Introduction to Deep Learning

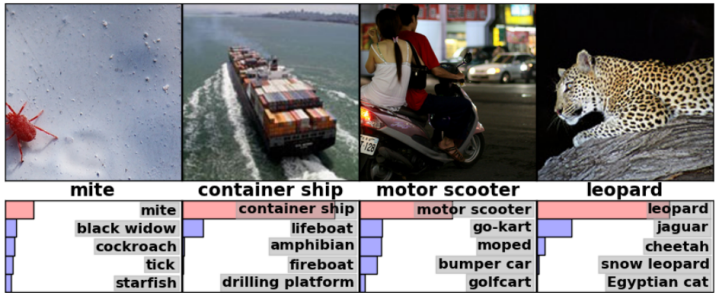
Master 2 AIDN / CMI 2020

Lecture 5: Convolutional Neural Networks (CNN)

(Adapted from a course of Charles Ollion - Olivier Grisel)

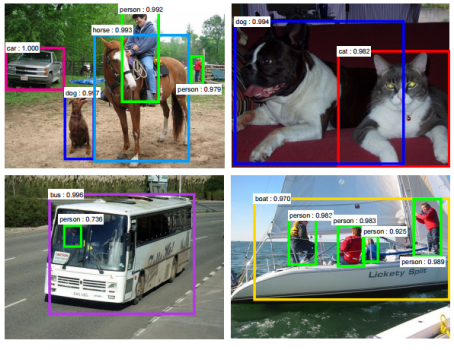
Prof. Nicolas Courty
ncourty@irisa.fr

Used everywhere for Vision

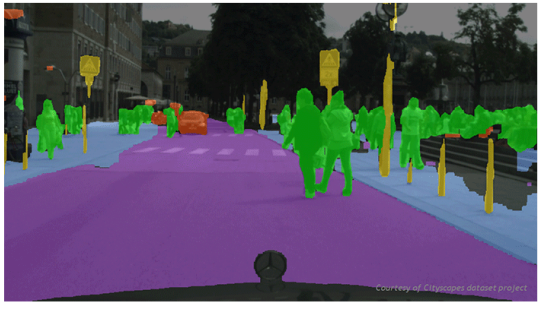


[Krizhevsky 2012]

[Ciresan et al. 2013]



[Faster R-CNN - Ren 2015]



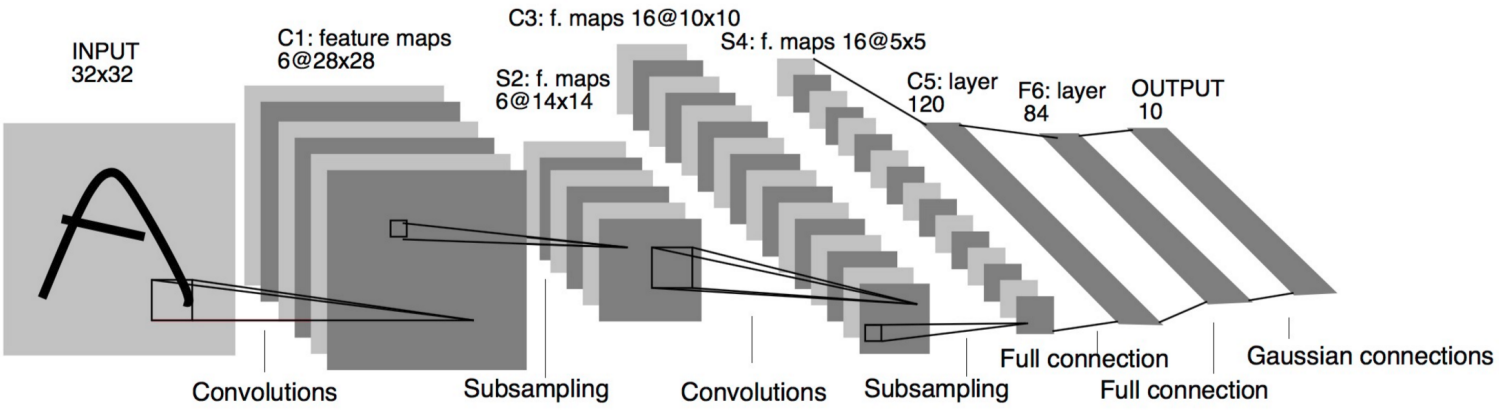
[NVIDIA dev blog]

Many other applications

- Speech recognition
- NLP (Natural Language Processing)
- Time series classification
- Protein/DNA binding prediction (more generally graphs) Any problem with a spatial (or sequential) structure !

ConvNets for image classification

CNN = Convolutional Neural Networks = ConvNet



LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.

Outline

- Convolutions
- CNNs for Image Classification
- CNN Architectures

Convolutions

Motivations

Standard Dense Layer for an image input:

```
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?

- $640 \times 480 \times 3 \times 1000 + 1000 = 922M$!
- Spatial organization of the input is destroyed by `Flatten`
- Never use Dense layers directly on large images. Most standard solution is convolution layers

Fully Connected Network: MLP

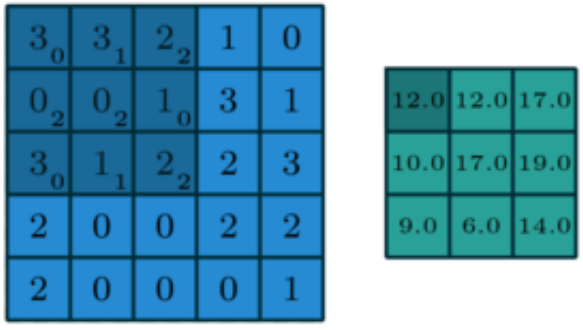
```
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

Convolutional Network

```
input_image = Input(shape=(28, 28, 1))
*x = Conv2D(32, 5, activation='relu')(input_image)
*x = MaxPool2D(2, strides=2)(x)
*x = Conv2D(64, 3, activation='relu')(x)
*x = MaxPool2D(2, strides=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
convnet = Model(inputs=input_image, outputs=x)
```

2D spatial organization of features preserved untill `Flatten`.

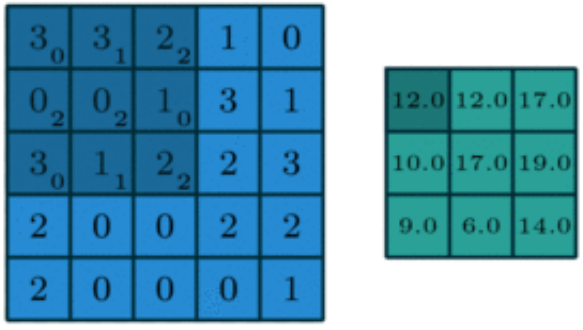
Convolution in a neural network



- x is a 3×3 chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ (small numbers)

These slides extensively use convolution visualisation by V. Dumoulin available at https://github.com/vdumoulin/conv_arithmetic

Convolution in a neural network



- x is a 3×3 chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ (small numbers)

The activation obtained by sliding the 3×3 window and computing:

$$z(x) = \text{relu}(\mathbf{w}^T x + b)$$

Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: strong prior for vision!

Animal Vision Analogy

Hubel & Wiesel, RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX (1959)

Why Convolution

Discrete convolution (actually cross-correlation) between two functions f and g:

$$(f \star g)(x) = \sum_{a+b=x} f(a).g(b) = \sum_{a} f(a).g(x+a)$$

2D-convolutions (actually 2D cross-correlation):

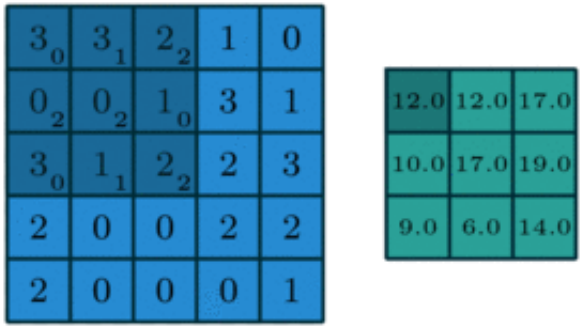
$$(f \star g)(x,y) = \sum_{n} \sum_{m} f(n,m).g(x+n, y+m)$$

f is a convolution kernel or filter applied to the 2-d map g (our image)

Example: convolution image

- Image: I of dimensions 5×5
- Kernel: k of dimensions 3×3

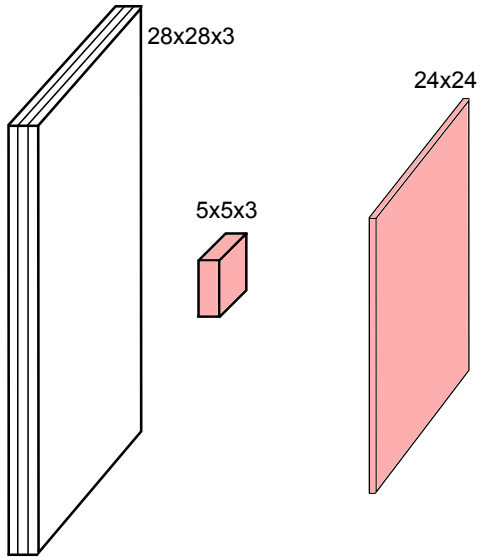
$$(\mathrm{k} \star \mathrm{I})(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} \mathrm{k}(n, m).\mathrm{I}(x + n - 1, y + m - 1)$$



Channels

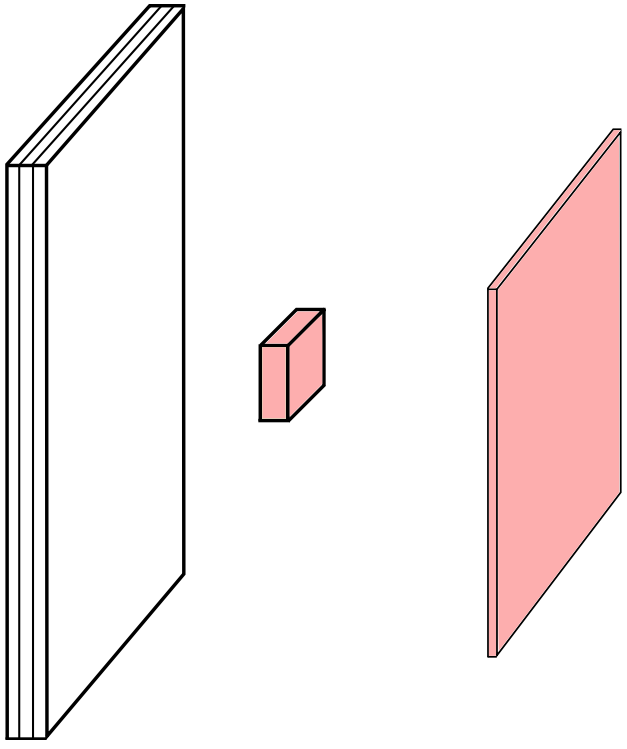
Colored image = tensor of shape `(height, width, channels)`

Convolutions are usually computed for each channel and summed:

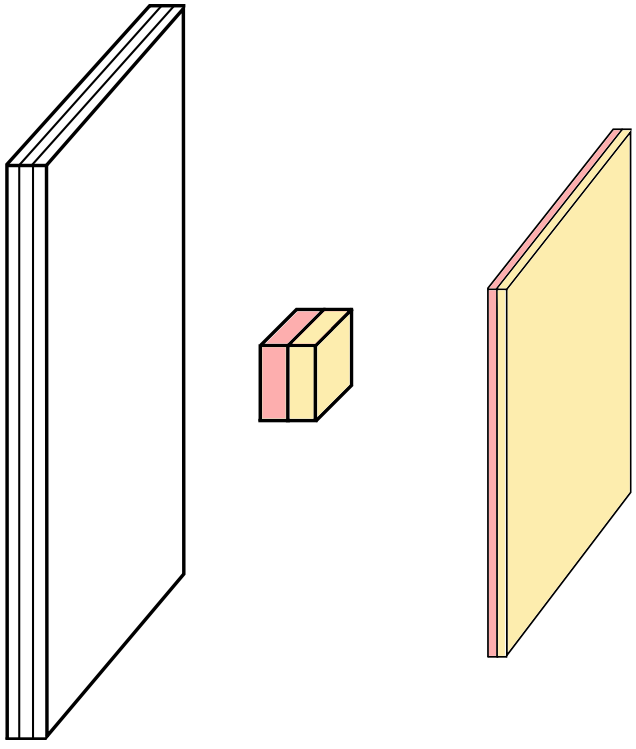


$$(k \star \mathrm{I}^{color}) = \sum_{c=0}^{2} \mathrm{k}^{c} \star \mathrm{I}^{c}$$

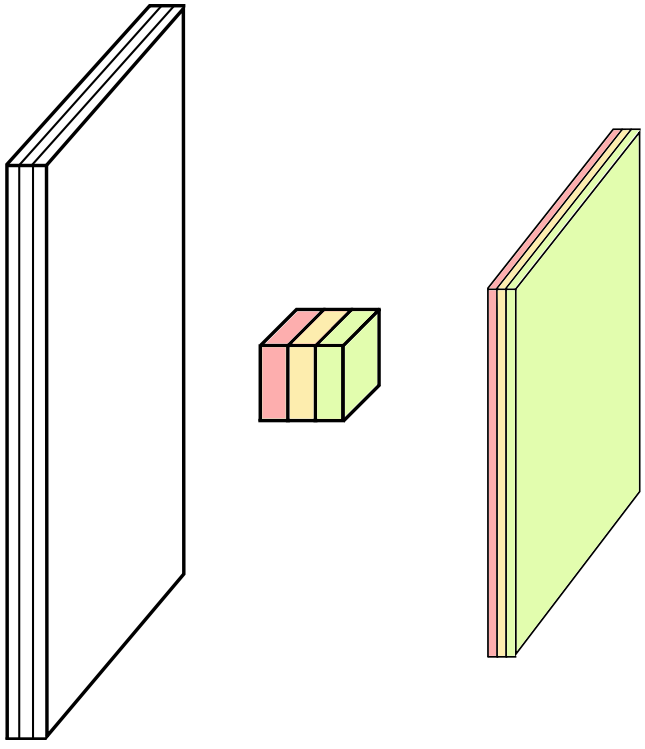
Multiple convolutions



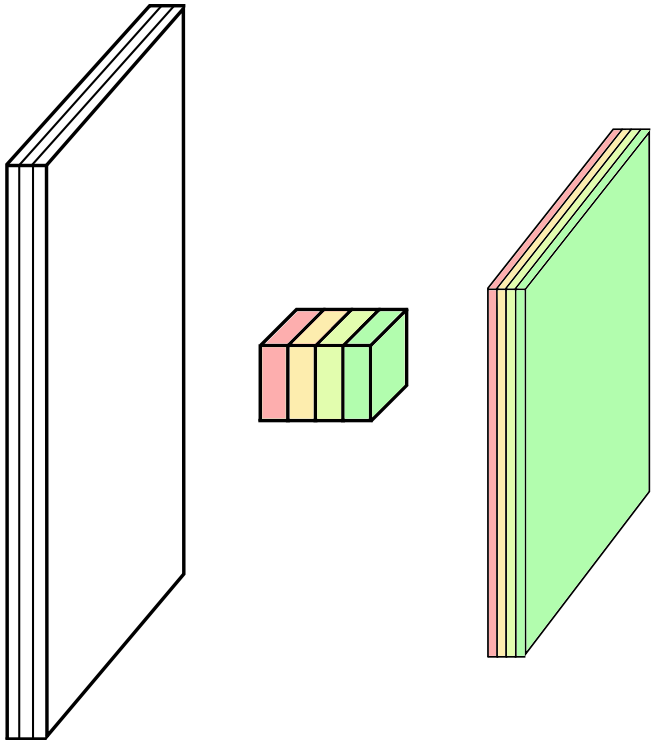
Multiple convolutions



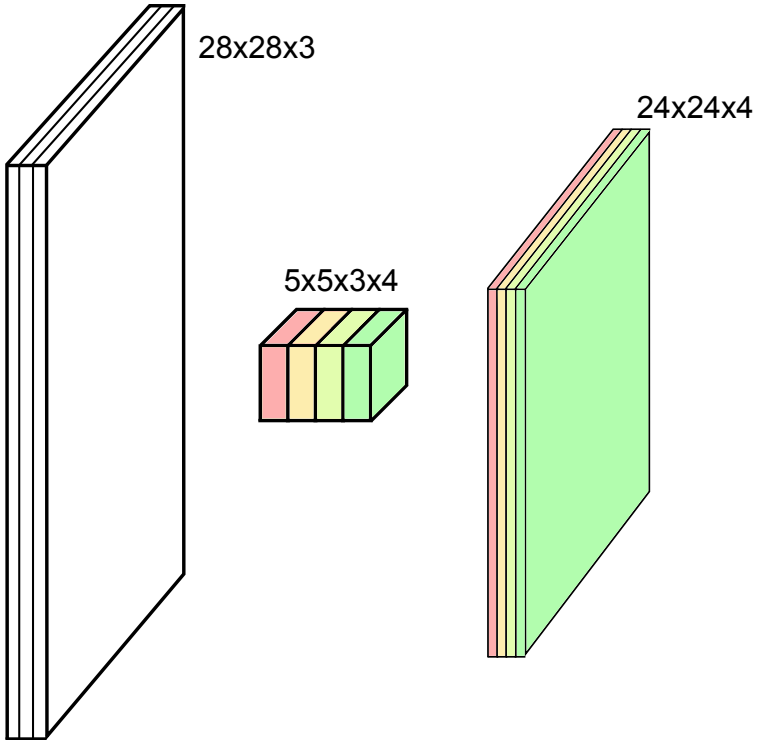
Multiple convolutions



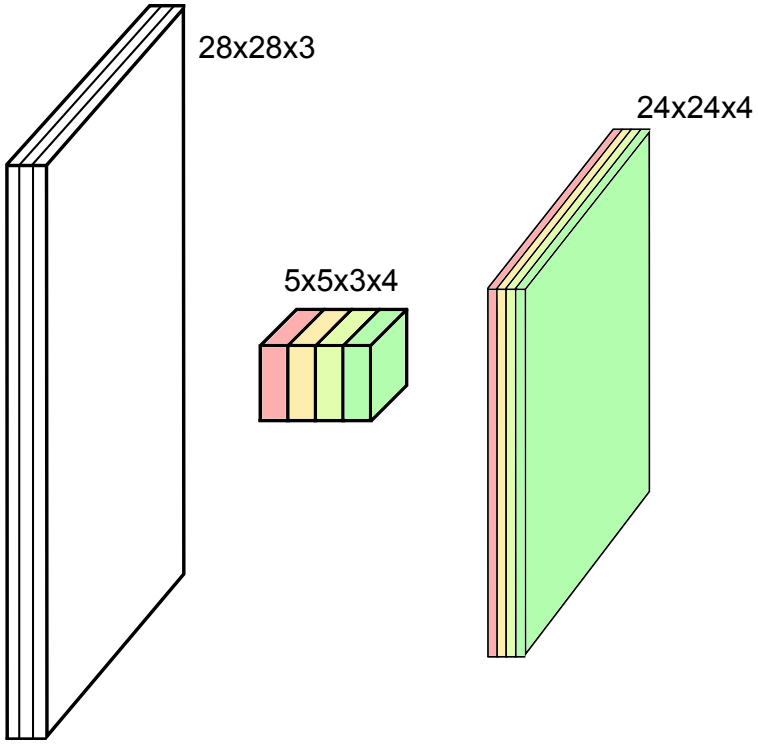
Multiple convolutions



Multiple convolutions



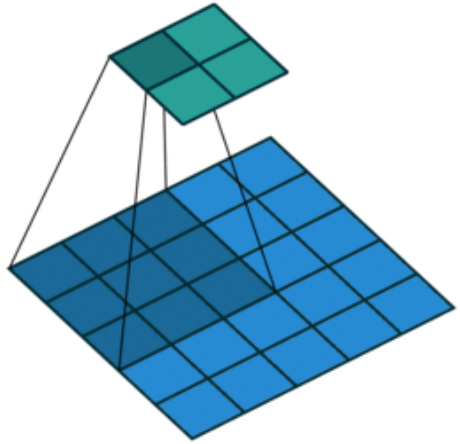
Multiple convolutions



- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: `length - kernel_size + 1`

Strides

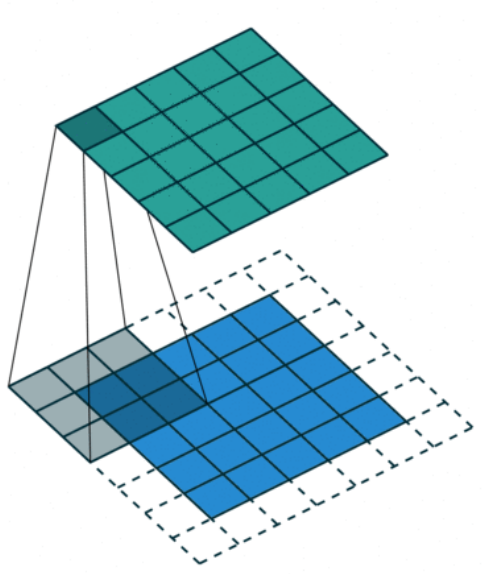
- Strides: increment step size for the convolution operator
- Reduces the size of the ouput map



Example with kernel size 3×3 and a stride of 2 (image in blue)

Padding

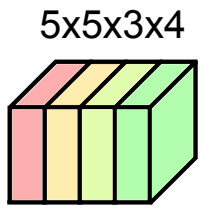
- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



Dealing with shapes

Kernel or Filter shape (F, F, C^i, C^o)

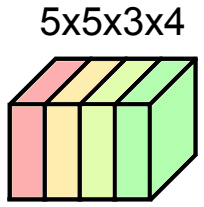
- $F \times F$ kernel size,
- C^i input channels
- C^o output channels



Dealing with shapes

Kernel or Filter shape (F, F, C^i, C^o)

- $F \times F$ kernel size,
- C^i input channels
- C^o output channels

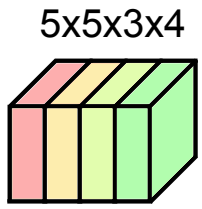


Number of parameters: $(F \times F \times C^i + 1) \times C^o$

Dealing with shapes

Kernel or Filter shape (F, F, C^i, C^o)

- $F \times F$ kernel size,
- C^i input channels
- C^o output channels



Number of parameters: $(F \times F \times C^i + 1) \times C^o$

Activations or Feature maps shape:

- Input (W^i, H^i, C^i)
- Output (W^o, H^o, C^o)

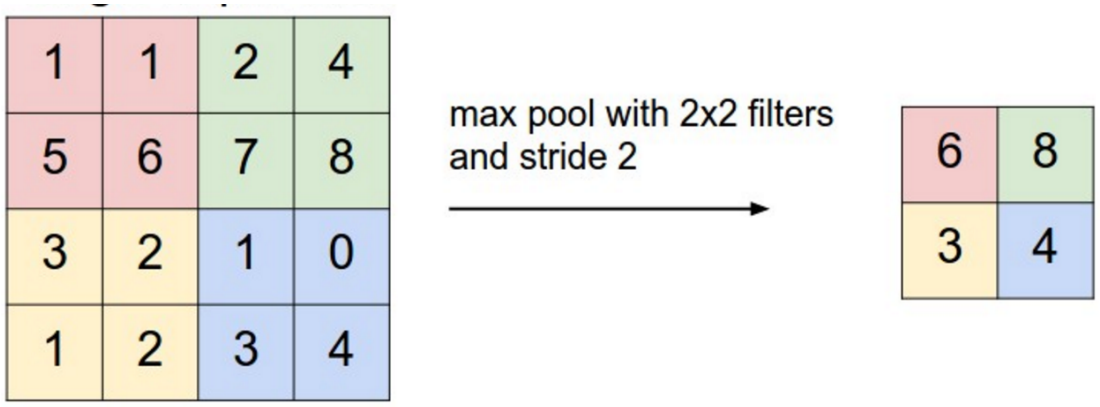
$W^o = (W^i - F + 2P)/S + 1$

Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units

Pooling

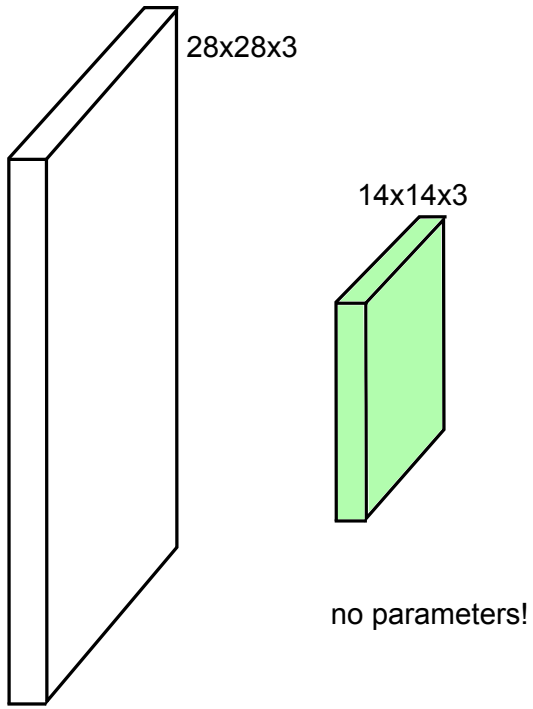
- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Schematic from Stanford http://cs231n.github.io/convolutional-networks

Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Architectures

Classic ConvNet Architecture

Input

Classic ConvNet Architecture

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Classic ConvNet Architecture

Input

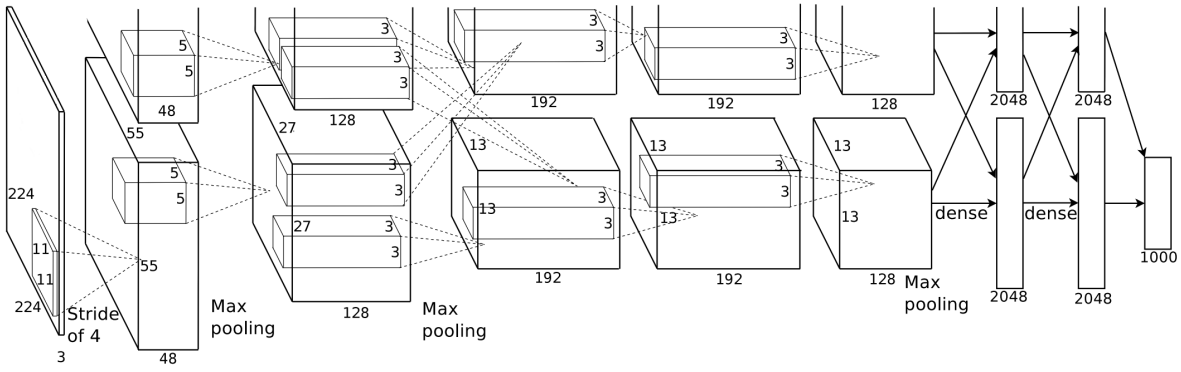
Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Output

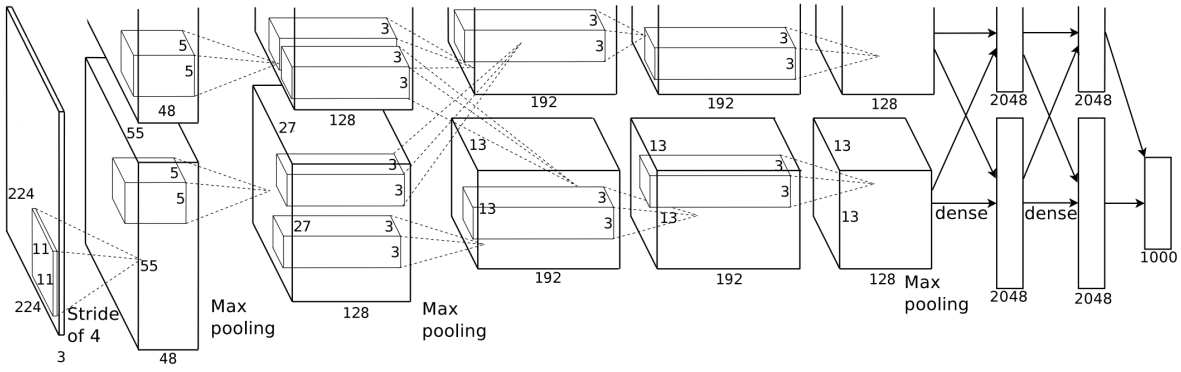
- Fully connected layers
- Softmax

AlexNet



Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

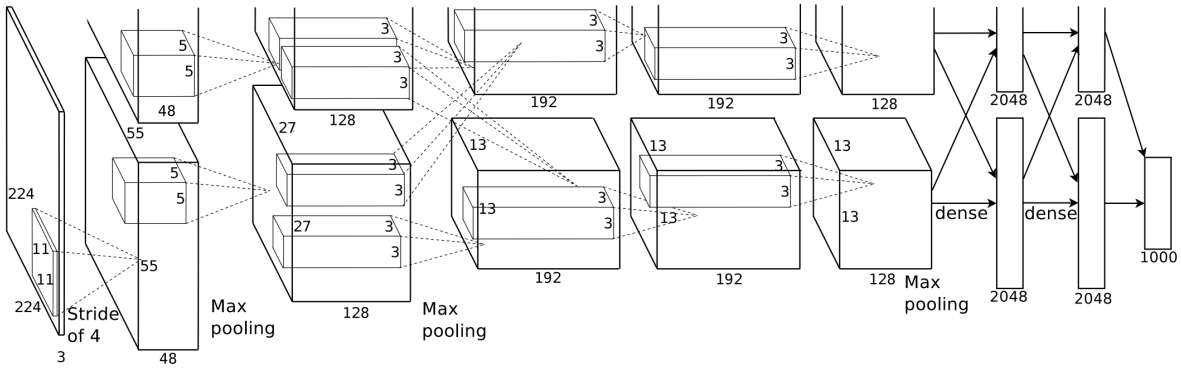
AlexNet



First conv layer: kernel 11x11x3x96 stride 4

Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

AlexNet

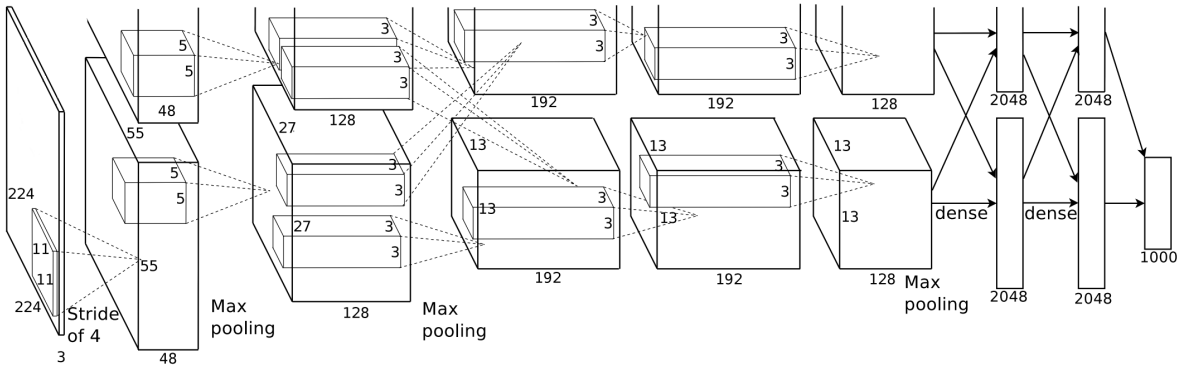


First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: `(11,11,3,96)`
- Output shape: `(55,55,96)`
- Number of parameters: `34,944`
- Equivalent MLP parameters: `43.7 x 1e9`

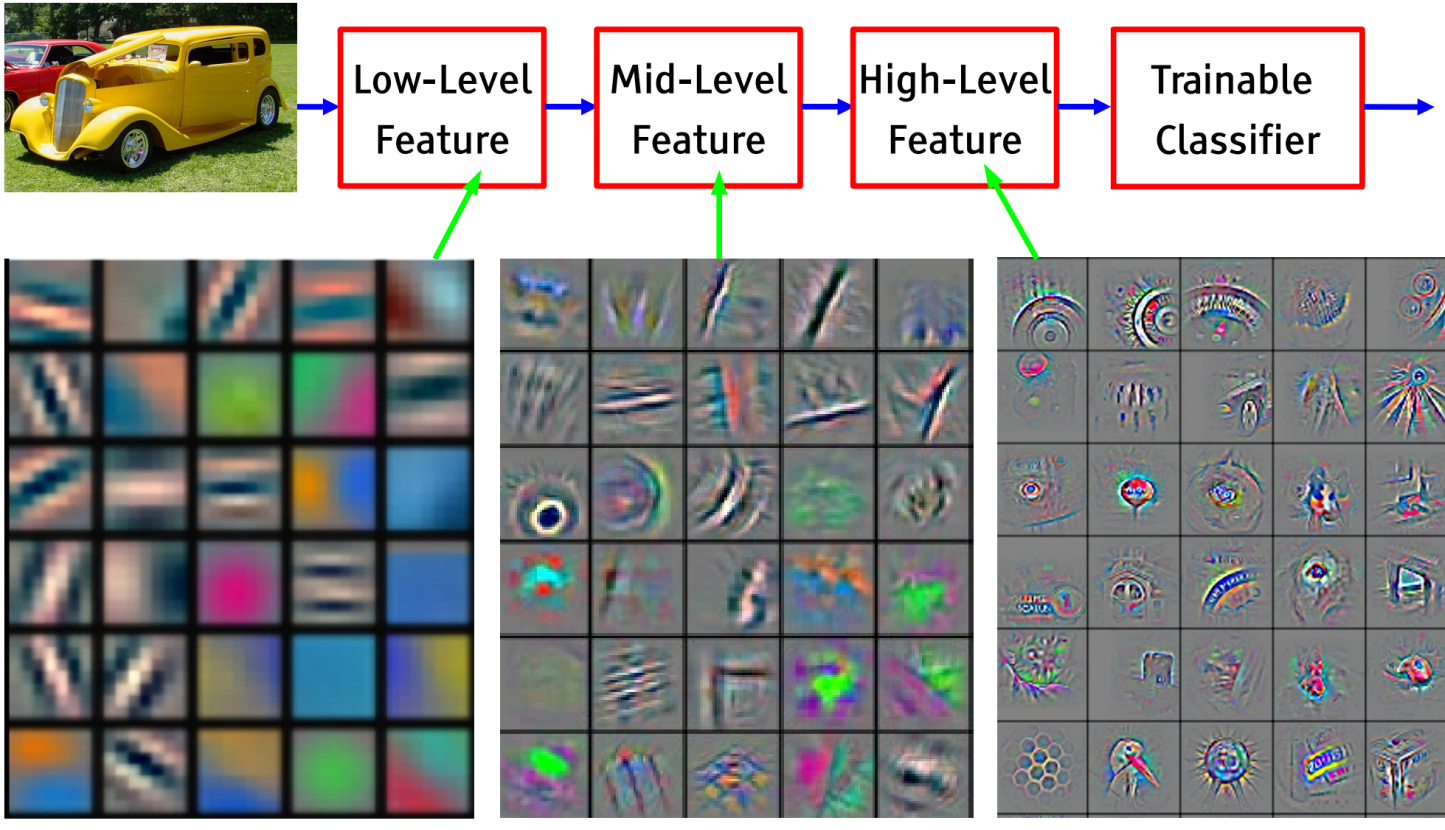
Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

AlexNet



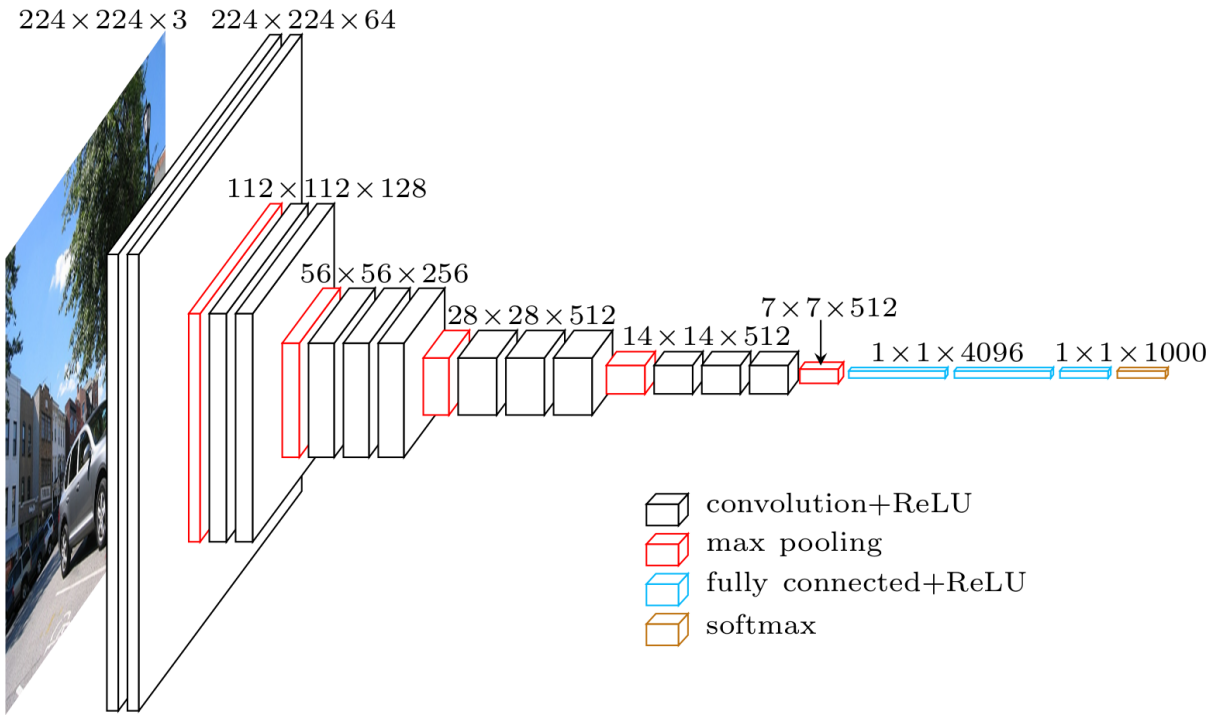

```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride 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)
```

Hierarchical representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

VGG-16



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

VGG in Keras

```
model.add(Convolution2D(64, 3, 3, activation='relu',input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

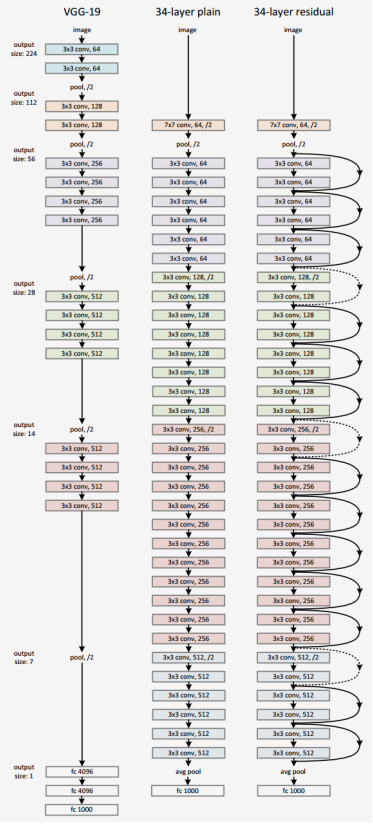
Memory and Parameters

```
           Activation maps          Parameters
INPUT:     [224x224x3]   = 150K     0
CONV3-64:  [224x224x64]  = 3.2M     (3x3x3)x64    =       1,728
CONV3-64:  [224x224x64]  = 3.2M     (3x3x64)x64   =      36,864
POOL2:     [112x112x64]  = 800K     0
CONV3-128: [112x112x128] = 1.6M     (3x3x64)x128  =      73,728
CONV3-128: [112x112x128] = 1.6M     (3x3x128)x128 =     147,456
POOL2:     [56x56x128]   = 400K     0
CONV3-256: [56x56x256]   = 800K     (3x3x128)x256 =     294,912
CONV3-256: [56x56x256]   = 800K     (3x3x256)x256 =     589,824
CONV3-256: [56x56x256]   = 800K     (3x3x256)x256 =     589,824
POOL2:     [28x28x256]   = 200K     0
CONV3-512: [28x28x512]   = 400K     (3x3x256)x512 =   1,179,648
CONV3-512: [28x28x512]   = 400K     (3x3x512)x512 =   2,359,296
CONV3-512: [28x28x512]   = 400K     (3x3x512)x512 =   2,359,296
POOL2:     [14x14x512]   = 100K     0
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =   2,359,296
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =   2,359,296
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =   2,359,296
POOL2:     [7x7x512]     =  25K     0
FC:        [1x1x4096]    = 4096     7x7x512x4096  = 102,760,448
FC:        [1x1x4096]    = 4096     4096x4096     =  16,777,216
FC:        [1x1x1000]    = 1000     4096x1000     =   4,096,000

TOTAL activations: 24M x 4 bytes ~=  93MB / image (x2 for backward)
TOTAL parameters: 138M x 4 bytes ~= 552MB (x2 for plain SGD, x4 for Adam)
```

Memory and Parameters

```
           Activation maps            Parameters
INPUT:     [224x224x3]   = 150K     0
*CONV3-64:  [224x224x64]  = 3.2M     (3x3x3)x64    =        1,728
*CONV3-64:  [224x224x64]  = 3.2M     (3x3x64)x64   =       36,864
POOL2:     [112x112x64]  = 800K     0
CONV3-128: [112x112x128] = 1.6M     (3x3x64)x128  =       73,728
CONV3-128: [112x112x128] = 1.6M     (3x3x128)x128 =      147,456
POOL2:     [56x56x128]   = 400K     0
CONV3-256: [56x56x256]   = 800K     (3x3x128)x256 =      294,912
CONV3-256: [56x56x256]   = 800K     (3x3x256)x256 =      589,824
CONV3-256: [56x56x256]   = 800K     (3x3x256)x256 =      589,824
POOL2:     [28x28x256]   = 200K     0
CONV3-512: [28x28x512]   = 400K     (3x3x256)x512 =    1,179,648
CONV3-512: [28x28x512]   = 400K     (3x3x512)x512 =    2,359,296
CONV3-512: [28x28x512]   = 400K     (3x3x512)x512 =    2,359,296
POOL2:     [14x14x512]   = 100K     0
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =    2,359,296
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =    2,359,296
CONV3-512: [14x14x512]   = 100K     (3x3x512)x512 =    2,359,296
POOL2:     [7x7x512]     =  25K     0
*FC:        [1x1x4096]    = 4096     7x7x512x4096  = 102,760,448
FC:        [1x1x4096]    = 4096     4096x4096     =  16,777,216
FC:        [1x1x1000]    = 1000     4096x1000     =   4,096,000

TOTAL activations: 24M x 4 bytes ~=  93MB / image (x2 for backward)
TOTAL parameters: 138M x 4 bytes ~= 552MB (x2 for plain SGD, x4 for Adam)
```

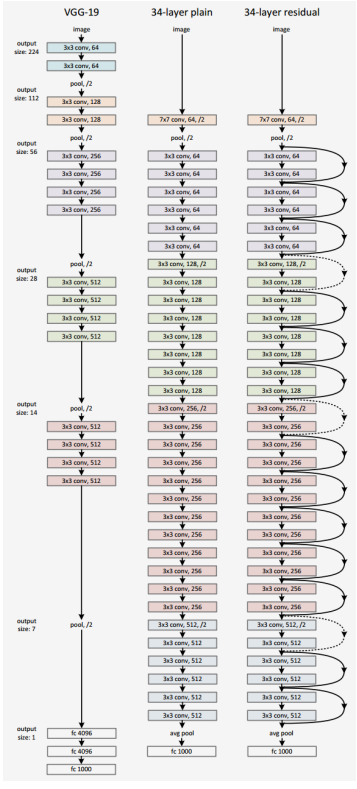
ResNet



- Even deeper models:
  - 34, 50, 101, 152 layers

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

ResNet



Learning with an identity module

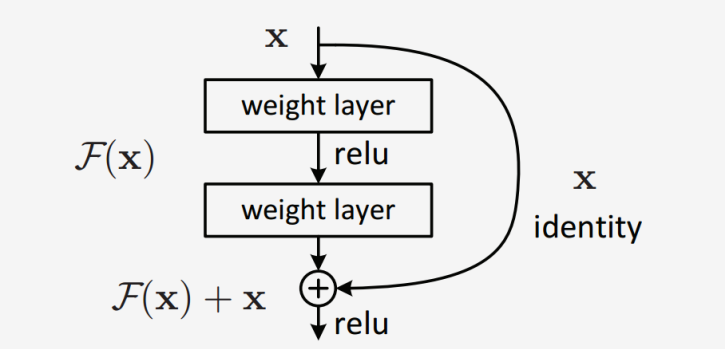
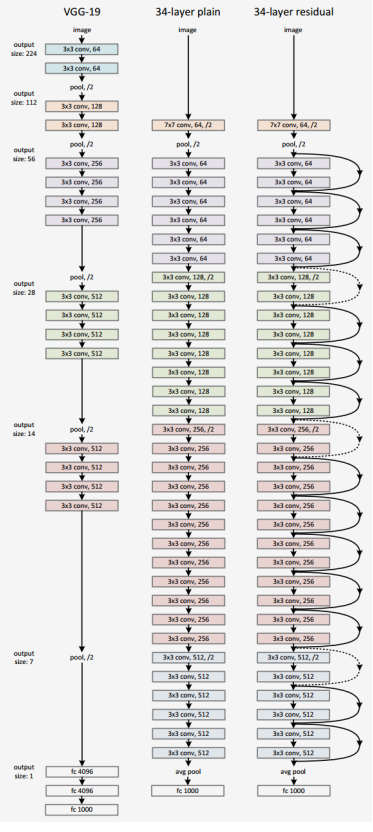


Figure 2. Residual learning: a building block.

- Good optimization properties

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

ResNet

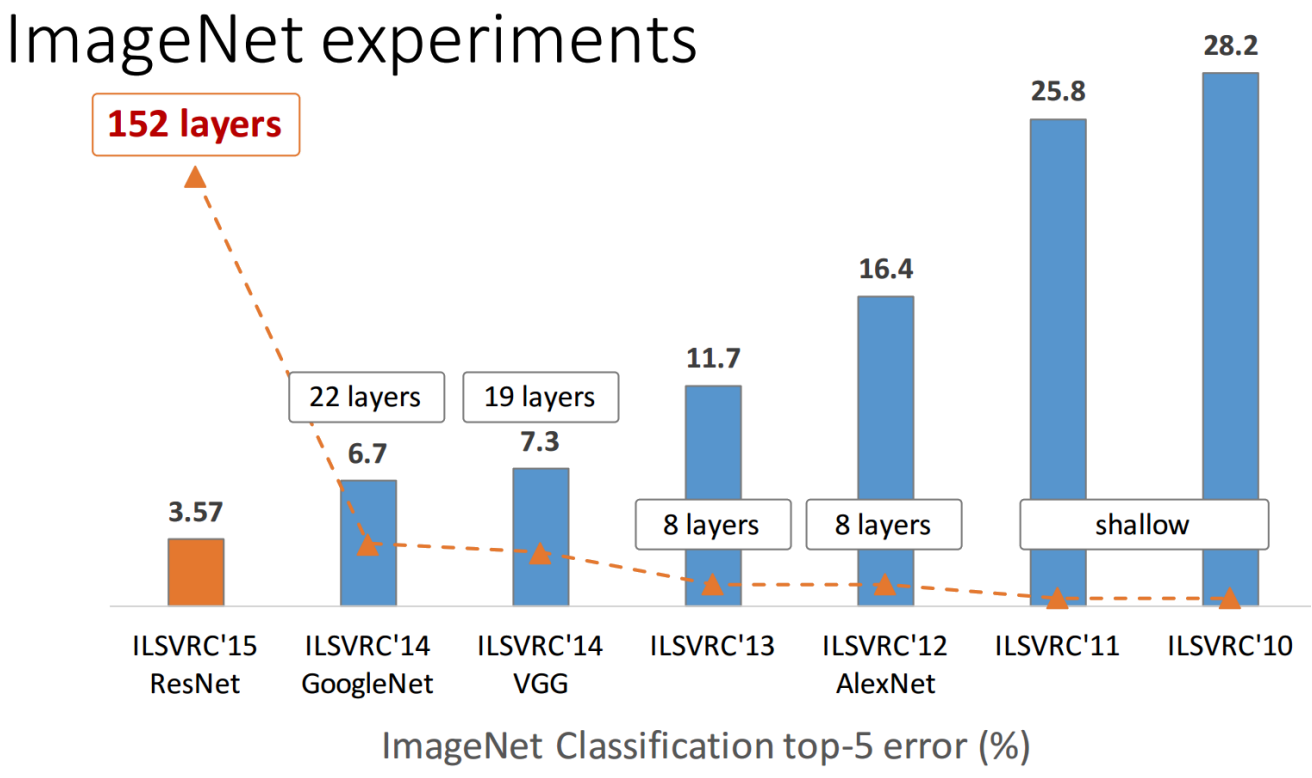


ResNet50 Compared to VGG:

- Superior accuracy in all vision tasks
  - 5.25% top-5 error vs 7.1%
- Less parameters
  - 25M vs 138M
- Computational complexity
  - 3.8B Flops vs 15.3B Flops
- Fully Convolutional until the last layer

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

Deeper is better



---

from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

State of the art

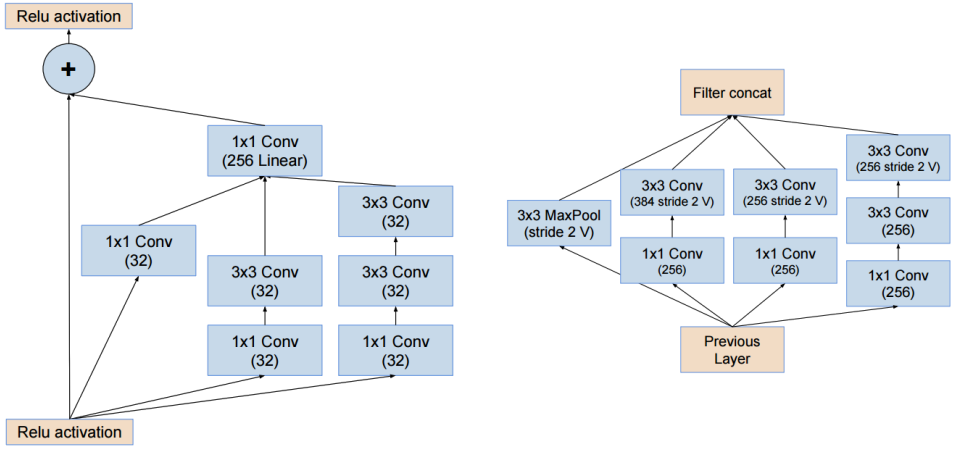
- Finding right architectures: Active area or research

Model	Params	×+	1/5-Acc (%)
Inception V3	23.8M	5.72B	78.0 / 93.9
Xception	22.8M	8.37B	79.0 / 94.5
Inception ResNet V2	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 (64x4d)	83.6M	31.5B	80.9 / 95.6
PolyNet	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131	79.5M	32.0B	81.5 / 95.8
Squeeze-Excite-Net	145.8M	42.3B	82.7 / **96.2**
GeNet-2	156M	–	72.1 / 90.4
Block-QNN-B, N=3	–	–	75.7 / 92.6
Hierarchical (2, 64)	64M	–	79.7 / 94.8
PNASNet-5 (4, 216)	86.1M	25.0B	**82.9** / 96.1
NASNet-A (6, 168)	88.9M	23.8B	82.7 / **96.2**
AmoebaNet-B (6, 190)	84.0M	22.3B	82.3 / 96.1
AmoebaNet-C (6, 168)	85.5M	22.5B	82.7 / 96.1
AmoebaNet-A (6, 190)	86.7M	23.1B	82.8 / 96.1
AmoebaNet-A (6, 204)	99.6M	26.2B	82.8 / **96.2**

Esteban Real, et al. Regularized Evolution for Image Classifier Architecture Search (Feb 2018)

State of the art

- Finding right architectures: Active area or research



Modular building blocks engineering

see also DenseNets, Wide ResNets, Fractal ResNets, ResNeXts, Pyramidal ResNets

---

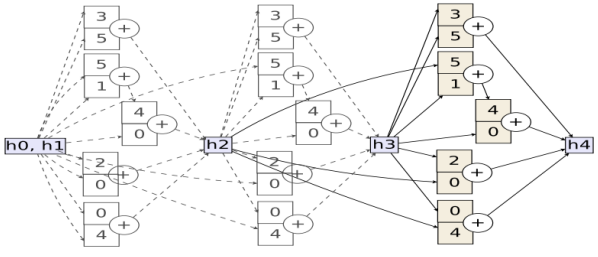
from He slides "Deep residual learning for image recognition." ICML. 2016.

State of the art

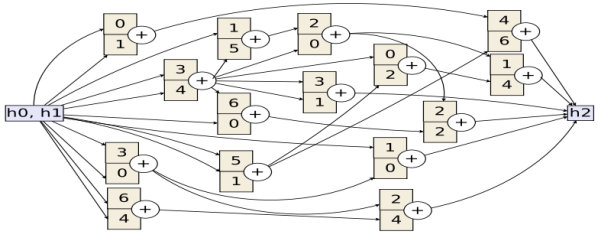
- Finding right architectures: Active area or research

Automated Architecture search:

- reinforcement learning
- evolutionary algorithms



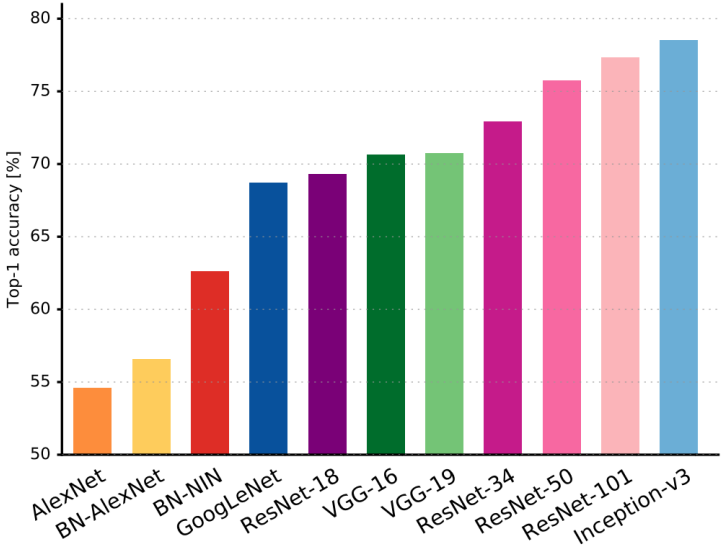
0 = sep. 3x3
1 = sep. 5x5
2 = sep. 7X7
3 = none
4 = avg. pool
5 = max pool
6 = dil. 3x3
7 = 1x7+7x1

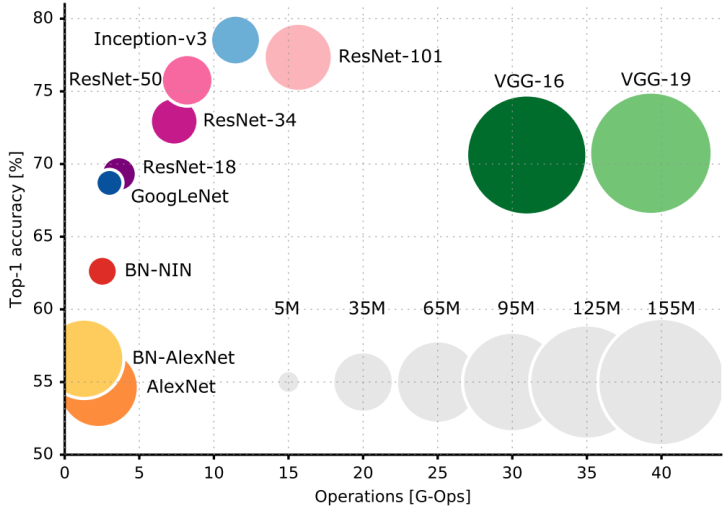


Esteban Real, et al. Regularized Evolution for Image Classifier Architecture Search (Feb 2018)

Comparison of models

Top 1-accuracy, performance and size on ImageNet





Canziani, Paszke, and Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications." (May 2016).

Pre-trained models

Pre-trained models

Training a model on ImageNet from scratch takes days or weeks.

Many models trained on ImageNet and their weights are publicly available!

Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

Pre-trained models

Training a model on ImageNet from scratch takes days or weeks.

Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

Data Augmentation

Data Augmentation

With Keras:

```
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```