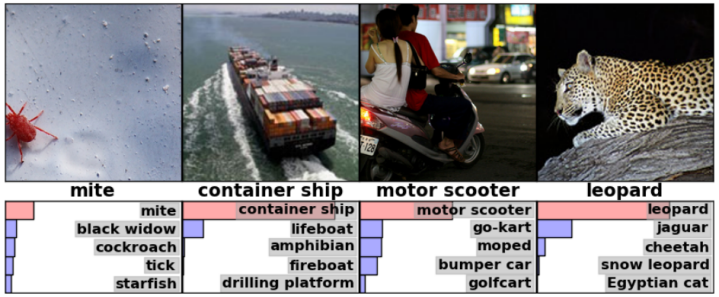
Week 3: Convolutional Neural Networks

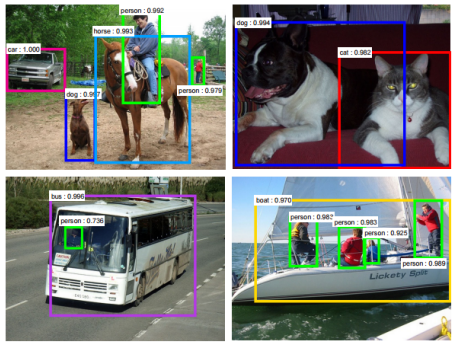
Matthew Willetts - Alexander Camuto

Used everywhere for Vision



[Krizhevsky 2012]

[Ciresan et al. 2013]



[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

Many other applications

Speech recognition & speech synthesis

Many other applications

Speech recognition & speech synthesis

Natural Language Processing

Many other applications

Speech recognition & speech synthesis

Natural Language Processing

Protein/DNA binding prediction

Many other applications

Speech recognition & speech synthesis

Natural Language Processing

Protein/DNA binding prediction

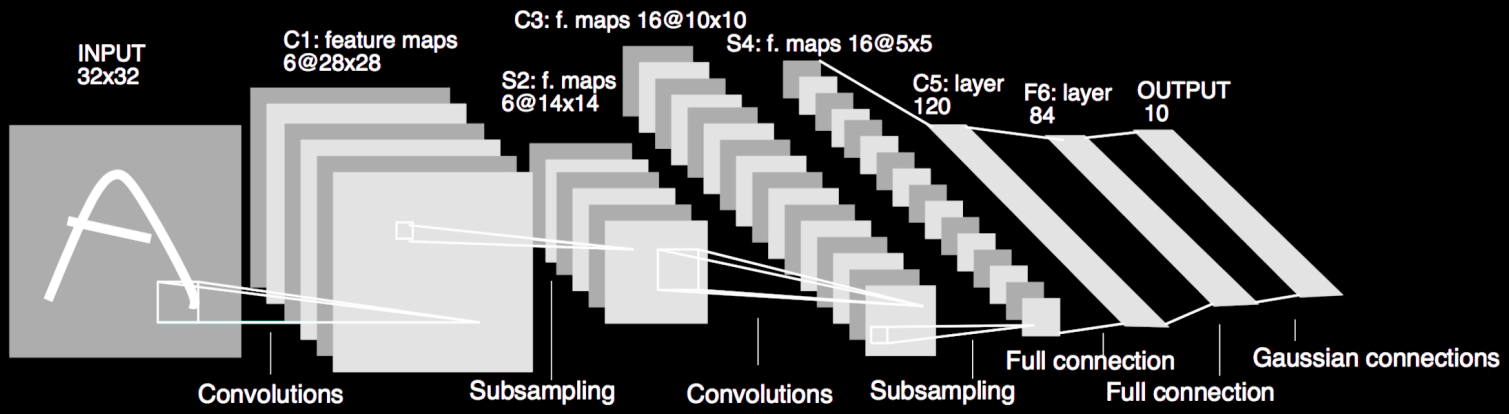
Any problem with a spatial (or sequential) structure

ConvNets for image classification

CNN = Convolutional Neural Networks = ConvNet

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

Outline

Convolutions

CNNs for Image Classification

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?

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?

$640 \times 480 \times 3 \times 1000 + 1000 = 922M!$

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`

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`

We 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)
```

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)
```

Fully Connected Network: MLP

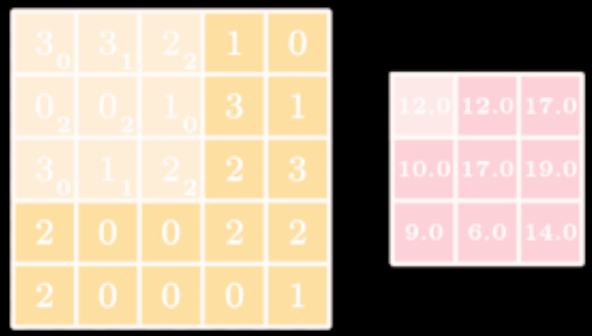
```
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

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`**.

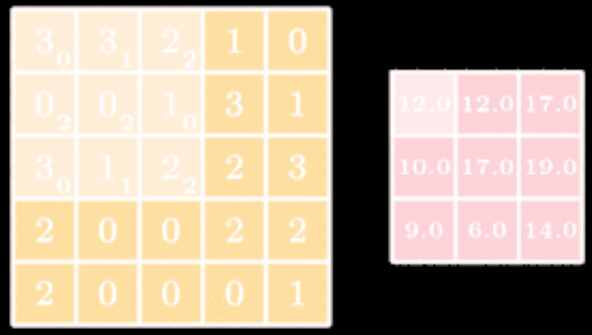
Convolution in a neural network



- $\boldsymbol{x}$ is a $\mathbf{3} \times \mathbf{3}$ chunk (pink area) of the image *(yellow array)*
- Each output neuron is parametrized with the $\mathbf{3} \times \mathbf{3}$ 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:

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

Motivations

Local connectivity

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

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!

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:

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

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):

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

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):

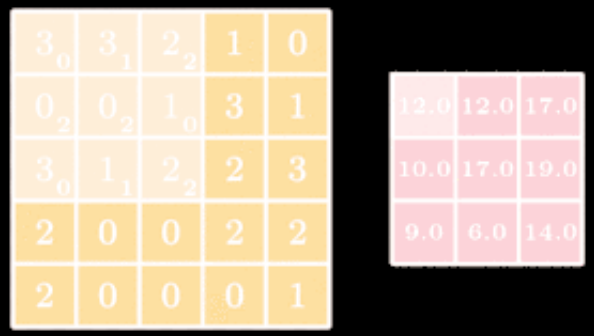
$$(f \star g)(x, y) = \sum_{n} \sum_{m} f(n, m).\, g(x+n, y+m)$$

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

Example: convolution image

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

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



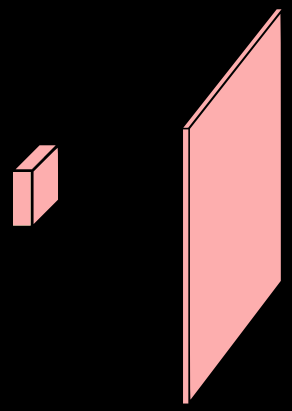
Channels

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

Channels

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

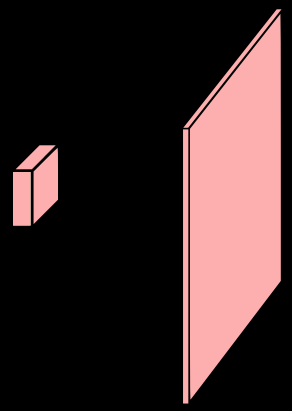
Convolutions are usually computed for each channel and summed:



Channels

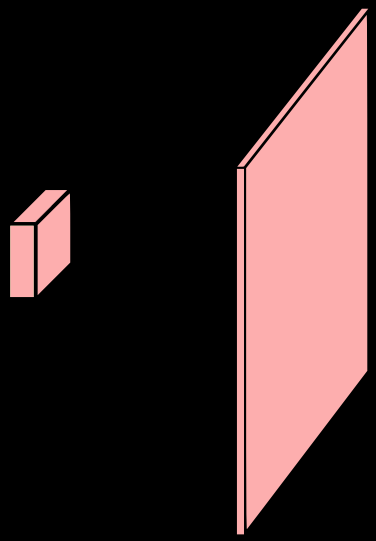
Colored image = tensor of shape `(height, width, channels)`

Convolutions are usually computed for each channel and summed:

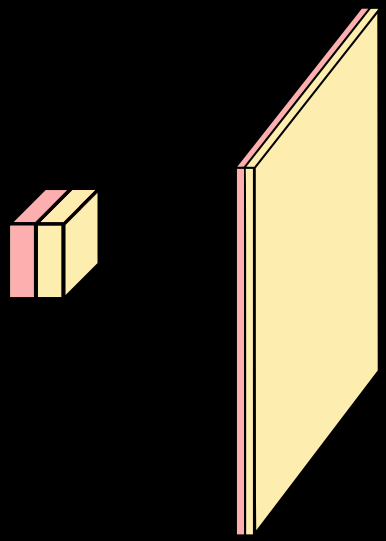


$$(k \star im^{color}) = \sum_{c=0}^{2} k^c \star im^c$$

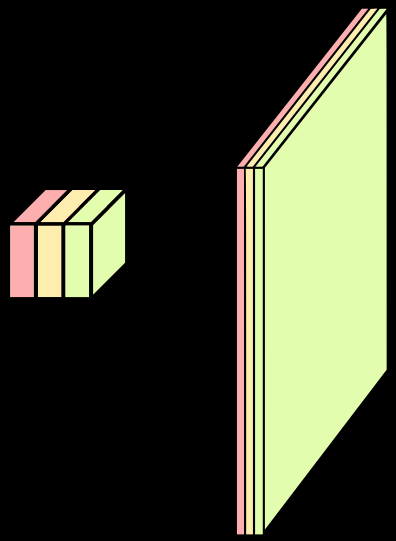
Multiple convolutions



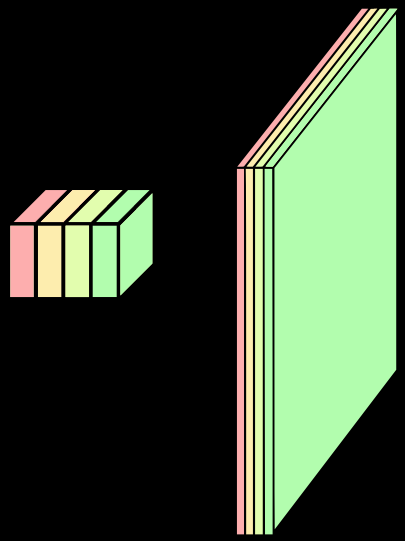
Multiple convolutions



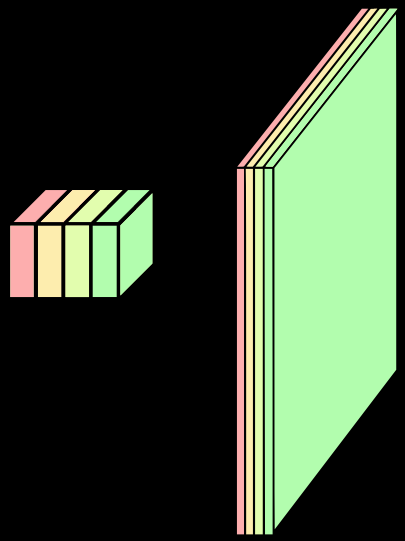
Multiple convolutions



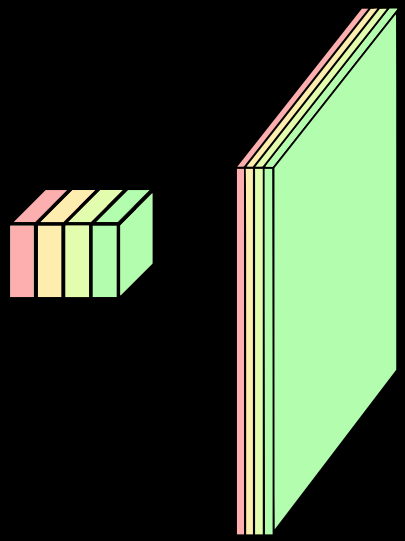
Multiple convolutions



Multiple convolutions



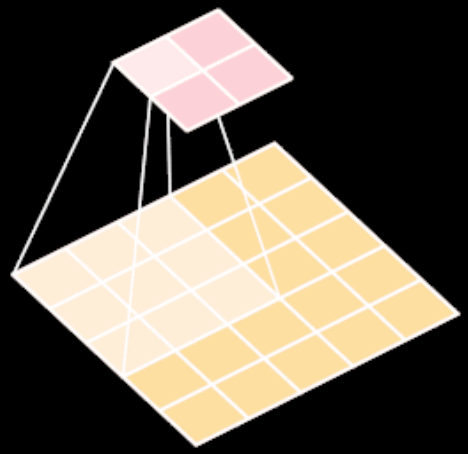
Multiple convolutions



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

Strides

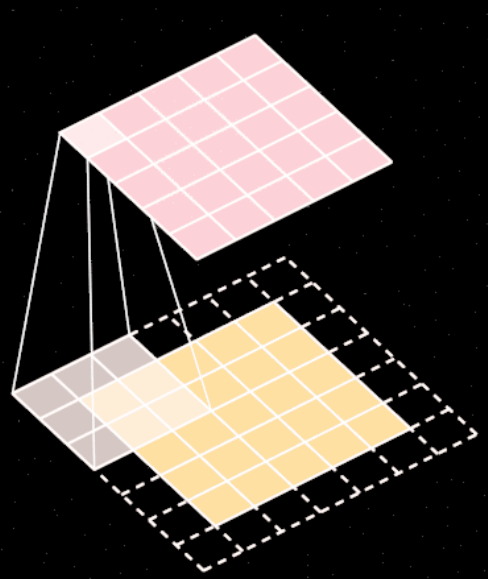
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



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

Padding

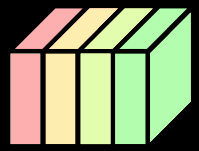
- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



Dealing with shapes

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

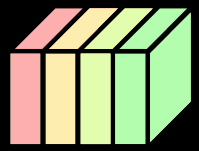
- $F \times F$ kernel size,
- C^i input channels
- C^o output channels



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

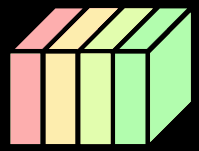


Number of parameters: $(F \times F \times C^i + 1) \times C^o$

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

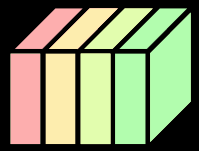
Activations or **Feature maps** shape:

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

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)

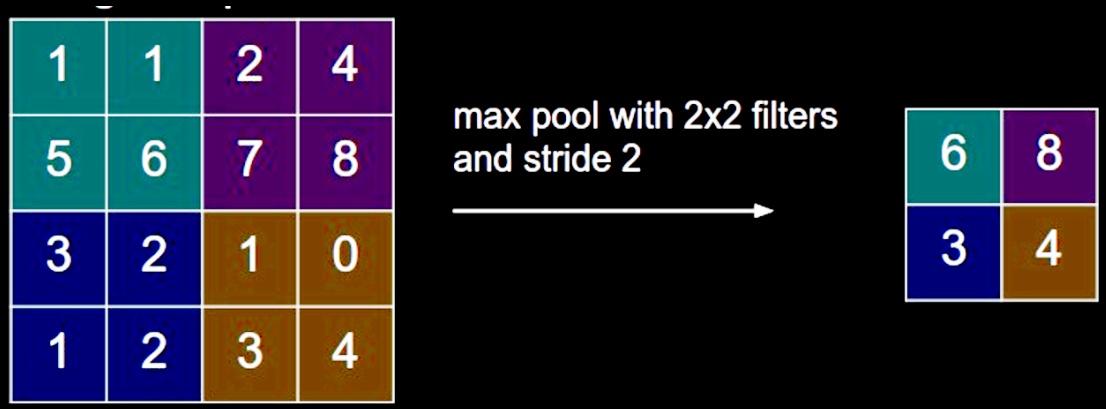
$W^o = (W^i - F + 2P)/S + 1$

Pooling

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

Pooling

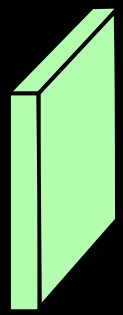
- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



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

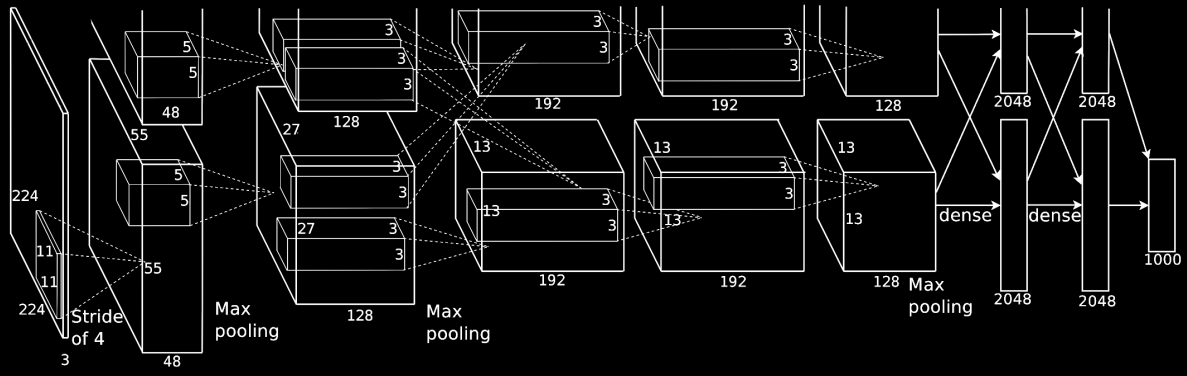
Conv blocks

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

Output

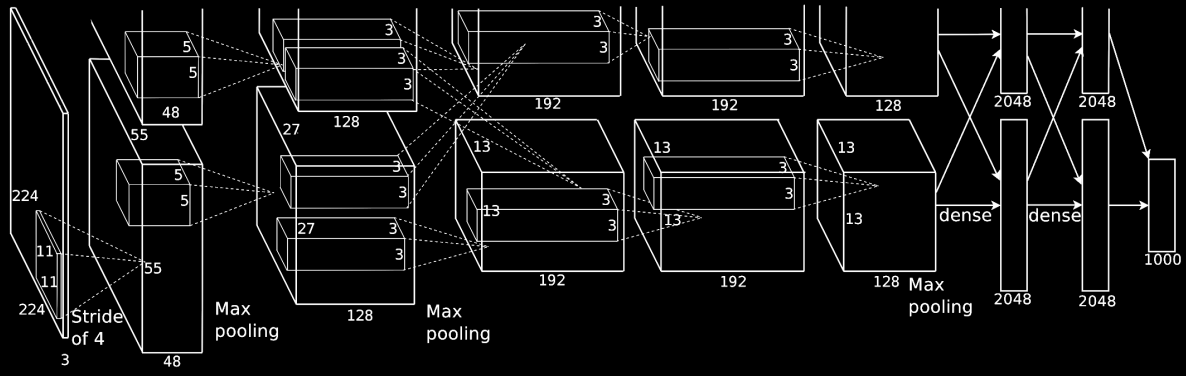
- Fully connected layers
- Softmax

AlexNet



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

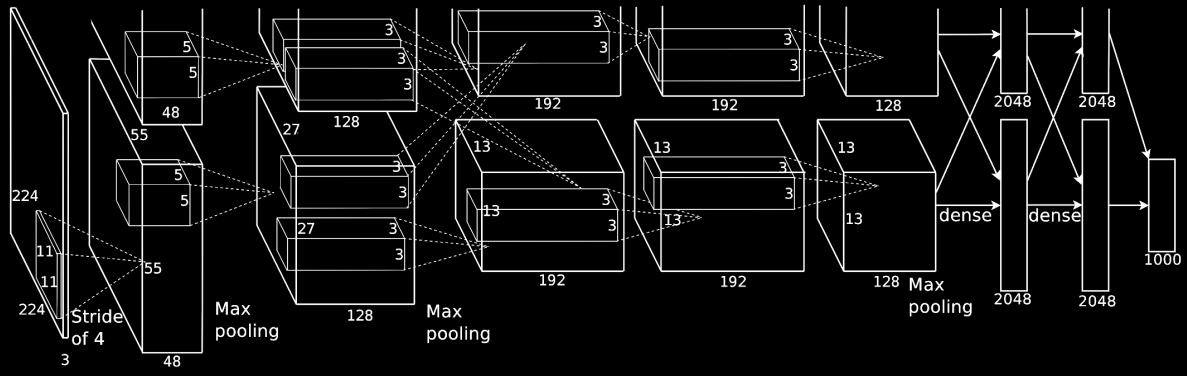
AlexNet



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

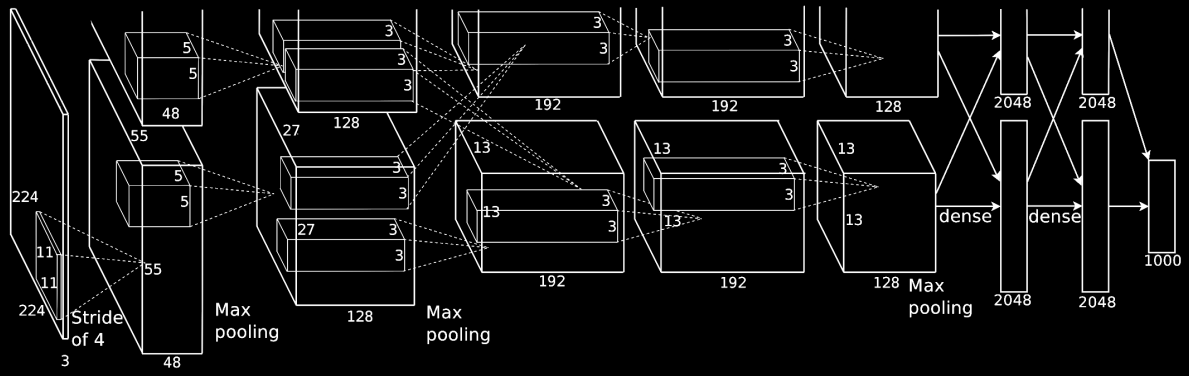


First conv layer: kernel 11x11x3x96 stride 4

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

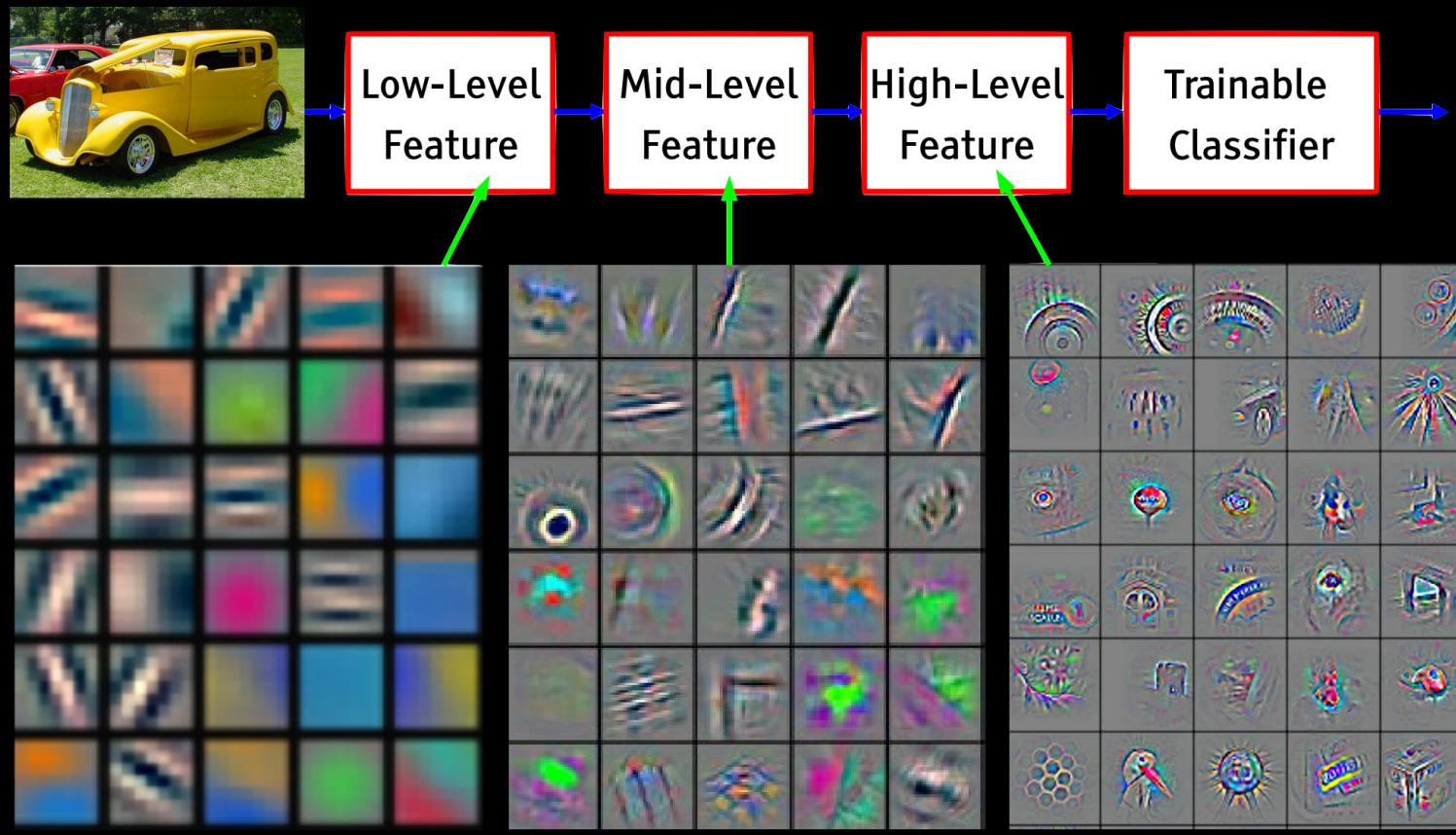
Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

AlexNet

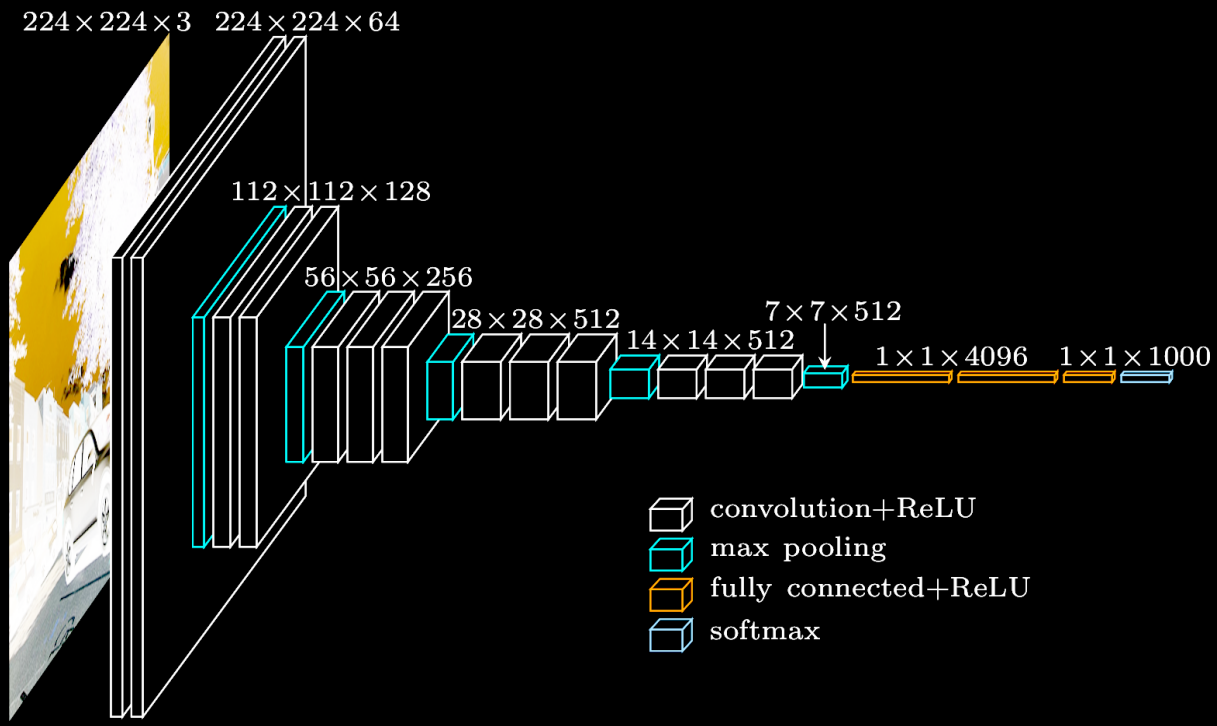



```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride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



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

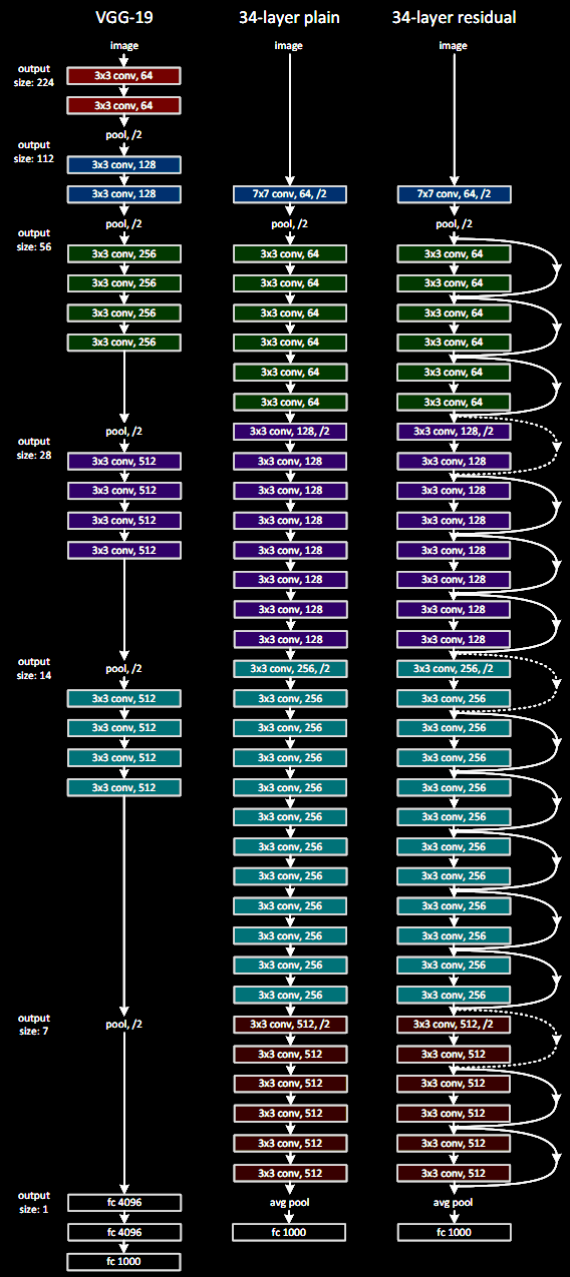
```
           Activation maps            Parameters
INPUT:     [224x224x3]   = 150K     0
CONV3-64:  [224x224x64]  = 3.2M     (3x3x3)x64    =       1,728
CONV3-64:  [224x224x64]  = 3.2M     (3x3x64)x64   =      36,864
POOL2:     [112x112x64]  = 800K     0
CONV3-128: [112x112x128] = 1.6M     (3x3x64)x128  =      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

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

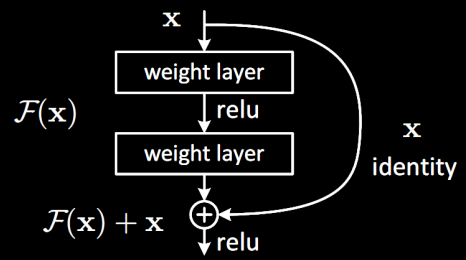
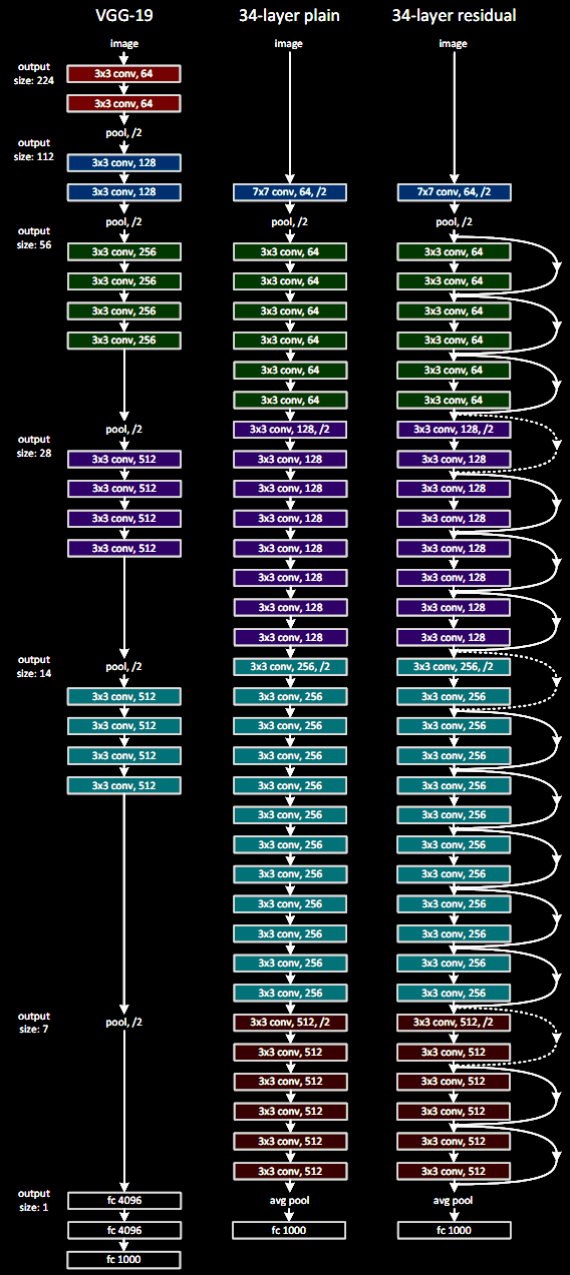


Figure 2. Residual learning: a building block.



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

ResNet

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

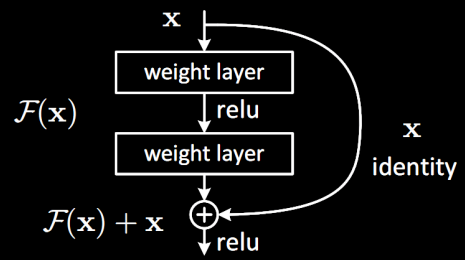
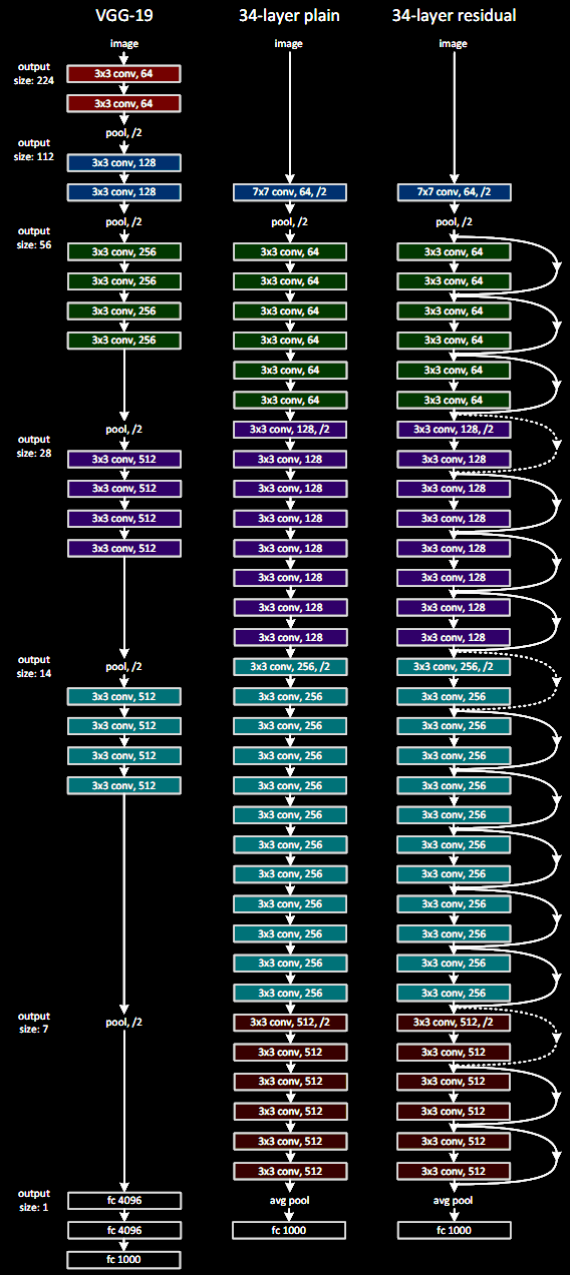


Figure 2. Residual learning: a building block.

- Good optimization properties

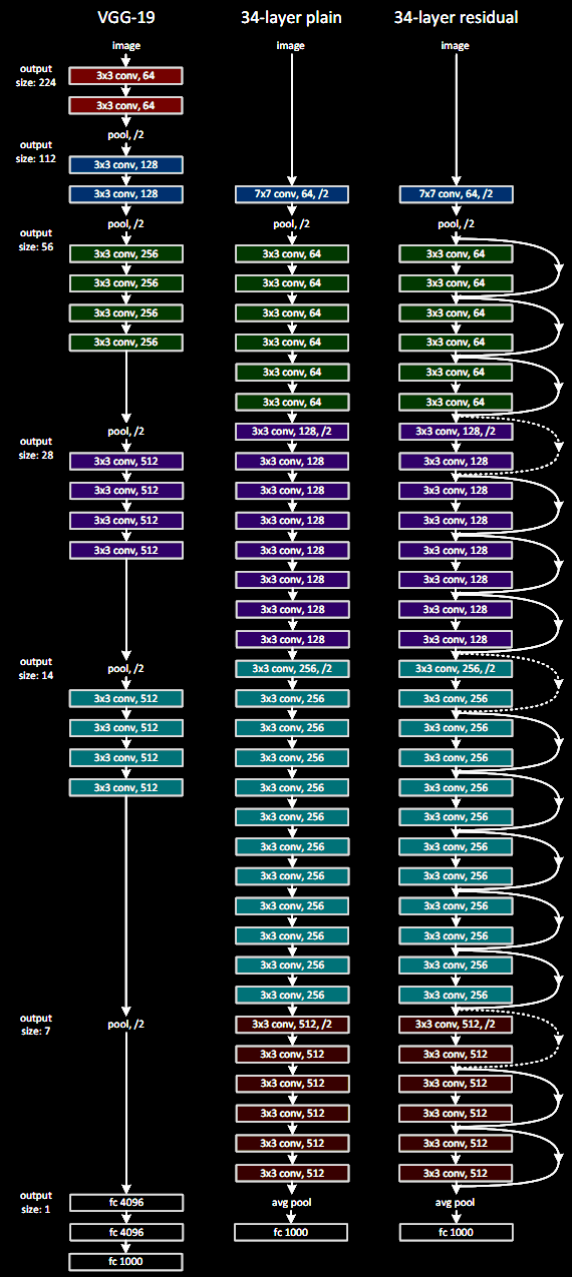


He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.

ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%



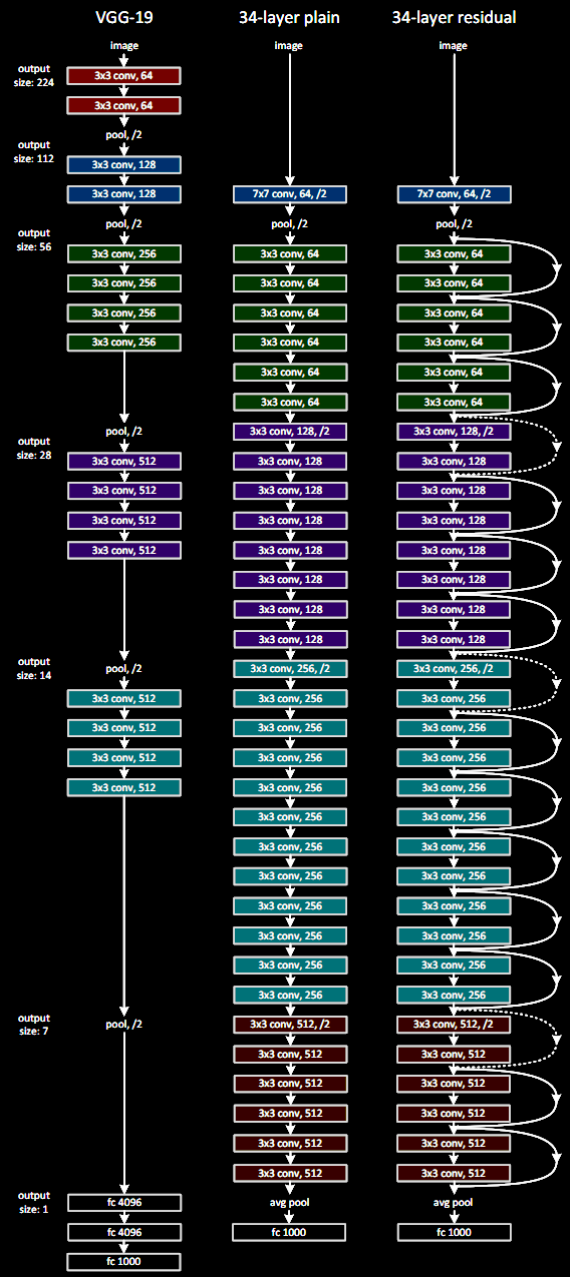
He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.

ResNet

ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%

Less parameters
25M vs 138M



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

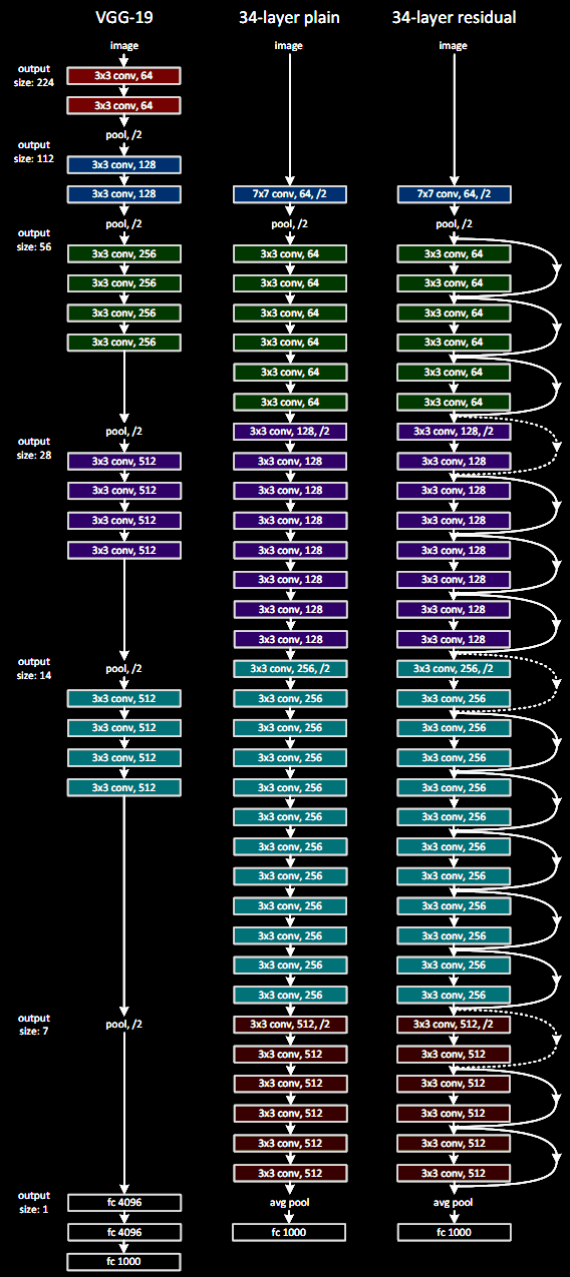
ResNet

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



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

ResNet

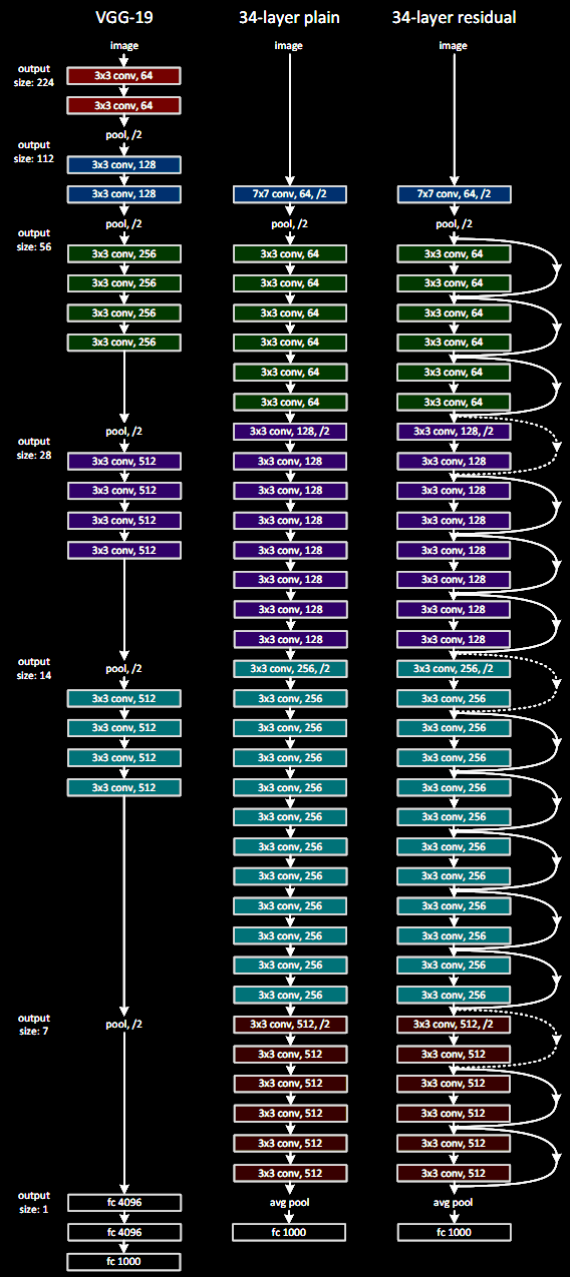
ResNet50 Compared to VGG:

Superior accuracy in all vision tasks
5.25% top-5 error vs 7.1%

Less parameters
25M vs 138M

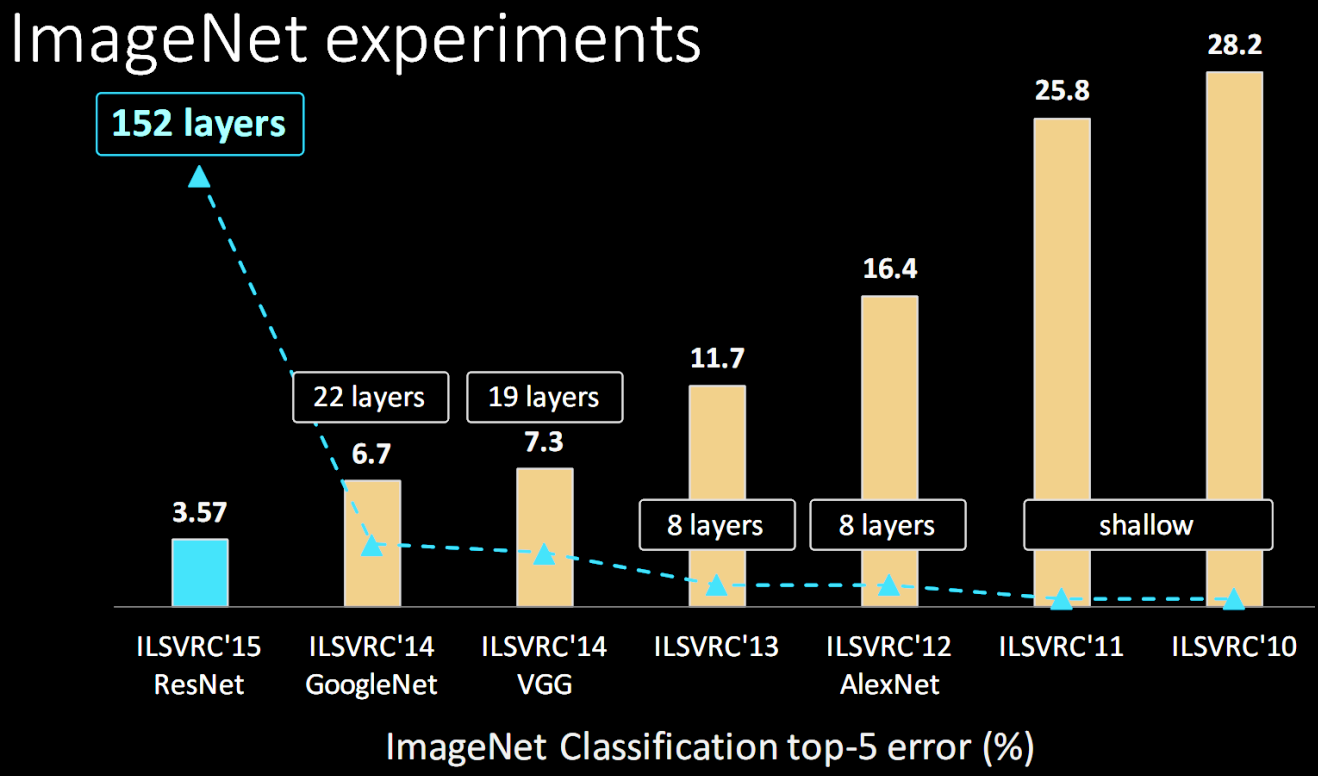
Computational complexity
3.8B Flops vs 15.3B Flops

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

State of the art

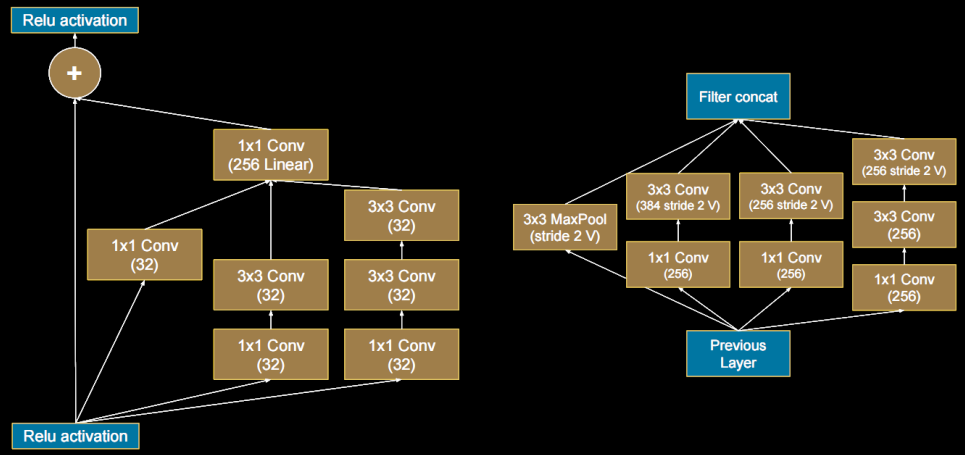
- Finding right architectures: Active area or research

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

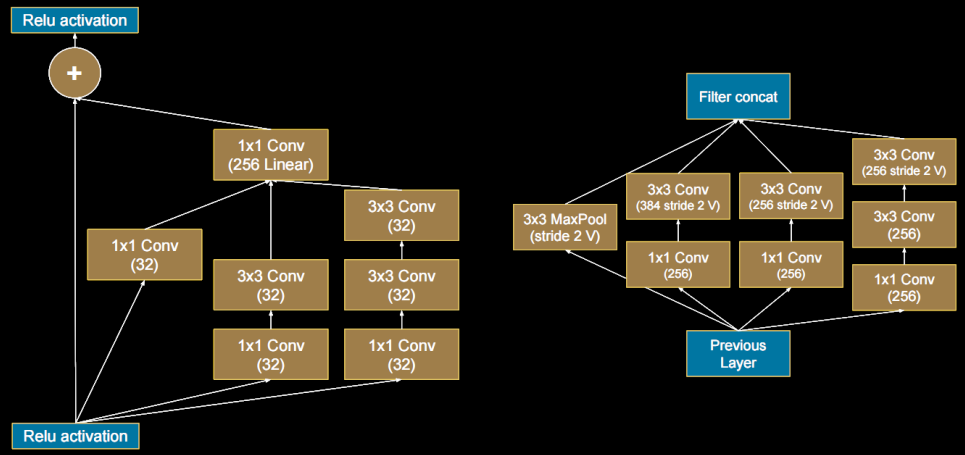


Modular building blocks engineering

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

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

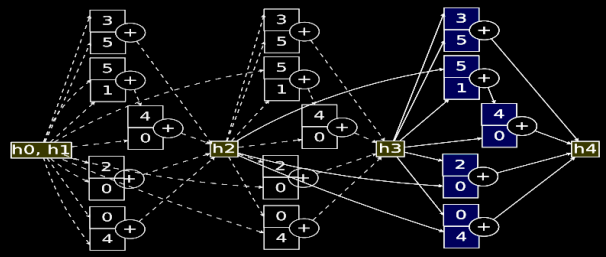
from He slides "Deep residual learning for image recognition." ICML. 2016.

State of the art

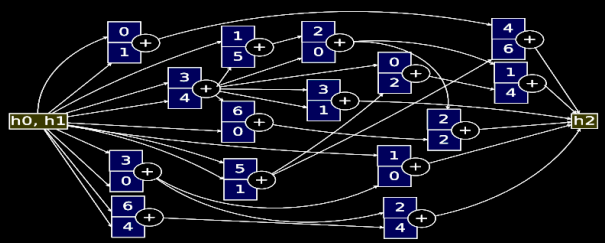
- Finding right architectures: Active area or research

Automated Architecture search:

- reinforcement learning
- evolutionary algorithms



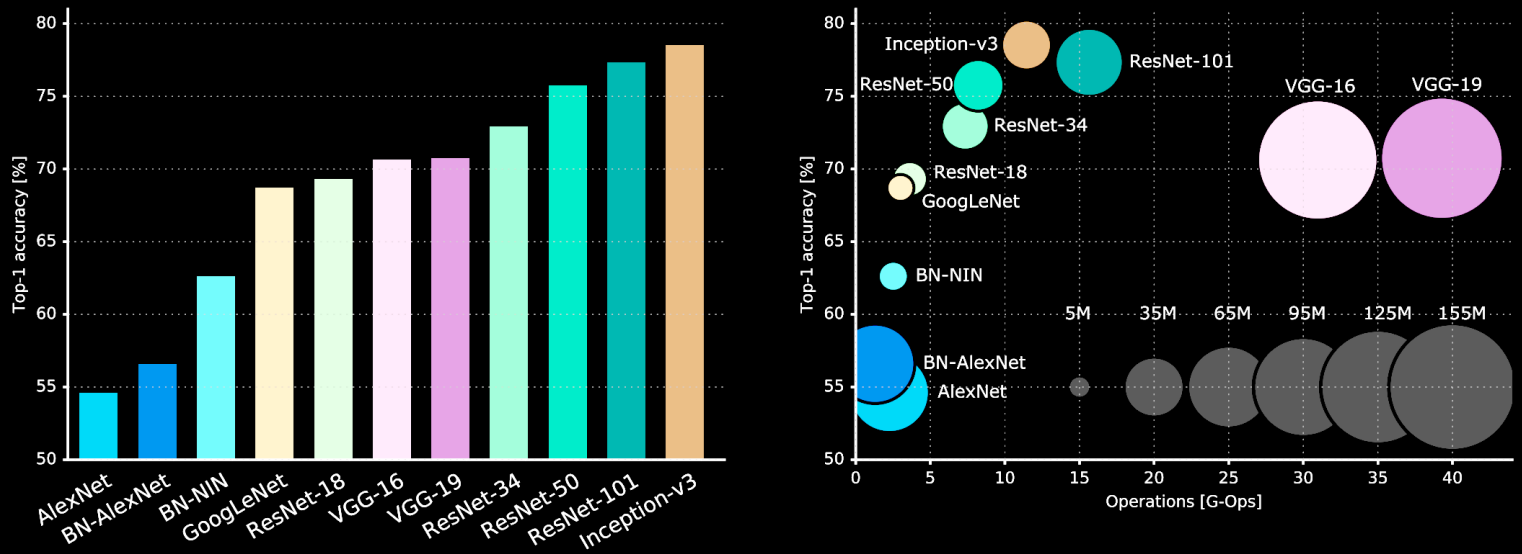
0 = sep. 3x3
1 = sep. 5x5
2 = sep. 7X7
3 = none
4 = avg. pool
5 = max pool
6 = dil. 3x3
7 = 1x7+7x1



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**.

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!

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)

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

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!

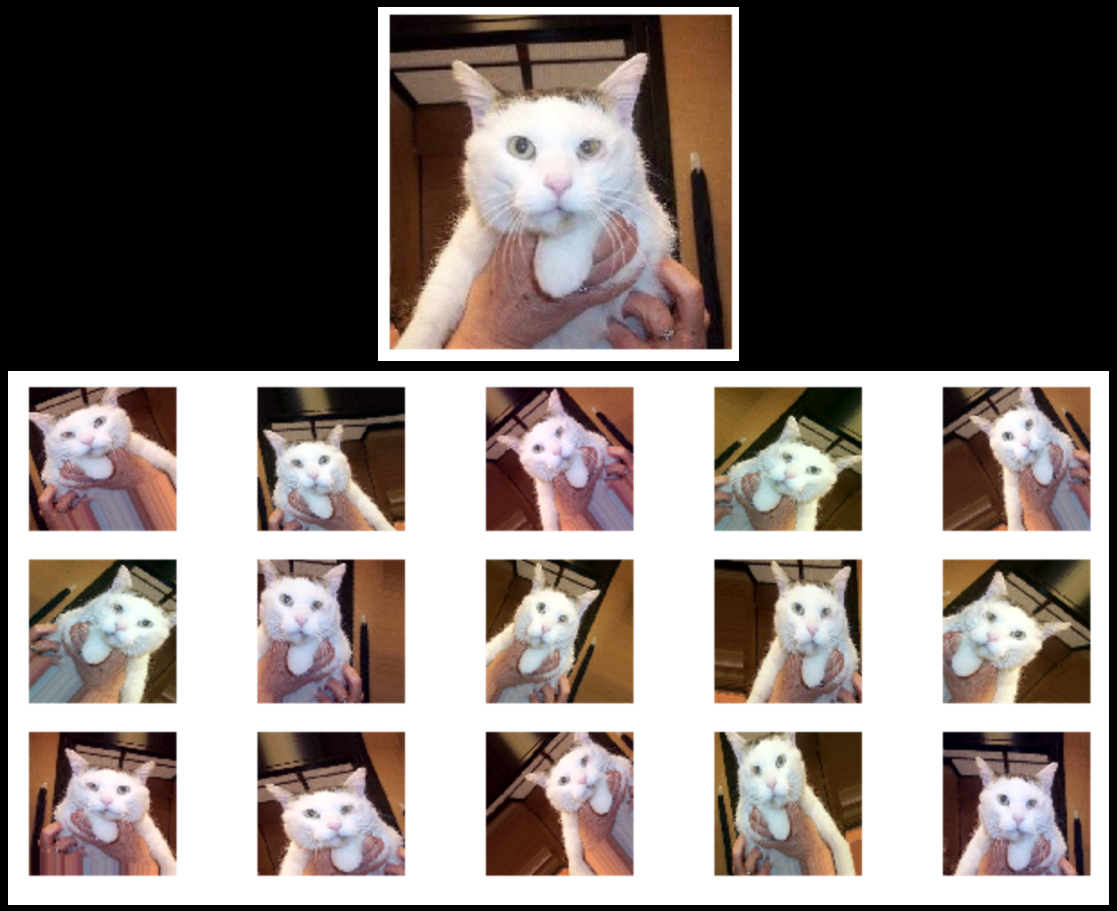
Fine-tuning

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



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)
```

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To im-

Skin Disease Diagnosis

LETTER

doi:10.1038/nature21056

Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva[1]*, Brett Kuprel[1]*, Roberto A. Novoa[2,3], Justin Ko[2], Susan M. Swetter[2,4], Helen M. Blau[5] & Sebastian Thrun[6]

Skin cancer, the most common human malignancy[1–3], is primarily diagnosed visually, beginning with an initial clinical screening and followed potentially by dermoscopic analysis, a biopsy and histopathological examination. Automated classification of skin lesions using images is a challenging task owing to the fine-grained variability in the appearance of skin lesions. Deep convolutional neural networks (CNNs)[4,5] show potential for general and highly variable tasks across many fine-grained object categories[6–11]. Here we demonstrate classification of skin lesions using a single CNN, trained end-to-end from images directly, using only pixels and disease labels as inputs. We train a CNN using a dataset of 129,450 clinical images—two orders of magnitude larger than previous datasets[12]—consisting of 2,032 different diseases. We test its performance against 21 board-certified dermatologists on biopsy-proven clinical images with two critical binary classification use cases: keratinocyte carcinomas versus benign seborrheic keratoses; and malignant melanomas versus benign nevi. The first case represents the identification of the most common cancers, the second represents the identification of the deadliest skin cancer. The CNN achieves performance on par with all tested experts across both tasks, demonstrating an artificial intelligence capable of classifying skin cancer with a level of competence comparable to dermatologists. Outfitted with deep neural networks, mobile devices

images (for example, smartphone images) exhibit variability in factors such as zoom, angle and lighting, making classification substantially more challenging[23,24]. We overcome this challenge by using a data-driven approach—1.41 million pre-training and training images make classification robust to photographic variability. Many previous techniques require extensive preprocessing, lesion segmentation and extraction of domain-specific visual features before classification. By contrast, our system requires no hand-crafted features; it is trained end-to-end directly from image labels and raw pixels, with a single network for both photographic and dermoscopic images. The existing body of work uses small datasets of typically less than a thousand images of skin lesions[16,18,19], which, as a result, do not generalize well to new images. We demonstrate generalizable classification with a new dermatologist-labelled dataset of 129,450 clinical images, including 3,374 dermoscopy images.

Deep learning algorithms, powered by advances in computation and very large datasets[25], have recently been shown to exceed human performance in visual tasks such as playing Atari games[26], strategic board games like Go[27] and object recognition[6]. In this paper we outline the development of a CNN that matches the performance of dermatologists at three key diagnostic tasks: melanoma classification, melanoma classification using dermoscopy and carcinoma classification. We restrict the comparisons to image-based classification.

Saliency Maps

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan **Andrea Vedaldi** **Andrew Zisserman**
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

Abstract

This paper addresses the visualisation of image classification models, learnt using deep Convolutional Networks (ConvNets). We consider two visualisation techniques, based on computing the gradient of the class score with respect to the input image. The first one generates an image, which maximises the class score [5], thus visualising the notion of the class, captured by a ConvNet. The second technique computes a class saliency map, specific to a given image and class. We show that such maps can be employed for weakly supervised object segmentation using classification ConvNets. Finally, we establish the connection between the gradient-based ConvNet visualisation methods and deconvolutional networks [13].

1 Introduction

With the deep Convolutional Networks (ConvNets) [10] now being the architecture of choice for large-scale image recognition [4, 8], the problem of understanding the aspects of visual appearance, captured inside a deep model, has become particularly relevant and is the subject of this paper.

Thanks to

Charles Ollion and Olivier Grisel, Paris-Saclay, for the slides

Chris Holmes and Gil McVean, as always