the convolving kernel (here 3x3)
- **stride** of the convolution: Sliding window step size for the cross-correlation
- **padding** = amount of implicit zero-paddings on both sides for padding number of points for each dimension

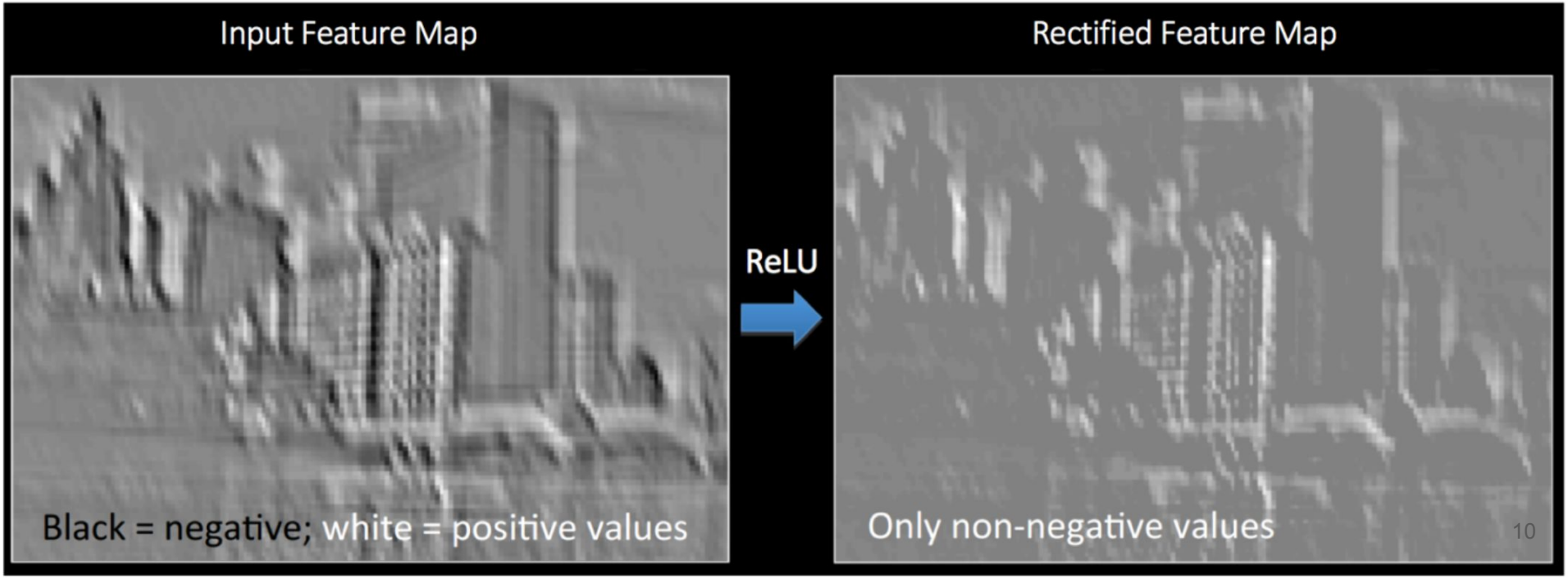


https://en.wikipedia.org/wiki/Cross-correlation

Effect: Non-linear (ReLU) activation Layer

Output = Max(zero, Input)





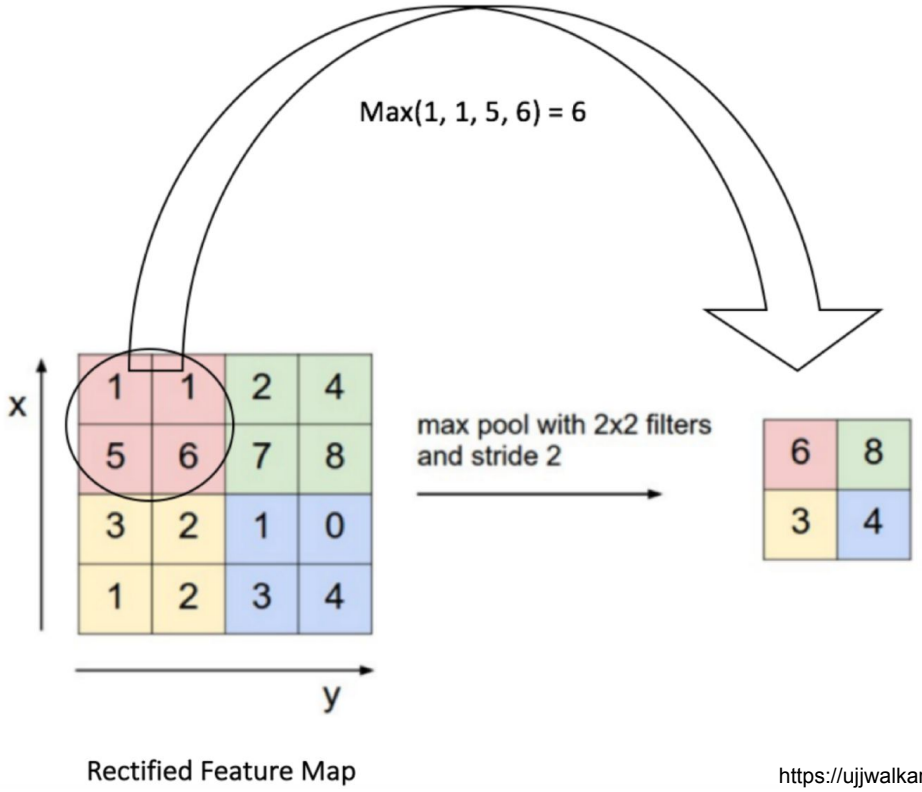
Non linear layers: ReLU (Rectified Linear Unit)

PyTorch class:
***class*torch.nn.ReLU(*inplace=False*)**
Applies the rectified linear unit function element-wise $ReLU(x)=max(0,x)$

- **Inplace:** can optionally do the operation in-place

(Max, Avg, Sum) Pooling (sub-sampling) layers

Figure 10 shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a 2×2 window.



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

(Max, Avg, Sum) Pooling (sub-sampling) layers

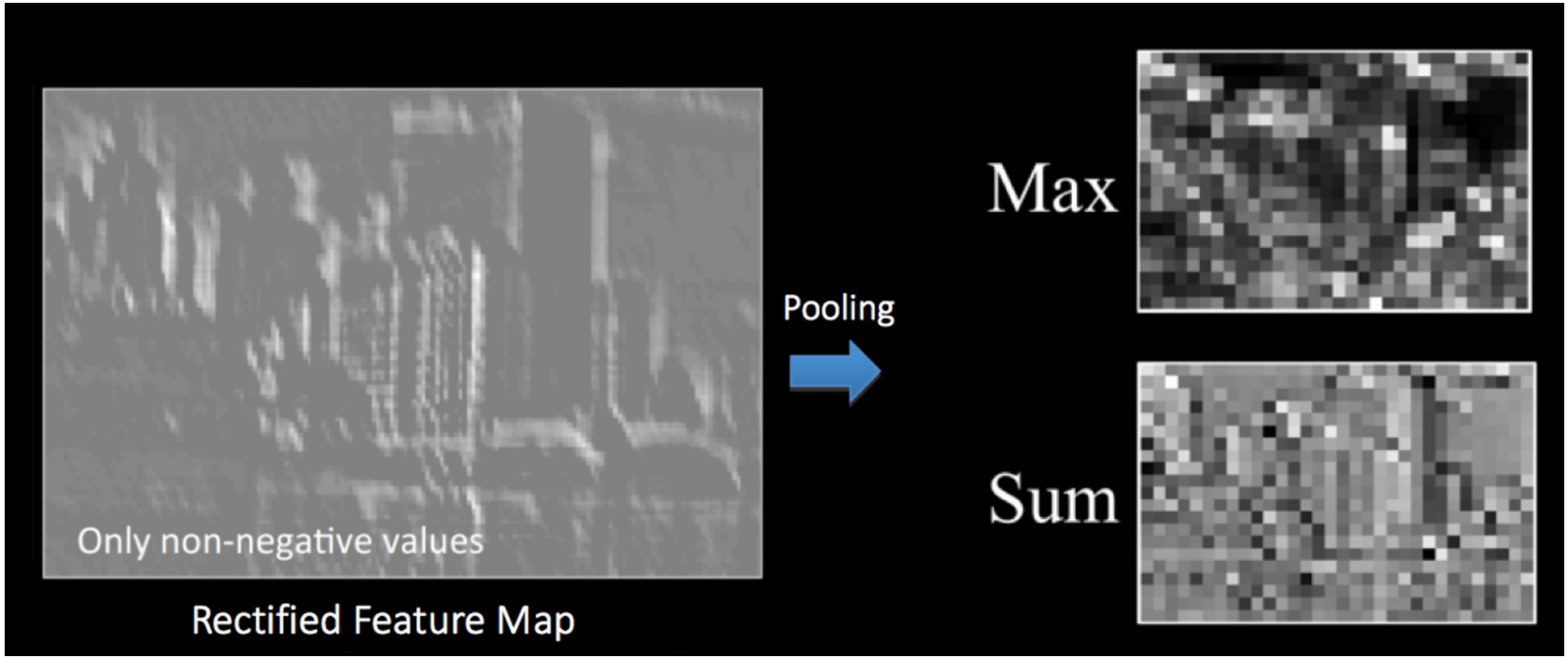
PyTorch class:

***Class* `torch.nn.`MaxPool2d(*kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False*)**

Applies a 2D max pooling over an input signal composed of several input planes.

- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window (default value: **kernel_size)**

Effect: (Max, Avg, Sum) Pooling layers

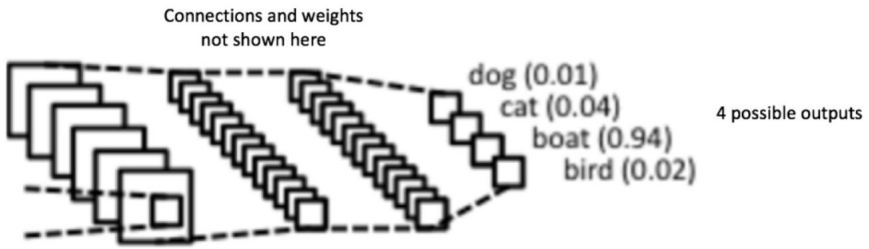


Fully Connected (FC, linear) layers

PyTorch class:

Class **torch.nn.Linear(*in_features, out_features, bias=True*)**

Applies a linear transformation to the incoming data: $y = Ax+b$



- **in_features**: size of each input sample
- **out_features**: size of each output sample
- **bias**: if set to False, the layer will not learn an additive bias

Other Parameters:

- **Batch size**: Nr of training examples seen before updating the weights (i.e. one forward/backward pass; the higher, the more memory space needed)
- **Epochs**: 1 epoch: 1 full iteration on the training set
- **Learning rate η**: Gradient Descent **step size**: rate at which weight parameters are updated when optimizing loss function **L**: $W := W - \eta \nabla \mathrm{L}$
- **Training set size**: number of data points used to train the network

Numpy vs PyTorch

```numpy
import numpy as np

# N is batch size: D_in is input dimension;
# H is hidden dimension; D_out is output dimension

N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = np.random.randn(N, D_in)
y = np.random.randn(N, D_out)

# Randomly initialize weights
w1 = np.random.randn(D_in, H)
w2 = np.random.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = np.matmul(x, w1)
    h_relu = np.maximum(h, 0)
    y_pred = np.matmul(h_relu, w2)

    # Compute and print loss
    loss = ((y_pred - y) ** 2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and
    # w2 with respect to loss  (we go backwards in time one operation at a time,
    # computing the gradient of each input and output parameter)
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = np.matmul(h_relu.T, grad_y_pred)
    grad_h_relu = np.matmul(grad_y_pred, w2.T)
    grad_h = grad_h_relu.copy()
    grad_w1 = np.matmul(x.T, grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

```pytorch
import torch

# N is batch size: D_in is input dimension;
# H is hidden dimension; D_out is output dimension

N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Randomly initialize weights
w1 = torch.randn(D_in, H)
w2 = torch.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = torch.matmul(x, w1)
    h_relu = h.clamp(min=0)
    y_pred = torch.matmul(h_relu, w2)

    # Compute and print loss
    loss = ((y_pred - y) ** 2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and
    # w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = torch.matmul(h_relu.t(), grad_y_pred)
    grad_h_relu = torch.matmul(grad_y_pred, w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = torch.matmul(x.t(), grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Add your model to the leaderboard!

For fun, a leaderboard is available here: https://lite.framacalc.org/xk3dIkVGeE (excel sheet without verification, so try to fill it in an honest and responsible way).
The baseline consists of 2 metrics: Precision and Speed, if you increase the speed of the network while keeping the same prediction you have improved!
To measure the speed of your network, do it during validation, remember that the batch size in validation is 4.
The evolution of the state of the art on Cifar 10 is available here:

- https://en.wikipedia.org/wiki/CIFAR-10#Research_Papers_Claiming_State-of-the-Art_Results_on_CIFAR-10

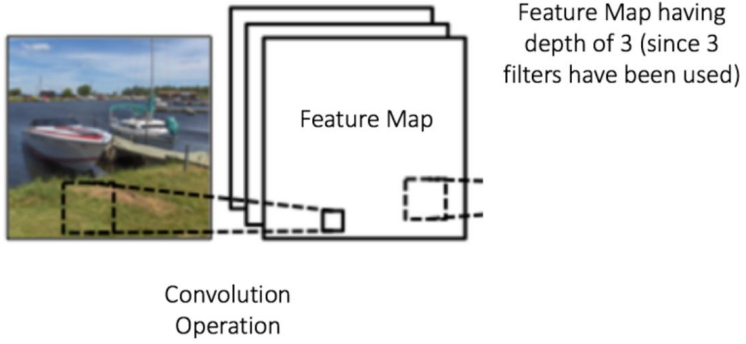
- http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

The Wikipedia link is the newest, the other, more complete. We went from 80% in 2010 to 98% in 2018.

Acknowledgements and References

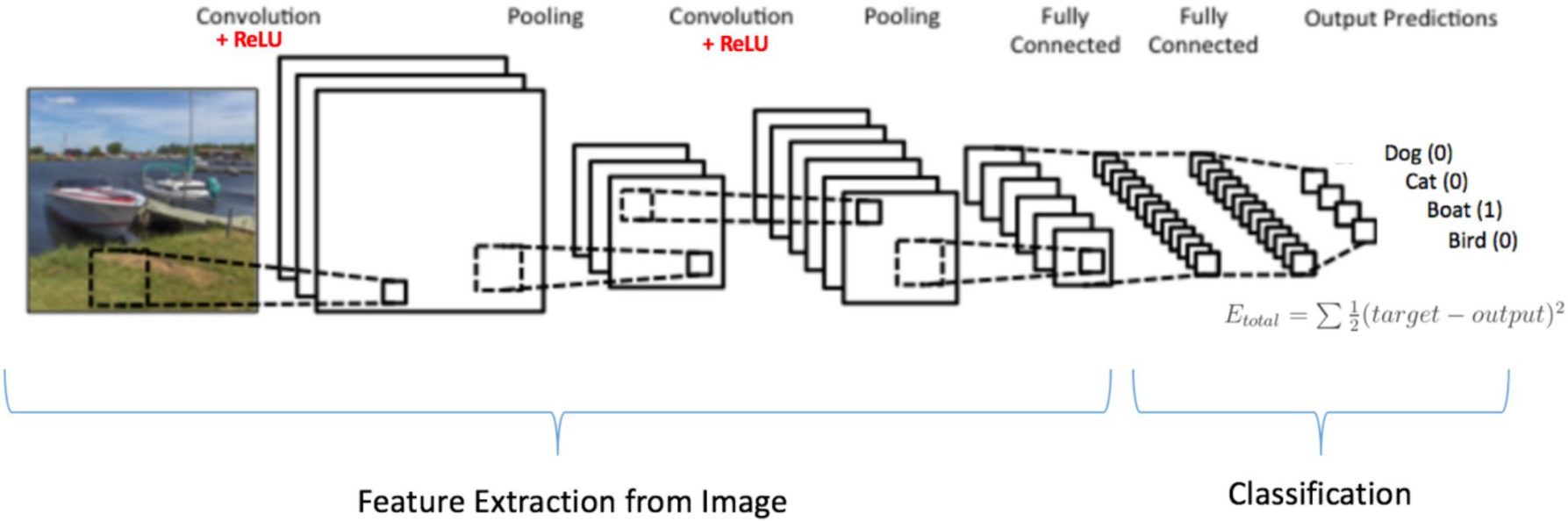
- ROB313 http://perso.ensta-paristech.fr/~manzaner/Cours/ROB313.html
- PyTorch Docs http://pytorch.org/docs/
- Convolution Arithmetic visual guide, V. Dumoulin and F. Visin.
- LeNet, Y. LeCun et al. 1998http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- R. Fergus 2015, http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf
- CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/
- A. Canziani Torch Tutorials https://github.com/Atcold/torch-Video-Tutorials
- E. Culurciello. https://towardsdatascience.com/neural-network-architectures-156e5bad51ba
- AlexNet, Krizhevsky et al. 2012. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Inspiration Appendix: Some successful neural network architectures



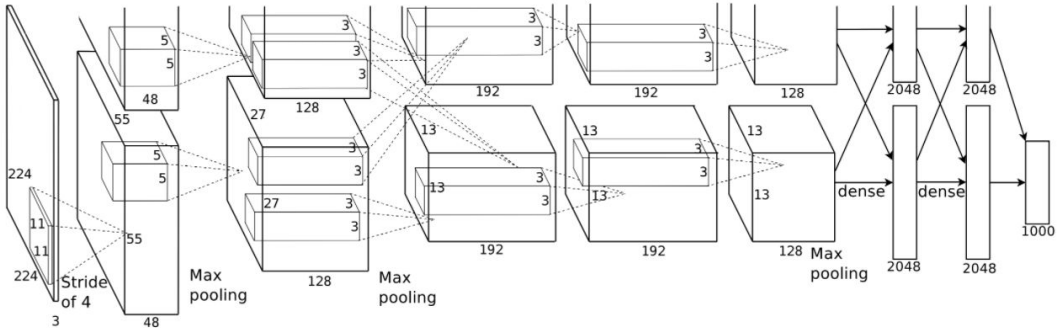
General Architecture

- Input Image = Boat
- Target Vector = [0, 0, 1, 0]



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
https://www.clarifai.com/technology

AlexNet



INPUT: [227x227x3]
CONV1: [55x55x96] 96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96] 3x3 filters at stride 2
CONV2: [27x27x256] 256 5x5 filters at stride 1, pad 2
MAX POOL2: [13x13x256] 3x3 filters at stride 2
CONV3: [13x13x384] 384 3x3 filters at stride 1, pad 1
CONV4: [13x13x384] 384 3x3 filters at stride 1, pad 1
CONV5: [13x13x256] 256 3x3 filters at stride 1, pad 1
MAX POOL3: [6x6x256] 3x3 filters at stride 2
FC6: [4096] 4096 neurons
FC7: [4096] 4096 neurons
FC8: [1000] 1000 neurons (softmax logits)

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

AlexNet (Uses Dropout)

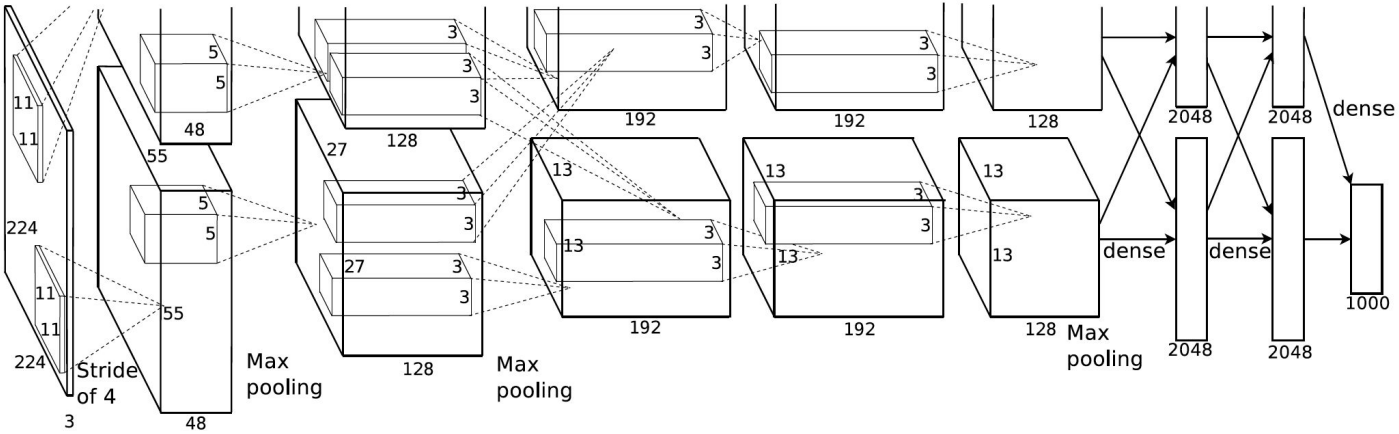


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

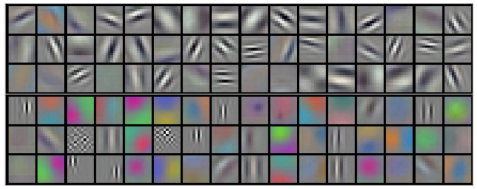
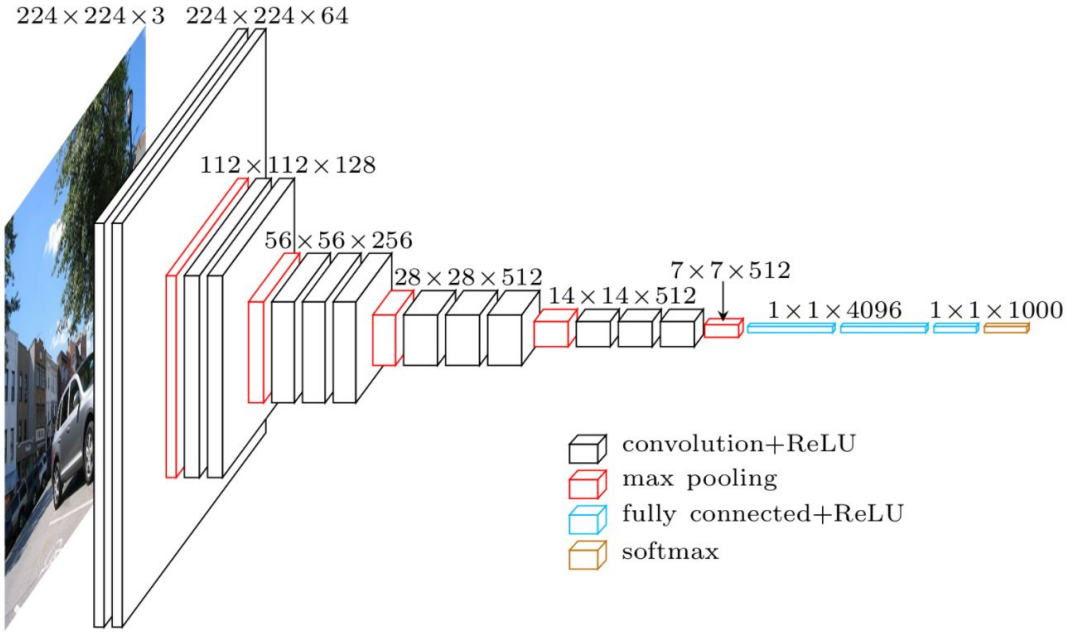


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

VGG-16



Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

ResNet

A block learns the residual w.r.t. identity

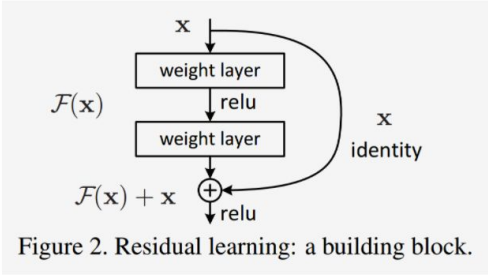
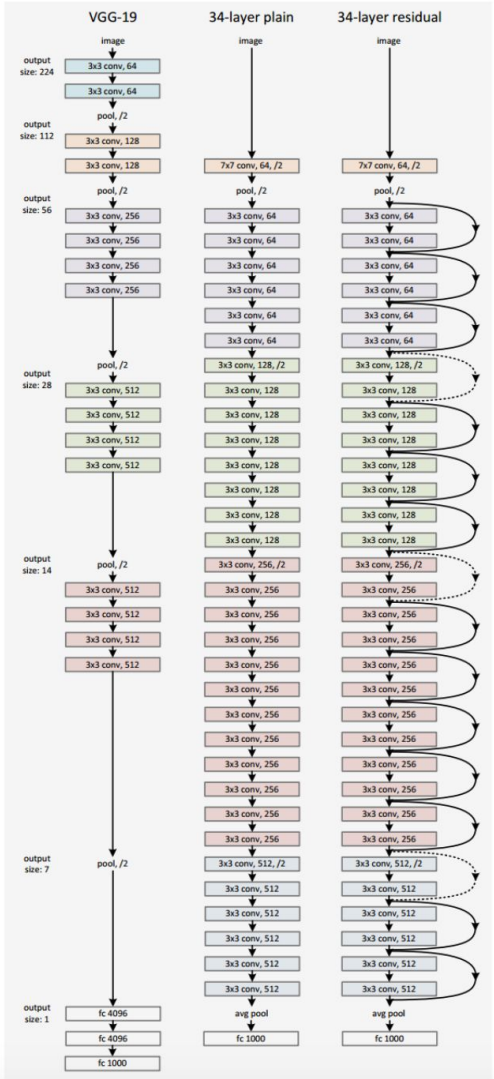


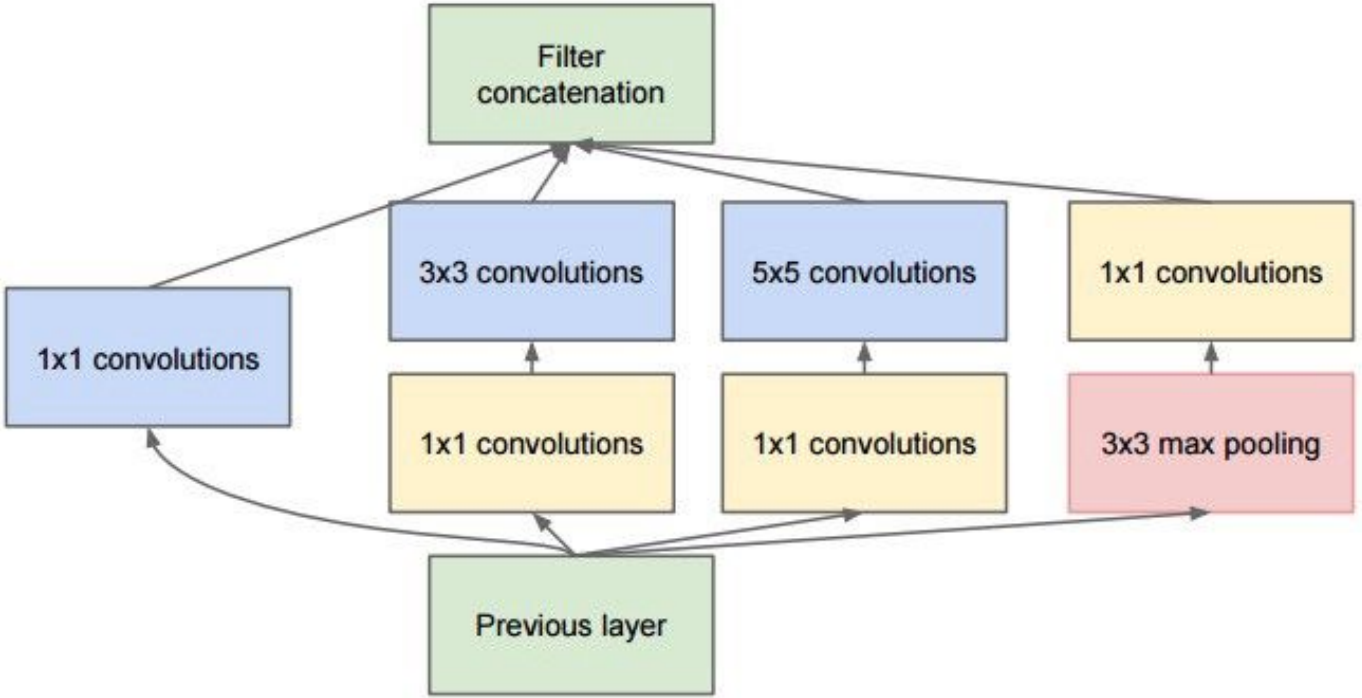
Figure 2. Residual learning: a building block.

- Good optimization properties

He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.



Inception model module



https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

Appendix

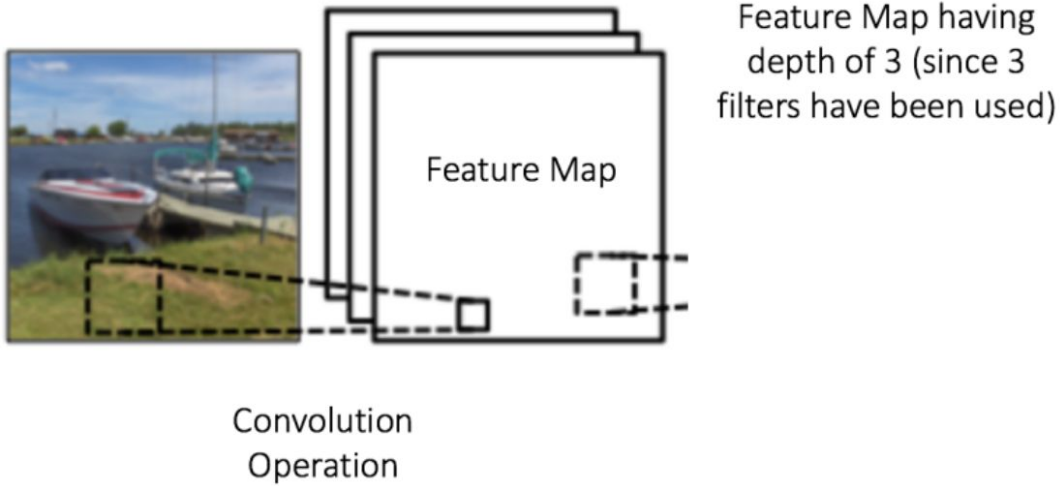
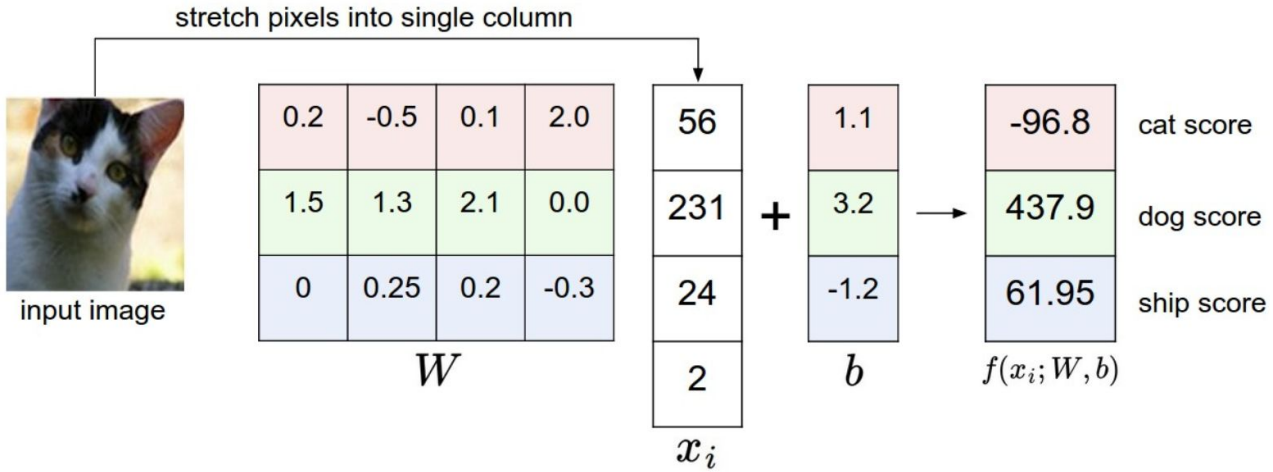

Feature Map having depth of 3 (since 3 filters have been used)
Feature Map
Convolution Operation

Image classification objective: mapping an image to a score



An example of mapping an image to class scores. For the sake of visualization, we assume the image only has 4 pixels (4 monochrome pixels, we are not considering color channels in this example for brevity), and that we have 3 classes (red (cat), green (dog), blue (ship) class). (Clarification: in particular, the colors here simply indicate 3 classes and are not related to the RGB channels.) We stretch the image pixels into a column and perform matrix multiplication to get the scores for each class. Note that this particular set of weights W is not good at all: the weights assign our cat image a very low cat score. In particular, this set of weights seems convinced that it's looking at a dog.

Architecture Ingredients: Convolution Transpose Layer (upsampling)

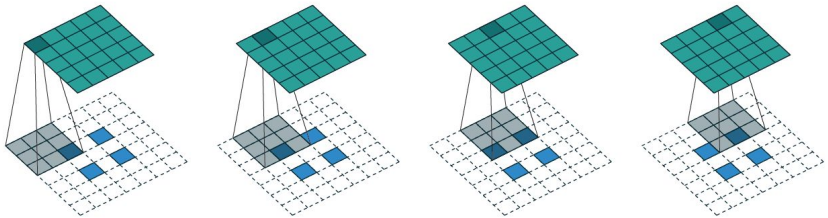


Figure 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $\tilde{i}' = 3$, $k' = k$, $s' = 1$ and $p' = 2$).

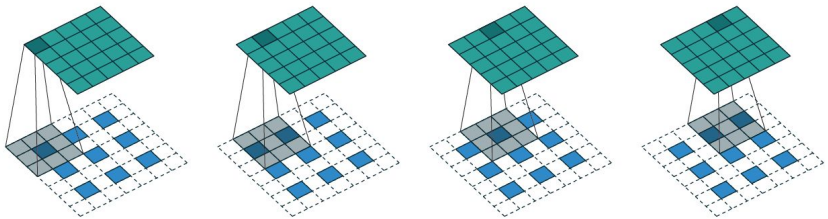


Figure 4.6: The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

https://github.com/vdumoulin/conv_arithmetic

Architecture Ingredients: Convolution Layer: Dilation

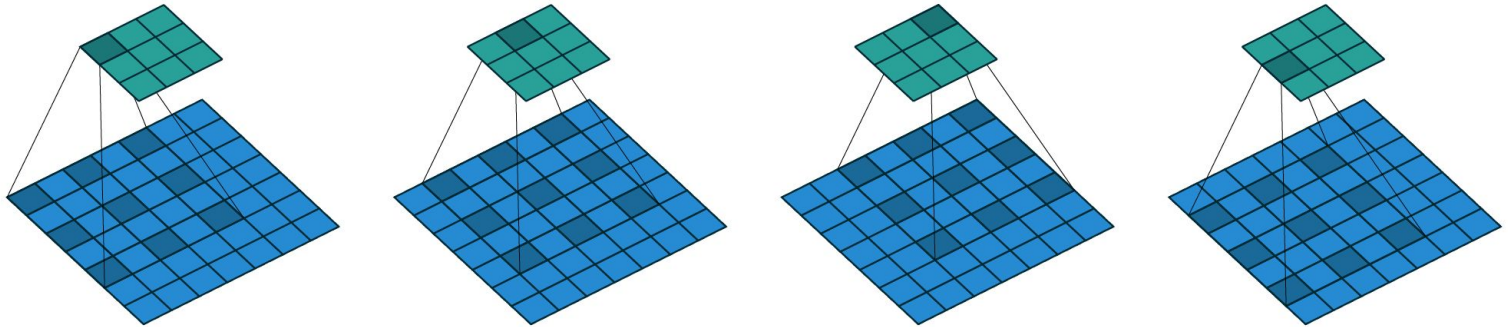


Figure 5.1: (Dilated convolution) Convolving a 3×3 kernel over a 7×7 input with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).

Figure 5.1 provides an example for $i = 7$, $k = 3$ and $d = 2$.

Dilated convolution is a way of increasing receptive view (global view) of the network exponentially and linear parameter accretion. Integrates knowledge of a wider context with less cost. However, aggressively increasing **dilation** factors fails to aggregate local features of small objects.
Hamaguchi'17
https://github.com/vdumoulin/conv_arithmetic

(Max, Avg, Sum) Pooling (sub-sampling) layers

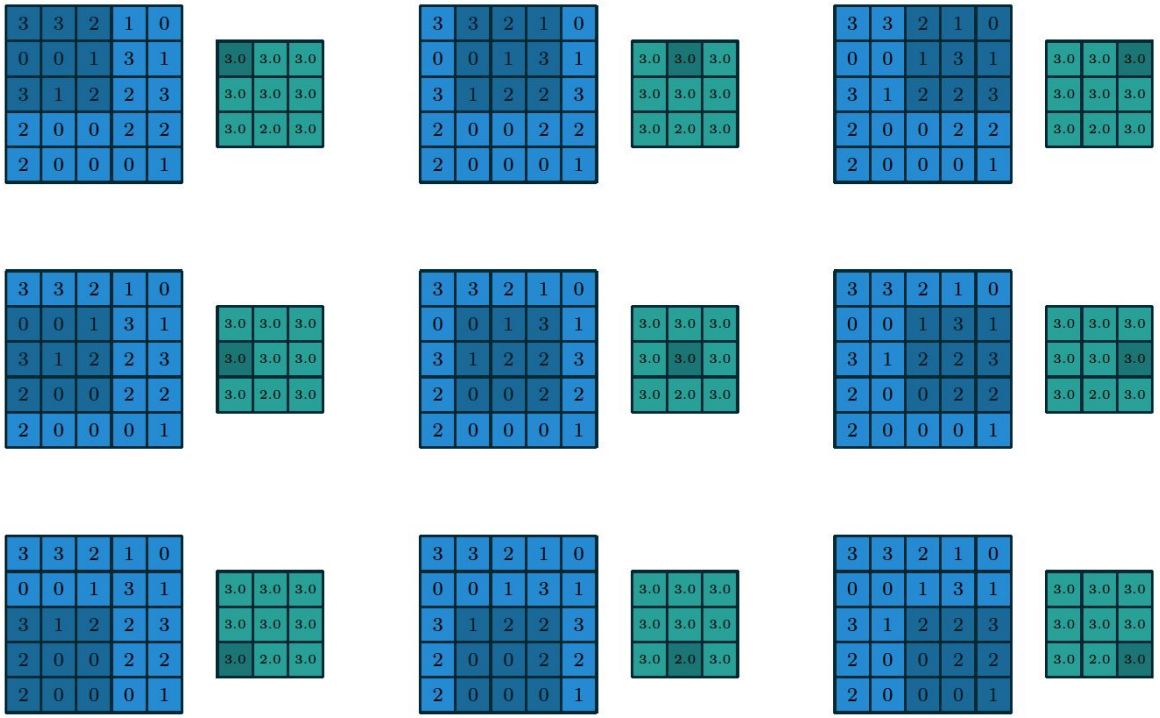


Figure 1.6: Computing the output values of a 3×3 max pooling operation on a 5×5 input using 1×1 strides.

Good exercises:

- Write the backpropagation equations
  http://neuralnetworksanddeeplearning.com/chap2.html
- http://neuralnetworksanddeeplearning.com/chap2.html#the_four_fundamental_equations_behind_backpropagation

Extras:

- Dropout regularization (see example) and other regularization techniques in PyTorch
- Pre-trained networks: see state of the art on ResNet