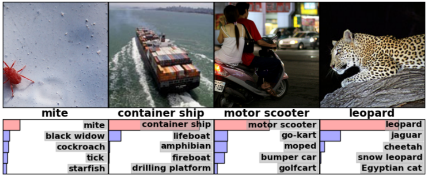
Convolutional Neural Networks (CNN)

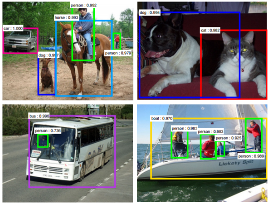
Alejandro Veloz

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

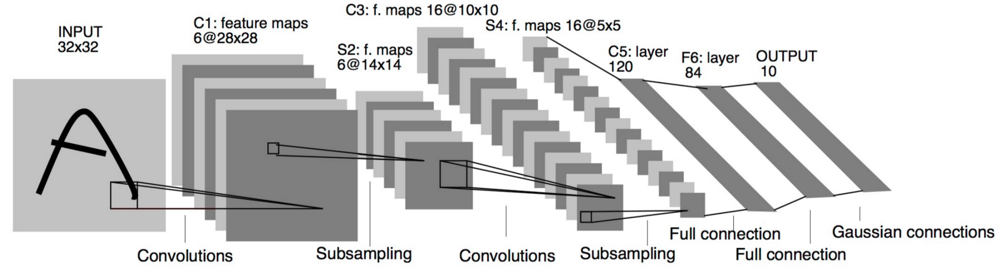
Natural Language Processing

Protein/DNA binding prediction

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`

We never use Dense layers directly on large images. Most standard solution is **convolution** layers

Fully Connected Network: MLP

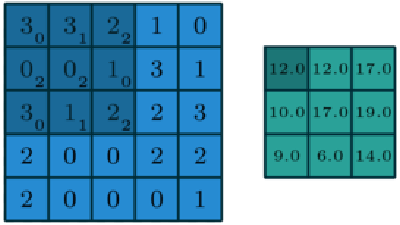
```
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

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‘.

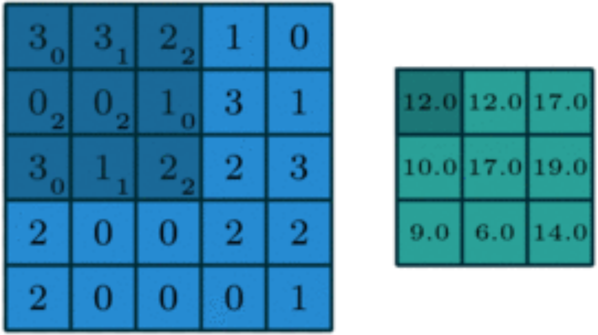
Convolution in a neural network



- 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)*

https://github.com/vdumoulin/conv_arithmetic

Convolution in a neural network

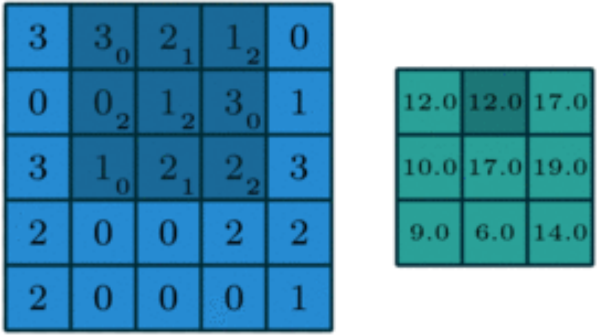


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

Convolution in a neural network

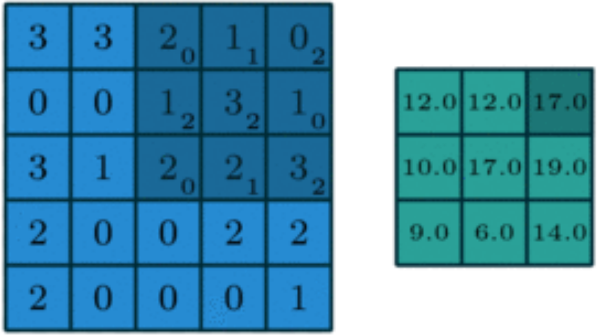


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

Convolution in a neural network

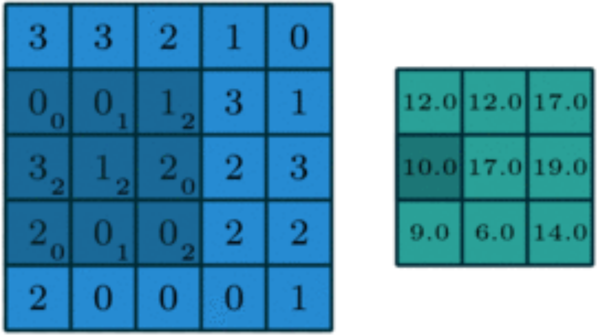


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

Convolution in a neural network

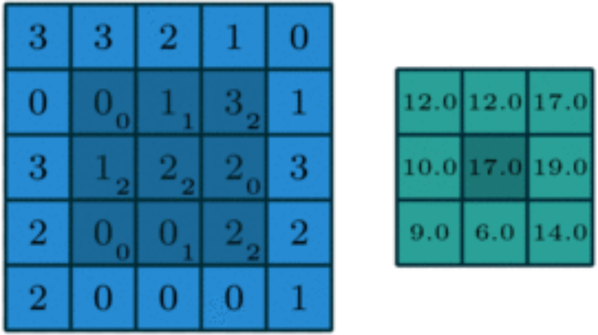


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

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:

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

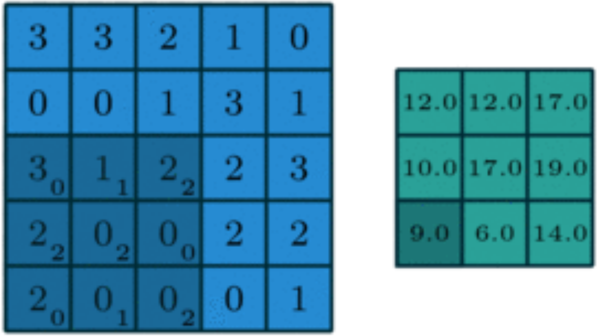
Convolution in a neural network

- 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:

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

Convolution in a neural network

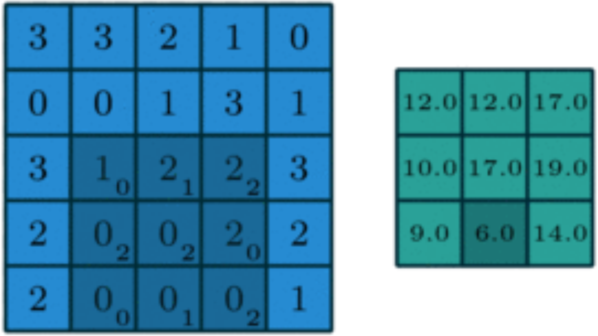


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

Convolution in a neural network

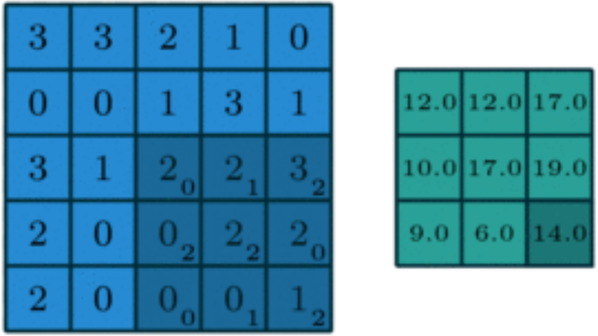


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

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

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 NEURONS 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):

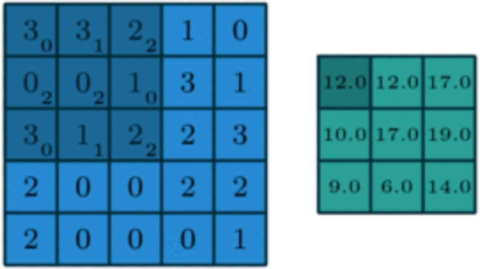
$$(f \star g)(x,y) = \sum_{n} \sum_{m} f(n,m)\, g(x+n, y+m)$$

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

Example: convolution image

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

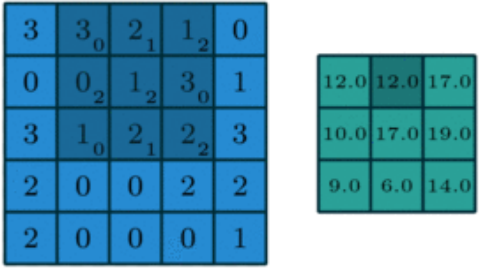
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Example: convolution image

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

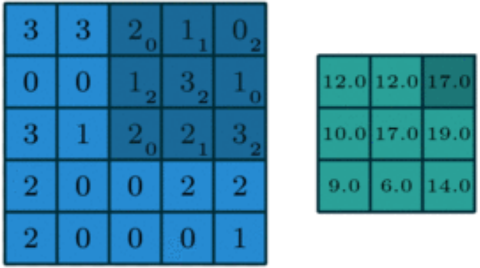
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Example: convolution image

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

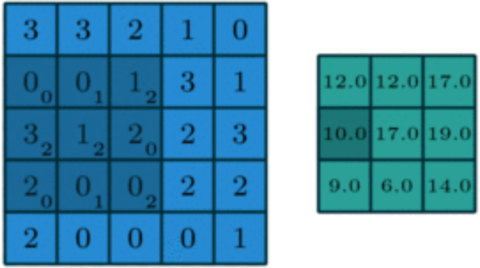
$$(k \star im)(x,y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n,m)\, im(x+n-1, y+m-1)$$



Example: convolution image

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

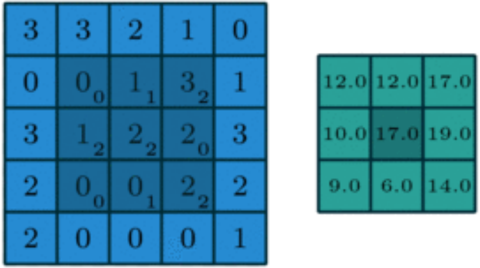
$$(k \star im)(x,y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n,m)\, im(x+n-1, y+m-1)$$



Example: convolution image

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

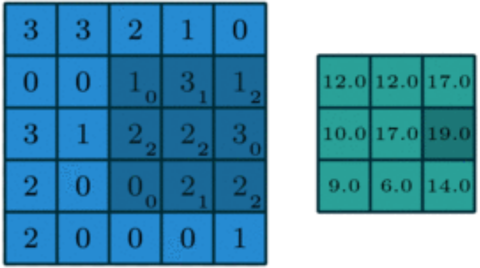
$$(k \star im)(x,y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n,m)\, im(x+n-1, y+m-1)$$



Example: convolution image

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

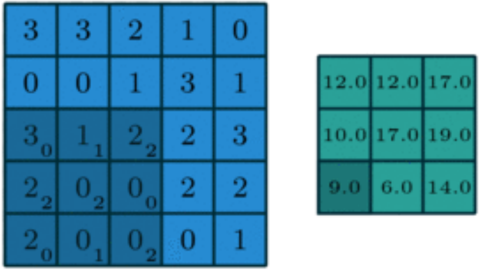
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Example: convolution image

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

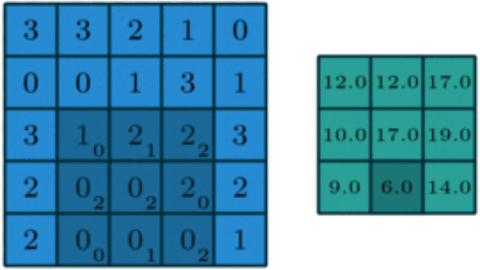
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Example: convolution image

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

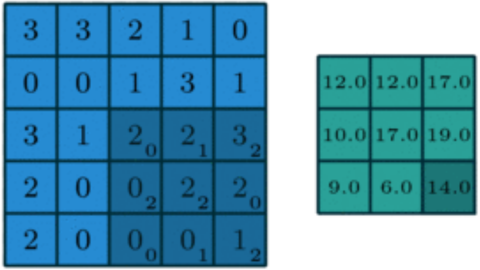
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Example: convolution image

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

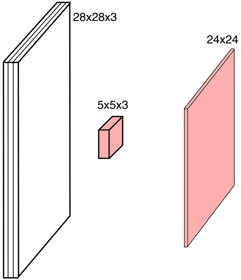
$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m)\, im(x + n - 1, y + m - 1)$$



Channels

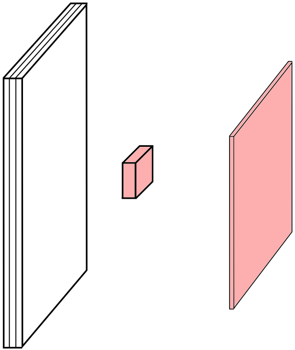
Colored image = tensor of shape `(height, width, channels)`

Convolutions are usually computed for each channel and summed:

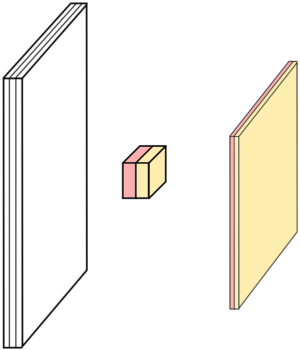


$$(k \star im^{color}) = \sum_{c=0}^{2} k^c \star im^c$$

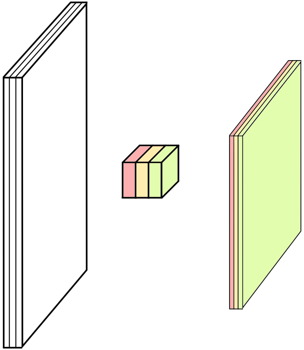
Multiple convolutions



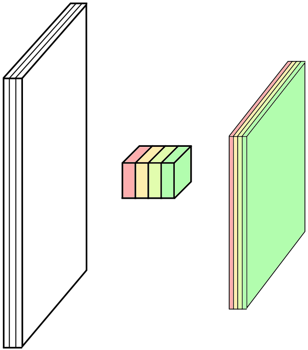
Multiple convolutions



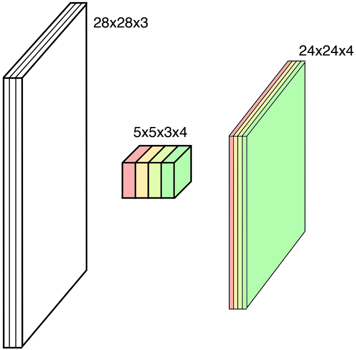
Multiple convolutions



Multiple convolutions



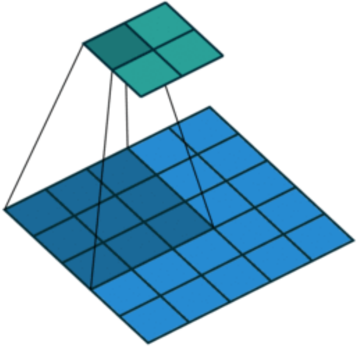
Multiple convolutions



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

Strides

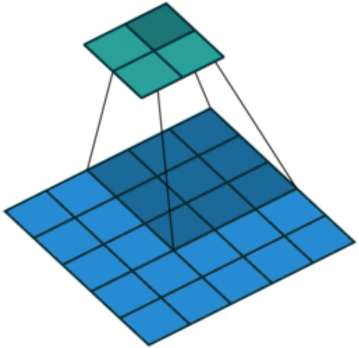
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



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

Strides

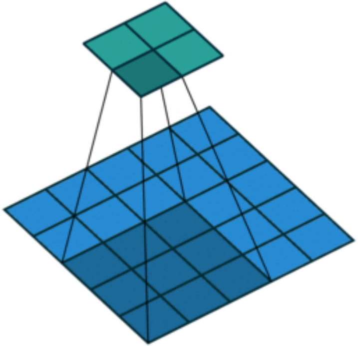
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



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

Strides

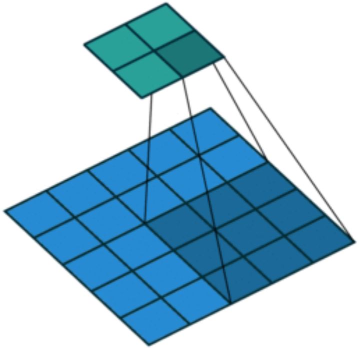
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



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

Strides

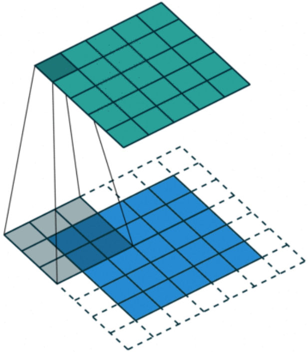
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



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

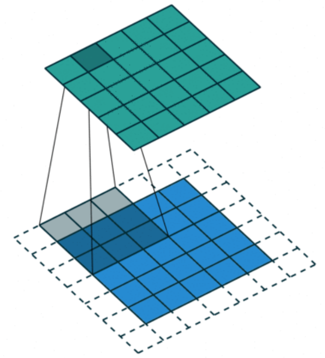
Padding

- 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



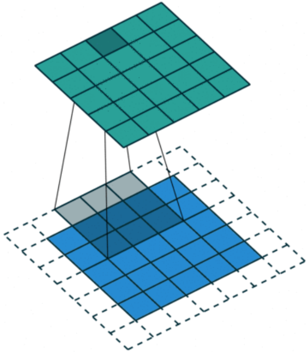
Padding

- 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



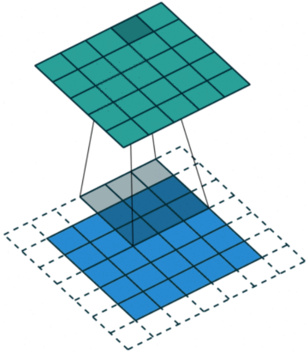
Padding

- 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



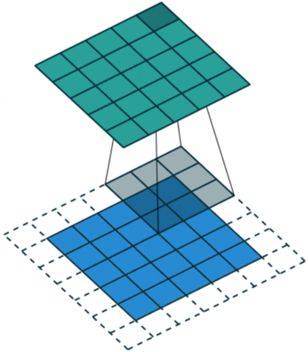
Padding

- 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



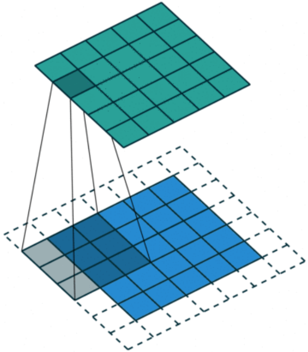
Padding

- 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



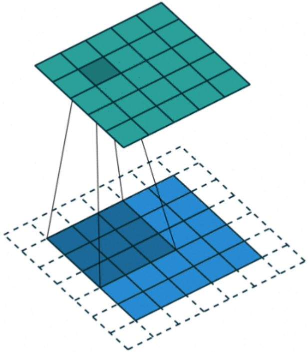
Padding

- 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



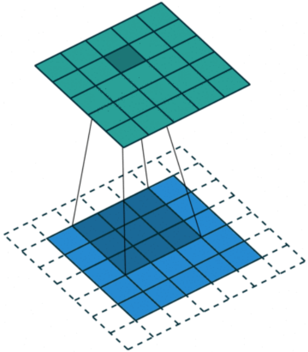
Padding

- 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



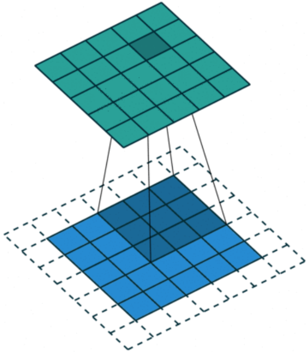
Padding

- 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



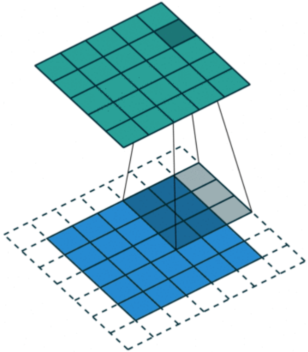
Padding

- 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



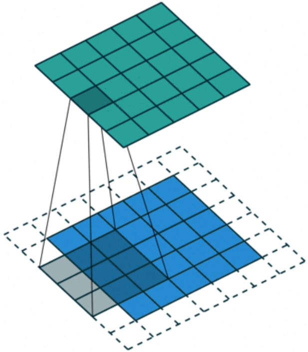
Padding

- 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



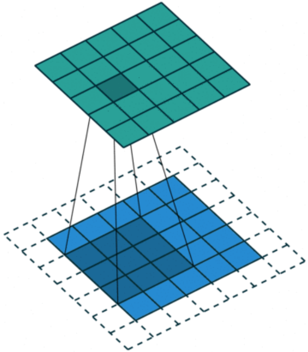
Padding

- 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



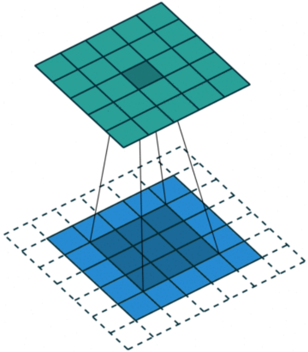
Padding

- 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



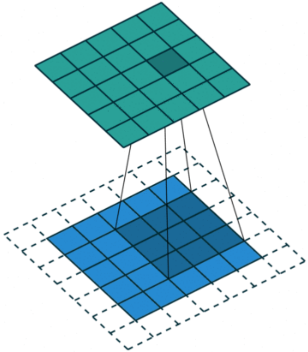
Padding

- 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



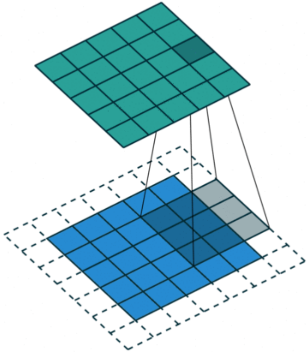
Padding

- 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



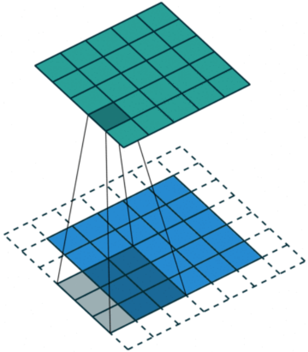
Padding

- 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



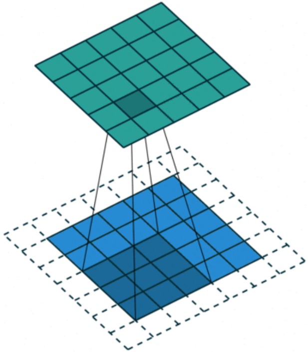
Padding

- 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



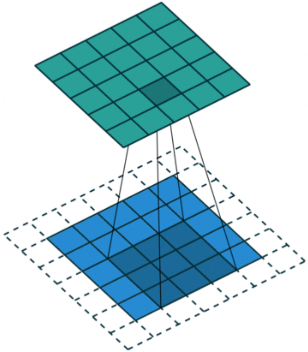
Padding

- 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



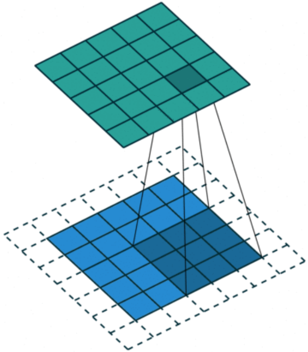
Padding

- 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



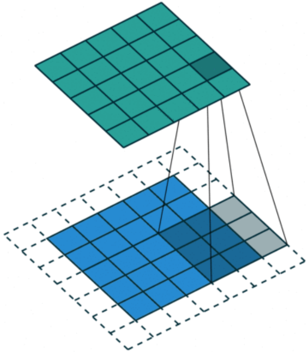
Padding

- 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



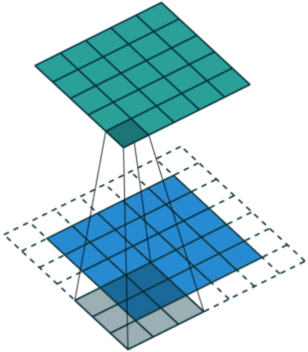
Padding

- 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



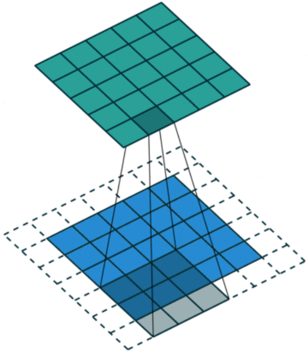
Padding

- 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



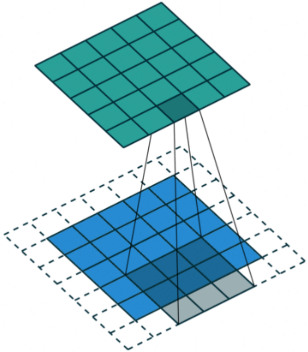
Padding

- 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



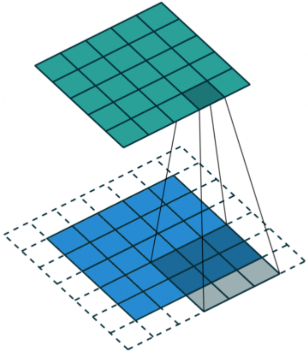
Padding

- 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



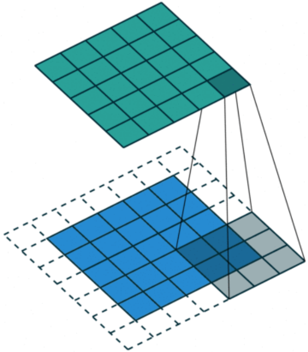
Padding

- 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



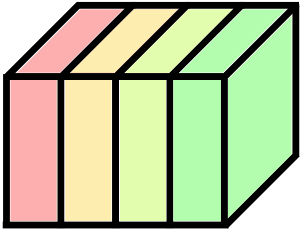
Padding

- 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

5x5x3x4



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:

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

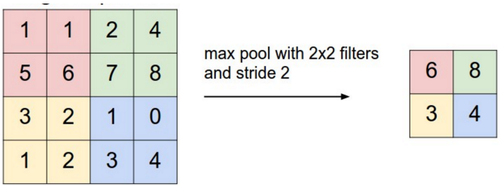
Activations or **Feature maps** shape:

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

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

Pooling

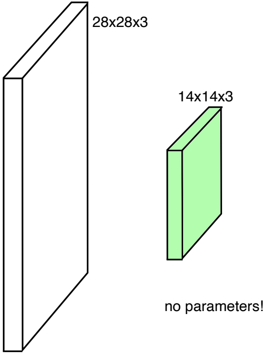
- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



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

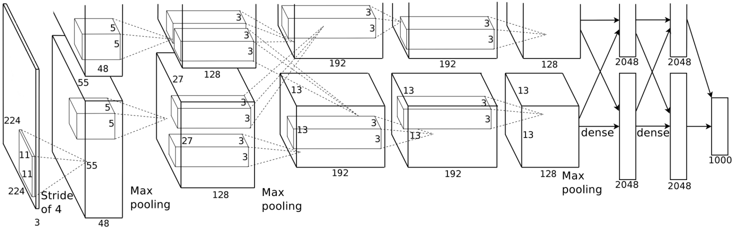
Conv blocks

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

Output

- Fully connected layers
- Softmax

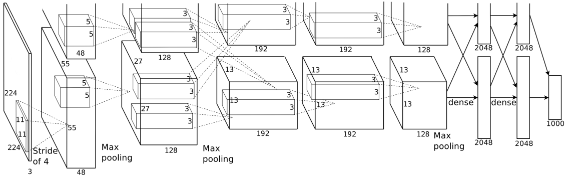
AlexNet



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

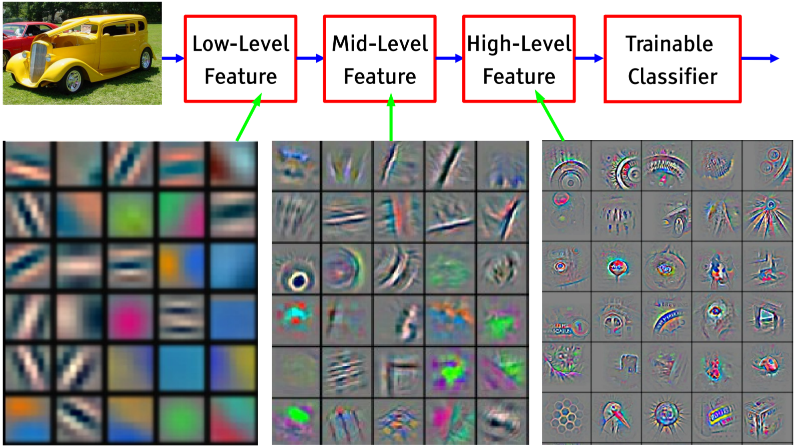
- Input: 227x227x3 image
- First conv layer: kernel 11x11x3x96 stride 4
- Kernel shape: `(11,11,3,96)`
- Output shape: `(55,55,96)`
- Number of parameters: `34,944`
- Equivalent MLP parameters: `43.7 x 1e9`

AlexNet



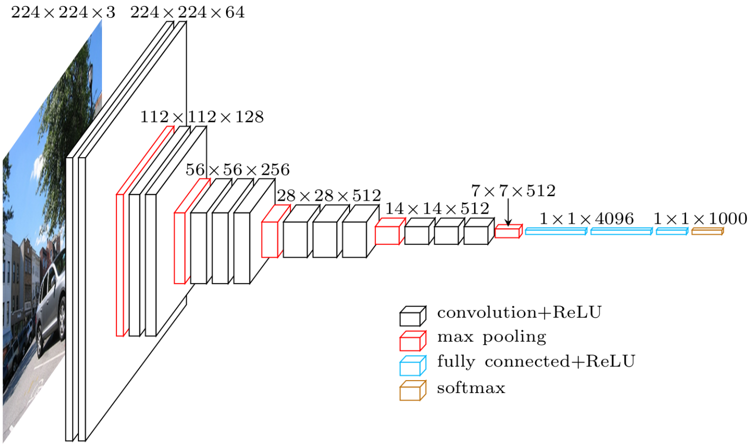

```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride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

ResNet

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

- Even deeper models: 34, 50, 101, 152 layers
- A block learns the residual w.r.t. identity

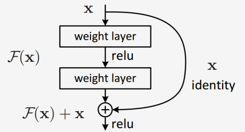
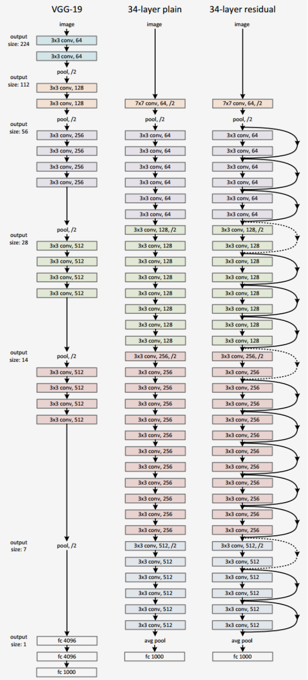


Figure 2. Residual learning: a building block.

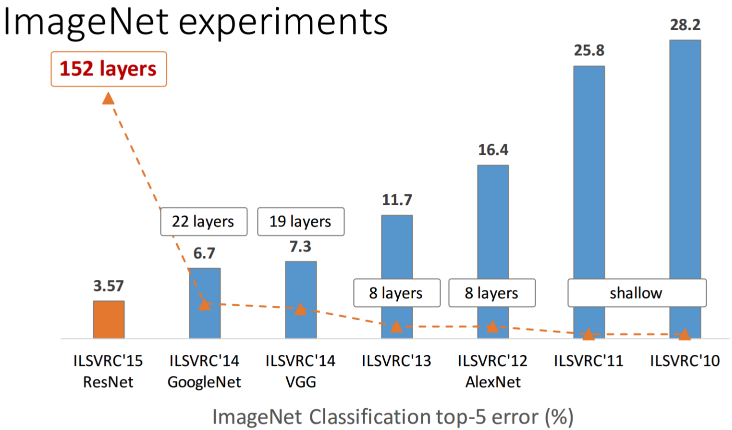
- Good optimization properties

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



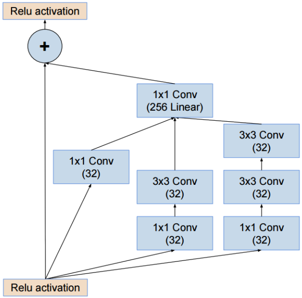
Deeper is better

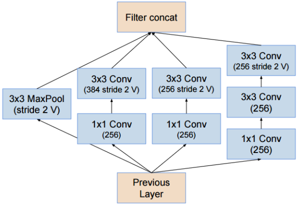


from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

State of the art

- Finding right architectures: Active area or research





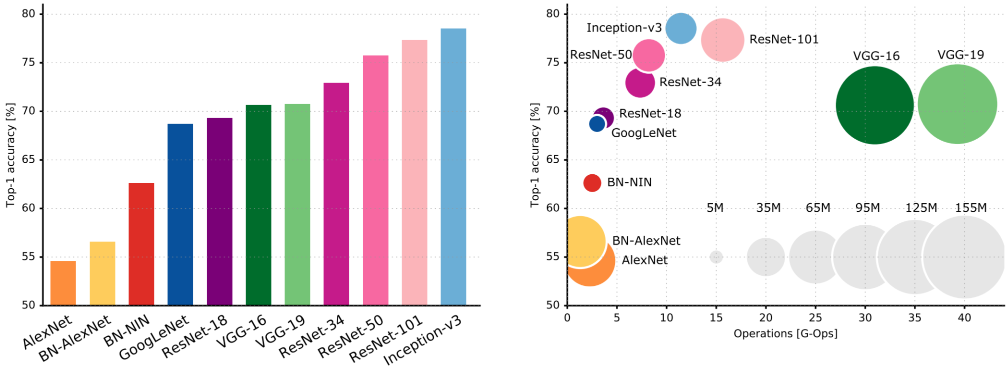
Modular building blocks engineering

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

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

State of the art

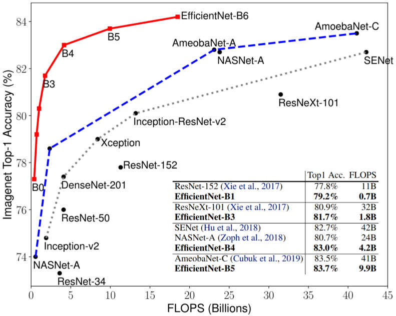
Top 1-accuracy, performance and size on ImageNet



See also: https://paperswithcode.com/sota/image-classification-on-imagenet

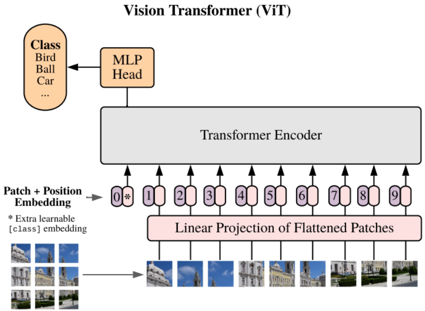
Canziani, Paszke, and Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications." (May 2016).

More ImageNet SOTA



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.2%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.7%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**83.0%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.7%**	**9.9B**

Figure 5. **FLOPS vs. ImageNet Accuracy** – Similar to Figure 1 except it compares FLOPS rather than model size.



- Mingxing Tan, Quoc V. Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, ICML 2019.
- Irwan Bello, LambdaNetworks: Modeling long-range Interactions without Attention, ICLR 2021.
- Dosovitskiy A. et al, An Image is worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

State of the art

Method	# Params	Extra Data	ImageNet Top-1	ImageNet Top-5	ImageNet-ReaL [6] Precision@1
ResNet-50 [24]	26M	–	76.0	93.0	82.94
ResNet-152 [24]	60M	–	77.8	93.8	84.79
DenseNet-264 [28]	34M	–	77.9	93.9	–
Inception-v3 [62]	24M	–	78.8	94.4	83.58
Xception [11]	23M	–	79.0	94.5	–
Inception-v4 [61]	48M	–	80.0	95.0	–
Inception-resnet-v2 [61]	56M	–	80.1	95.1	–
ResNeXt-101 [78]	84M	–	80.9	95.6	85.18
PolyNet [87]	92M	–	81.3	95.8	–
SENet [27]	146M	–	82.7	96.2	–
NASNet-A [90]	89M	–	82.7	96.2	82.56
AmoebaNet-A [52]	87M	–	82.8	96.1	–
PNASNet [39]	86M	–	82.9	96.2	–
AmoebaNet-C + AutoAugment [12]	155M	–	83.5	96.5	–
GPipe [29]	557M	–	84.3	97.0	–
EfficientNet-B7 [63]	66M	–	85.0	97.2	–
EfficientNet-B7 + FixRes [70]	66M	–	85.3	97.4	–
EfficientNet-L2 [63]	480M	–	85.5	97.5	–
ResNet-50 Billion-scale SSL [79]	26M	3.5B labeled Instagram	81.2	96.0	–
ResNeXt-101 Billion-scale SSL [79]	193M	3.5B labeled Instagram	84.8	–	–
ResNeXt-101 WSL [42]	829M	3.5B labeled Instagram	85.4	97.6	88.19
FixRes ResNeXt-101 WSL [69]	829M	3.5B labeled Instagram	86.4	98.0	89.73
Big Transfer (BiT-L) [33]	928M	300M labeled JFT	87.5	98.5	90.54
Noisy Student (EfficientNet-L2) [77]	480M	300M unlabeled JFT	88.4	98.7	90.55
Noisy Student + FixRes [70]	480M	300M unlabeled JFT	88.5	98.7	–
Vision Transformer (ViT-H) [14]	632M	300M labeled JFT	88.55	–	90.72
EfficientNet-L2-NoisyStudent + SAM [16]	480M	300M unlabeled JFT	88.6	98.6	–
Meta Pseudo Labels (EfficientNet-B6-Wide)	390M	300M unlabeled JFT	90.0	98.7	**91.12**
Meta Pseudo Labels (EfficientNet-L2)	480M	300M unlabeled JFT	**90.2**	**98.8**	91.02

Meta Pseudo Labels, Hieu Pham et al. (Jan 2021)

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**.

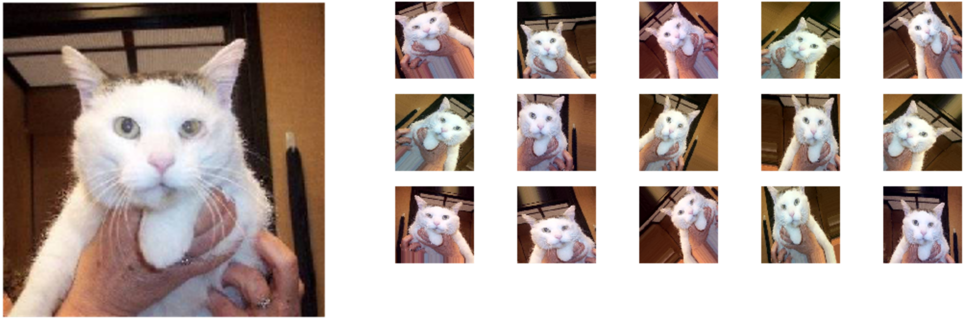
Many models trained on ImageNet and their weights are publicly available!

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



See also: RandAugment and Unsupervised Data Augmentation for Consistency Training.

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)
```

Beyond Image Classification

Beyond Image Classification

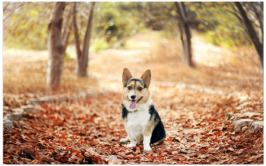
Limitations of CNNs

- Mostly on centered images
- Only a single object per image
- Not enough for many real world vision tasks

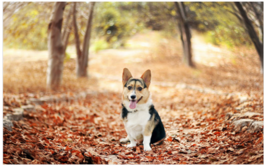
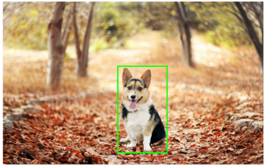
Beyond Image Classification

Classification

single
object



Beyond Image Classification

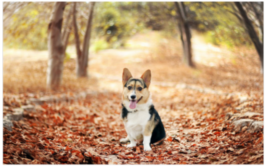
	Classification	Classif + Localisation
single object		

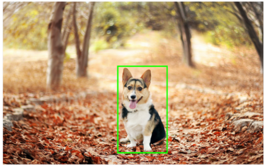
Beyond Image Classification

Classification

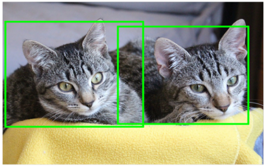
Classif + Localisation

single object





multiple objects

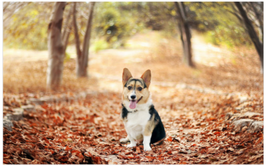


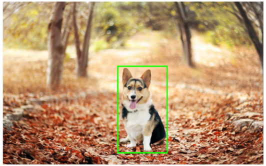
Object Detection

Beyond Image Classification

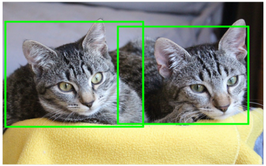
Classification | Classif + Localisation

single object





multiple objects

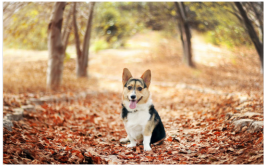


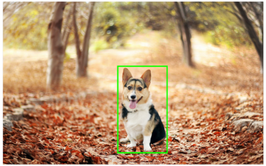
Object Detection | Semantic Segmentation

Beyond Image Classification

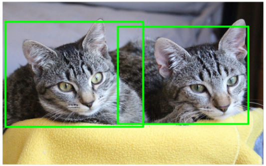
Classification | Classif + Localisation

single object





multiple objects



Object Detection | Instance Segmentation

Beyond Image Classification

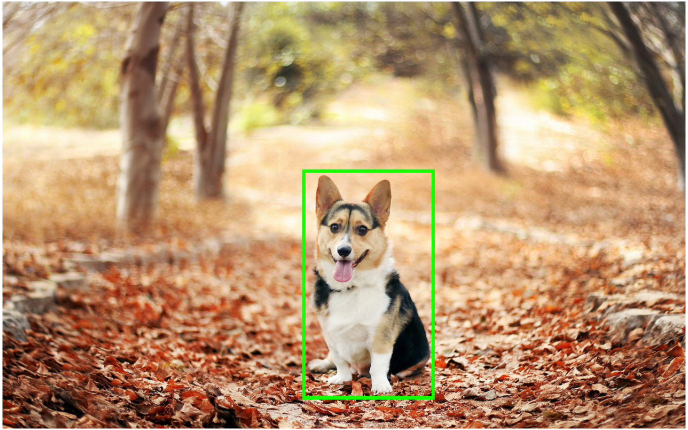
Simple Localization as regression

Detection Algorithms

Fully convolutional Networks

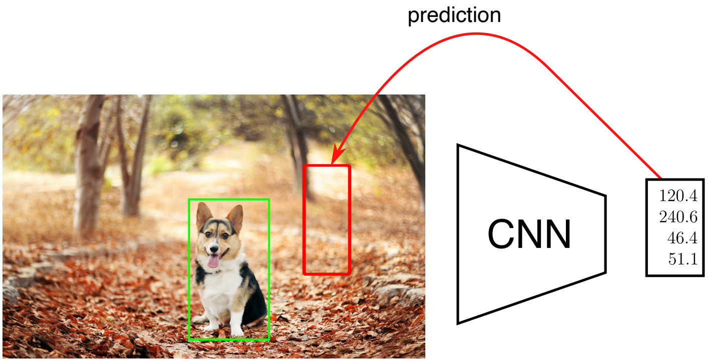
Semantic & Instance Segmentation

Localization

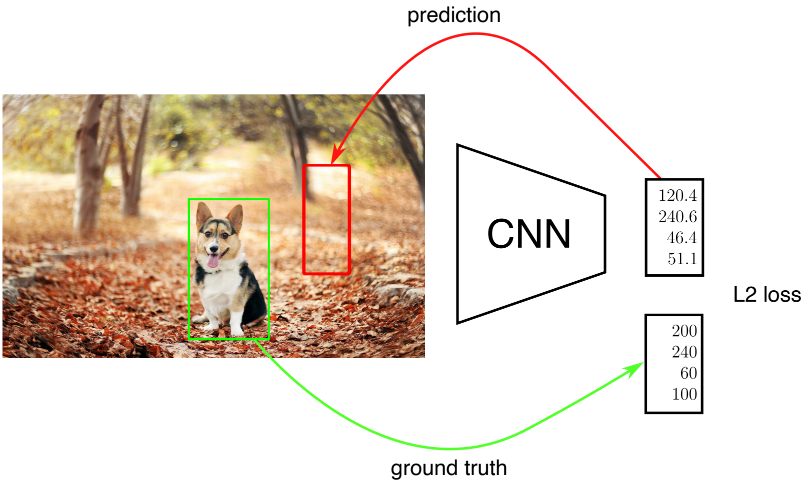


- Single object per image
- Predict coordinates of a bounding box `(x, y, w, h)`
- Evaluate via Interection over Union (IoU)

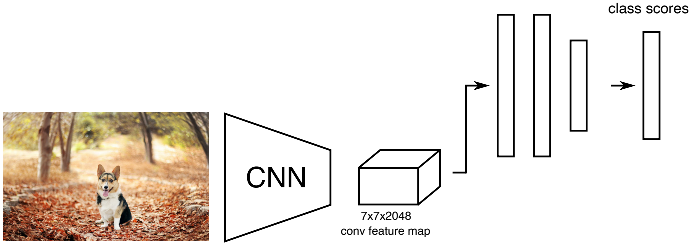
Localization as regression



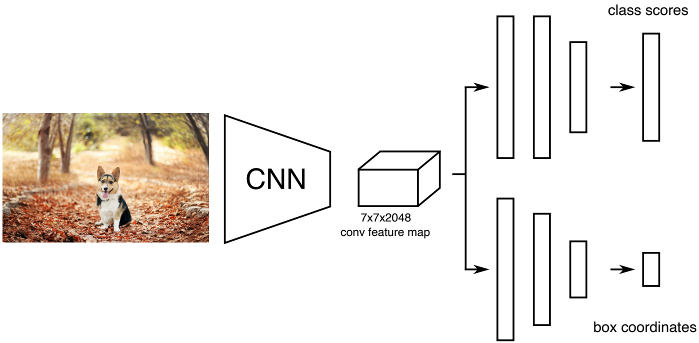
Localization as regression



Classification + Localization

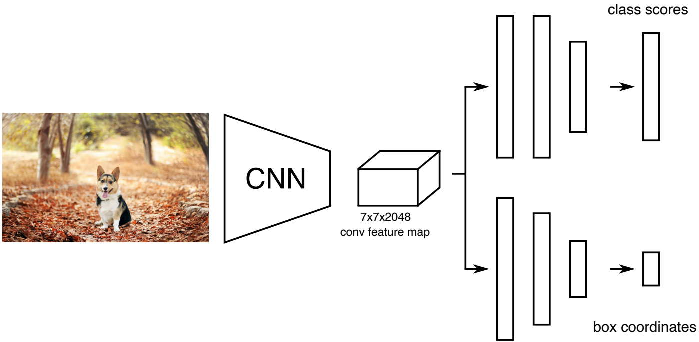


Classification + Localization



- Use a pre-trained CNN on ImageNet (e.g. ResNet)
- The "localization head" is trained seperately with regression
- Possible end-to-end finetuning of both tasks
- At test time, use both heads

Classification + Localization



C classes, 4 output dimensions (1 box).

Predict exactly N objects: predict $(N \times 4)$ coordinates and $(N \times K)$ class scores.

Object detection

We don't know in advance the number of objects in the image. Object detection relies on *object proposal* and *object classification*.

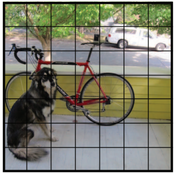
Object proposal: find regions of interest (RoIs) in the image.

Object classification: classify the object in these regions.

Two main families:

- Single-Stage: A grid in the image where each cell is a proposal (SSD, YOLO, RetinaNet).
- Two-Stage: Region proposal then classification (Faster-RCNN).

YOLO

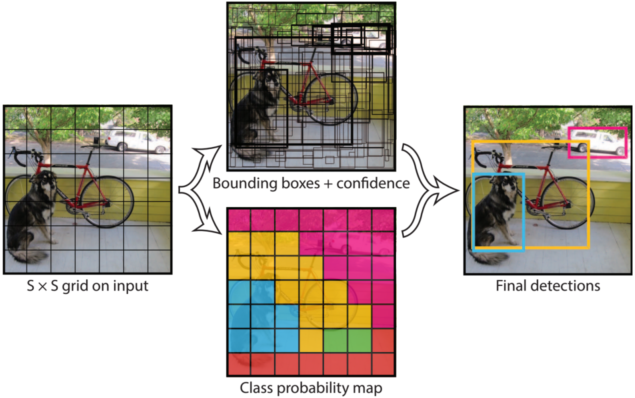


S × S grid on input

For each cell of the $S \times S$ predict: B **boxes** and **confidence scores** C ($5 \times B$ values) + **classes** c

Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” CVPR (2016)

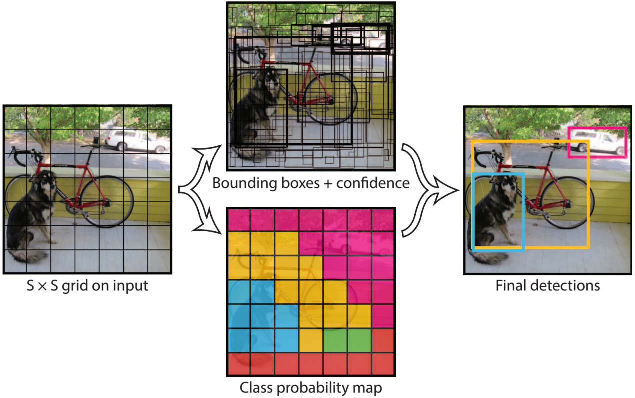
YOLO



For each cell of the $S \times S$ predict: B **boxes** and **confidence scores** C ($5 \times B$ values) + **classes** c

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." CVPR (2016)

YOLO



Final detections: $C_j * prob(c) > \text{threshold}$

Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” CVPR (2016)

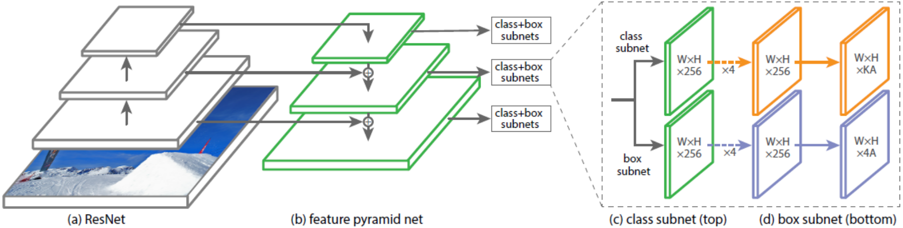
YOLO

Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” CVPR (2016)

- After ImageNet pretraining, the whole network is trained end-to-end
- The loss is a weighted sum of different regressions

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}
$$

RetinaNet



Lin, Tsung-Yi, et al. “Focal loss for dense object detection.” ICCV 2017.

Single stage detector with:

- Multiple scales through a *Feature Pyramid Network*
- Focal loss to manage imbalance between background and real objects

See: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4

Box Proposals

Instead of having a predefined set of box proposals, find them on the image:

- **Selective Search** - from pixels (not learnt, no longer used).
- **Faster - RCNN** - Region Proposal Network (RPN).

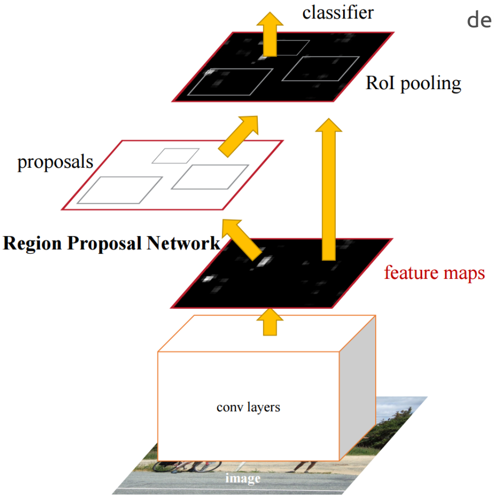
Girshick, Ross, et al. "Fast r-cnn." ICCV 2015

Crop-and-resize operator (**RoI-Pooling**):

- Input: convolutional map + N regions of interest
- Output: tensor of $N \times 7 \times 7 \times$ depth boxes
- Allows to propagate gradient only on interesting regions, and efficient computation

Faster-RCNN

Ren, Shaoqing, et al. “Faster r-cnn: Towards real-time object detection with region proposal networks.” NIPS 2015



- Train jointly **RPN** and other head
- 200 box proposals, gradient propagated only in positive boxes
- Region proposal is translation invariant, compared to YOLO

Measuring performance

method	test size shorter edge/max size	feature pyramid	align	mAP@[0.5:0.95]	AP_s	AP_m	AP_l
R-FCN [17]	600/1000			32.1	12.8	34.9	46.1
Faster R-CNN (2fc)	600/1000			30.3	9.9	32.2	47.4
Deformable [3]	600/1000		√	34.5	14.0	37.7	50.3
G-RMI [13]	600/1000			35.6	-	-	-
FPN [19]	800/1200	√		36.2	18.2	39.0	48.2
Mask R-CNN [7]	800/1200	√	√	38.2	20.1	41.1	50.2
RetinaNet [20]	800/1200	√		37.8	20.2	41.1	49.2
RetinaNet ms-train [20]	800/1200	√		39.1	21.8	42.7	50.2
Light head R-CNN	800/1200		√	**39.5**	21.8	43.0	50.7
Light head R-CNN ms-train	800/1200		√	**40.8**	22.7	44.3	52.8
Light head R-CNN	800/1200	√	√	**41.5**	25.2	45.3	53.1

Measures: mean Average Precision **mAP** at given **IoU** thresholds

Zeming Li et al. Light-Head R-CNN: In Defense of Two-Stage Object Detector 2017

- AP @0.5 for class “cat”: average precision for the class, where $IoU(box^{pred}, box^{true}) > 0.5$

State-of-the-art

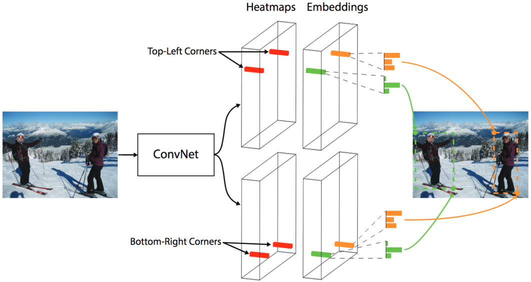
Model	FLOPs	# Params	AP_{val}	$AP_{test\text{-}dev}$
SpineNet-190 (1536) [11]	2076B	176.2M	52.2	52.5
DetectoRS ResNeXt-101-64x4d [43]	—	—	—	$55.7^{\dagger}$
SpineNet-190 (1280) [11]	1885B	164M	52.6	52.8
SpineNet-190 (1280) w/ self-training [71]	1885B	164M	54.2	54.3
EfficientDet-D7x (1536) [56]	410B	77M	54.4	55.1
YOLOv4-P7 (1536) [60]	—	—	—	$55.8^{\dagger}$
Cascade Eff-B7 NAS-FPN (1280)	1440B	185M	54.5	54.8
w/ Copy-Paste	1440B	185M	(+1.4) **55.9**	(+1.2) **56.0**
w/ self-training Copy-Paste	1440B	185M	(+2.5) **57.0**	(+2.5) **57.3**

Ghiasi G. et al. Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation, 2020

- Larger image sizes, larger and better models, better augmented data
- https://paperswithcode.com/sota/object-detection-on-coco

Other works

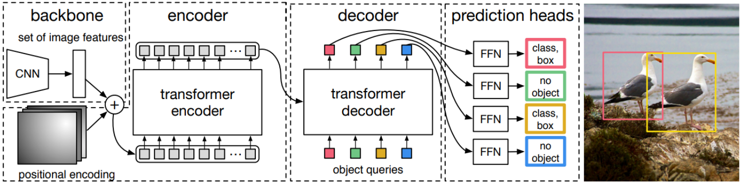
- New approaches try to avoid using anchors
- CornerNet only predicts the two extreme edges of a box:



Law, Hei, and Deng, Jia. “CornerNet: Detecting Objects as Paired Keypoints” ECCV 2018

Other works

- New approaches try to avoid using anchors
- DeTr uses a Transformer to map a set of features to a set of boxes (with different cardinality)

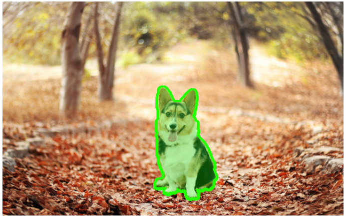


Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. “End-to-End Object Detection with Transformers” ECCV 2020

The loss is a pair-wise matching between ground truth and prediction set.

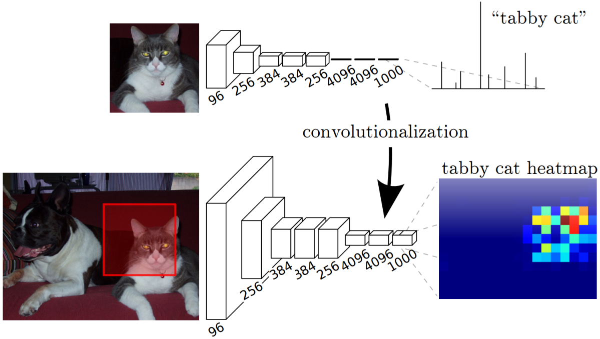
Segmentation

Output a class map for each pixel (here: dog vs background)



- **Instance segmentation**: specify each object instance as well (two dogs have different instances)
- This can be done through **object detection** + **segmentation**

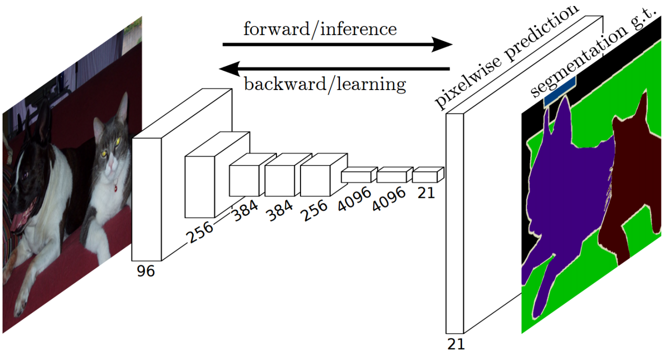
Convolutionize



Long, Jonathan, et al. "Fully convolutional networks for semantic segmentation." CVPR 2015

- Slide the network with an input of `(224, 224)` over a larger image. Output of varying spatial size
- **Convolutionize**: change Dense `(4096, 1000)` to 1×1 Convolution, with `4096, 1000` input and output channels
- Gives a coarse **segmentation** (no extra supervision)

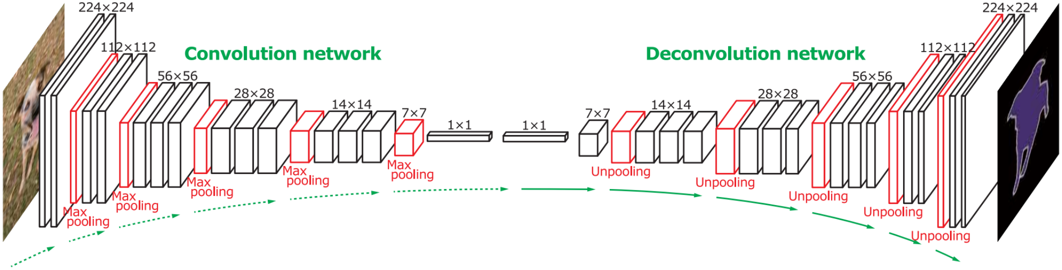
Fully Convolutional Network



Long, Jonathan, et al. "Fully convolutional networks for semantic segmentation." CVPR 2015

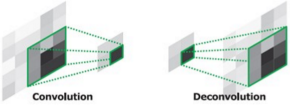
- Predict / backpropagate for every output pixel
- Aggregate maps from several convolutions at different scales for more robust results

Deconvolution

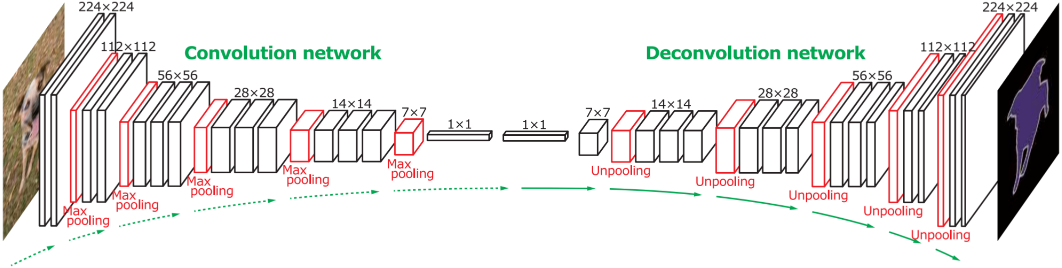


Noh, Hyeonwoo, et al. “Learning deconvolution network for semantic segmentation.” ICCV 2015

- “Deconvolution”: transposed convolutions



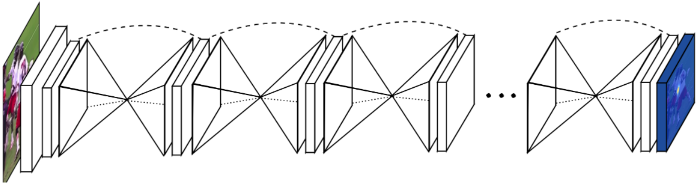
Deconvolution



Noh, Hyeonwoo, et al. “Learning deconvolution network for semantic segmentation.” ICCV 2015

- **skip connections** between corresponding convolution and deconvolution layers
- **sharper masks** by using precise spatial information (early layers)
- **better object detection** by using semantic information (late layers)

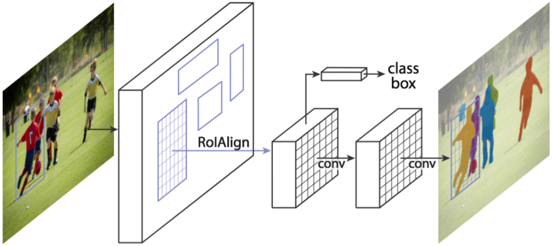
Hourglass network



Newell, Alejandro, et al. “Stacked Hourglass Networks for Human Pose Estimation.” ECCV 2016

- U-Net like architectures repeated sequentially.
- Each block refines the segmentation for the following.
- Each block has a segmentation loss.

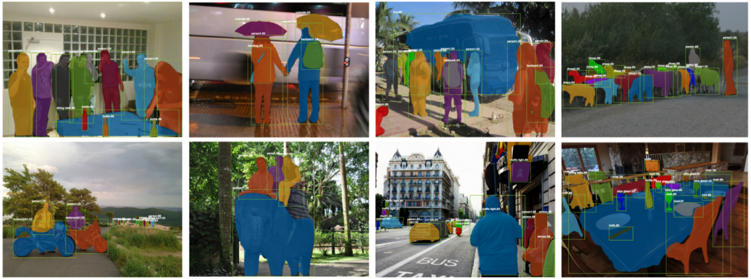
Mask-RCNN



K. He and al. Mask Region-based Convolutional Network (Mask R-CNN) NIPS 2017

Faster-RCNN architecture with a third, binary mask head

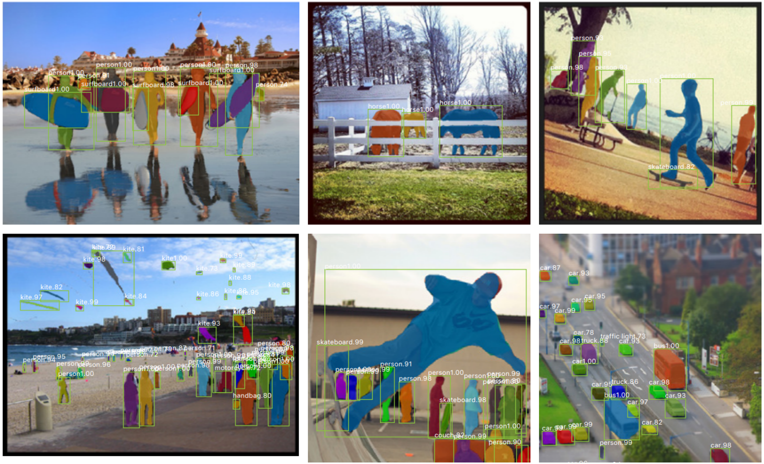
Results



K. He and al. Mask Region-based Convolutional Network (Mask R-CNN) NIPS 2017

- Mask results are still coarse (low mask resolution)
- Excellent instance generalization

Results



He, Kaiming, et al. “Mask r-cnn.” Internal Conference on Computer Vision (ICCV), 2017.

State-of-the-art & links

Most benchmarks and recent architectures are reported here:

https://paperswithcode.com/area/computer-vision

Tensorflow

object detection API

Pytorch

Detectron https://github.com/facebookresearch/Detectron

- Mask-RCNN, Retina Net and other architectures
- Focal loss, Feature Pyramid Networks, etc.