Deep Learning

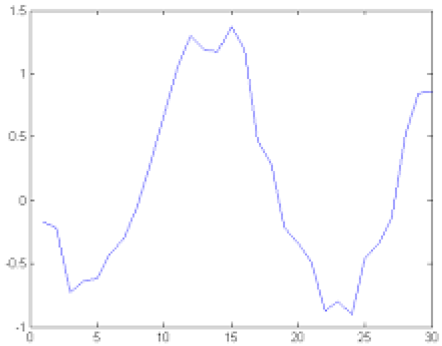
Convolutional Neural Networks

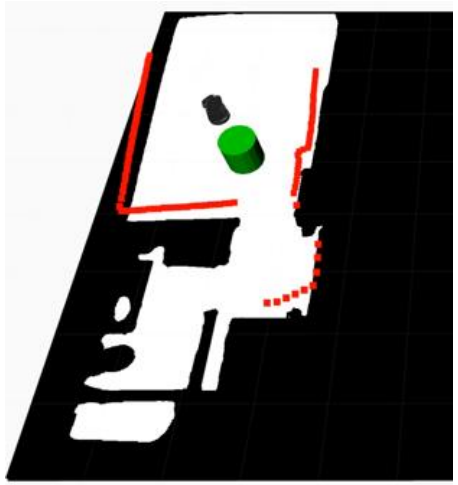
Danijel Skočaj
University of Ljubljana
Faculty of Computer and Information Science

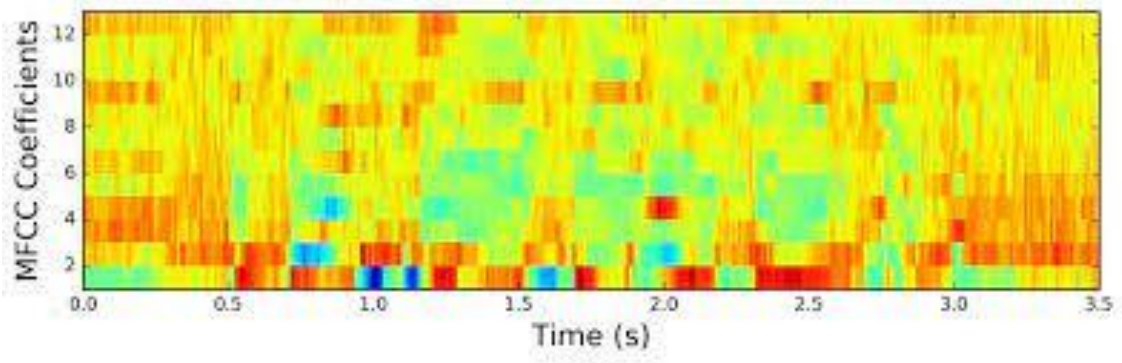
Academic year: 2022/23

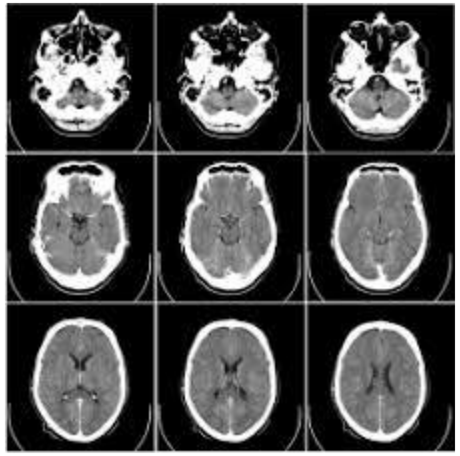
Convolutional neural networks

- Data in vectors, matrices, tensors
- Neigbourhood, spatial arrangement
- 2D: Images,time-fequency representations





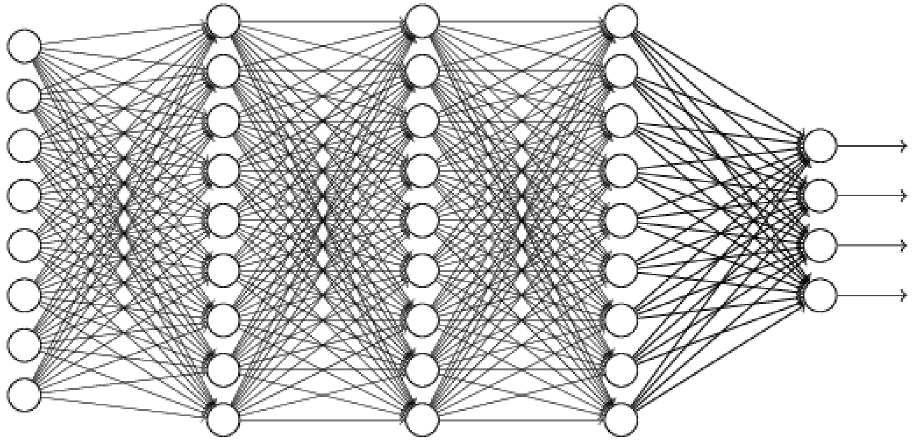




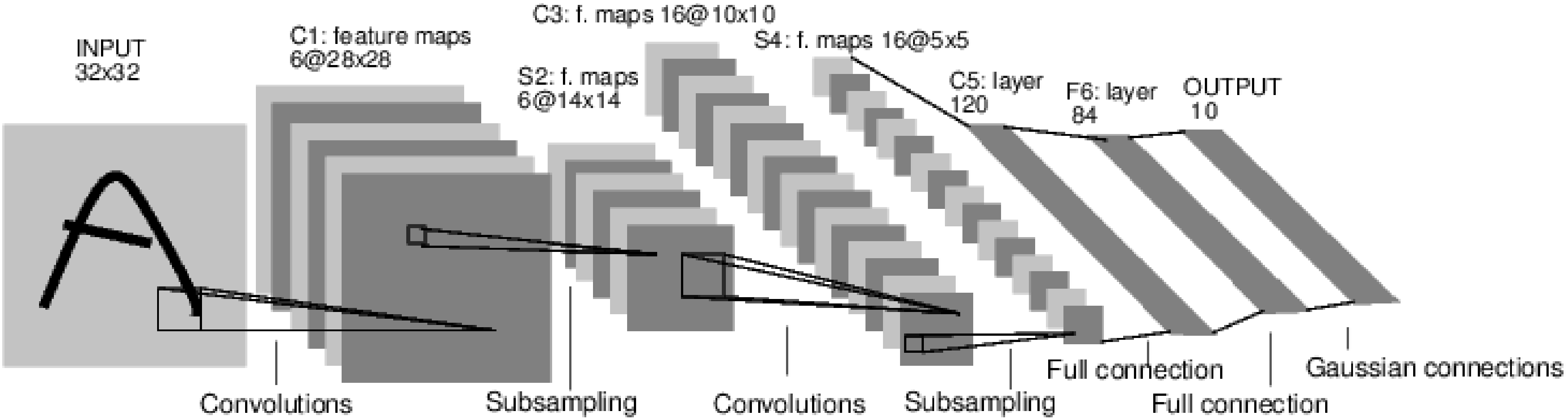
- 1D: sequential signals, text, audio, speech, time series,…
- 3D: volumetric images, video, 3D grids

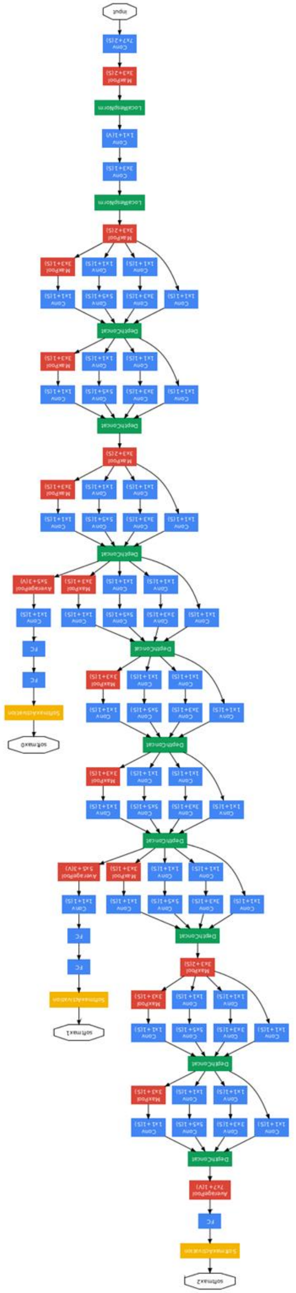
Convolutional neural networks

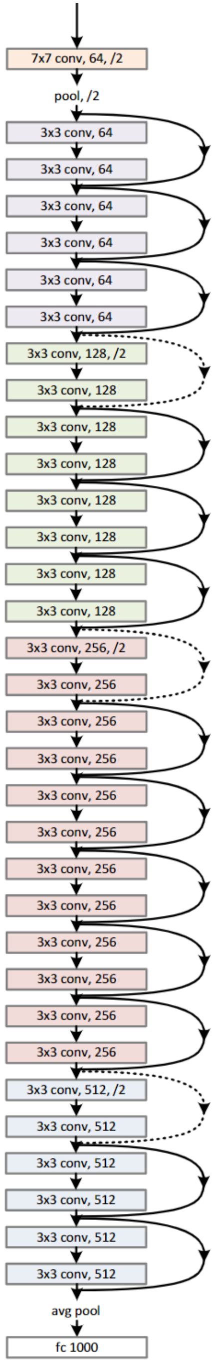
- From feedforward fully-connected neural networks …



- … to convolutional neural networks

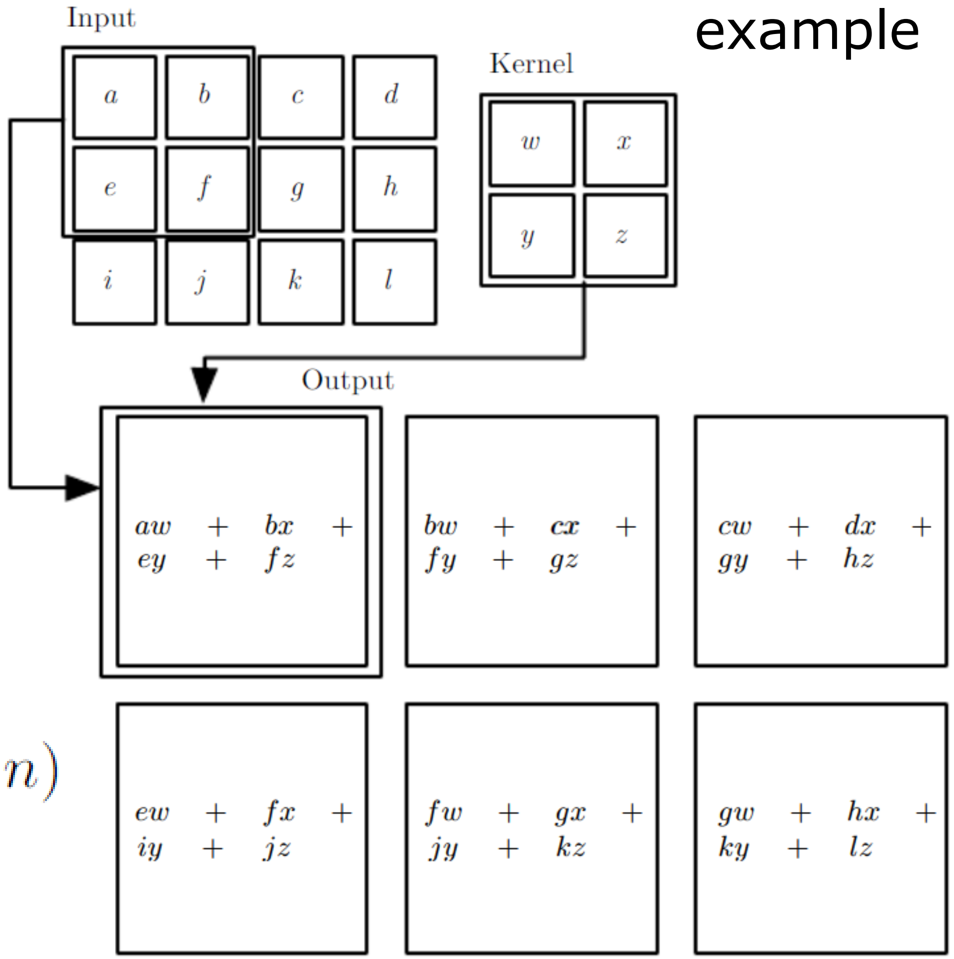






Convolution

example



- Convolution operation:

$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$

- Discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- Two-dimensional convolution:

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

- Convolution is commutative:

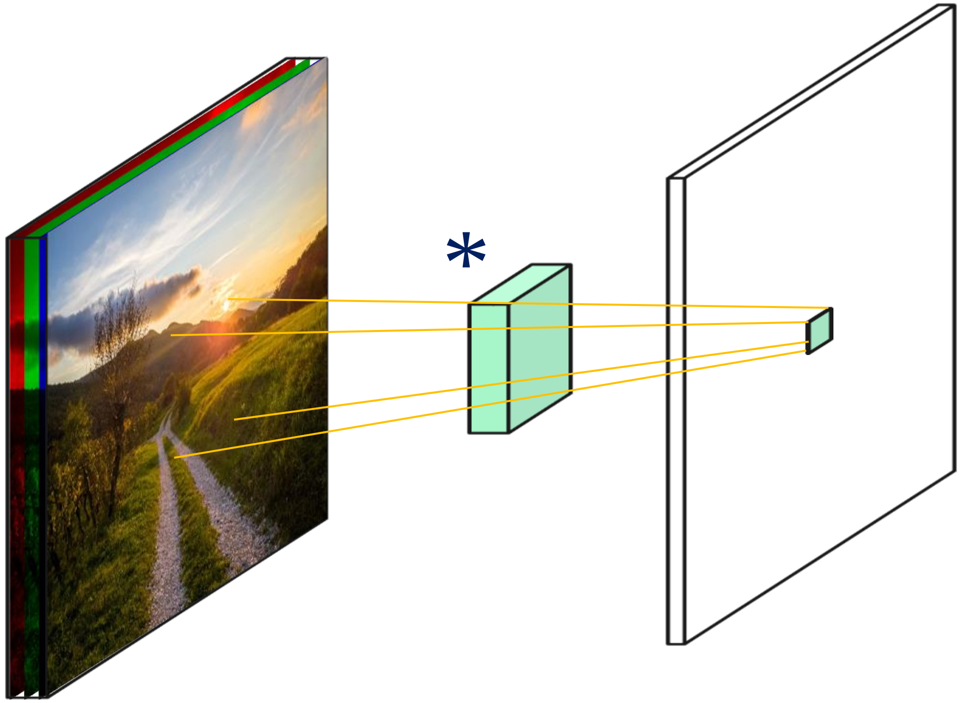
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

- Cross-correlation:

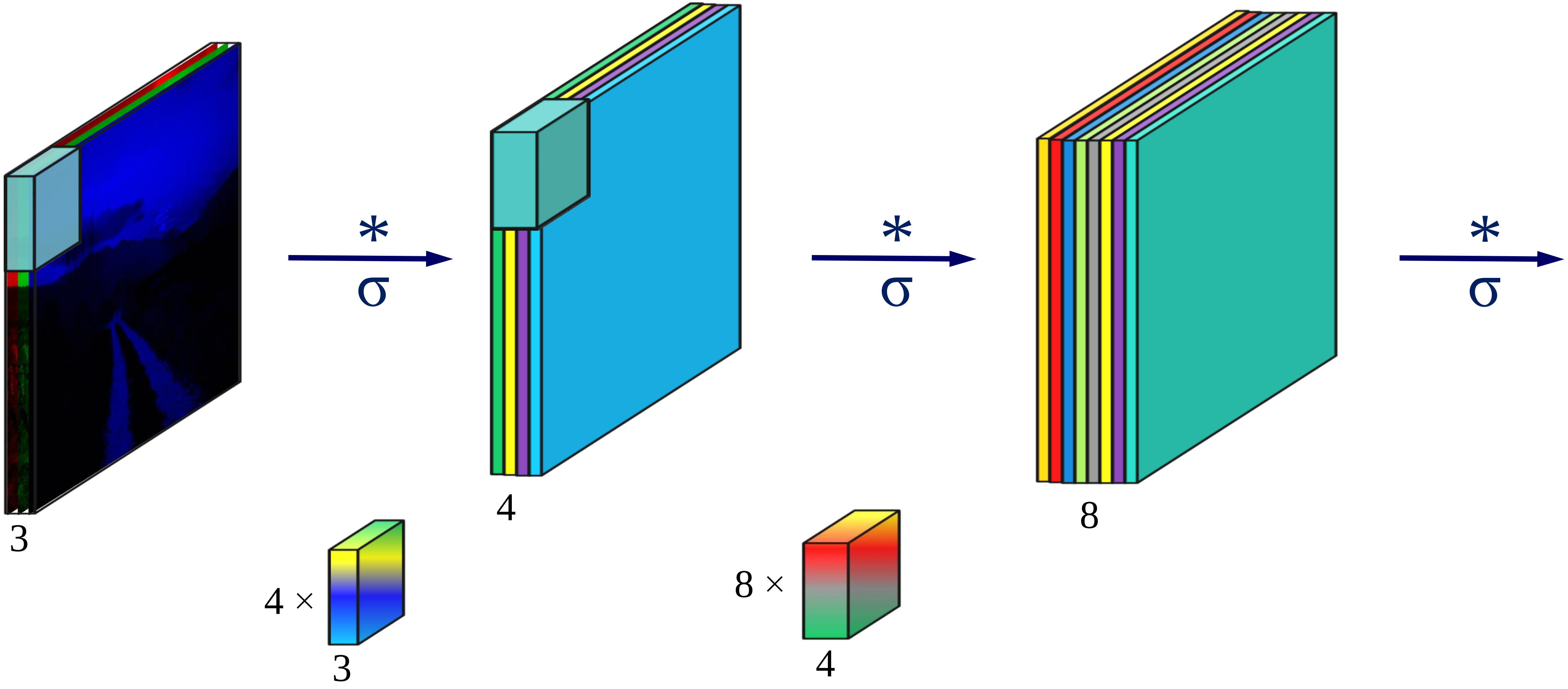
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

flipped kernel

Convolution layer

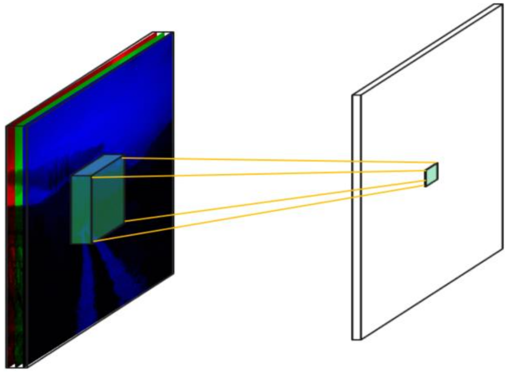


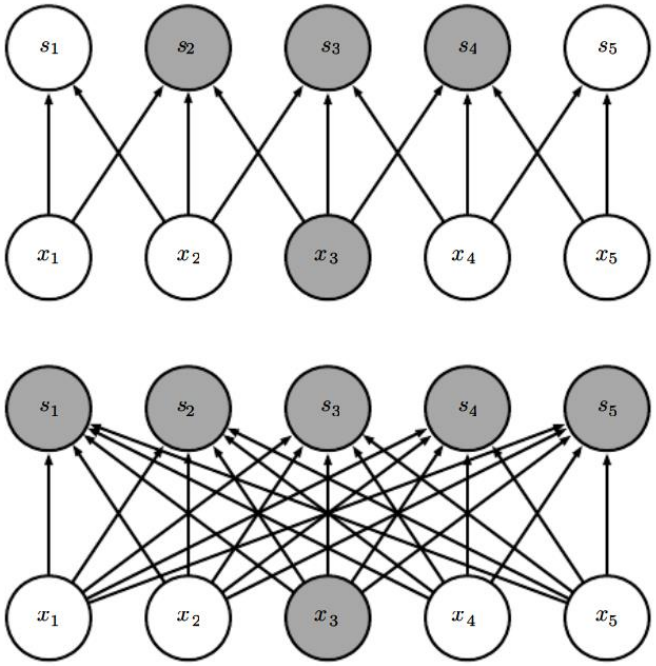
Convolution layer



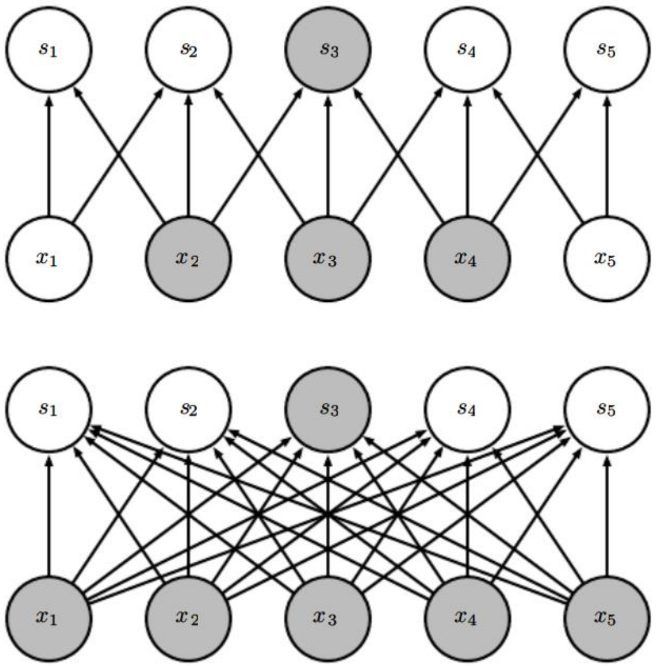
Sparse connectivity

- Local connectivity – neurons are only locally connected (**receptive field**)
  - Reduces memory requirements
  - Improves statistical efficiency
  - Requires fewer operations

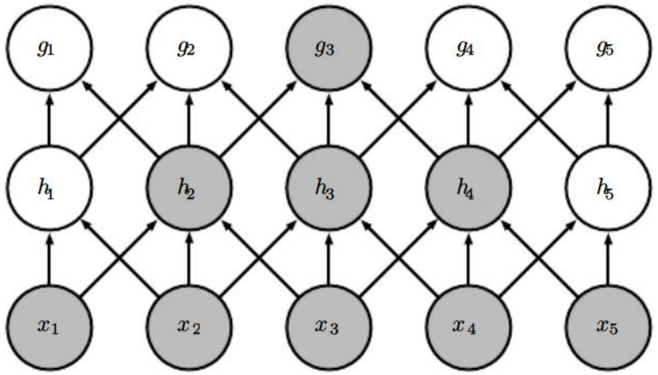




from below



from above

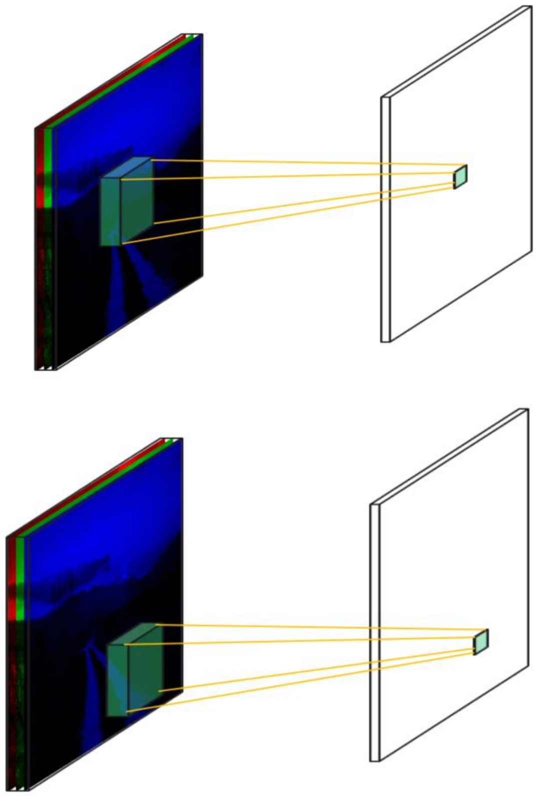


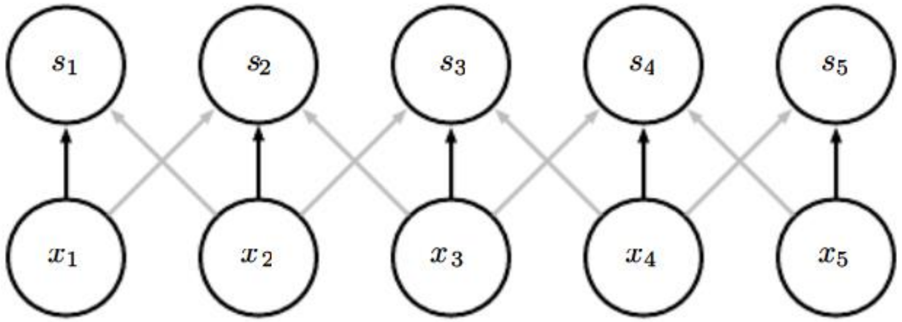
The receptive field of the units in the deeper layers is large
=> Indirect connections!

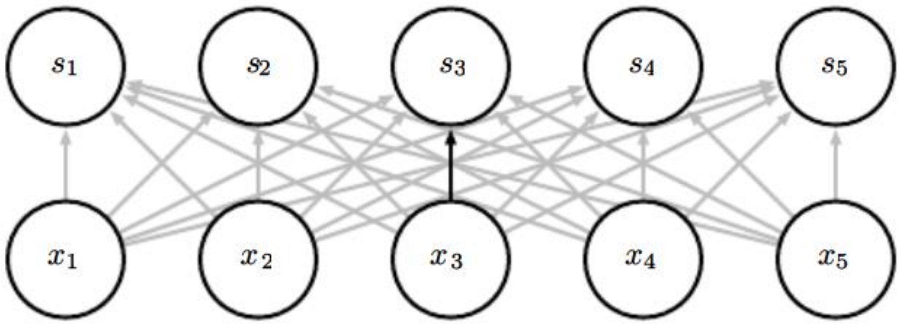
Parameter sharing

- **Neurons share weights!**
  - Tied weights
- Every element of the kernel is used at every position of the input
- All the neurons at the same level detect the same feature (everywhere in the input)
- Greatly reduces the number of parameters!

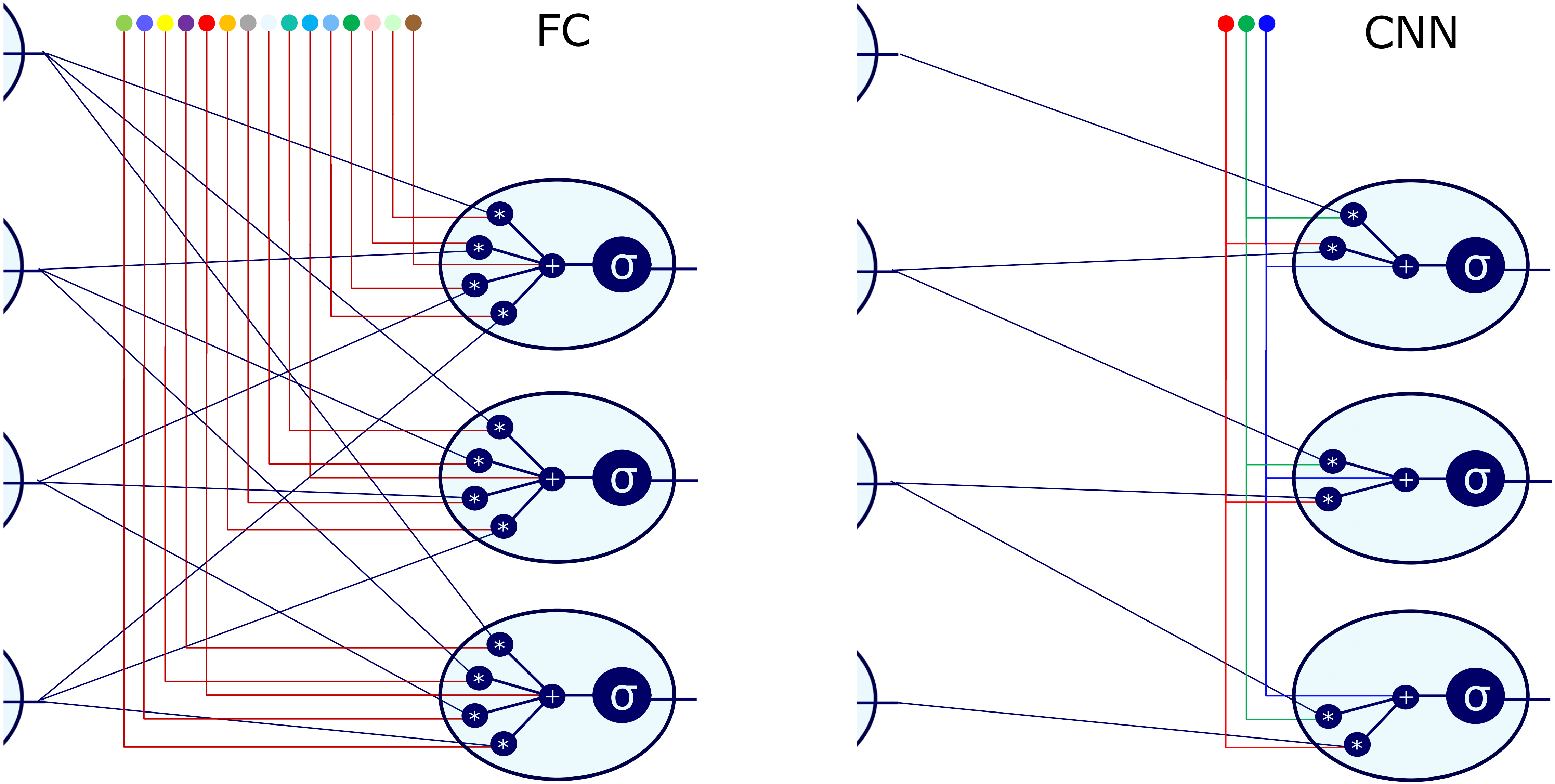
- **Equivariance to translation**
  - Shift, convolution = convolution, shift
  - Object moves => representation moves







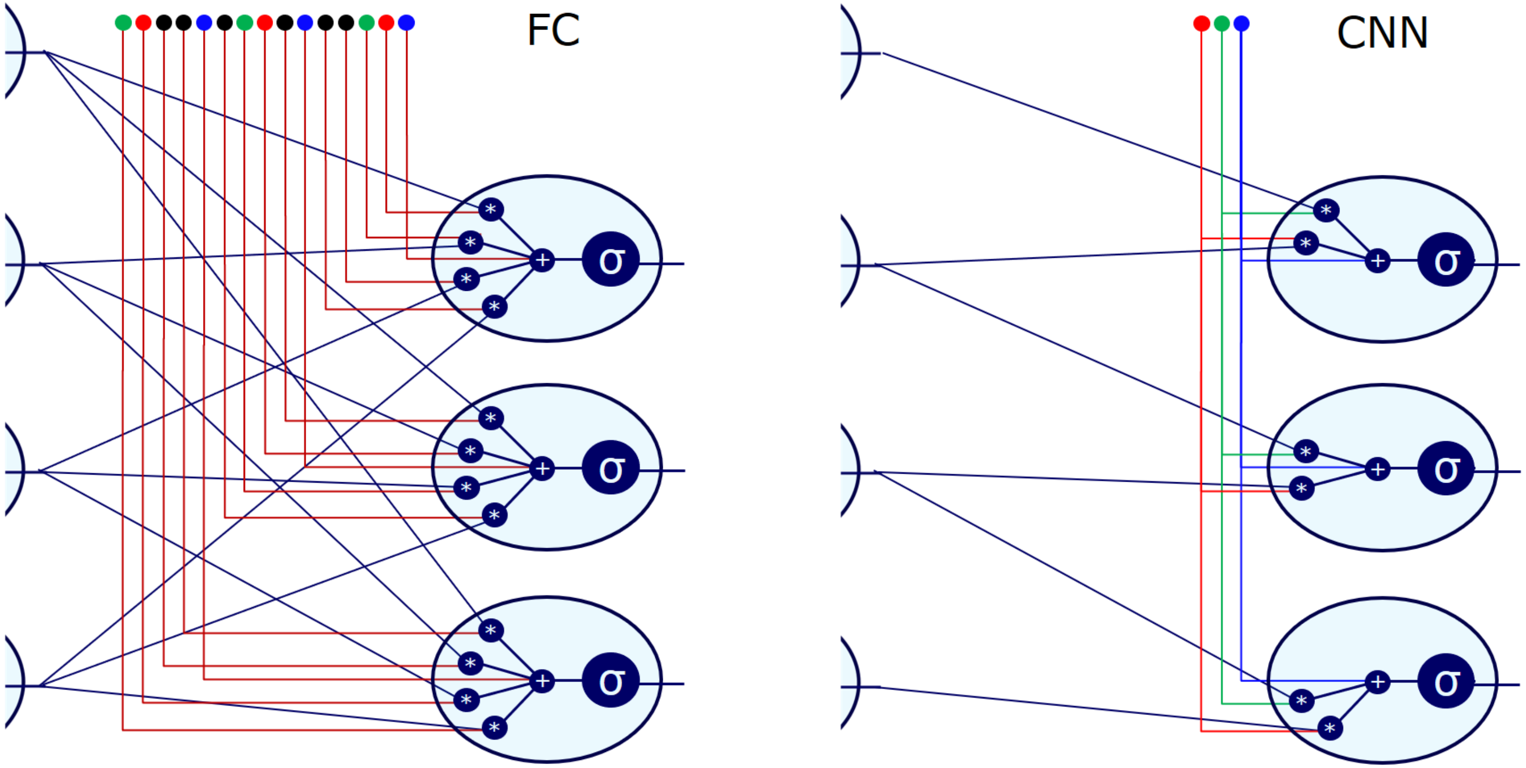
CNN as FC networks



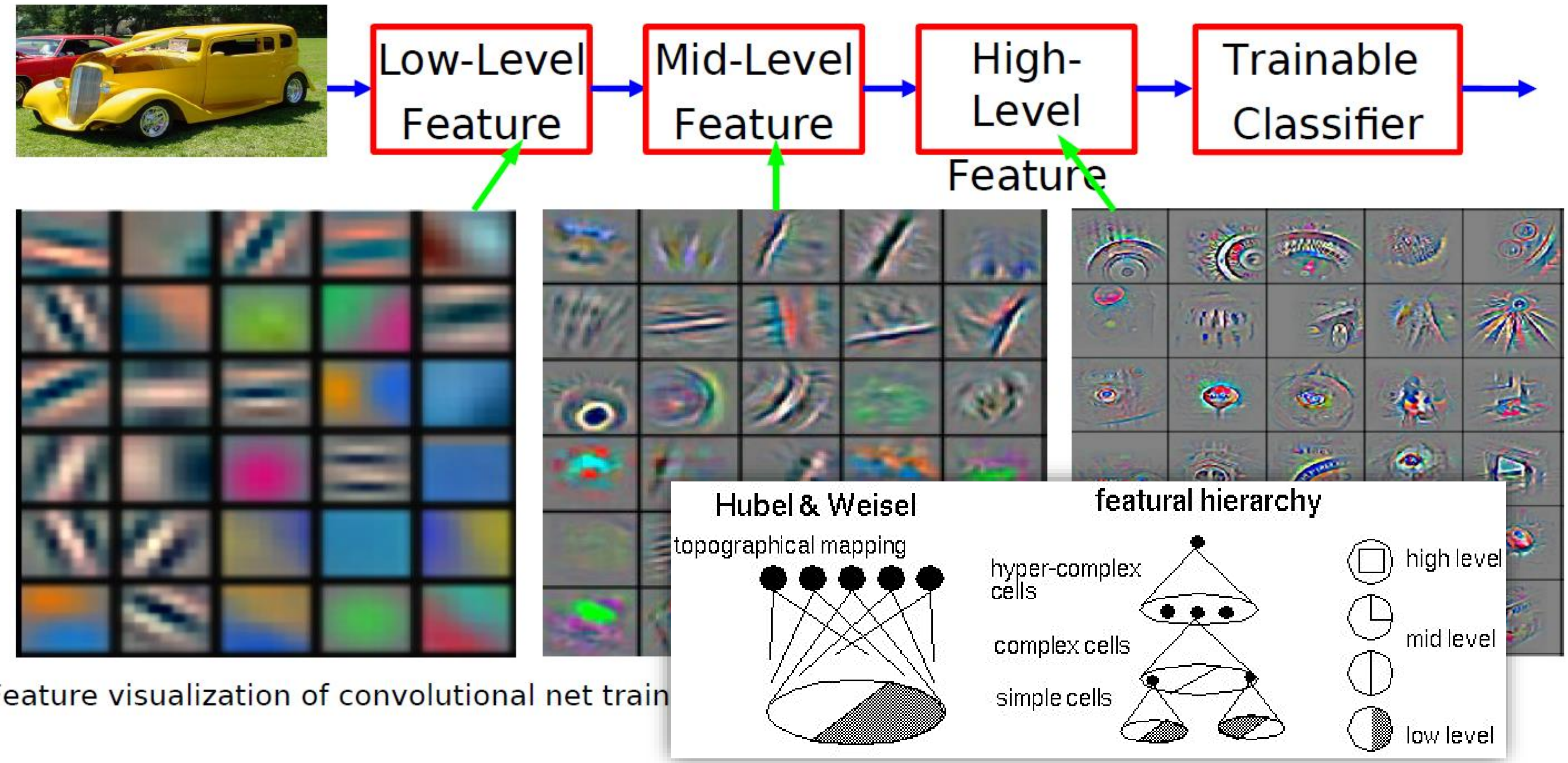
CNN as FC networks

- Fully connected network with an infinitively strong prior over its weights
  - Weights are zero outside the kernel region
  - Tied weights

  => learns only local interactions and is equivariant to translations

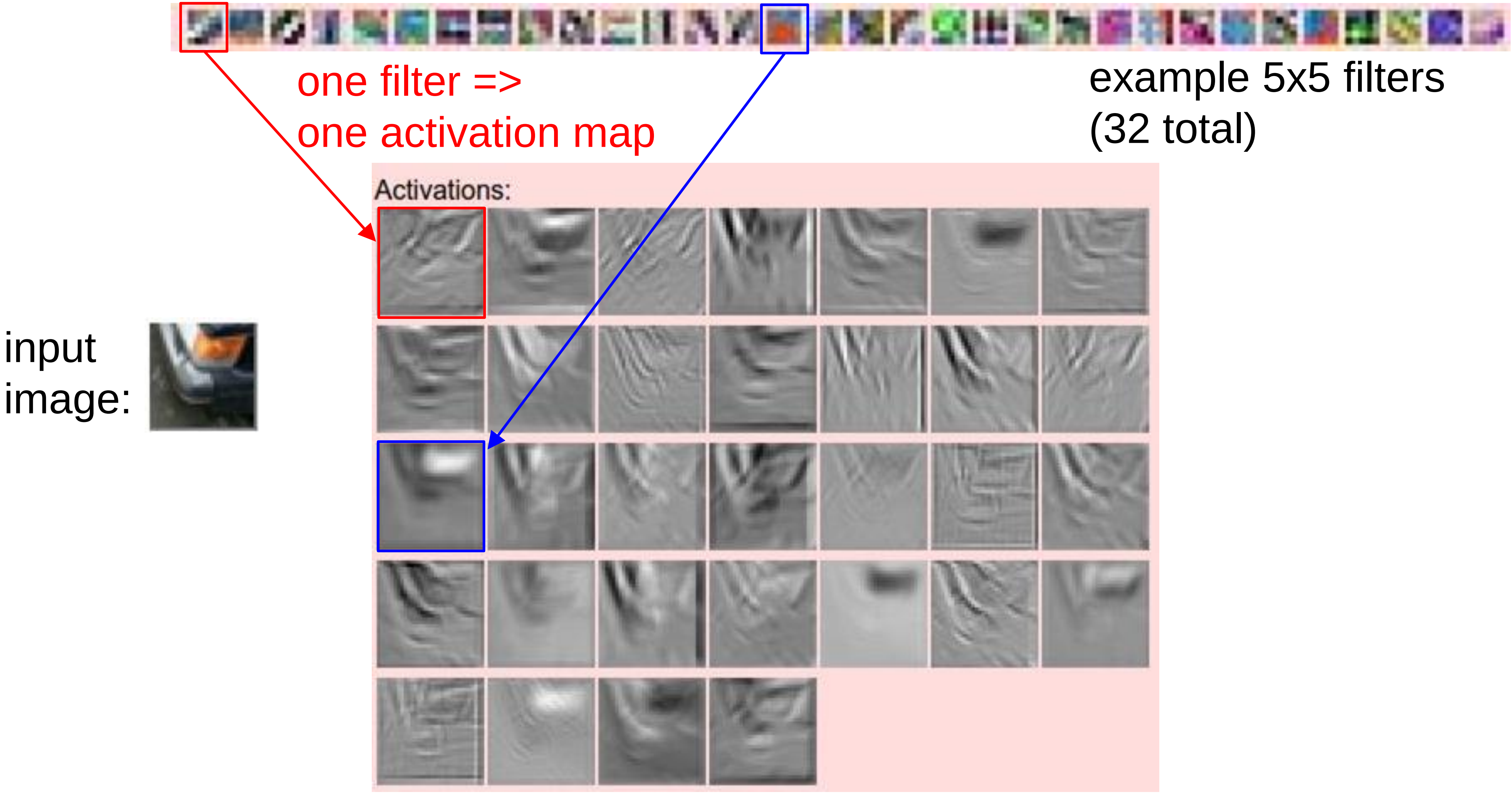


Convolutional neural network



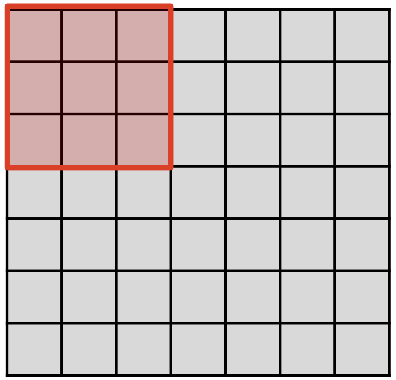
Feature visualization of convolutional net train

Convolutional neural network



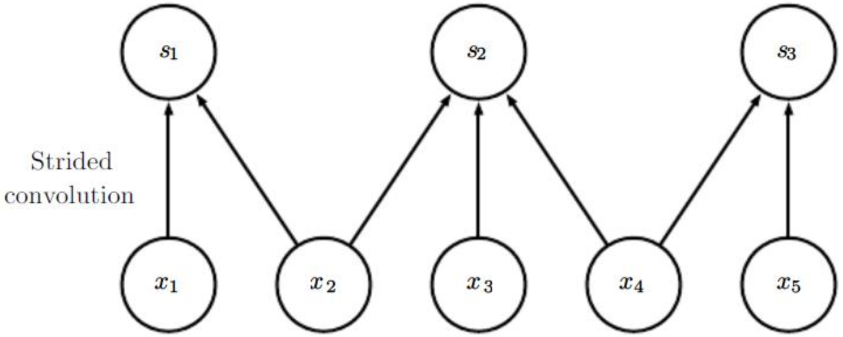
Stride

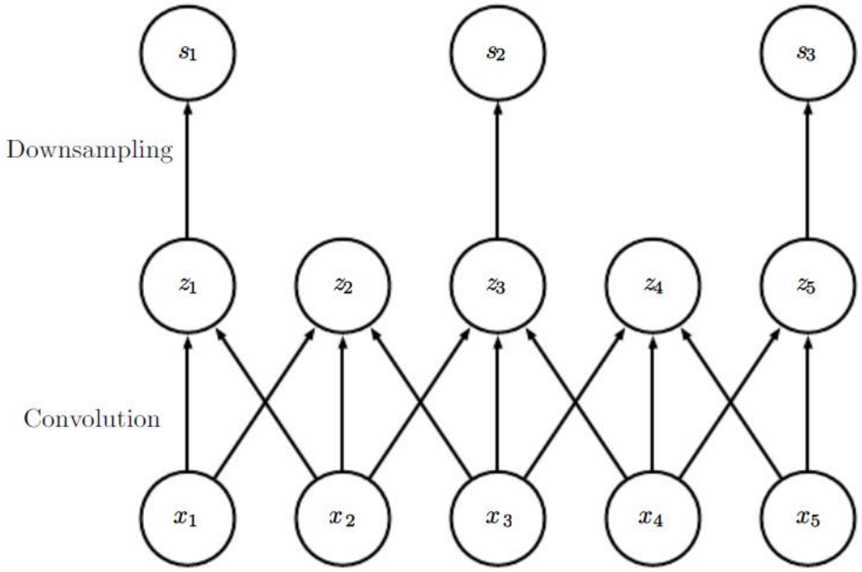
- Step for convolution filter



Stride=1
Stride=2

Convolution with stride>1
is equivalent to
convolution + downsampling

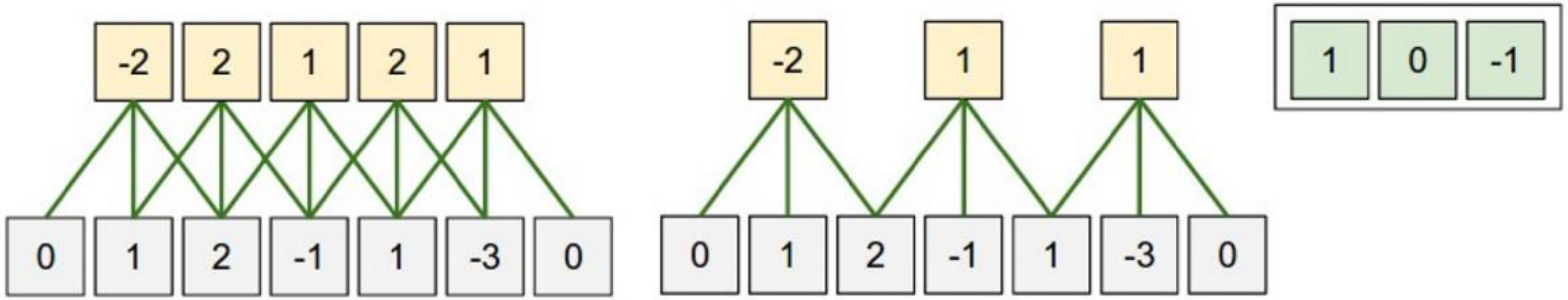




- Output size:

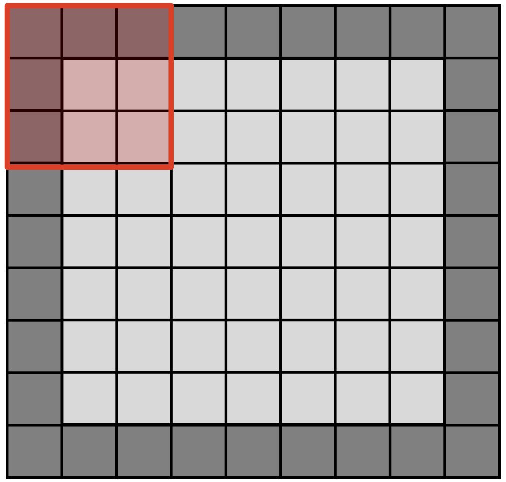
$$\frac{N-F}{S}+1$$

- Example:

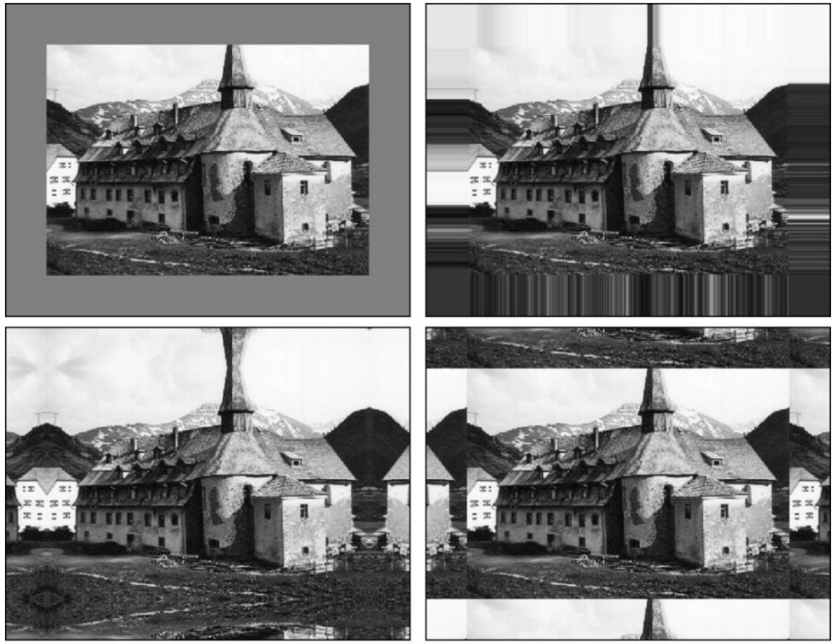


Padding

- Extend the image to facilitate processing of border pixels

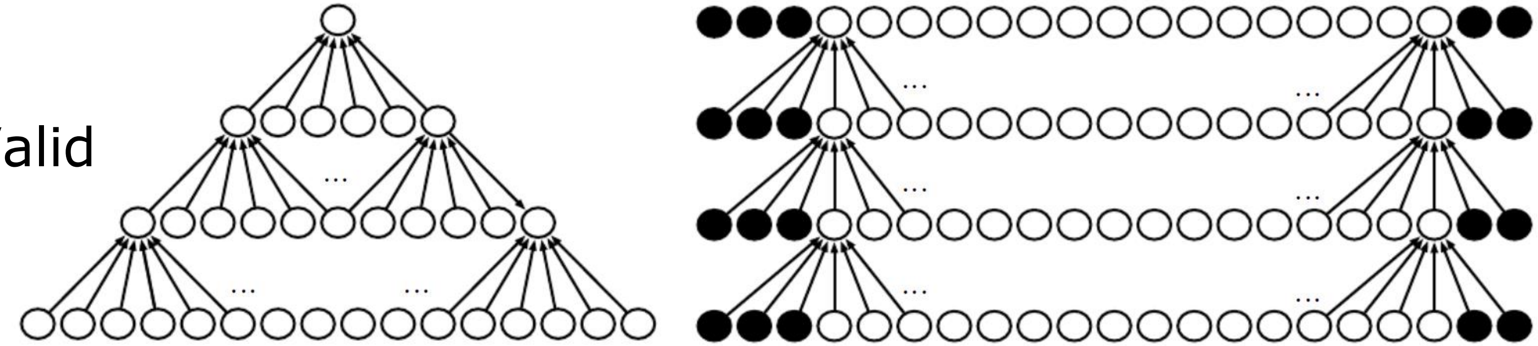


- Ussualy pad with 0
- To preserve size pad on every side $\frac{F-1}{2}$ pixels



- Zero padding prevents shrinking network size
  - Valid: no zero-padding – output is smaller than input
  - Same: keeps the size of the input

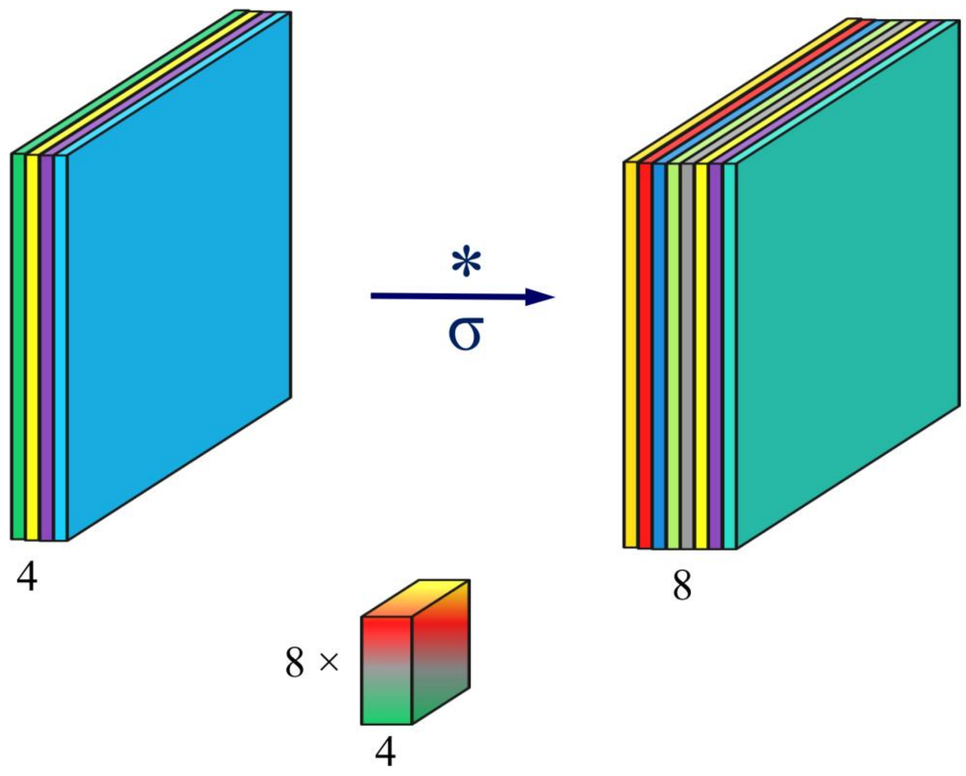
Valid



Same

Convolution layer parameters

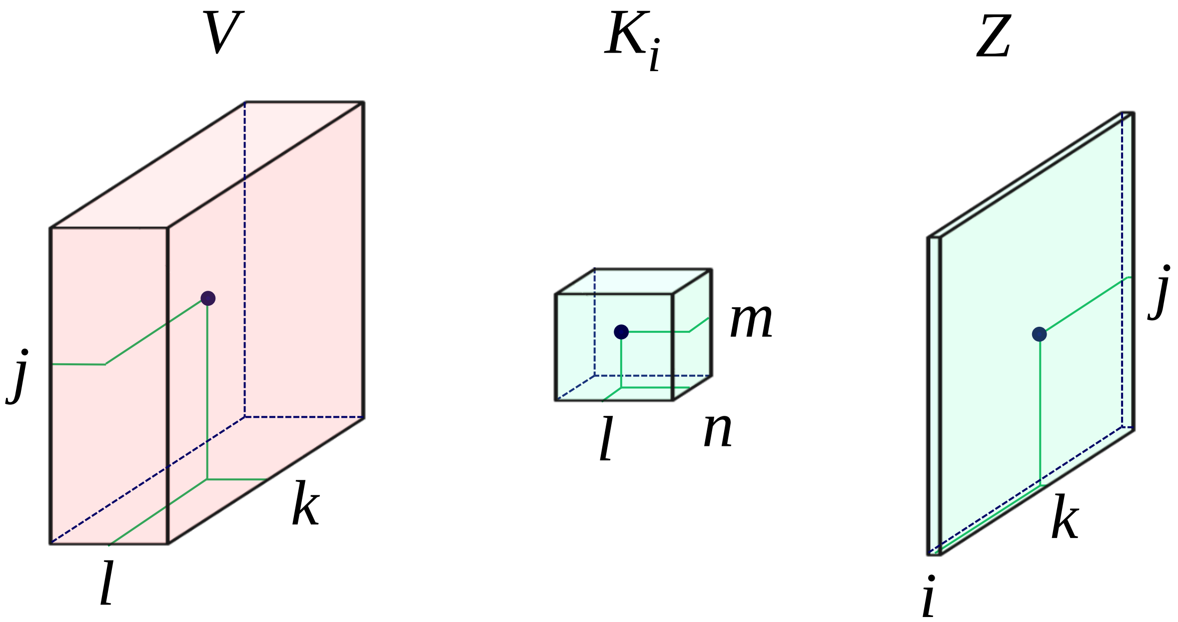
- Hyperparameters:
  - K - Number of filters
  - F – Filter size
  - S – Stride
  - P – Padding
- Input: volume of the size $W_1 \times H_1 \times D_1$
- Output volume size:
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - $D_2 = K$
- Number of parameters
  - Number of weights: $K \cdot F \cdot F \cdot D_1$
  - Number of biases: K



Executing convolution

- V – input
- K – kernel
- Z – output
- i – output channel
- j,k: input/output row, column
- l: input channel
- m,n: offset rows, columns

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

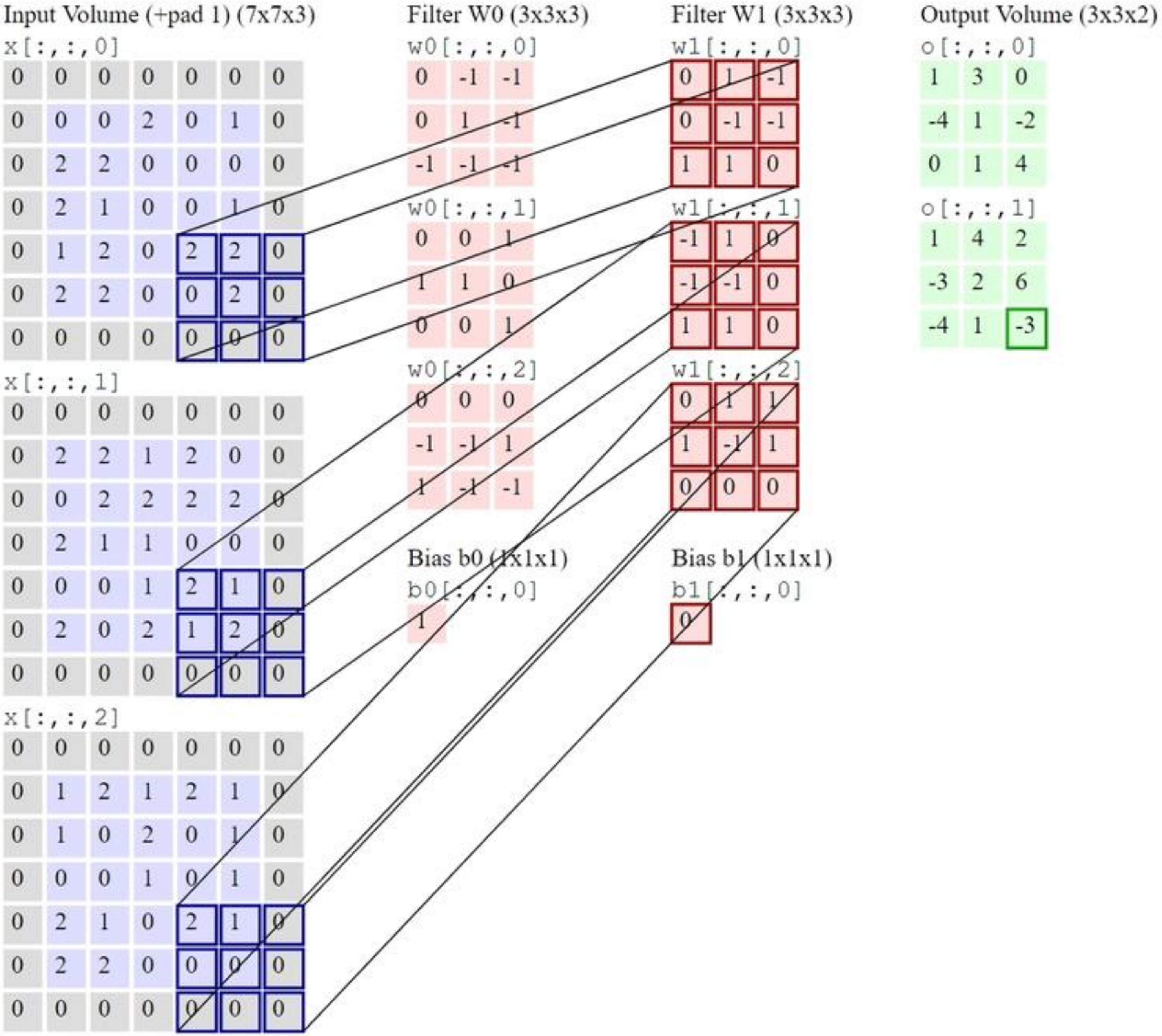


- s: stride

$$Z_{i,j,k} = c(\mathsf{K}, \mathsf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right]$$

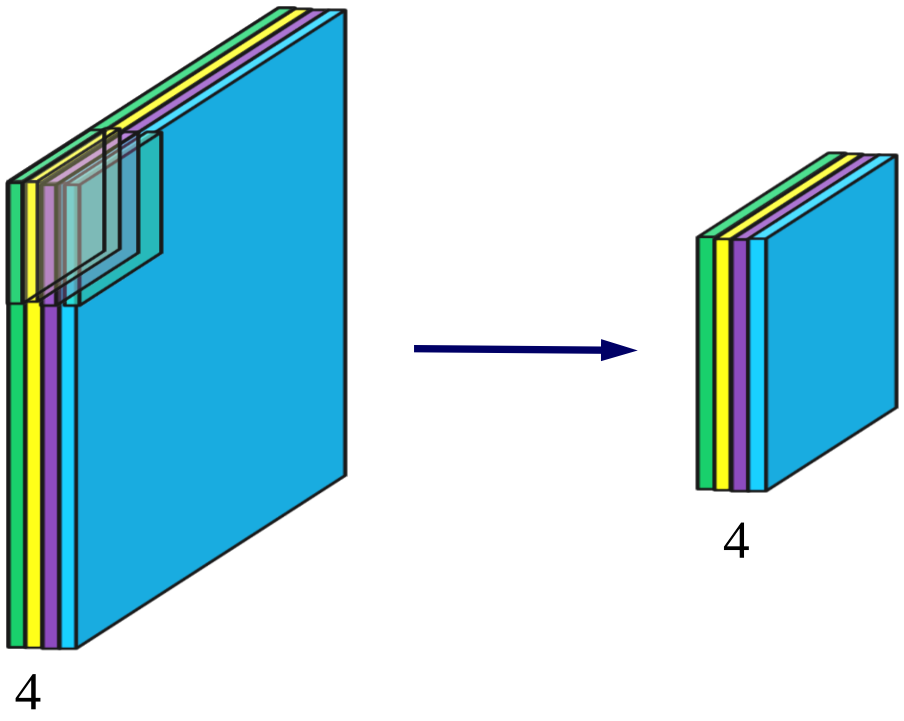
Example

- Input size: 5x5x3
- Kernel size: 3x3x3
- Num. of filters: 2
- Stride: 2
- Padding: 1
- Output size: 3x3x2

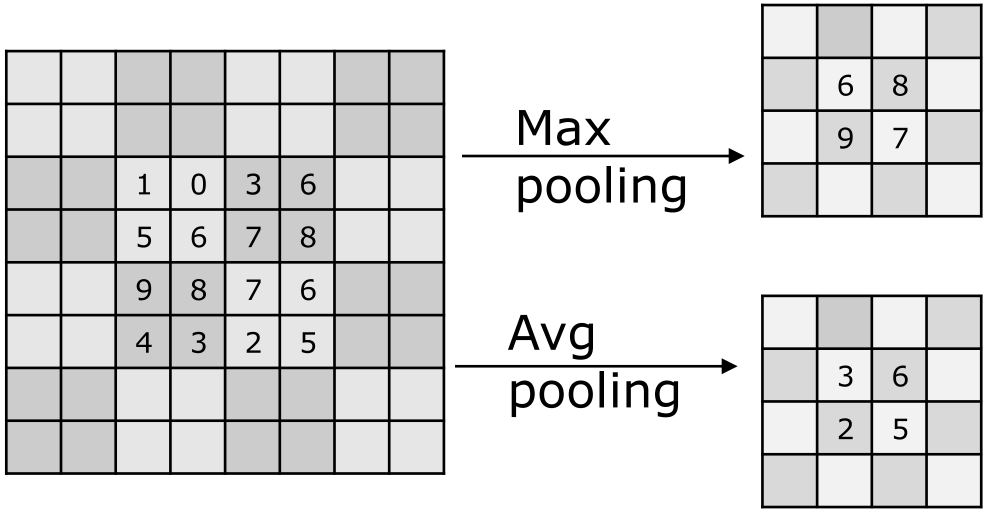


Pooling layer

- Downsampling – reduces the volume size (width and height)
- Process each activation map independently – keeps the volume depth unchanged

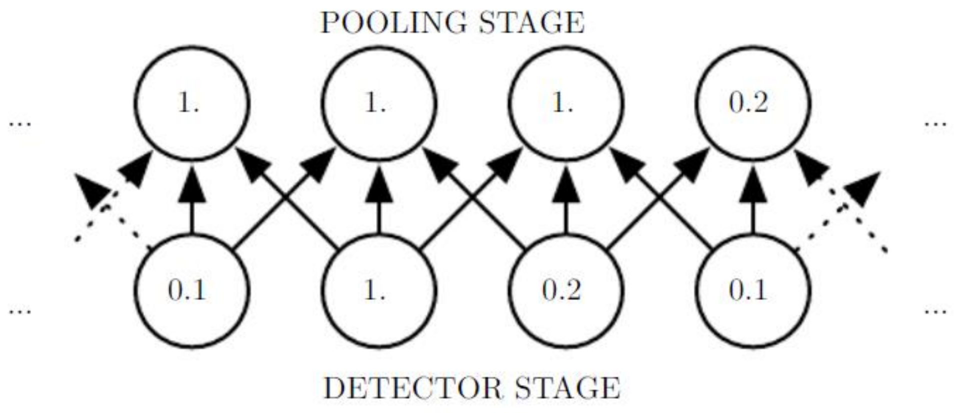


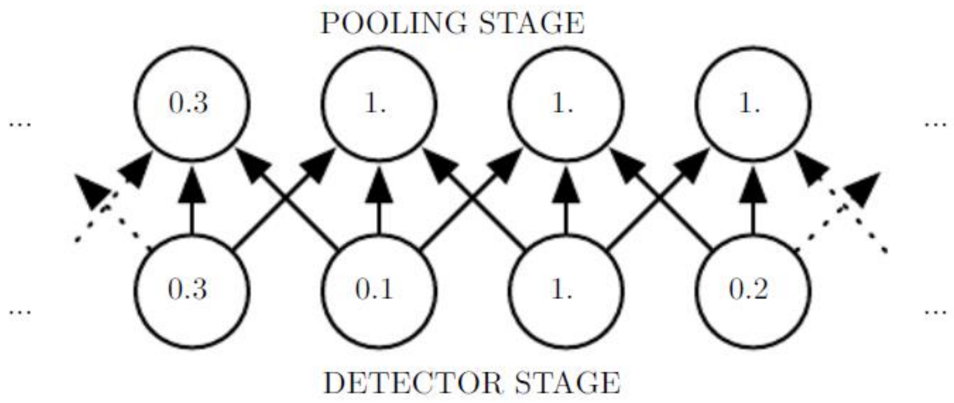
- Example with
  - F=2
  - S=2



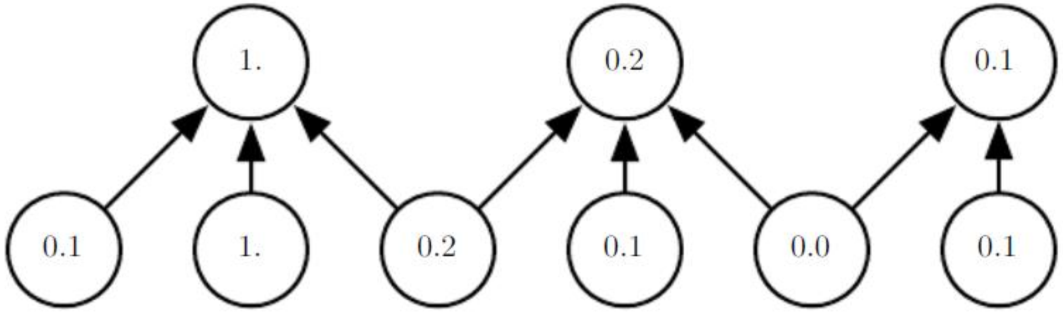
Pooling

- Max pooling introduces translation invariance



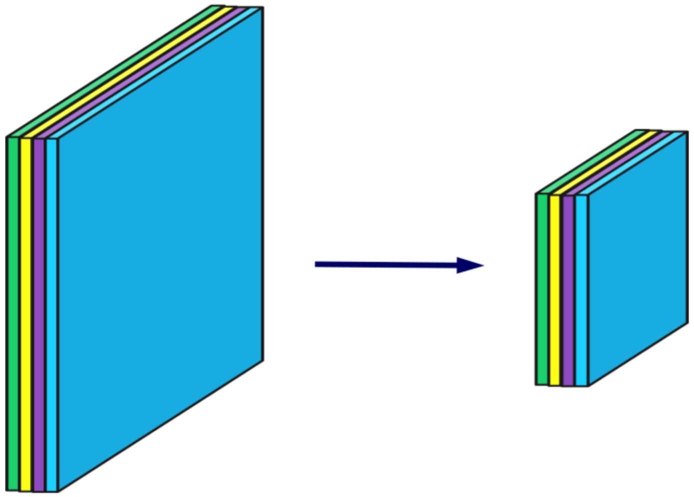


- Pooling with downsampling
  - Reduces the representation size
  - Reduces computational cost
  - Increases statistical efficiency



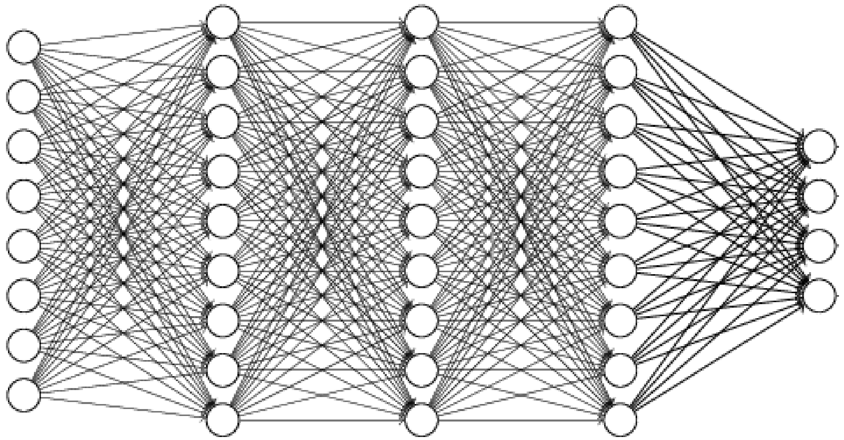
Pooling layer parameters

- Hyperparameters:
  - F – Filter size
  - S – Stride (ussualy >1)
- Input: volume of the size $W_1 \times H_1 \times D_1$
- Output volume size:
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_2 = D_1$
- Number of parameters: 0



Fully connected layer

- Every neuron at the layer $l\text{-}1$ is connected to every neuron at the layer l

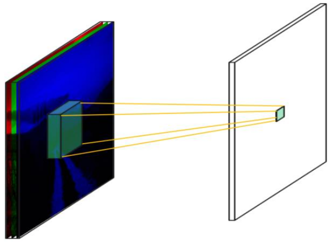


- Usually added at the end of the network to perform classification
- Hyperparameters:
  - N – Number of neurons
- Input: $N_{l\text{-}1}$ neurons
- Output size: N_l neurons
- Number of parameters:
  - Number of weights: $N_{l\text{-}1} \cdot N_l$
  - Number of biases: N_l

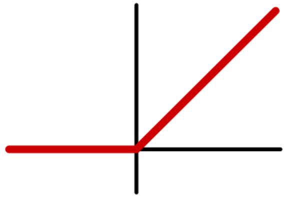
CNN layers

- Layers used to build ConvNets:
  - INPUT:
    raw pixel values

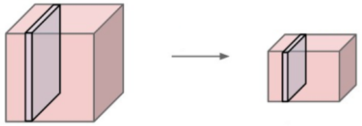
  - CONV:
    convolutional layer



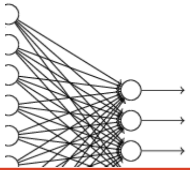
  - (BN: batch normalisation)
  - (ReLU:)
    introducing nonlinearity

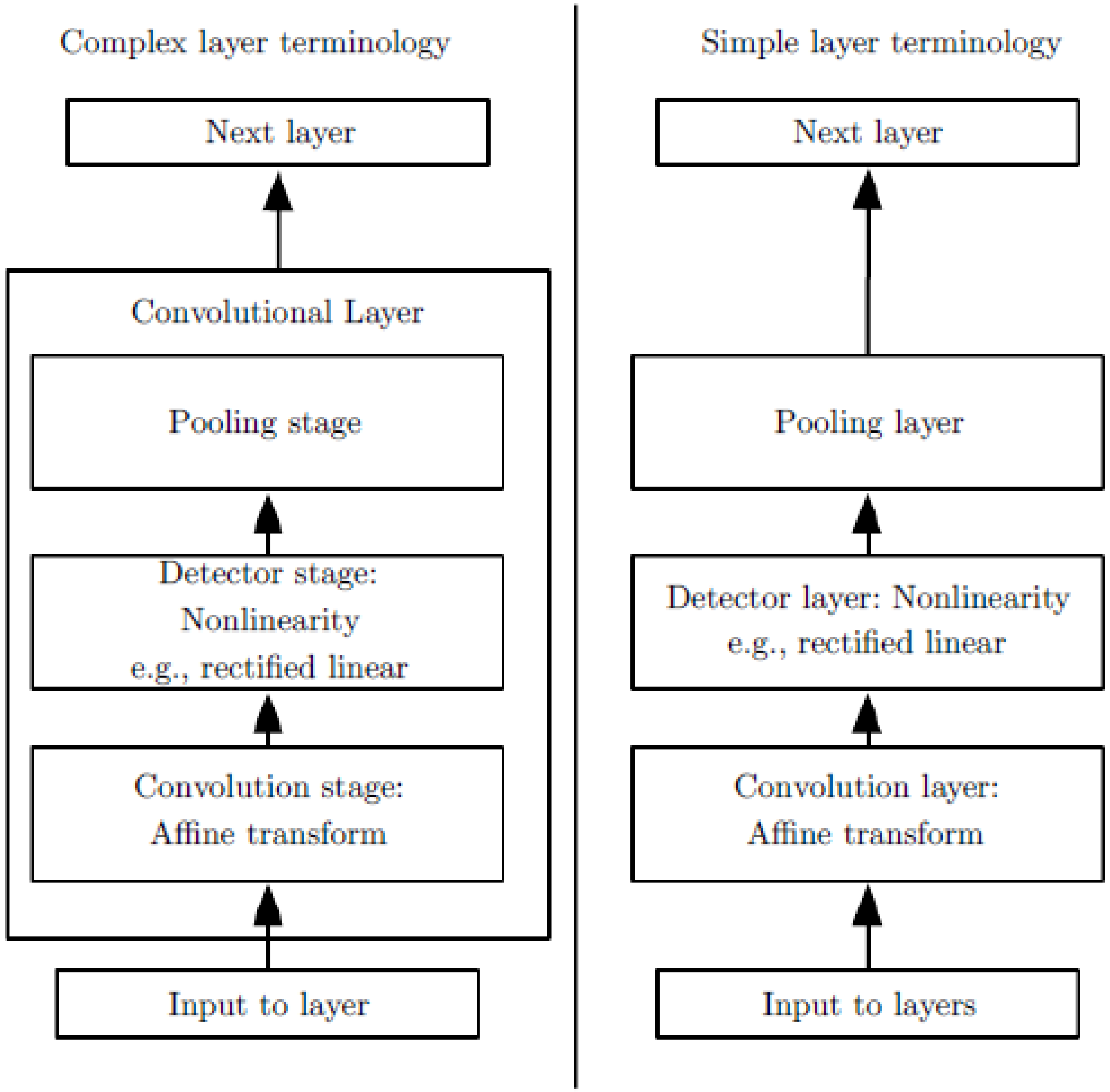


  - POOL:
    downsampling



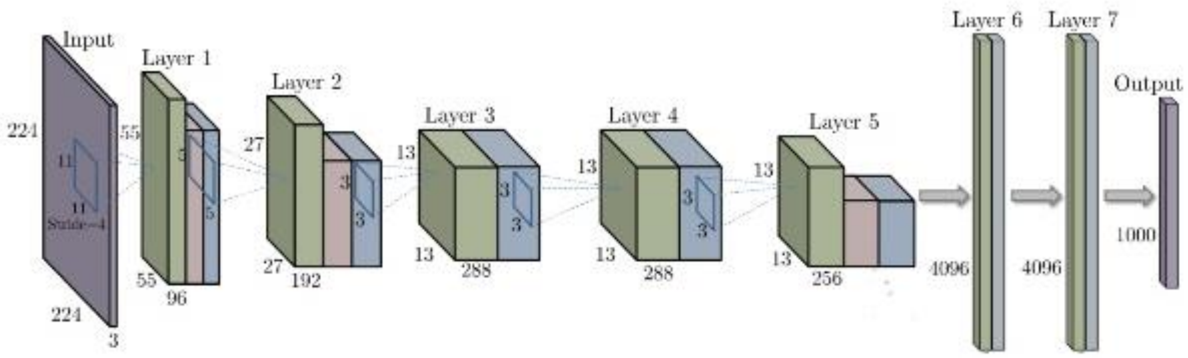
  - FC:
    for computing class scores
  - SoftMax



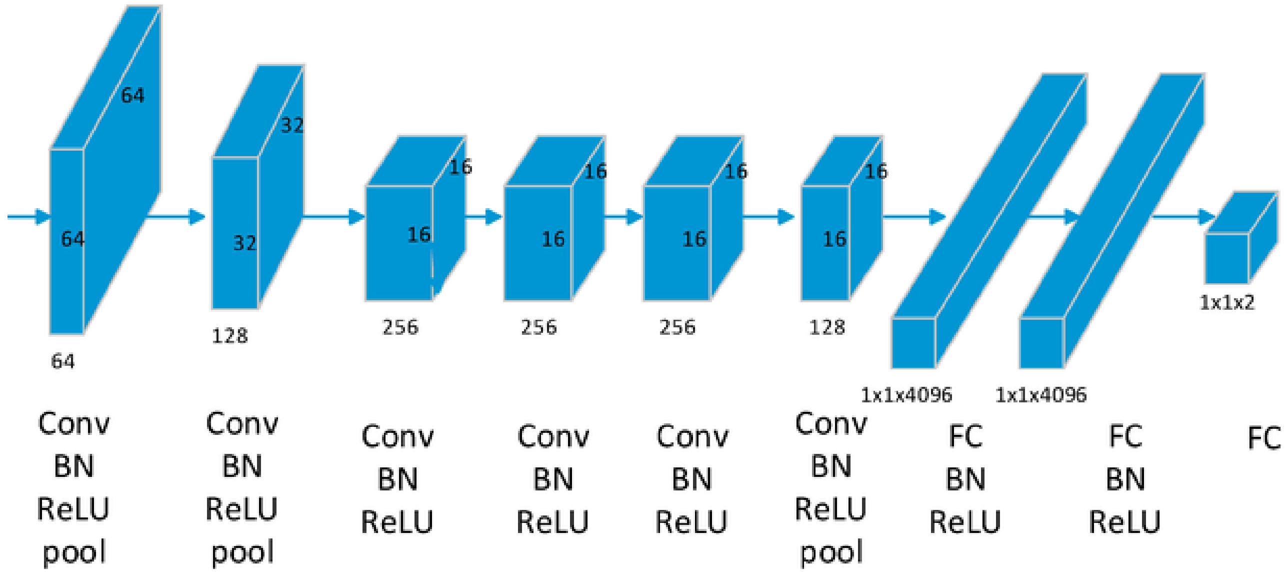


CNN architecture

- Stack the layers in an appropriate order

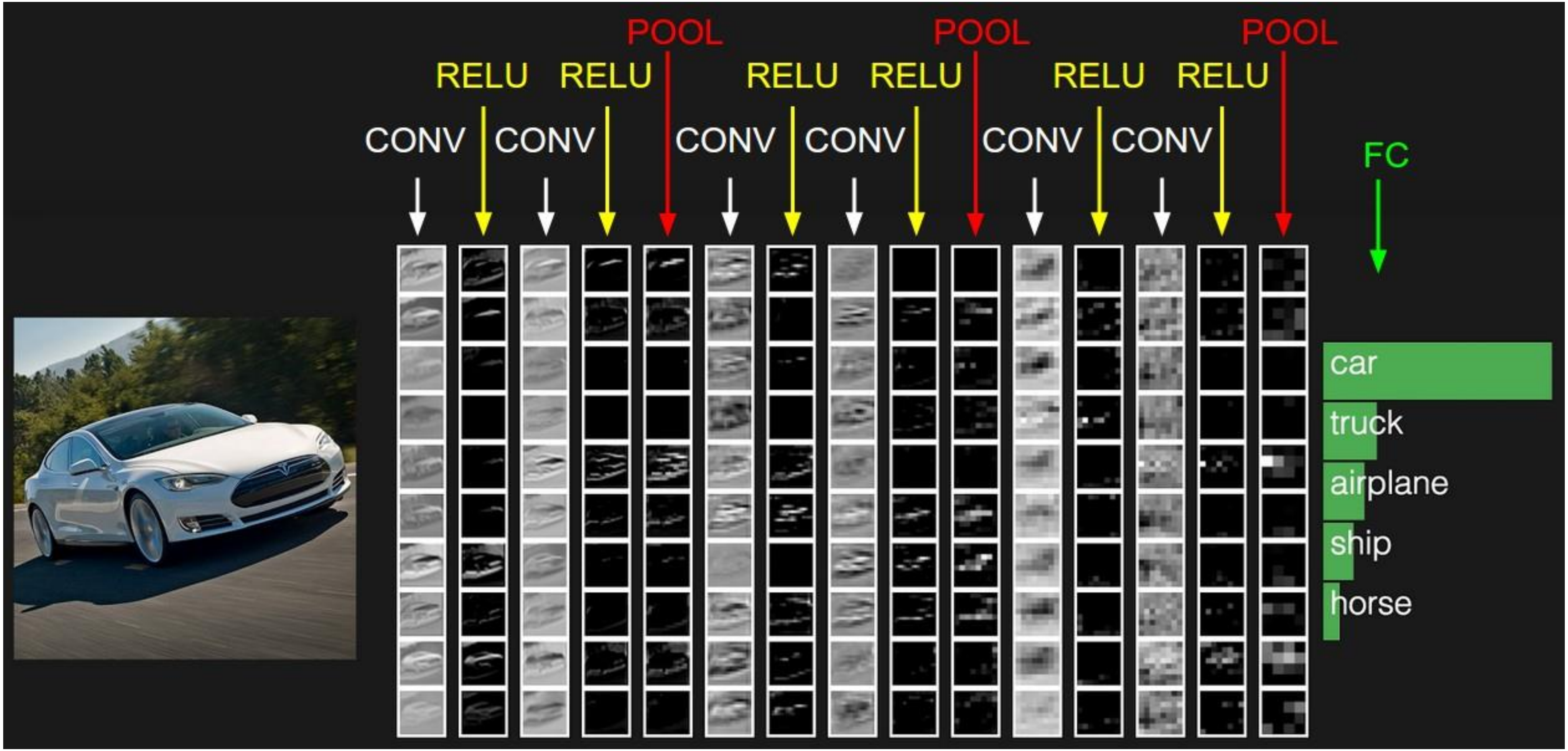


Babenko et. al.



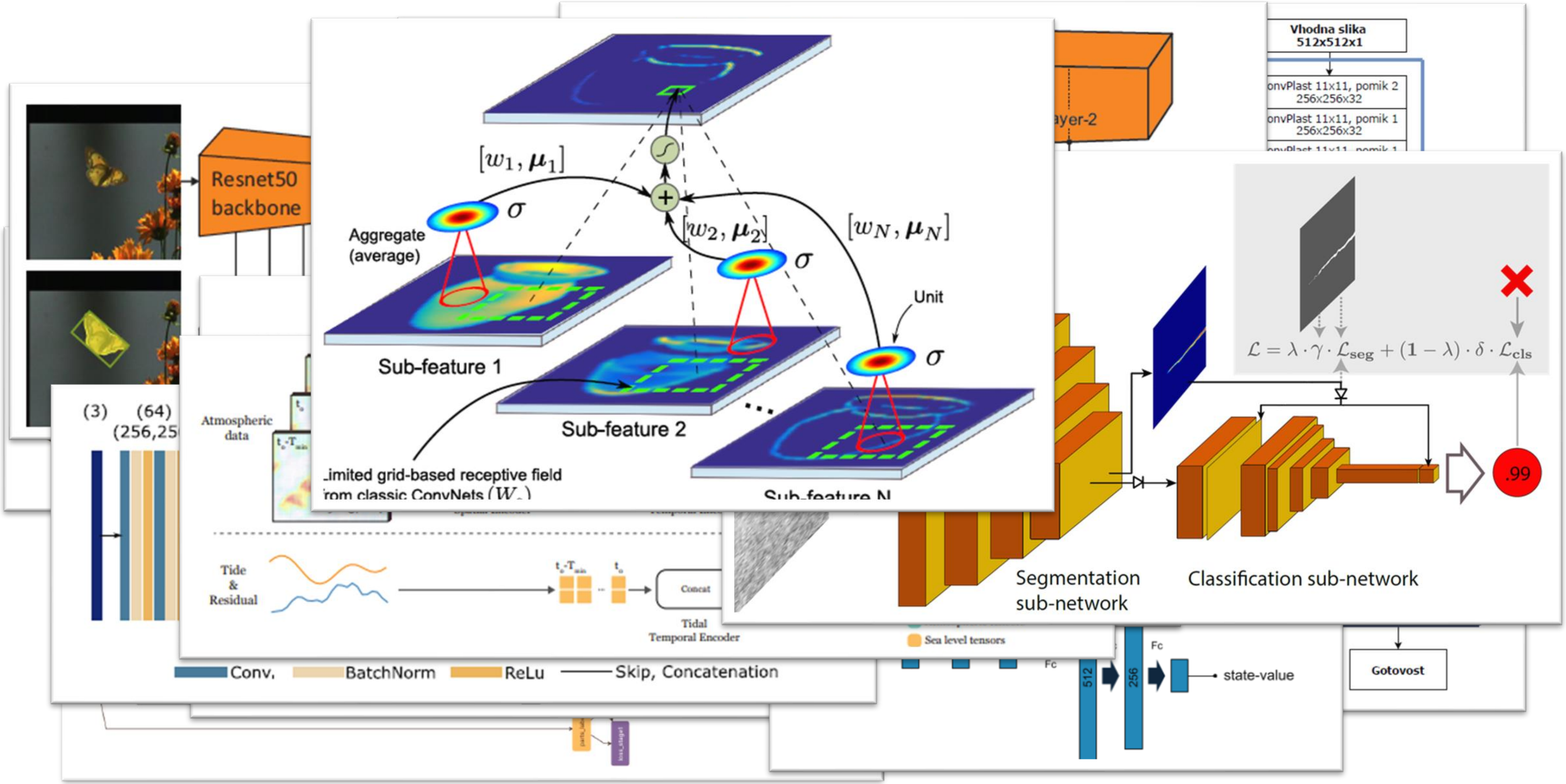
Hu et. al.

CNN architecture

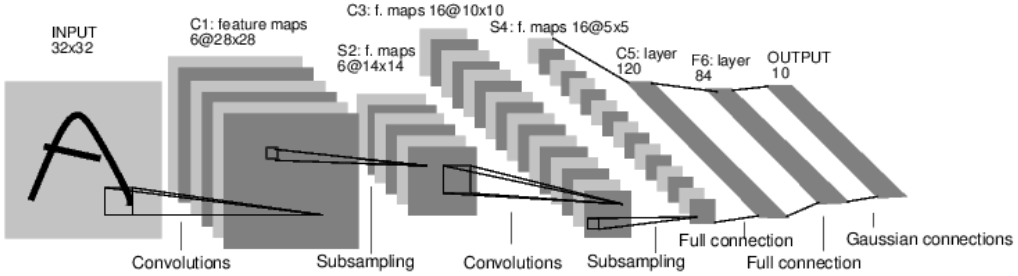


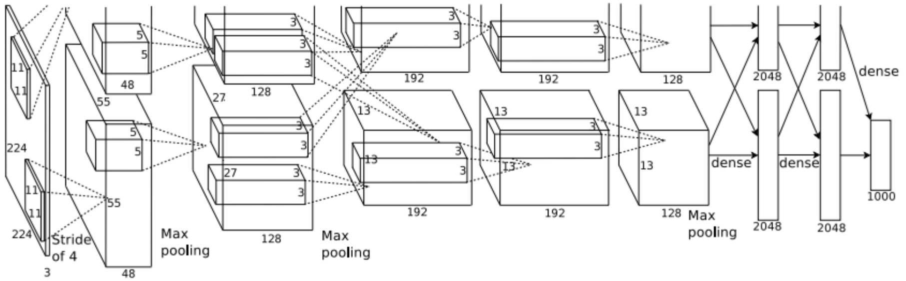
Slide credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

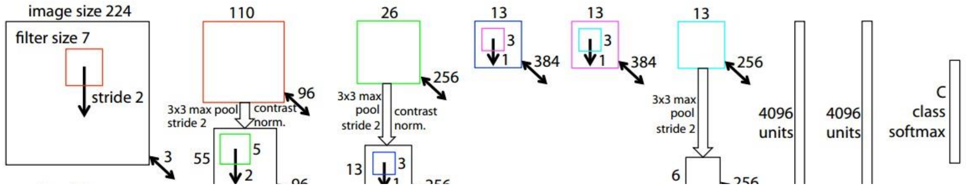
CNN architectures

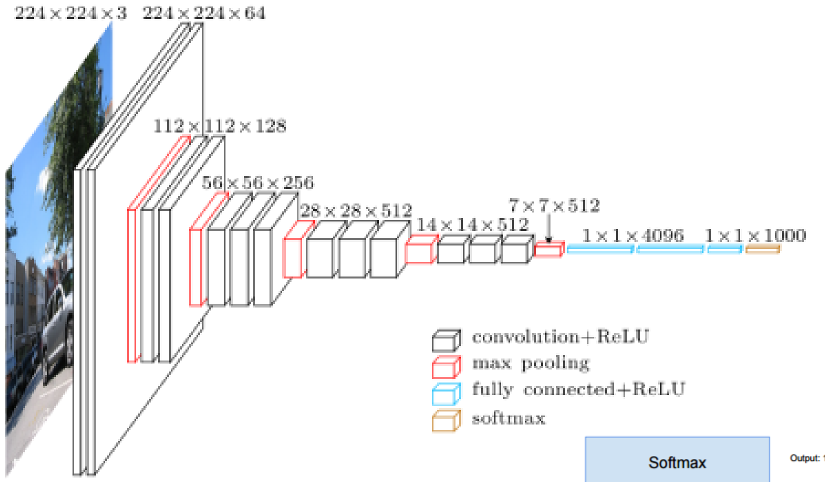


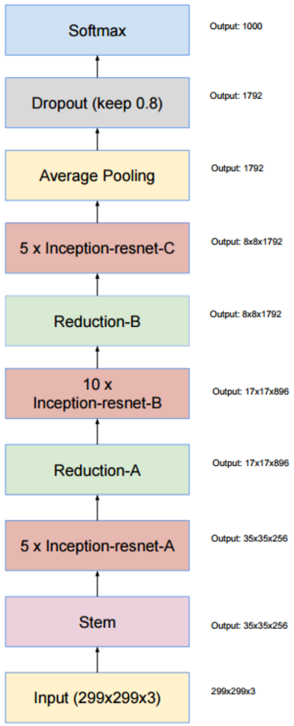
Case studies

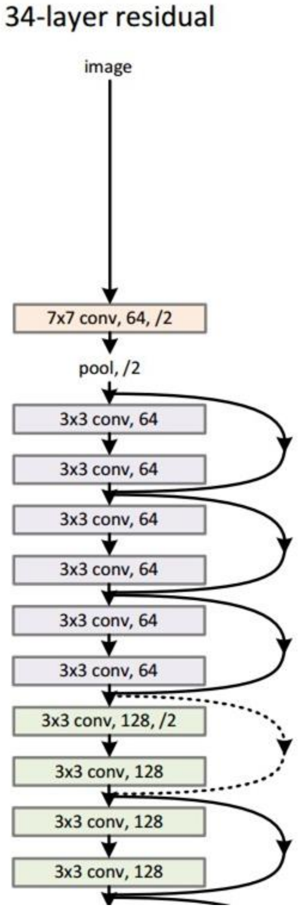


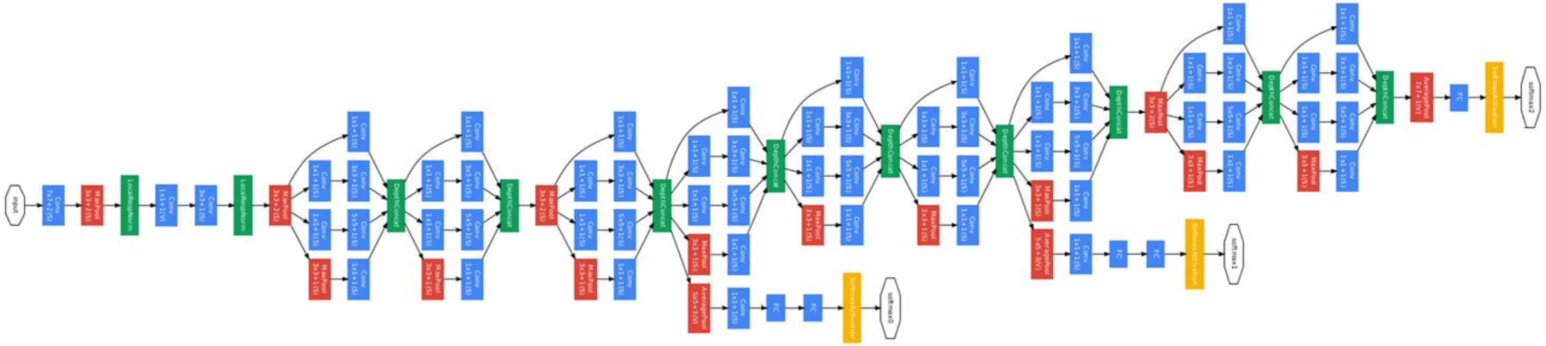












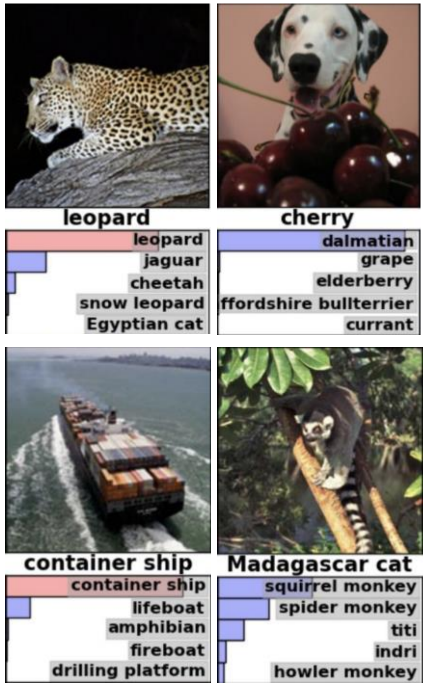
Increasing perfomance

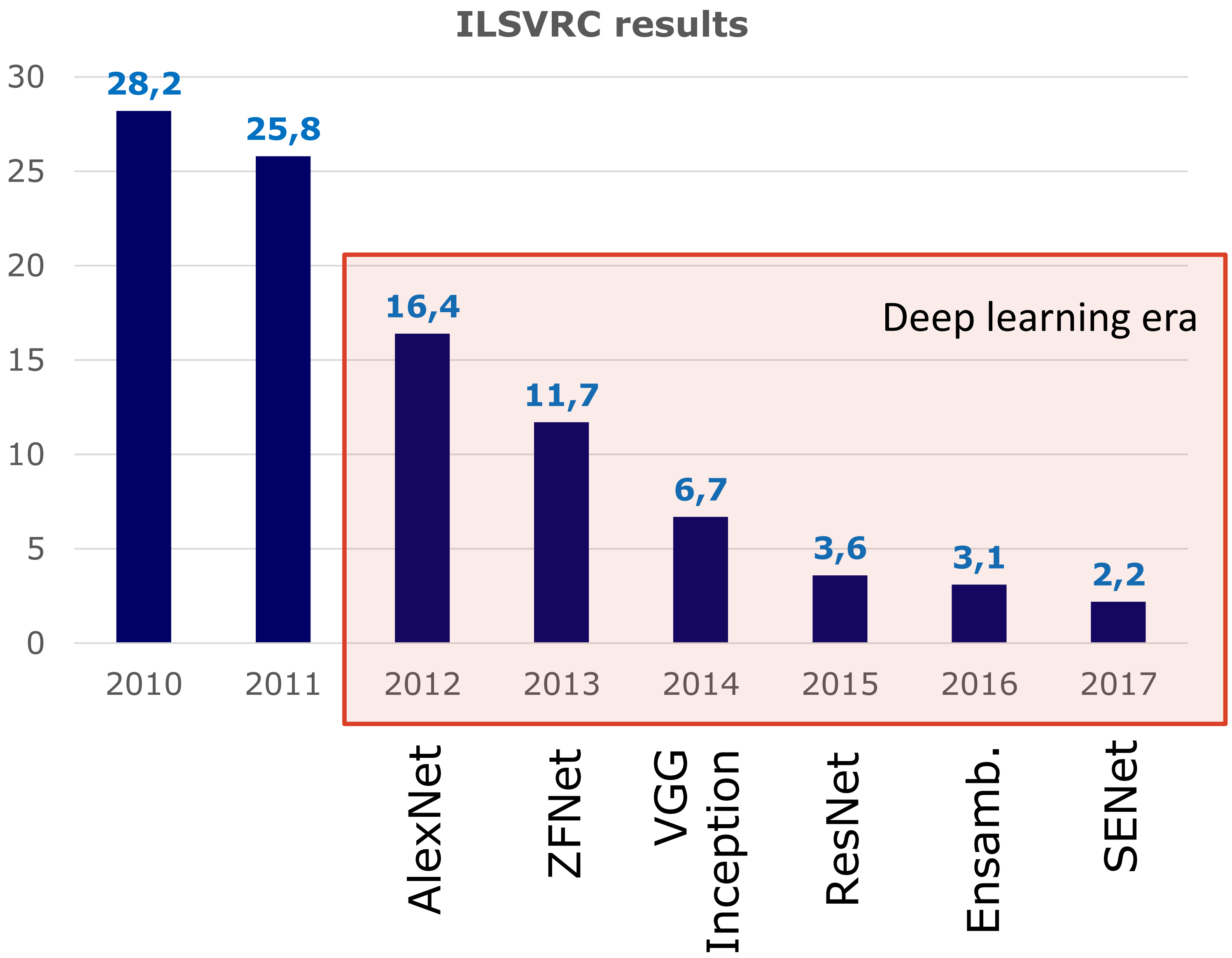
IMAGENET

1k categories

1,3M images

Top5 classification



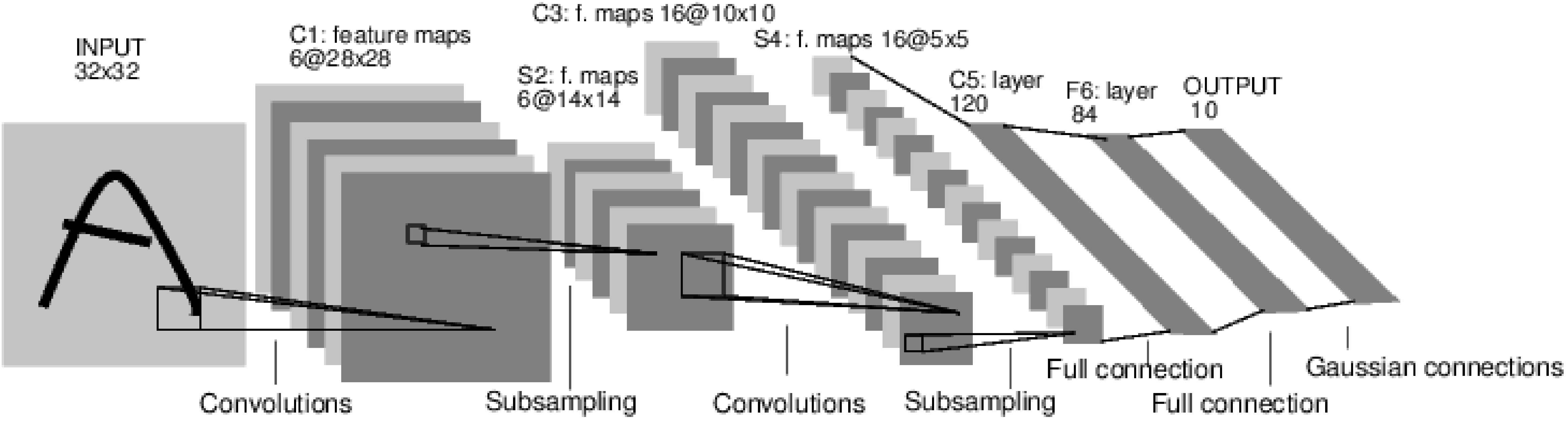


Architectures overview

- paperswithcode.com *[paperswithcode.com, 2022]*
  - Top 20 architectures in Convolutional Neural Networks

Method	Year	Papers
ResNet Deep Residual Learning for Image Recognition	2015	1461
VGG Very Deep Convolutional Networks for Large-Scale Image Recognition	2014	369
DenseNet Densely Connected Convolutional Networks	2016	300
AlexNet ImageNet Classification with Deep Convolutional Neural Networks	2012	280
VGG-16 Very Deep Convolutional Networks for Large-Scale Image Recognition	2014	258
MobileNetV2 MobileNetV2: Inverted Residuals and Linear Bottlenecks	2018	201
EfficientNet EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks	2019	154
Darknet-53 YOLOv3: An Incremental Improvement	2018	142
ResNeXt Aggregated Residual Transformations for Deep Neural Networks	2016	120
GoogLeNet Going Deeper with Convolutions	2014	119
Xception Xception: Deep Learning With Depthwise Separable Convolutions	2017	94
SqueezeNet SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size	2016	71
Inception-v3 Rethinking the Inception Architecture for Computer Vision	2015	67
CSPDarknet53 YOLOv4: Optimal Speed and Accuracy of Object Detection	2020	46
MobileNetV1 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications	2017	44
LeNet	1998	44
Darknet-19 YOLO9000: Better, Faster, Stronger	2016	44
WideResNet Wide Residual Networks	2016	42
ShuffleNet ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices	2017	36
MobileNetV3 Searching for MobileNetV3	2019	34

LeNet-5



CONV
5x5

POOL
F=2, S=2

CONV
5x5

POOL
F=2, S=2

FC

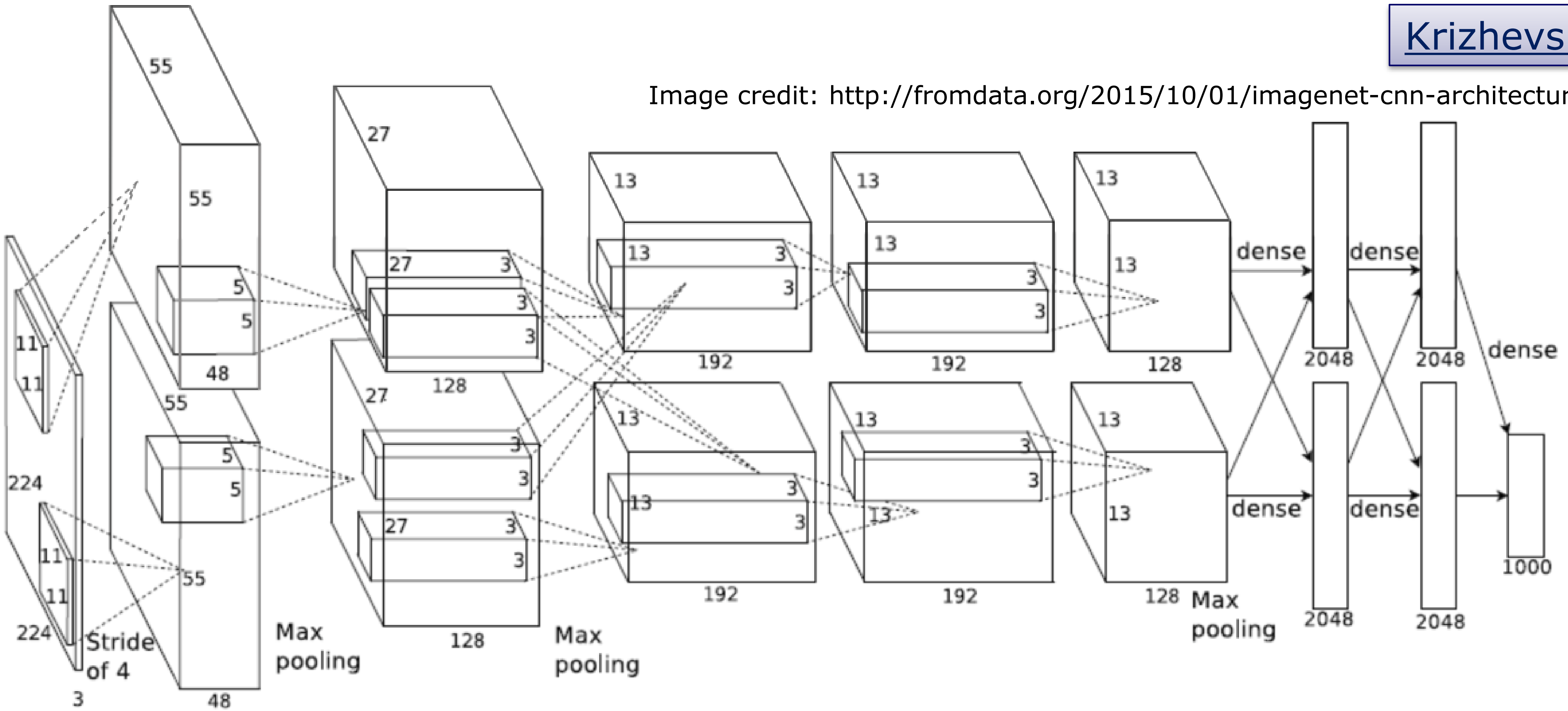
FC

[LeCun et al., 1998]

AlexNet

[Krizhevsky, 2012]

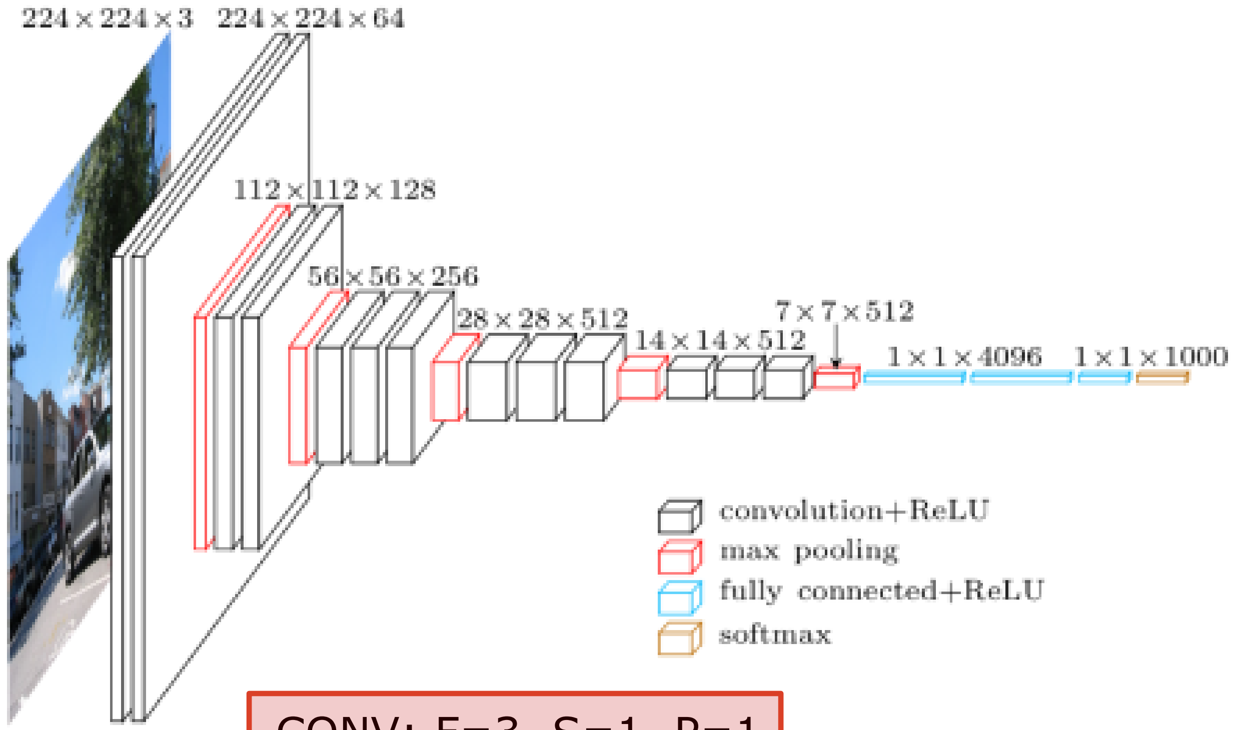
Image credit: http://fromdata.org/2015/10/01/imagenet-cnn-architecture-image/



CONV1	POOL	CONV2	POOL	CONV3	CONV4	CONV5	POOL	FC6	FC7	FC8
F=11	F=3	F=5	F=3	F=3	F=3	F=3	F=3	4096	4096	1000
S=4	S=2	S=1	S=2	S=1	S=1	S=1	S=2			
		P=2		P=1	P=1	P=1				

- ReLU, data augmentation, Dropout, Momentum, L2 regularisation

VGG



CONV: F=3, S=1, P=1
POOL: F=2, S=2

- Classical CNN backbone shape
- VGG16, VGG19

[Simonyan & Zisserman, 2014]

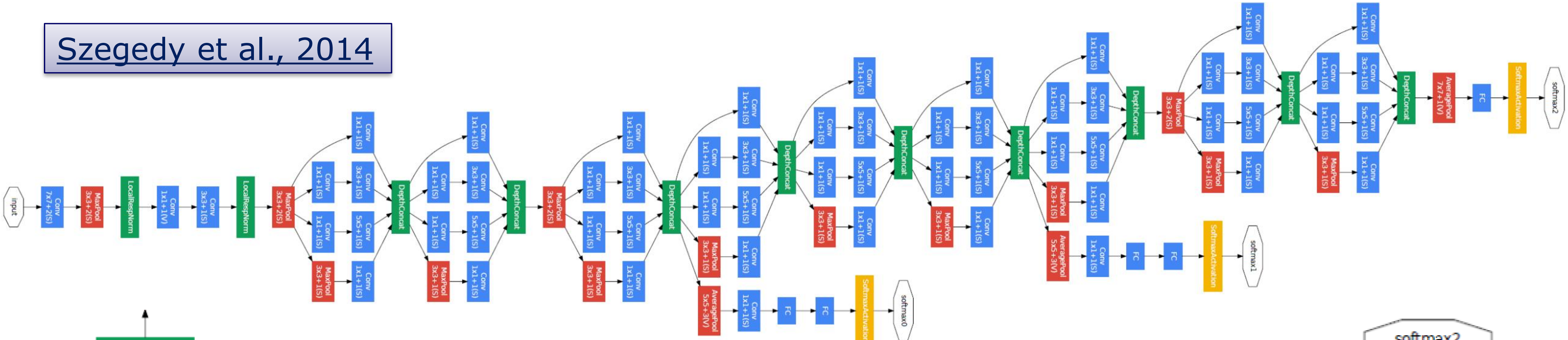
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

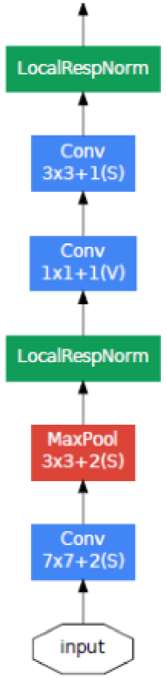
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

GoogLeNet / Inception

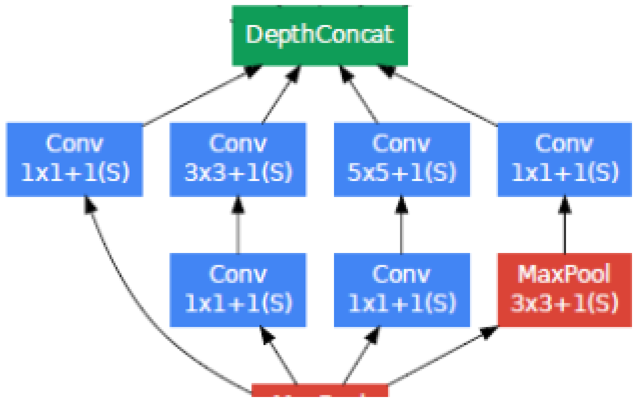
Szegedy et al., 2014



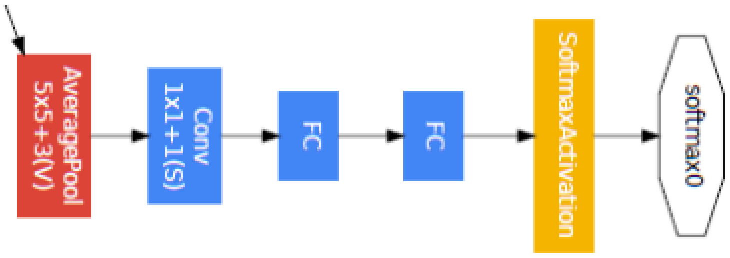
Stem network



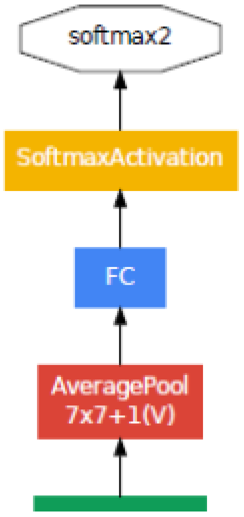
Inception module



Auxiliary output



Clasifier output

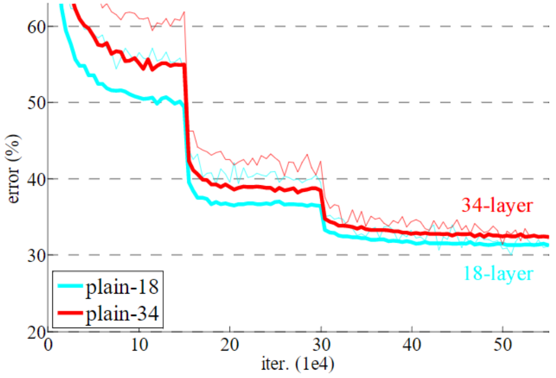


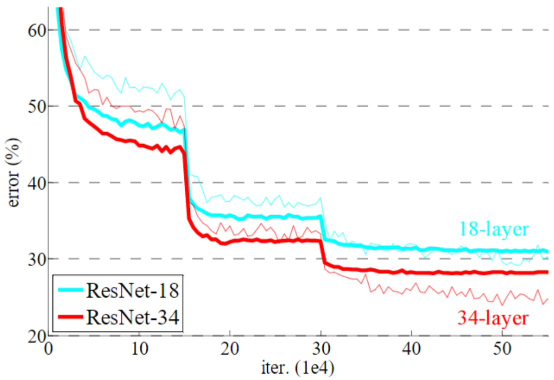
ResNet

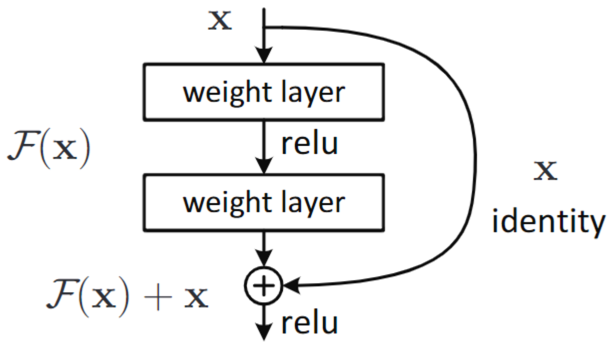
- Going deeper!
- Plain deep networks do not work
- Shortcut connections!
  - Figth vanishing gradient problem
- Learn residual functions

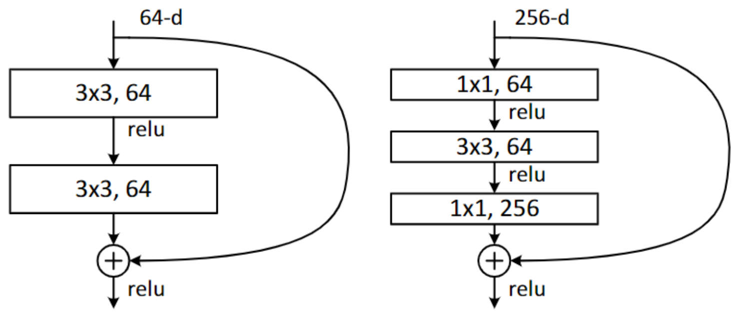
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

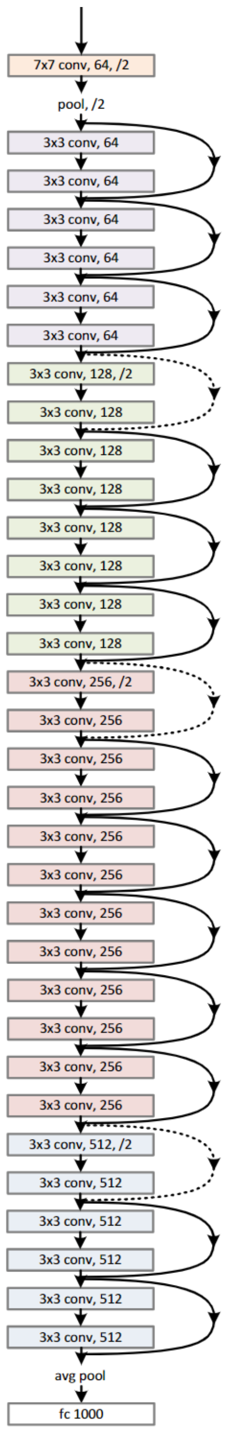
- Bottleneck building blocks
- Very deep networks:
  - 152, 101, 50, 34, 18

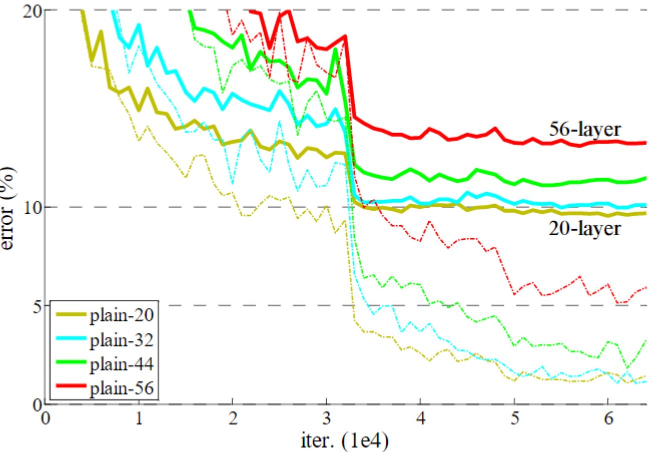


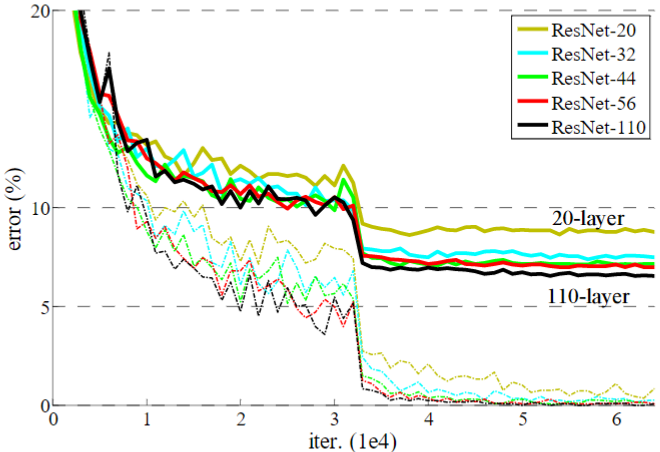


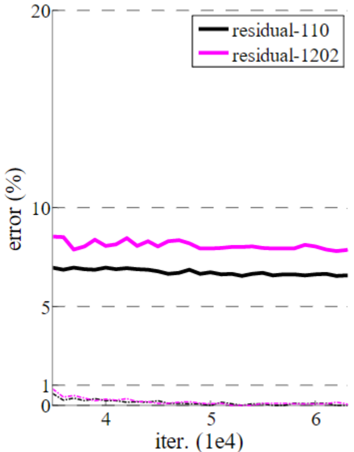












	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

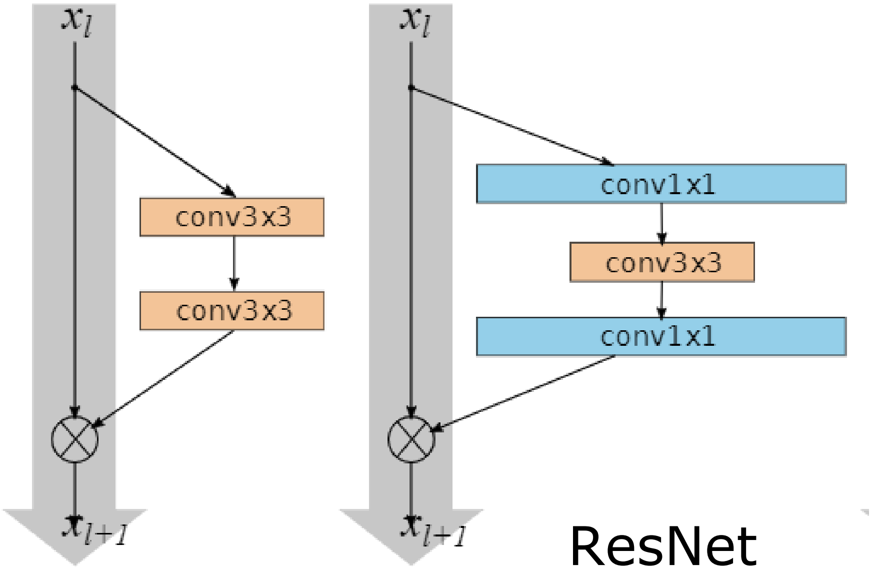
He et al., 2015

Wide ResNet

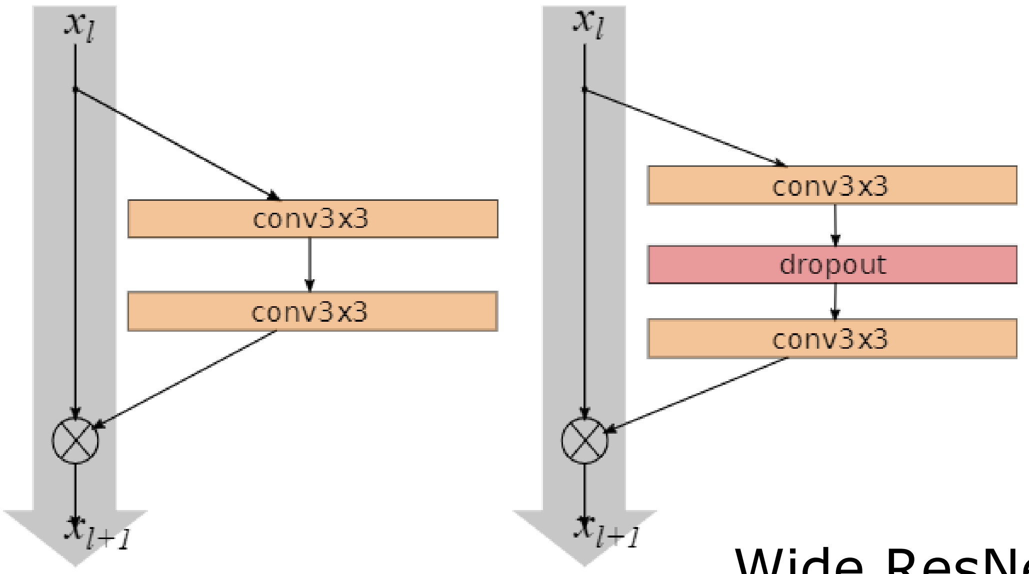
- Wide Residual Networks
  - Width instead of depth
  - Adding more feature planes
  - Parallelisable

[Zagoruyko et al. 2016]

group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3, 16]$
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$



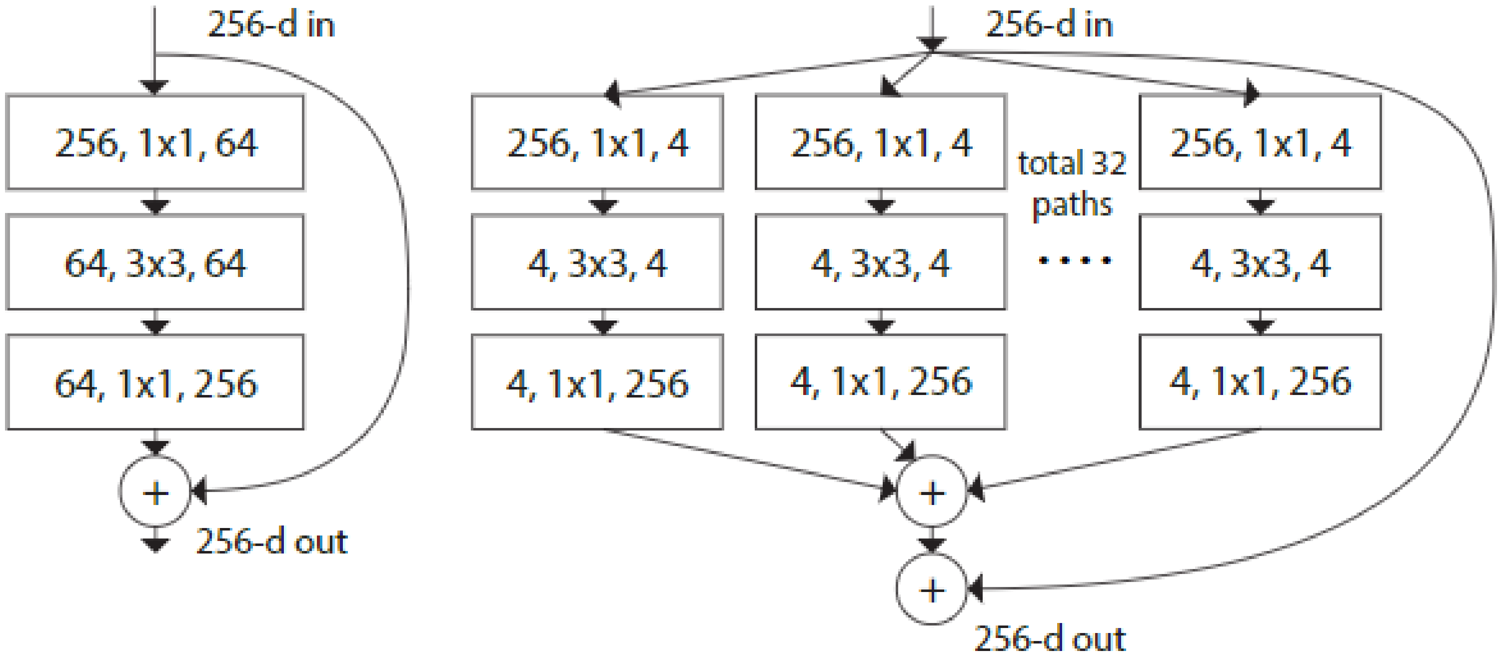
ResNet



Wide ResNet

ResNeXt

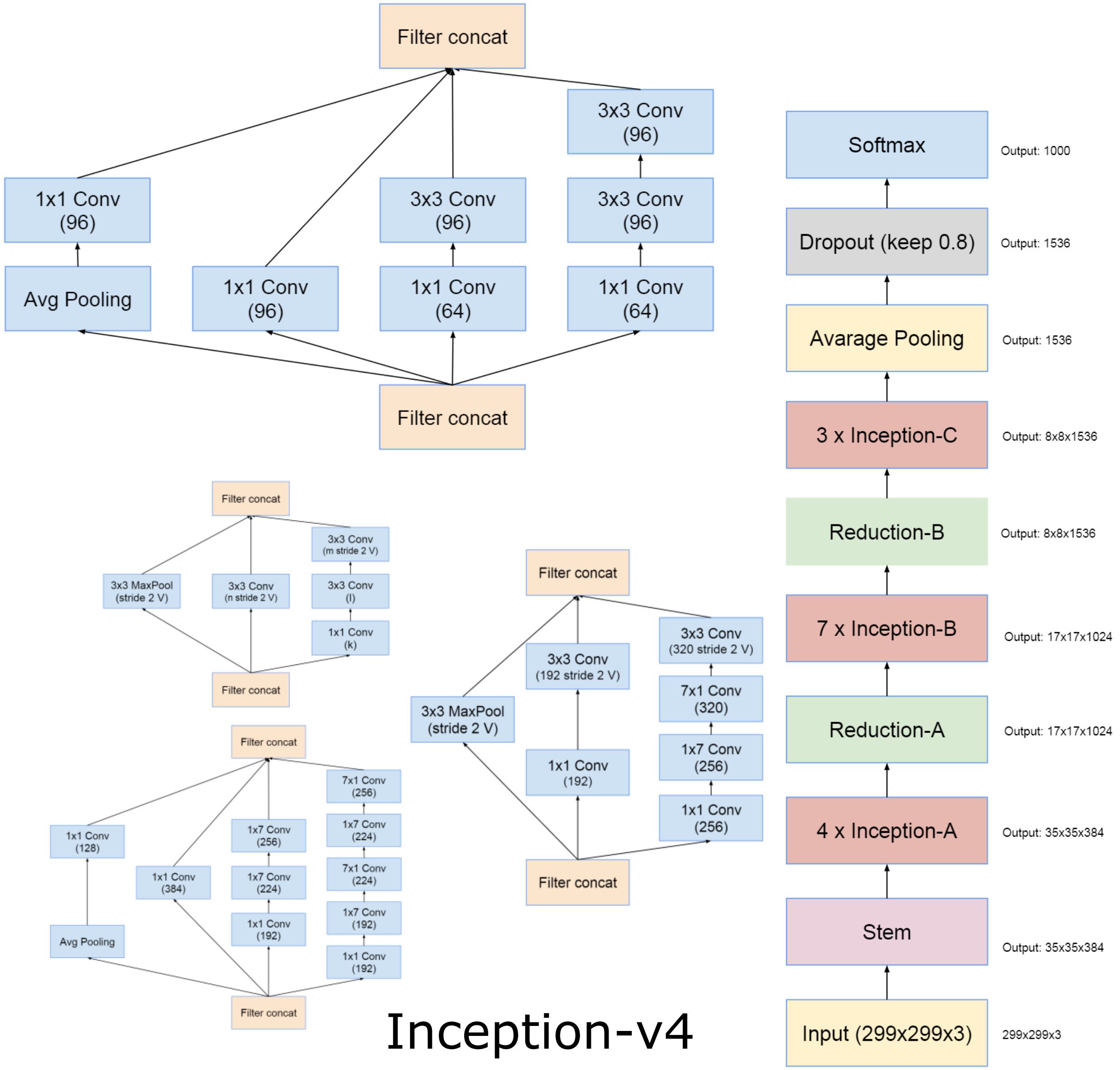
- Aggregated Residual Transformations for Deep Neural Networks
  - ResNet blocks widened with multiple pathways
  - That are summed together (in contrast to Inception)



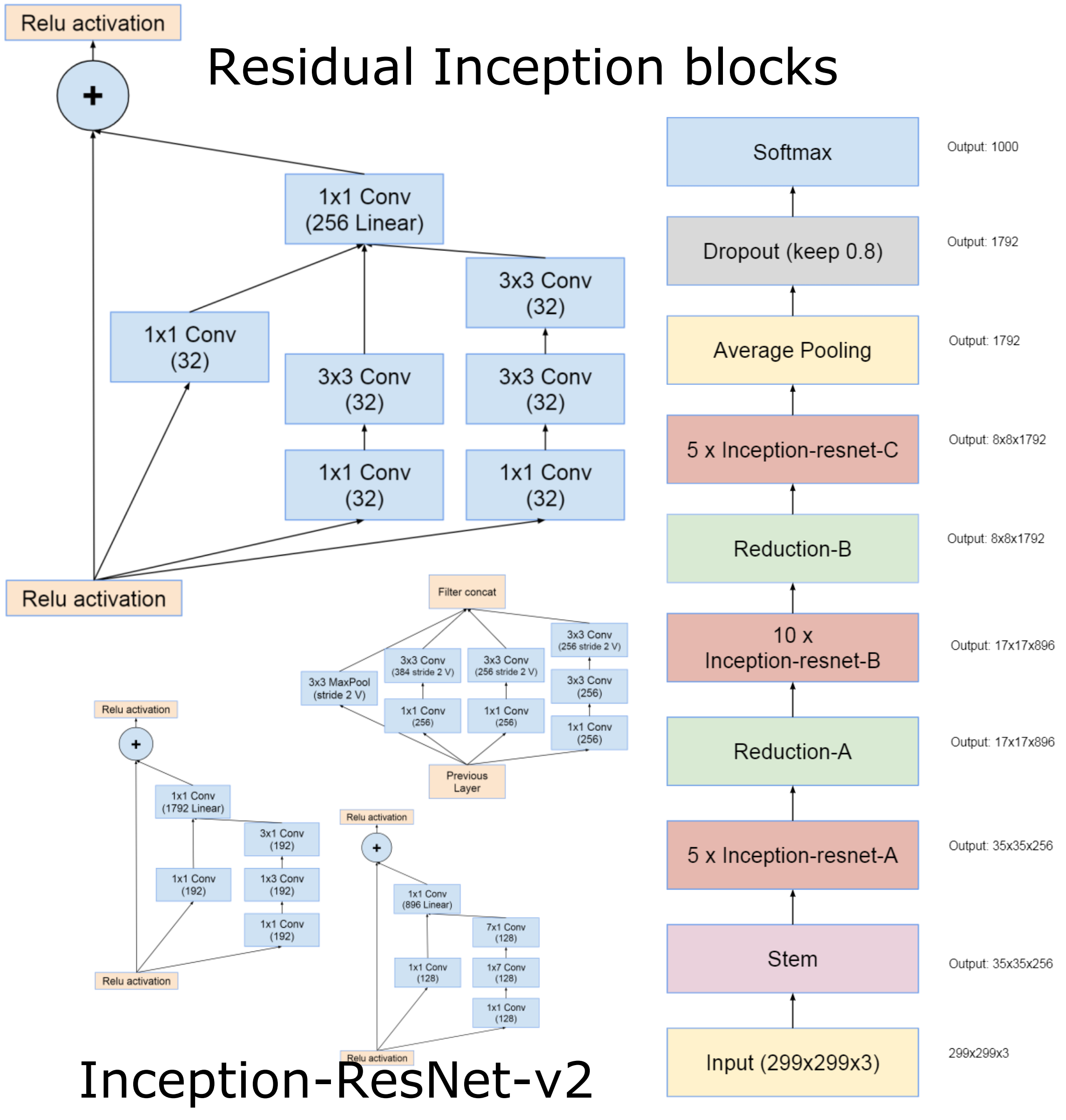
Xie et al. 2016

Inception-v4 and Inception-ResNet

Szegedy et al., 2016



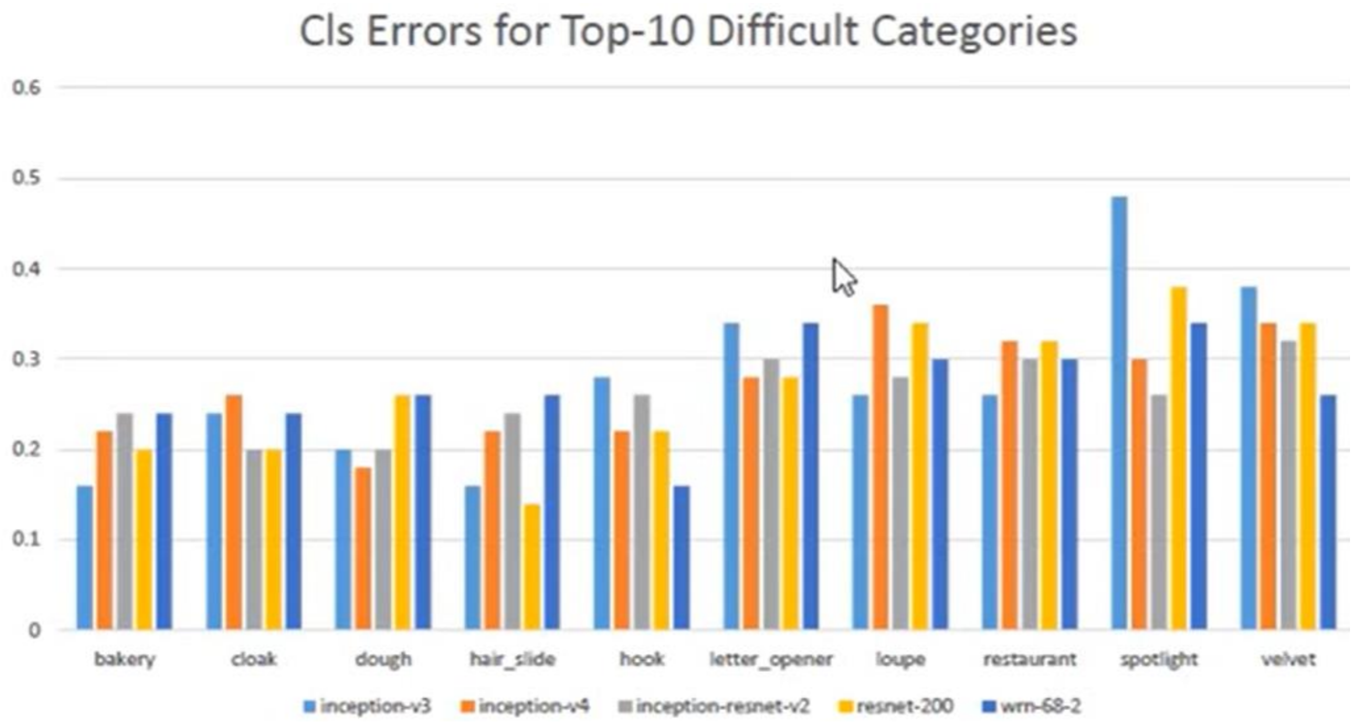
Inception-v4



Inception-ResNet-v2

Ensemble methods

- Merge the results of different models



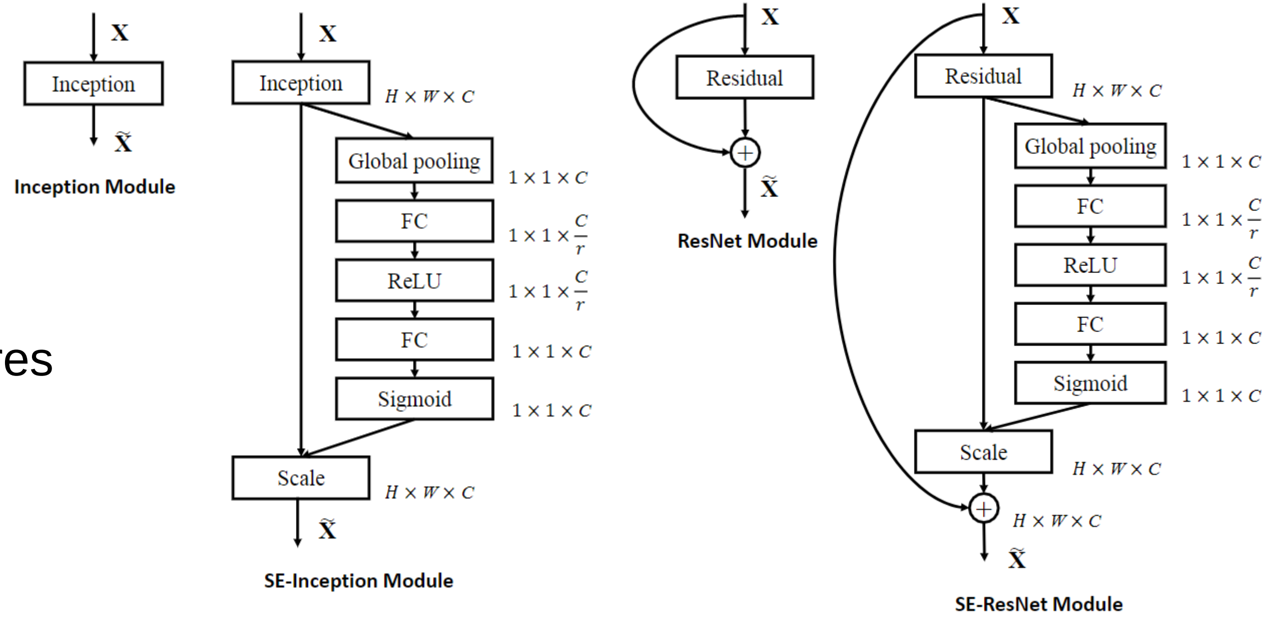
Trimps-Soushen@ILSVRC2016

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

SENet

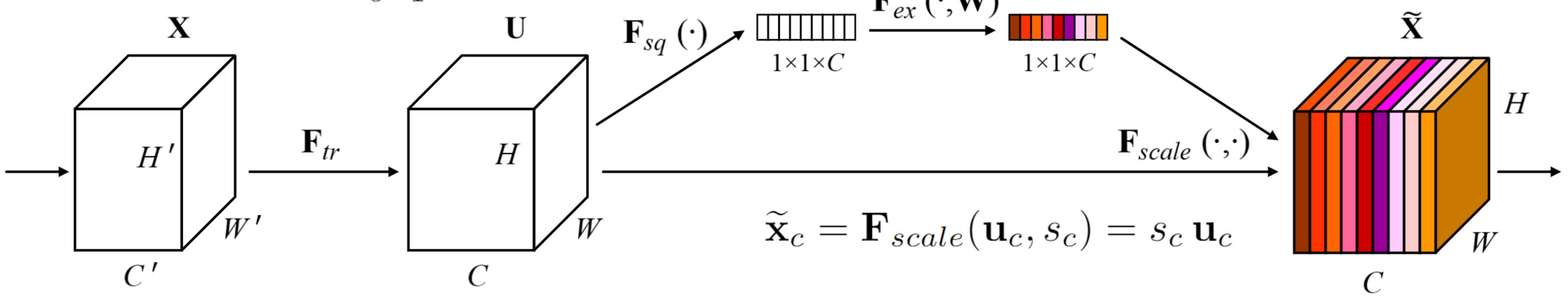
- Squeeze-and-Excitation Networks
- Explore chanel relationships
- SE blocks
  - adaptively recalibrate
    channel-wise feature responses
  - Can be stacked together in SENet architectures
  - Can improve CNN architectures

[Hu et al., 2017]

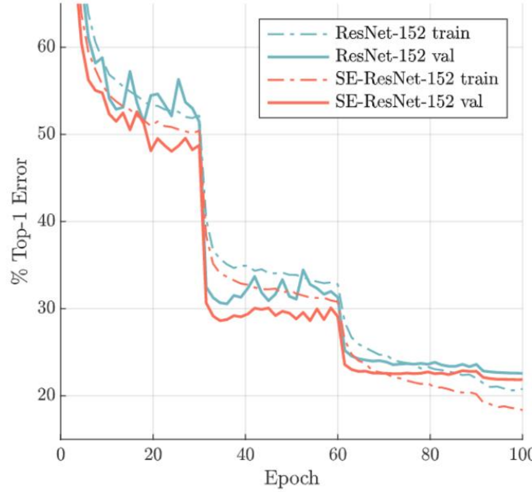


$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s$$

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z}))$$

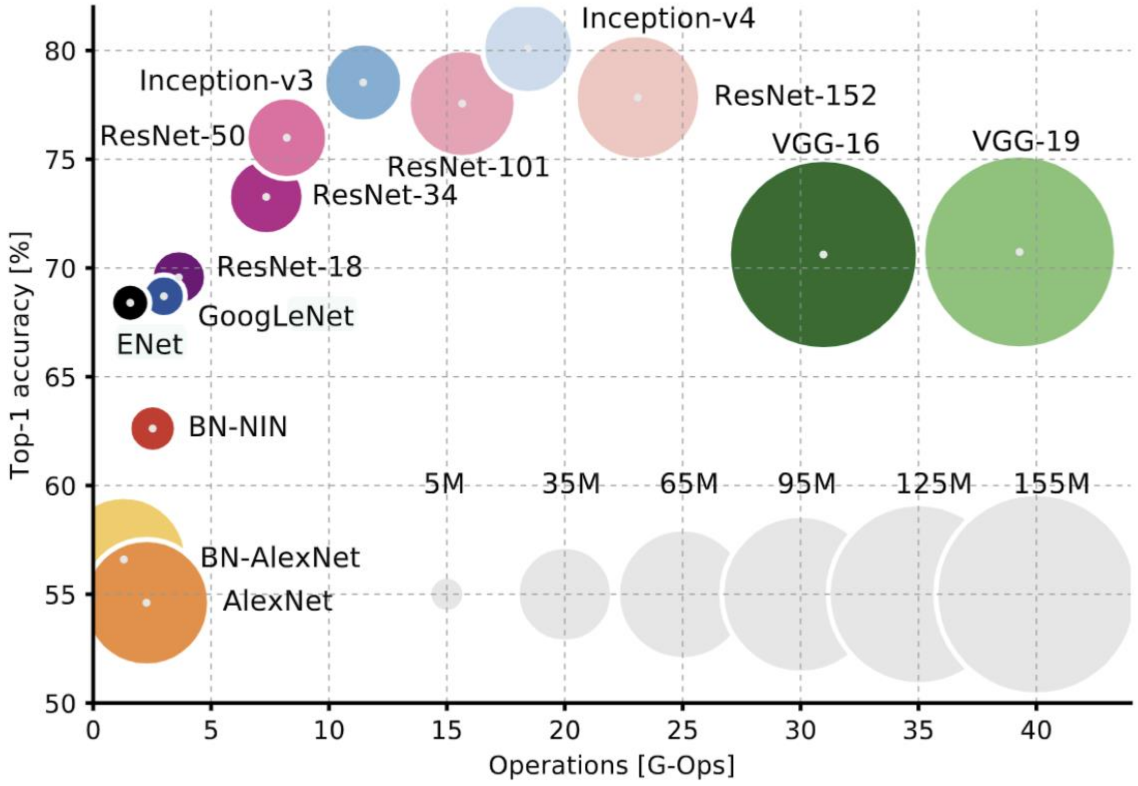


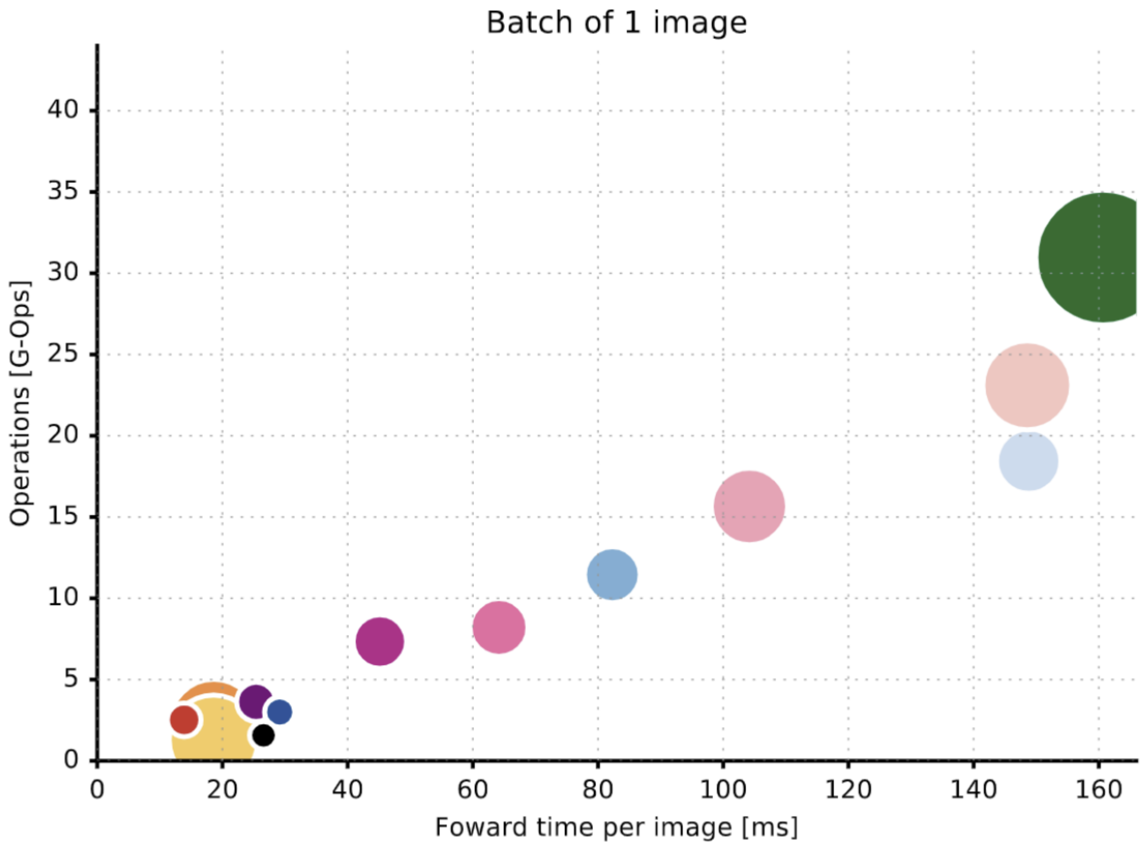
$$\widetilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c\,\mathbf{u}_c$$



Benchmarking architectures

- Accuracy, number of parameters, number of operations and inference time



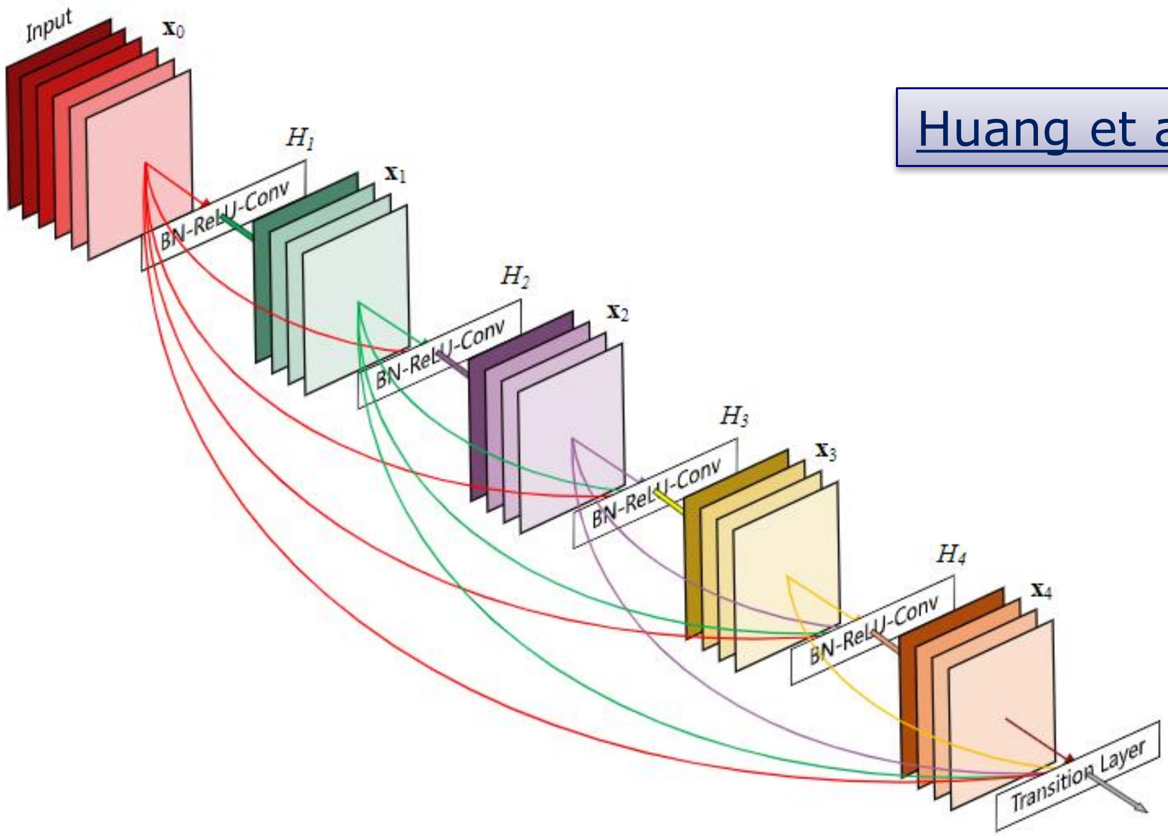


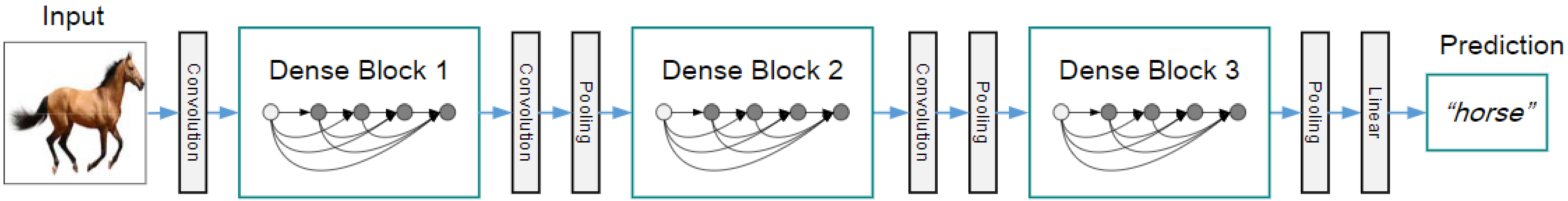
[Canziani et al., 2017]

DenseNet

[Huang et al. 2017](#)

- Densely Connected Convolutional Networks
  - Every layer connected to every other layer in a feed-forward fashion
  - Dense connectivity
  - Model compactness
  - Strong gradient flow
  - Implicit deep supervision
  - Feature reuse

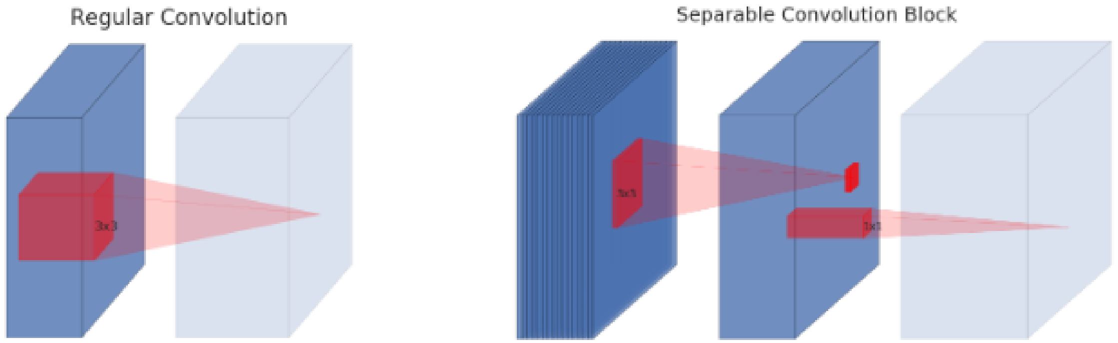


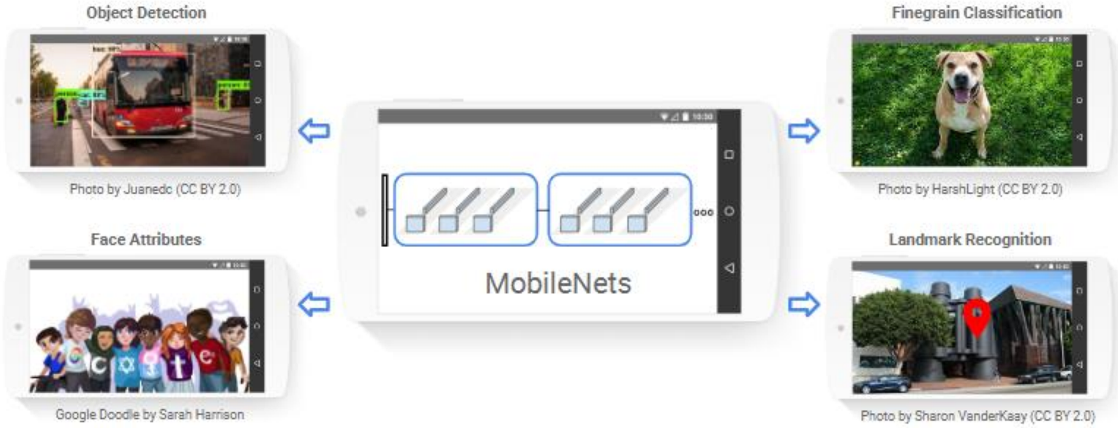


MobileNets

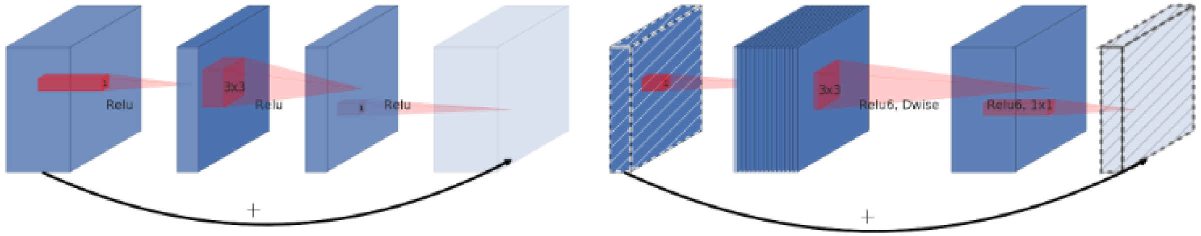
- Efficient Convolutional Neural Networks for Mobile Applications
- Efficient models for mobile and embedded vision applications
- Depthwise separable convolution:
  - Depthwise convolution
  - Pointwise (1x1) convolution

[Howard et al. 2017]





- MobileNetV2: Inverted Residuals and Linear Bottlenecks



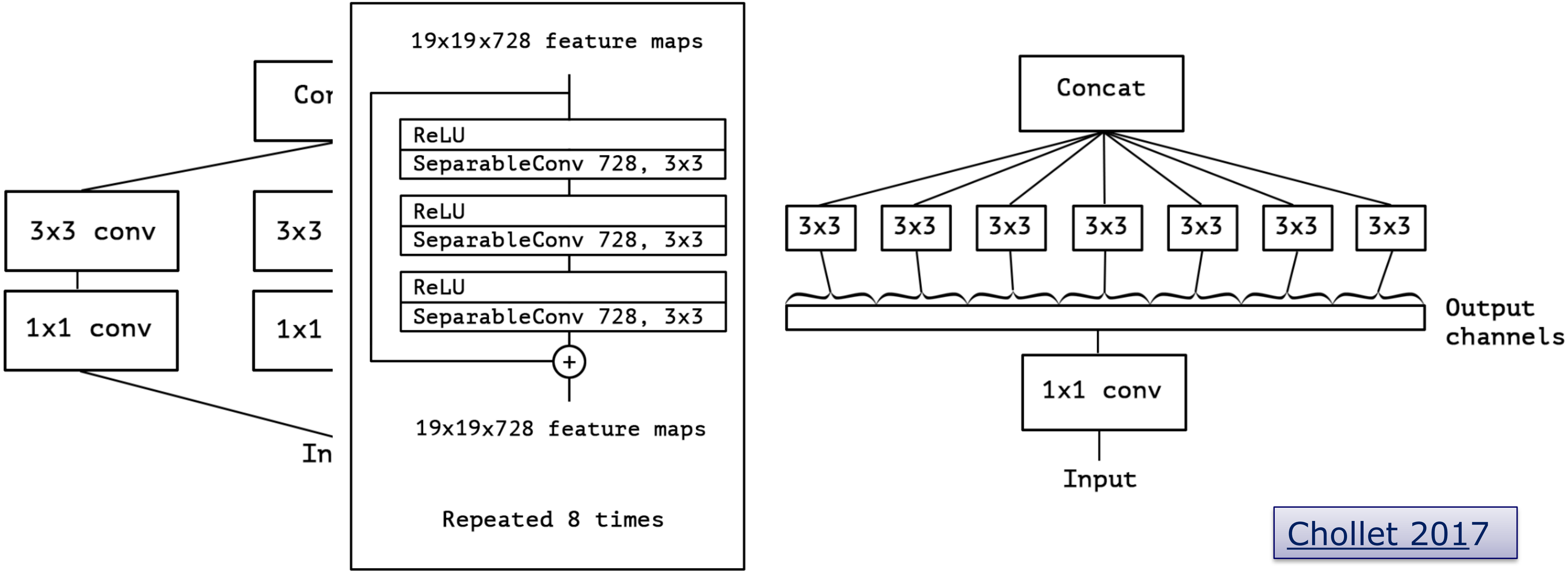
[Sandler et al. 2018]

- MobileNetV3: NAS+ NetAdapt

[Howard et al. 2019]

Xception

- Extreme Inception – mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled
- Inception modules replaced by depthwise separable convolutions with residual conn.
- Continuum between regular and depthwise separable convolution

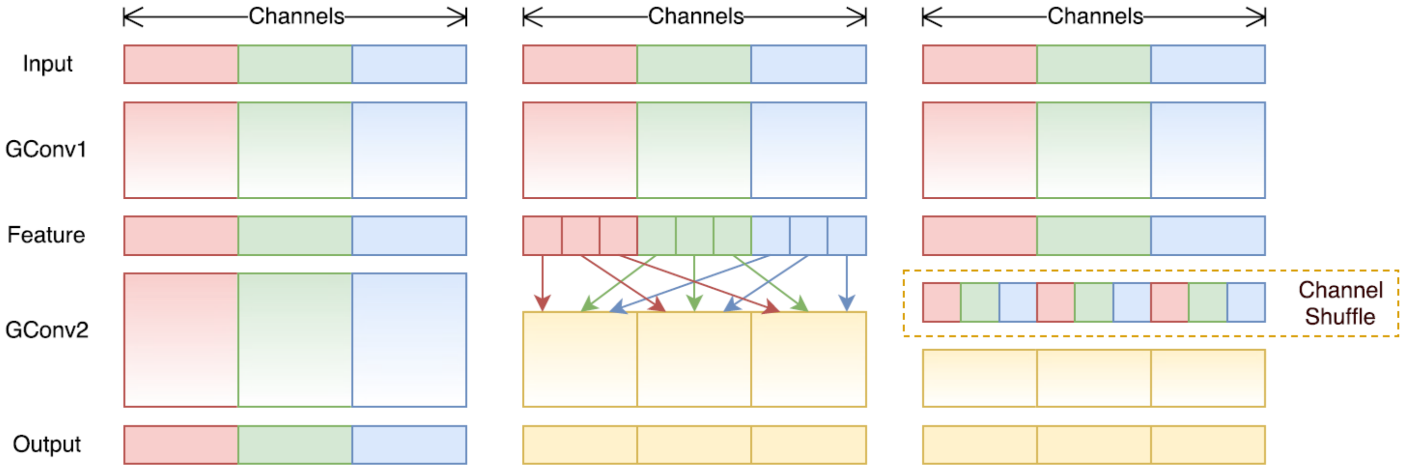


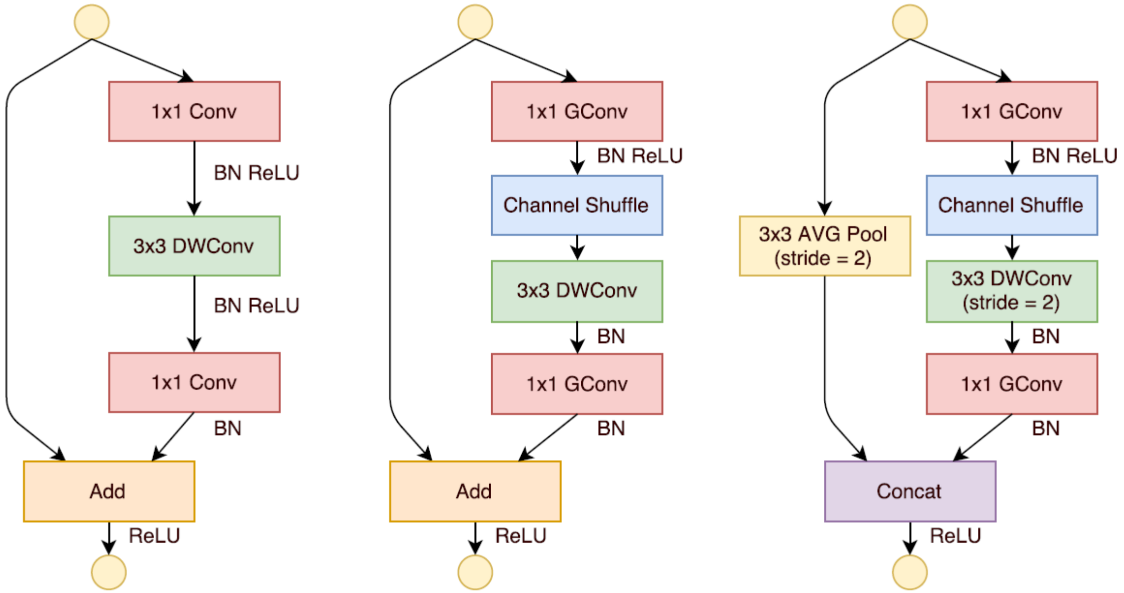
Chollet 2017

ShuffleNet

- An Extremely Efficient Convolutional Neural Network for MobileDevices
- Group convolutions
- Bottleneck units
- Channel shuffle
- Allows using wider feature maps
- 13x faster than AlexNet at the same accuracy
- Typicaly outperforms other architectures at the same MFLOPS

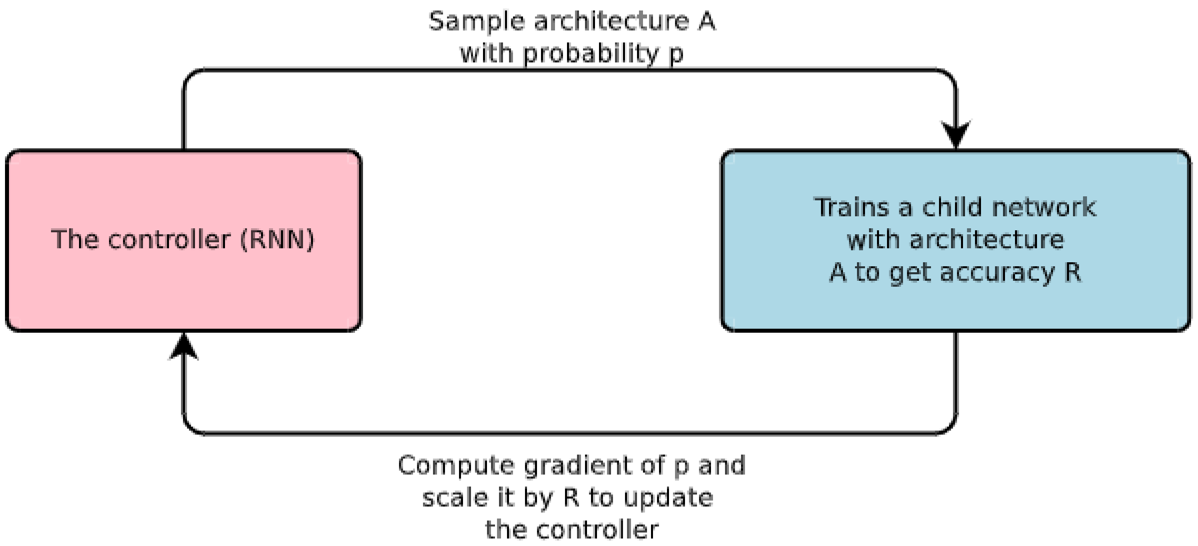
Zhang et al. 2018

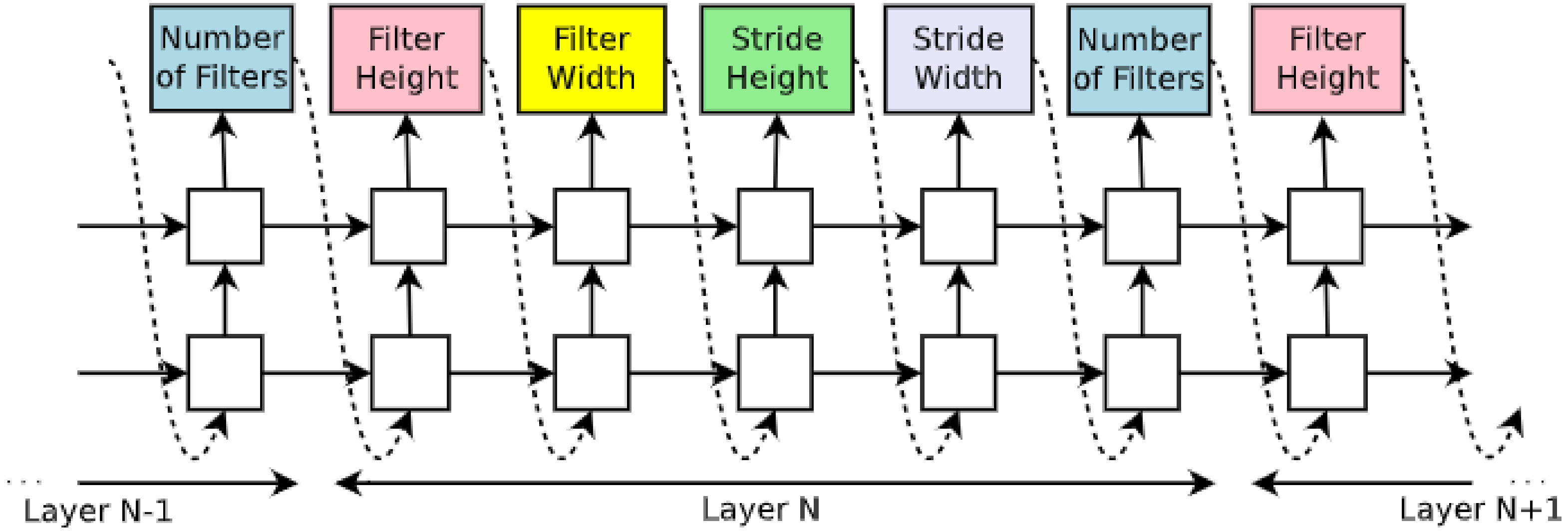




Neural Architecture Search

- Neural architecture search with reinforcement learning
- Recurrent network to generate model descriptions of neural networks
- Maximising the expected accuracy of the generated architectures on a validation set



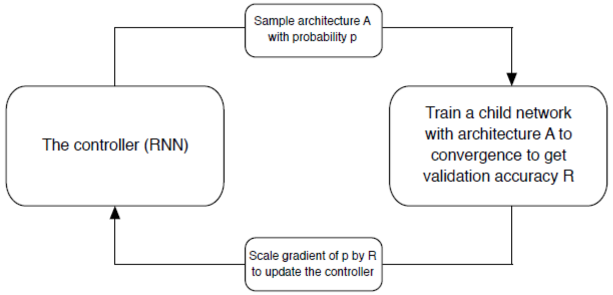


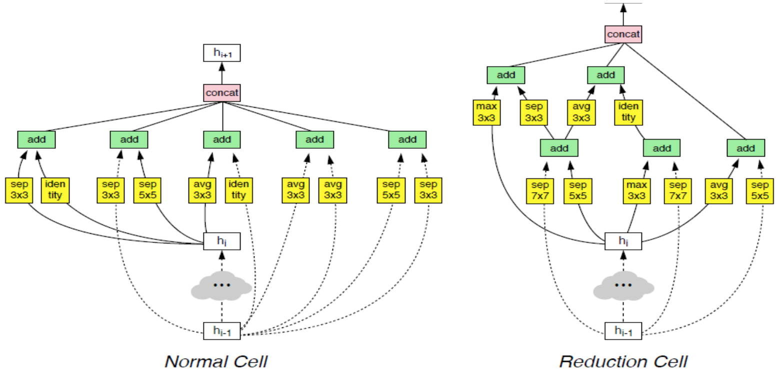
[Zoph & Le 2016]

NASNet

[Zoph et al. 2017]

- Neural Architecure Search
  - Search the space of architetures to find the optimal one given available resources
  - 500 GPUs across 4 days resulting in 2,000 GPU-hours on NVidia P100

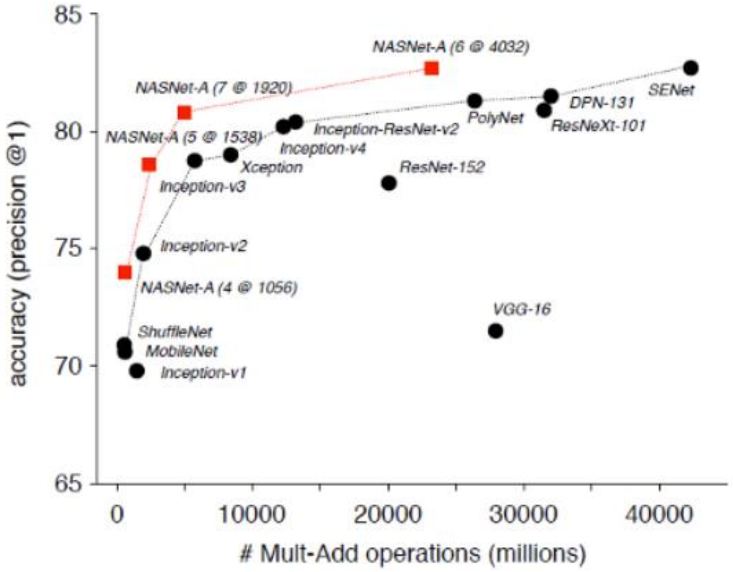


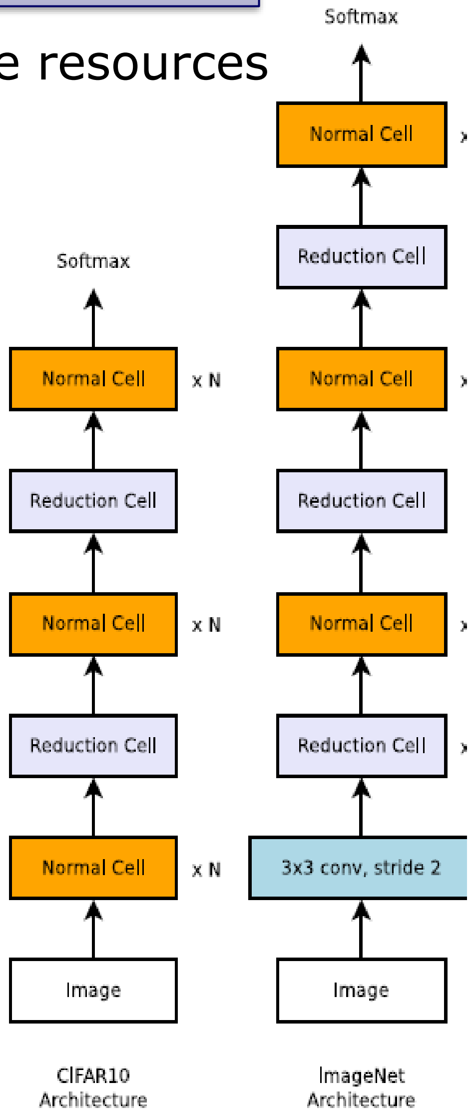


Best convolutional cells (NASNet-A) for CIFAR-10

Available operations to select from:

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv





- Other architecture search methods:
  - AmoebaNet, Real et al., 2018
  - MoreMNAS, Chu et. al, 2019, …

EfficientNet

- Scaling the network in

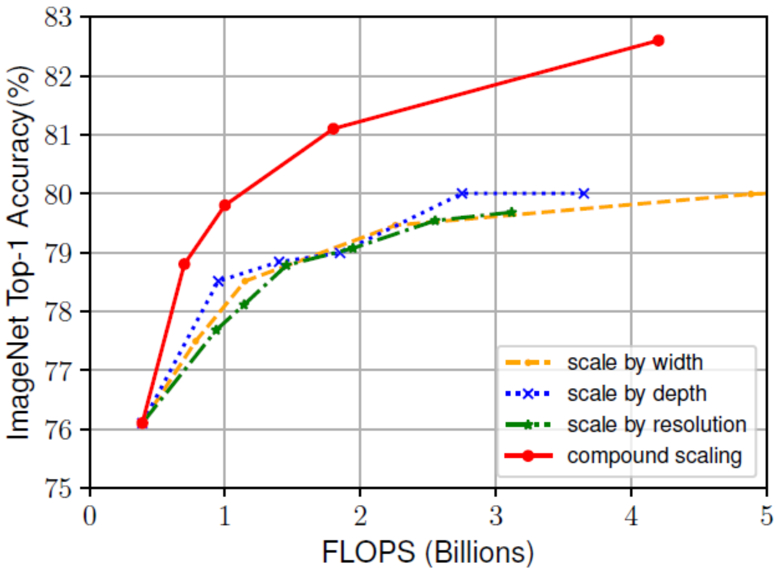
$$\text{depth: } d = \alpha^{\phi}$$

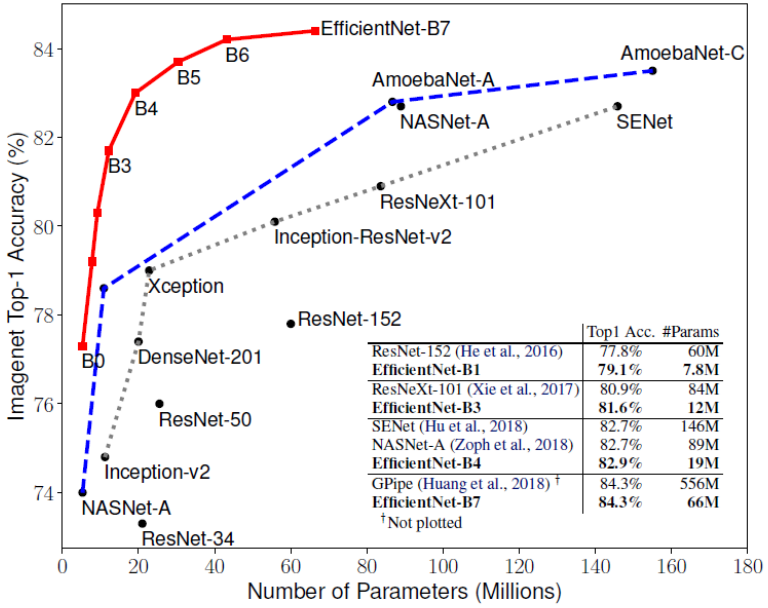
$$\text{width: } w = \beta^{\phi}$$

$$\text{resolution: } r = \gamma^{\phi}$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

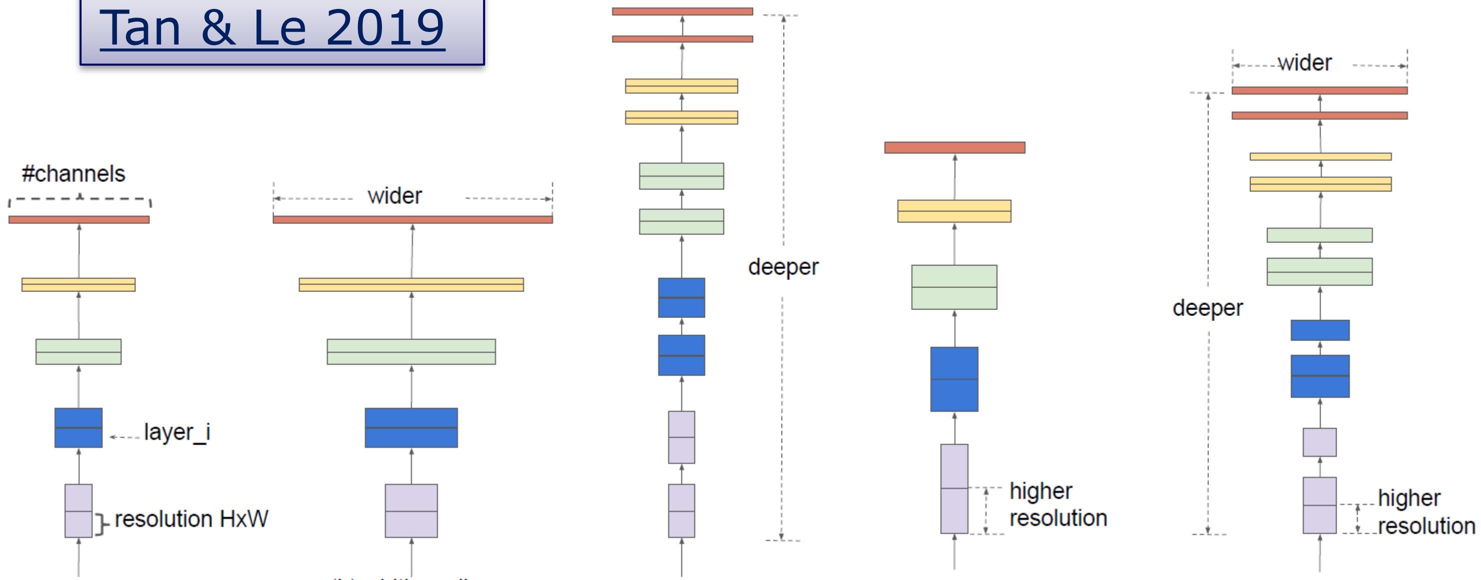




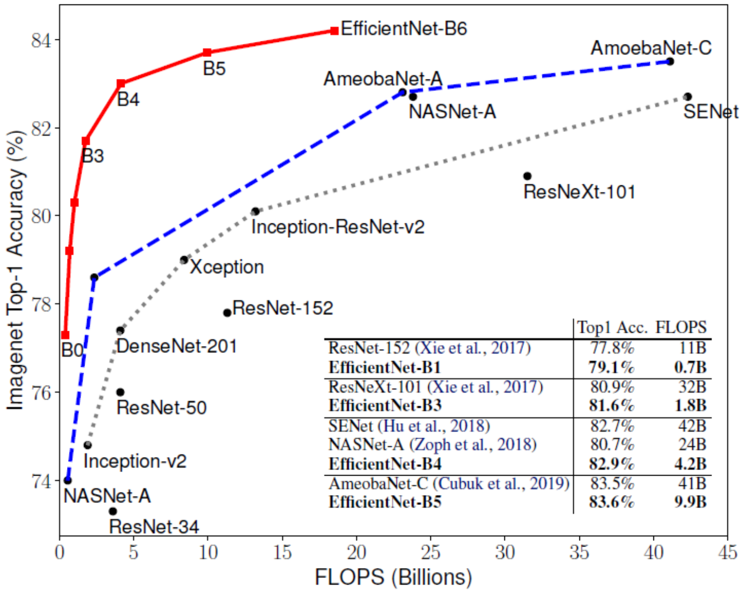
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

[Tan & Le 2019]



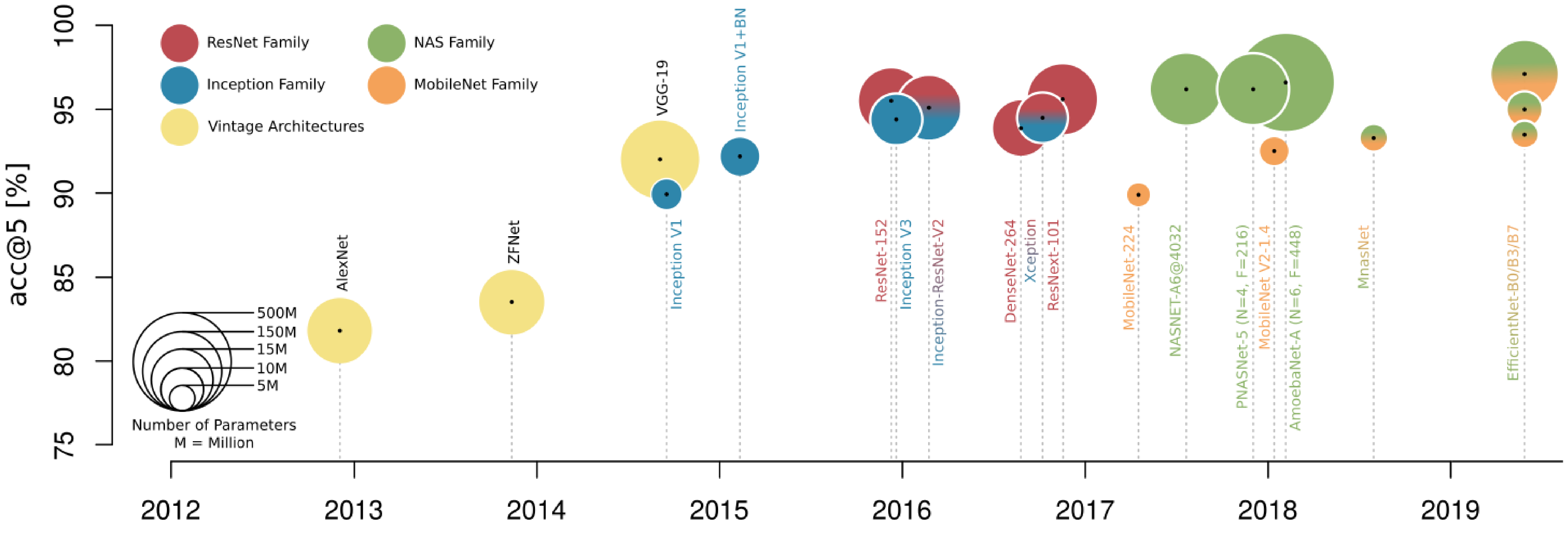
(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling (e) compound scaling



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Architectures overview

- Date of publication, main type



Hoeser & Kuenzer, 2020

Architectures overview

- Main charasteristics

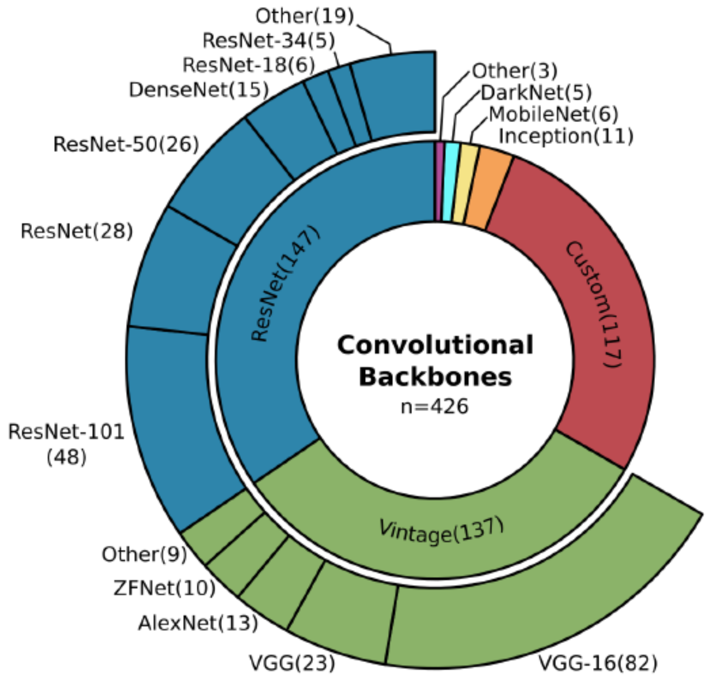
Architecture	Year	Bottleneck	Factorisation	Residual	NAS	M Parameters	$acc@5$ [%]
AlexNet [3]	2012					62	81.8
ZFNet [38]	2013					62	83.5
VGG-19 [39]	2014					144	91.9
Inception-V1 + BN [41]	2015	✓				11	92.2
ResNet-152 [43]	2015	✓		✓		60	95.5
Inception-V3 [42]	2015	✓	✓			24	94.4
DenseNet-264 [46]	2016	✓		✓		34	93.9
Xception [45]	2016	✓	✓	✓		23	94.5
ResNeXt-101 [44]	2016	✓		✓		84	95.6
MobileNet-224 [50]	2017	✓	✓	✓		4.2	89.9
NasNet [49]	2017	✓	✓	✓	✓	89	96.2
MobileNet V2 [108]	2018	✓	✓	✓		6.1	92.5
MnasNet [51]	2018	✓	✓	✓	✓	5.2	93.3
EfficientNet-B7 [52]	2019	✓	✓	✓	✓	66	97.1

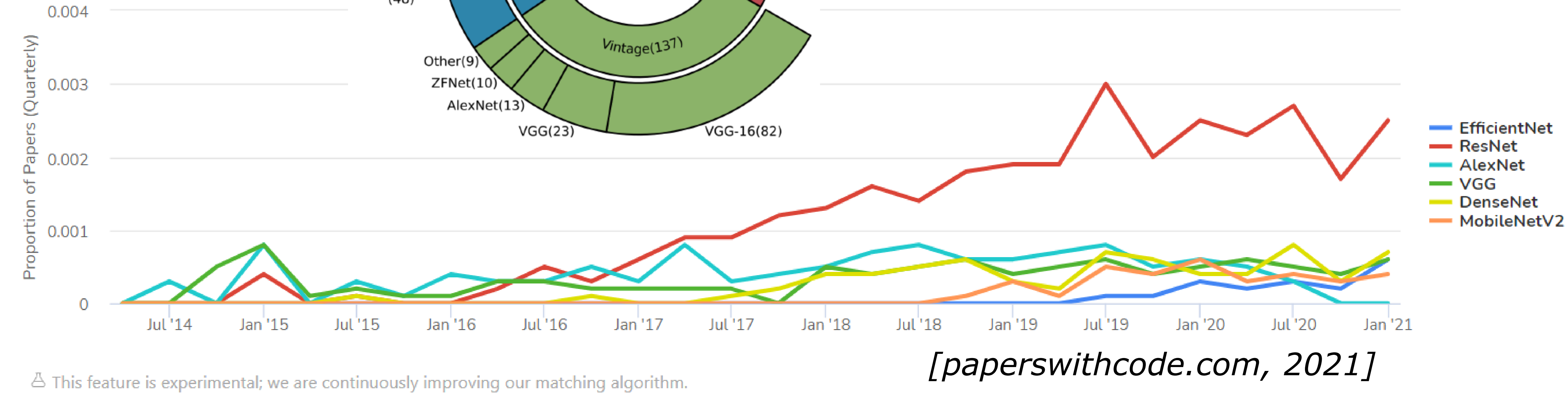
Hoeser & Kuenzer, 2020

Architectures overview

(remote sensing domain)

Hoeser et. al 2020

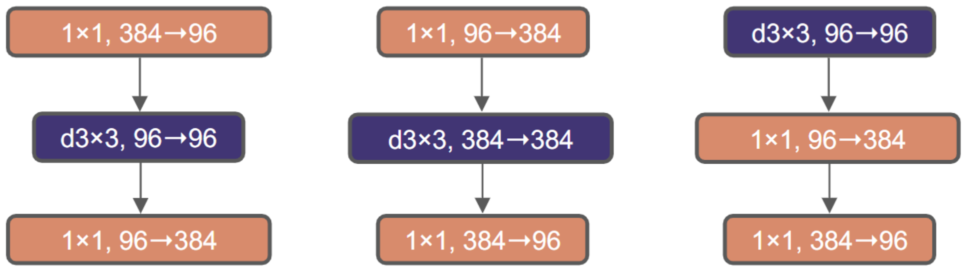


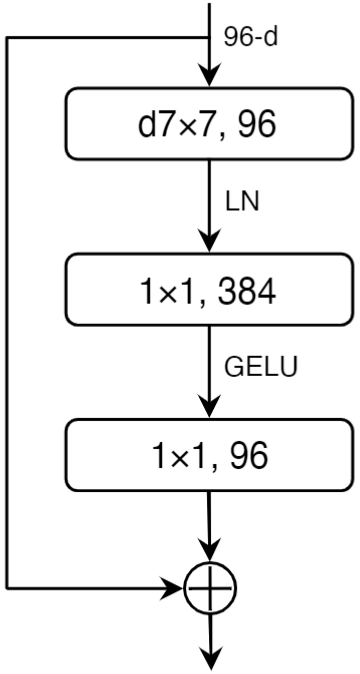


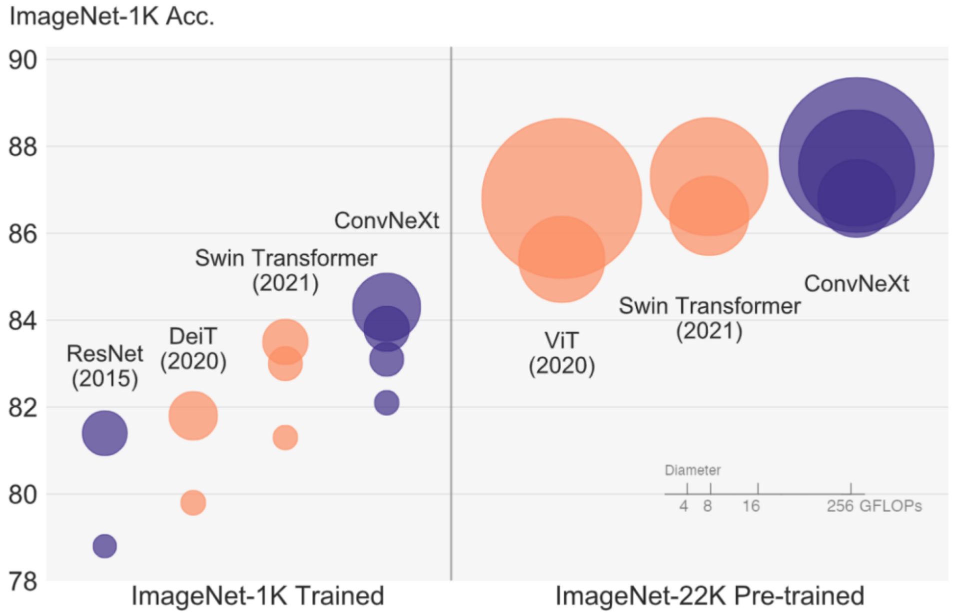
[paperswithcode.com, 2021]

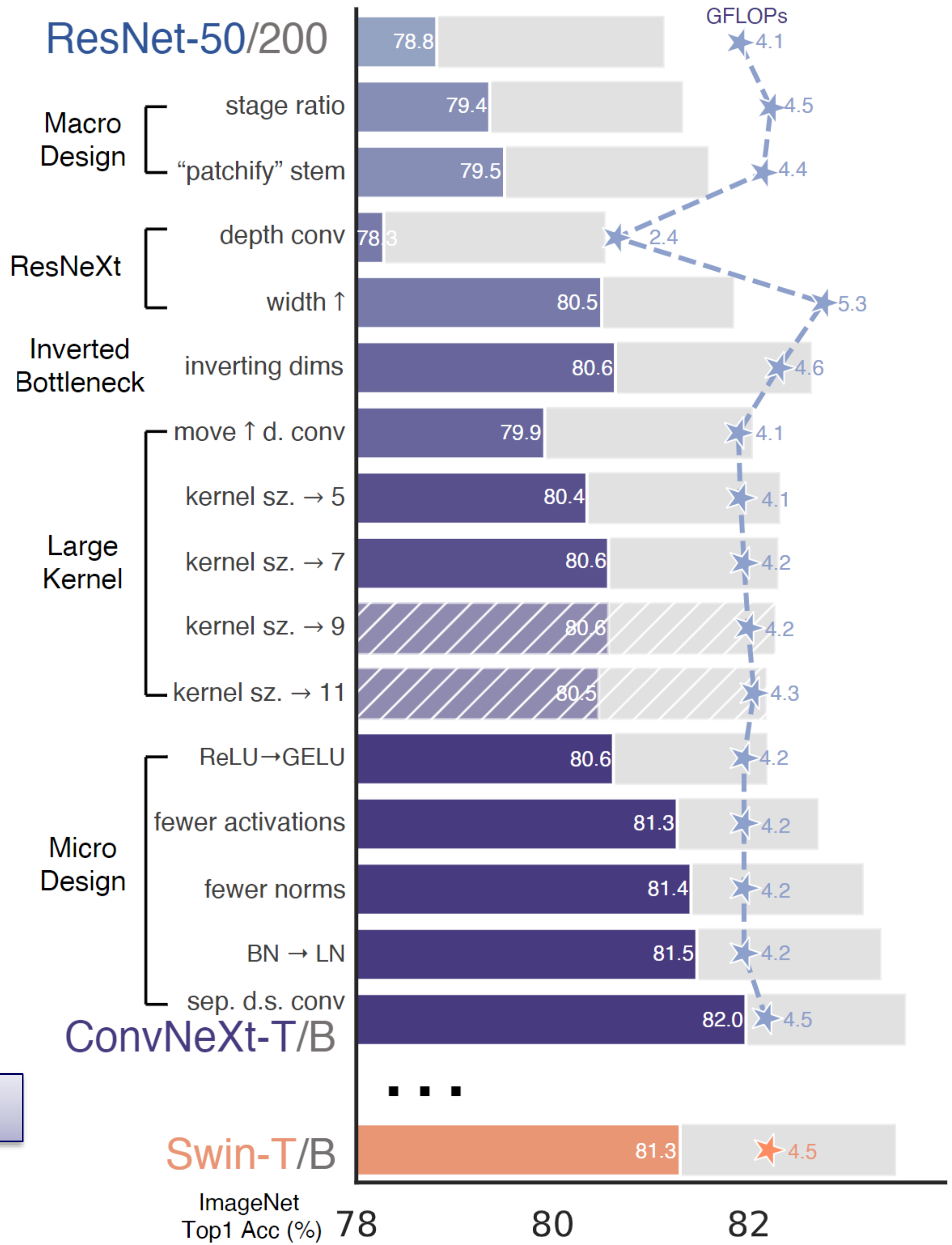
ConvNext

- A ConvNet for the 2020s
- Transformer-inspired modifications of ResNet









[Liu et al. 2022]

Architectures overview

- paperswithcode.com *[paperswithcode.com, 2022]*
  - Top 20 methods in Convolutional Neural Networks

Method	Year	Papers
ResNet Deep Residual Learning for Image Recognition	2015	1461
VGG Very Deep Convolutional Networks for Large-Scale Image Recognition	2014	369
DenseNet Densely Connected Convolutional Networks	2016	300
AlexNet ImageNet Classification with Deep Convolutional Neural Networks	2012	280
VGG-16 Very Deep Convolutional Networks for Large-Scale Image Recognition	2014	258
MobileNetV2 MobileNetV2: Inverted Residuals and Linear Bottlenecks	2018	201
EfficientNet EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks	2019	154
Darknet-53 YOLOv3: An Incremental Improvement	2018	142
ResNeXt Aggregated Residual Transformations for Deep Neural Networks	2016	120
GoogLeNet Going Deeper with Convolutions	2014	119
Xception Xception: Deep Learning With Depthwise Separable Convolutions	2017	94
SqueezeNet SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size	2016	71
Inception-v3 Rethinking the Inception Architecture for Computer Vision	2015	67
CSPDarknet53 YOLOv4: Optimal Speed and Accuracy of Object Detection	2020	46
MobileNetV1 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications	2017	44
LeNet	1998	44
Darknet-19 YOLO9000: Better, Faster, Stronger	2016	44
WideResNet Wide Residual Networks	2016	42
ShuffleNet ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices	2017	36
MobileNetV3 Searching for MobileNetV3	2019	34

Architectures overview

[paperswithcode.com, 2021]

Image Models

ResNet
1092 papers with code

AlexNet
287 papers with code

VGG
274 papers with code

DenseNet
235 papers with code

MobileNetV2
137 papers with code

▸ See all 102 methods

Image Model Blocks

Residual Block
1396 papers with code

Bottleneck Residual Block
1096 papers with code

Dense Block
258 papers with code

Squeeze-and-Excitation Block
147 papers with code

Inception Module
136 papers with code

▸ See all 79 methods

Convolutions

Convolution
8346 papers with code

1x1 Convolution
2562 papers with code

Grouped Convolution
451 papers with code

Pointwise Convolution
414 papers with code

Depthwise Convolution
406 papers with code

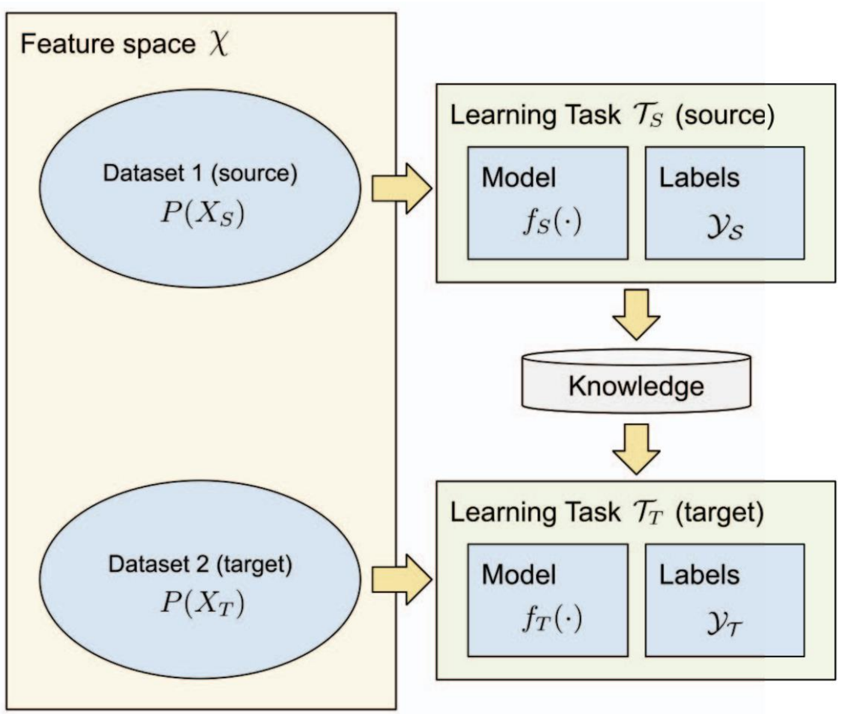
▸ See all 35 methods

Pretrained models

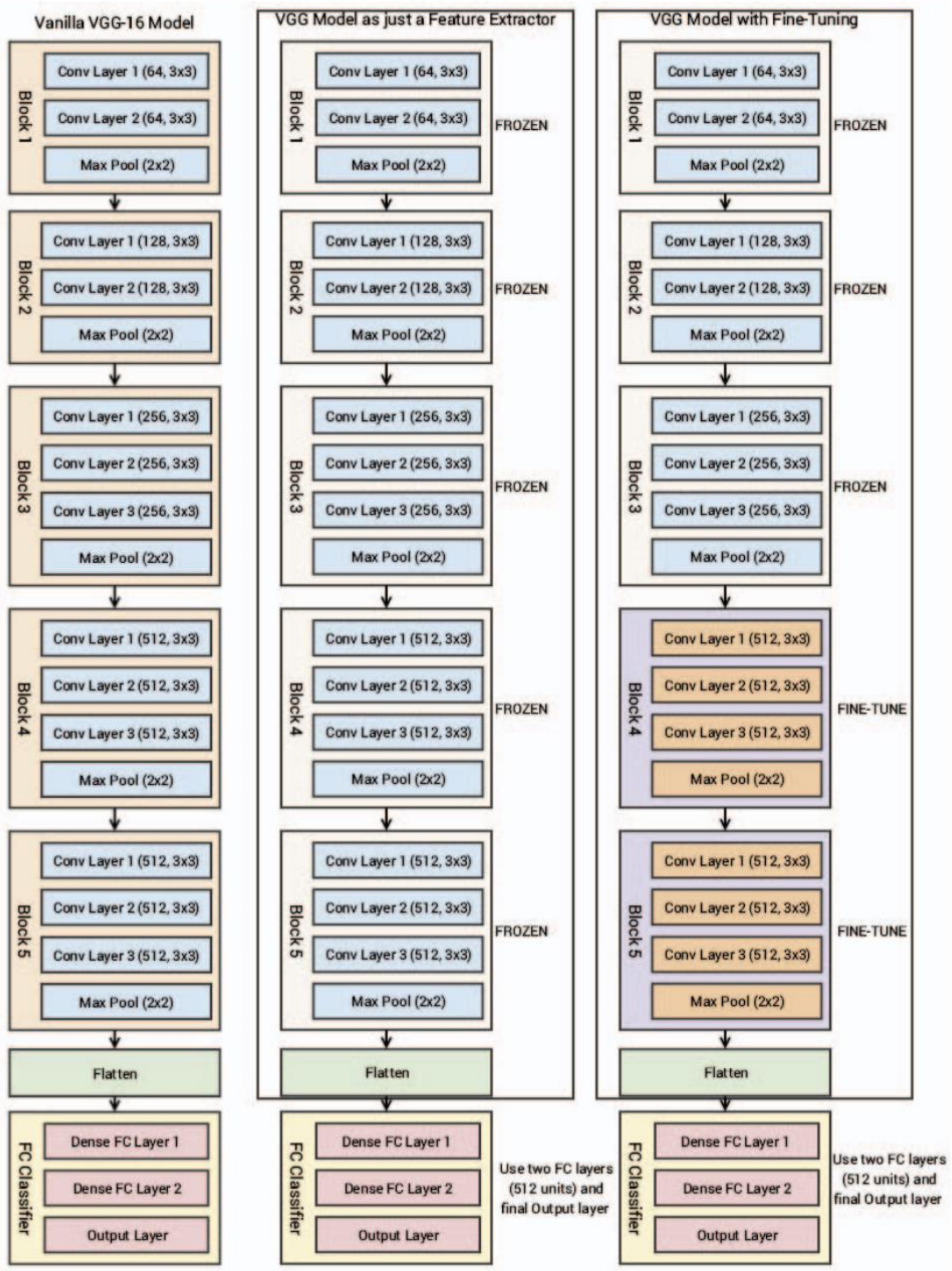
```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
mobilenet_v3_large = models.mobilenet_v3_large(pretrained=True)
mobilenet_v3_small = models.mobilenet_v3_small(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
```

Transfer learning

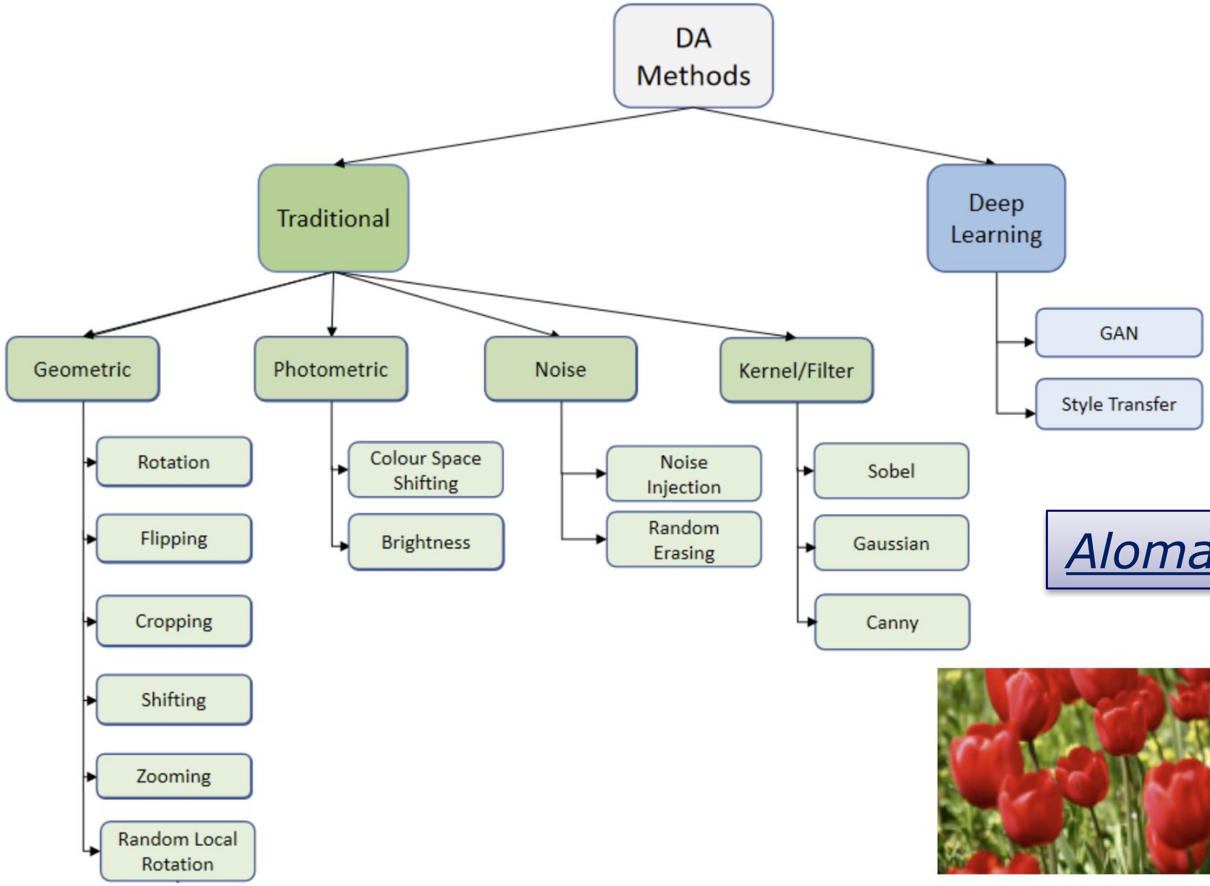
- Train on a large related dataset
- Fine-tune on the target dataset
- Heavily used



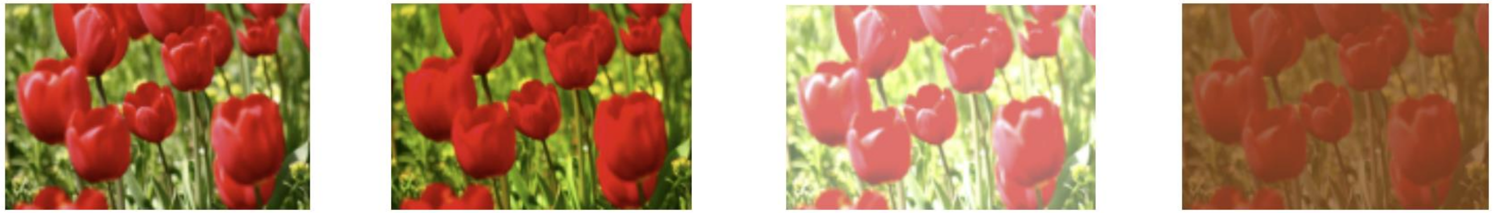
[Ribani & Marengoni 2019]



Data augmentation

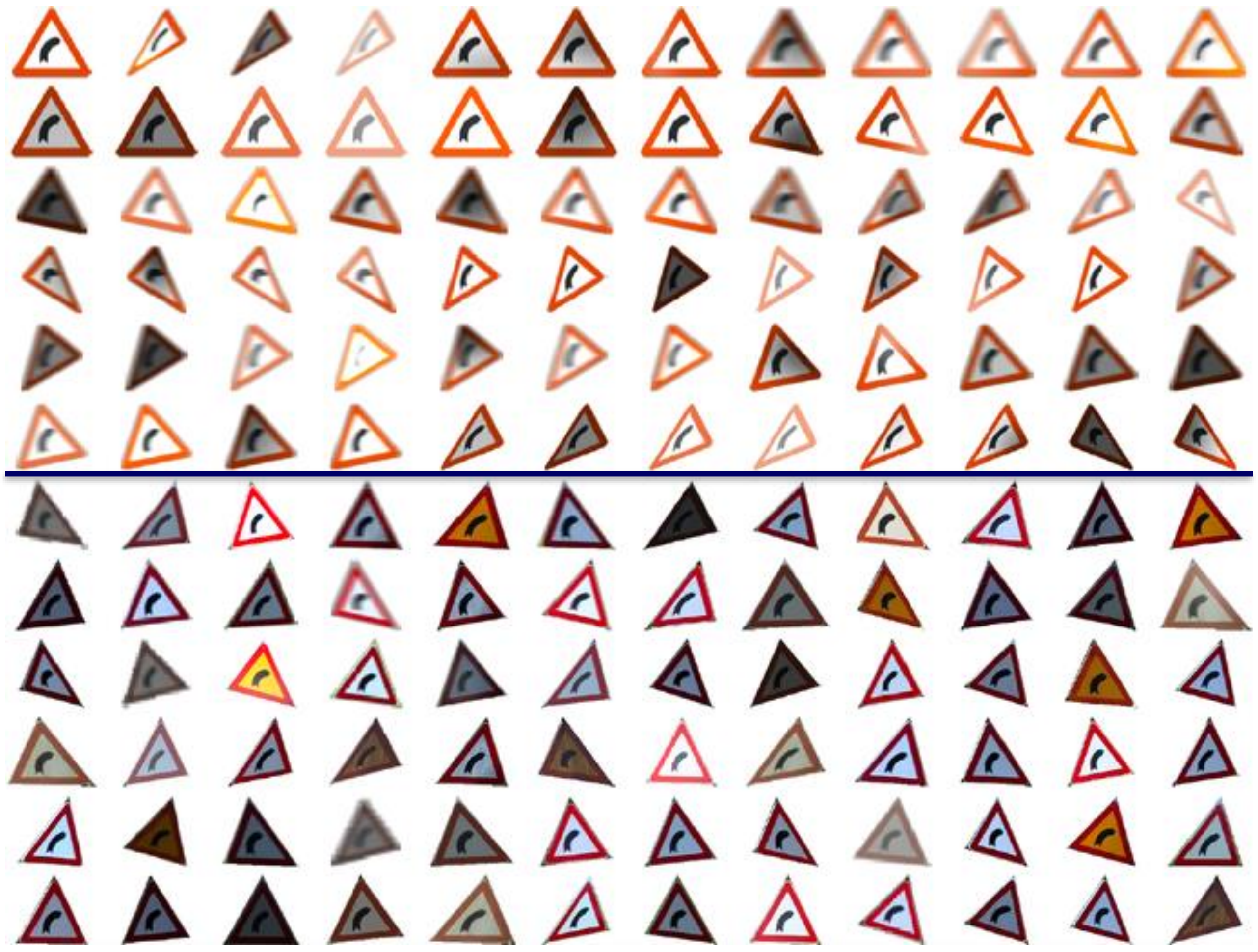


[Alomar et al., 2023]



Data augmentation

- Any meaningful transformations
- Random mix/combinations of :
  - translation
  - rotation
  - stretching
  - shearing,
  - lens distortions, …
- Distribution of augmented images (features, parameters) should correspond to the distribution of original training images!
- The augmented images could be generated in advance or on the fly during training
- Simple to implement, use it!
- Especially useful for small datasets
- Kind of regularisation

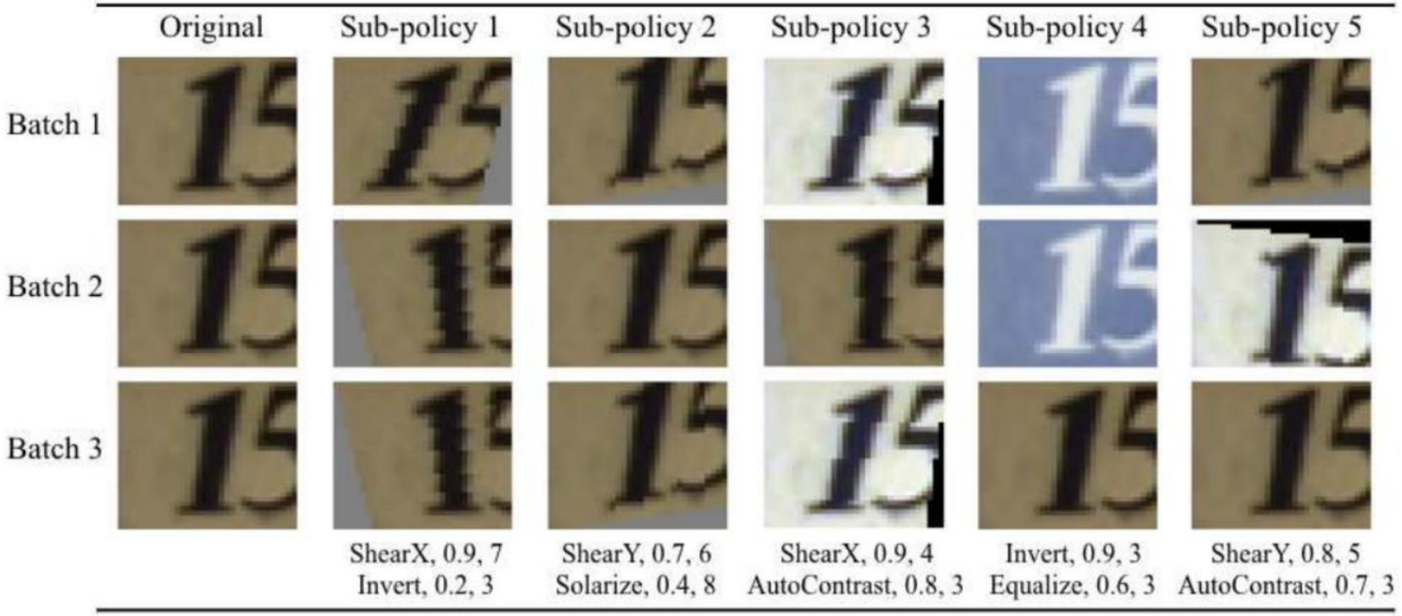


Tabernik & Skočaj, 2020

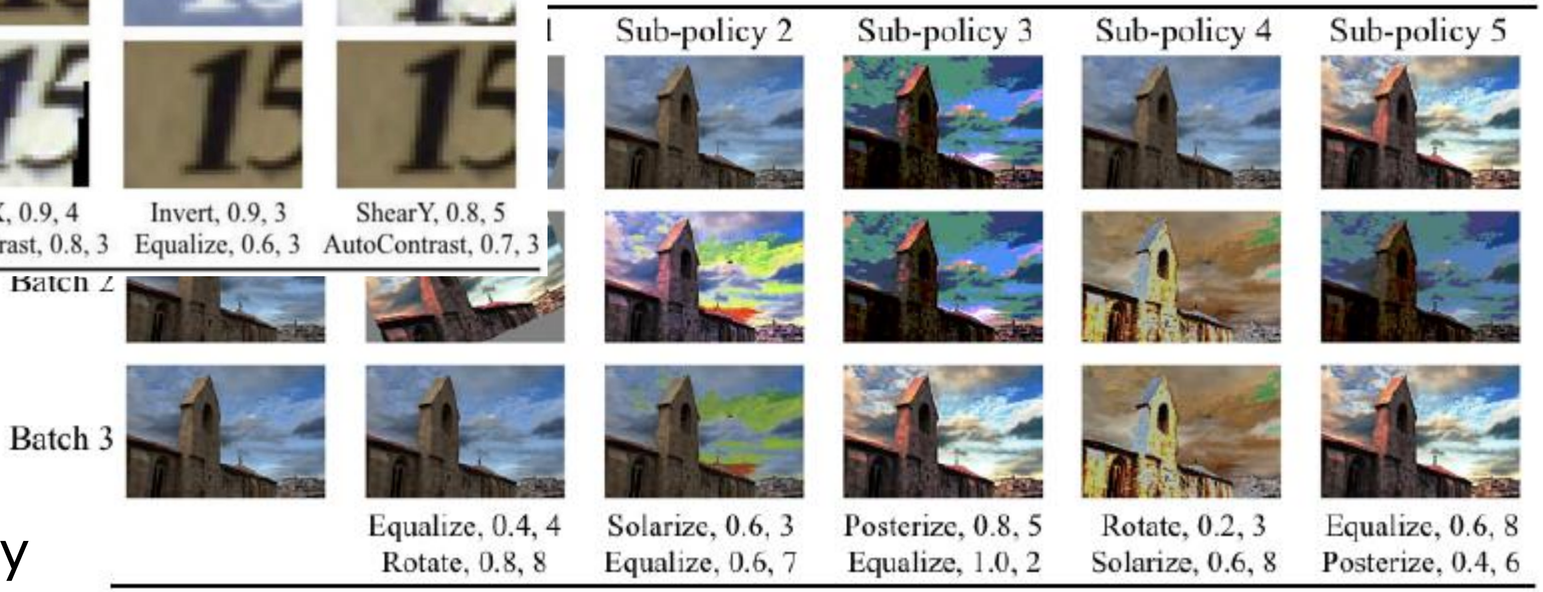
Automatic Data Augmentation - AutoAugment

[Cubuk et. al 2018]

- Trained like Neural architecture search
  - Search for optimal augmentation parameters (operations; probability, magnitude)
- Proximal Policy Optimization Algorithms



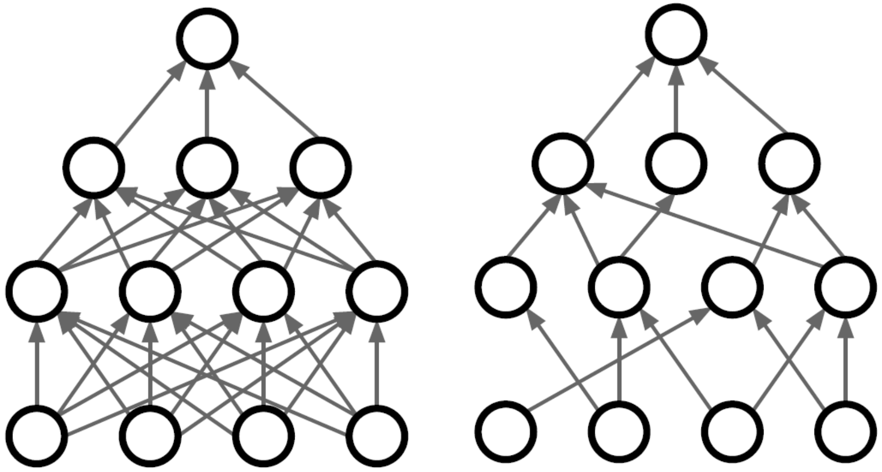
Dataset	GPU hours	Best published results	Our results
CIFAR-10	5000	2.1	1.5
CIFAR-100	0	12.2	10.7
SVHN	1000	1.3	1.0
Stanford Cars	0	5.9	5.2
ImageNet	15000	3.9	3.5



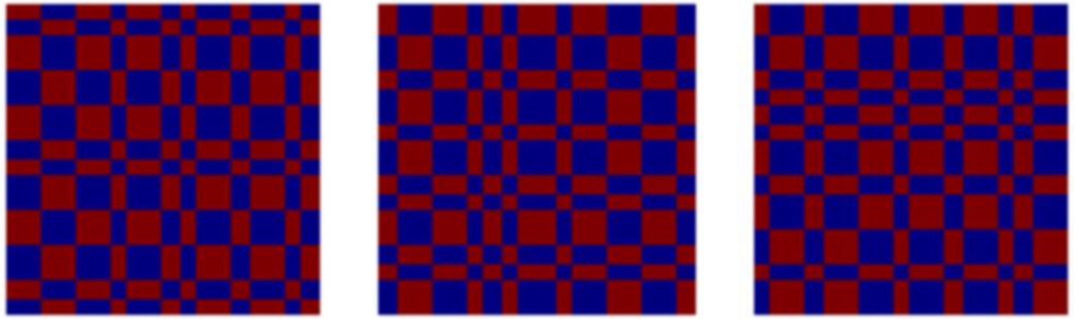
- Learning Augmentation Strategies from Data
- Transfer learning -> transfer augmentation policy

Regularisation

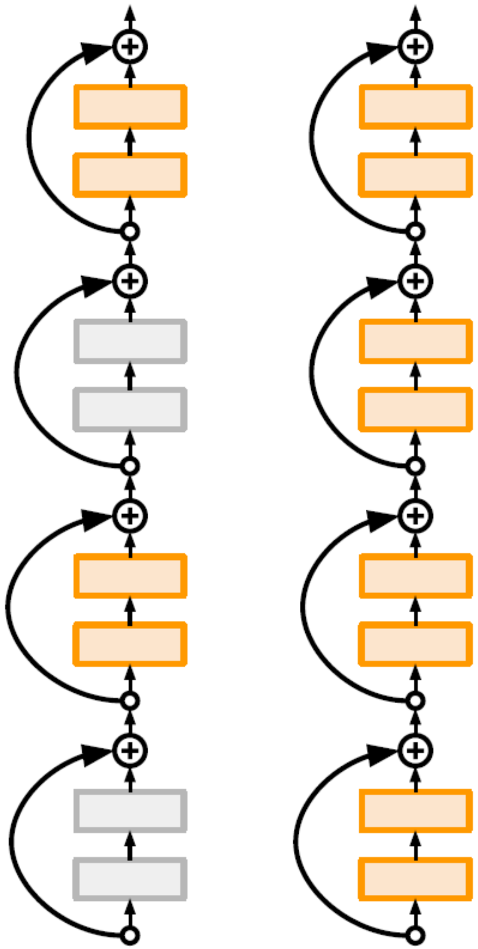
- Data Augmentation
- L2 regularisation
- Dropout
- Batch Normalization
- DropConnect
- Fractional Max Pooling
- Stochastic Depth
- Cutout / Random Crop
- Mixup



Wan et. al 2013



Graham et. al 2014

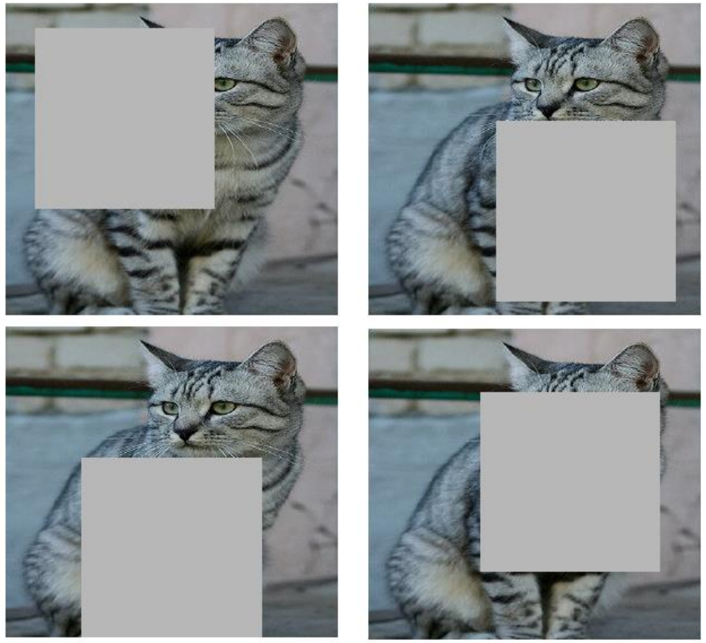


Huang et. al 2016

Regularisation: Cutout

- Radomly mask image regions

DeVries & Taylor, 2017

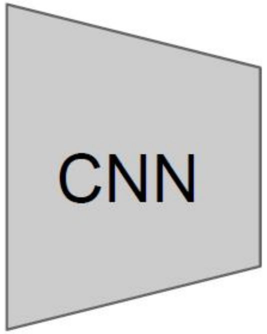


Method	C10	C10+	C100	C100+	SVHN
ResNet18 [5]	10.63 ± 0.26	4.72 ± 0.21	36.68 ± 0.57	22.46 ± 0.31	-
ResNet18 + cutout	9.31 ± 0.18	3.99 ± 0.13	34.98 ± 0.29	21.96 ± 0.24	-
WideResNet [22]	6.97 ± 0.22	3.87 ± 0.08	26.06 ± 0.22	18.8 ± 0.08	1.60 ± 0.05
WideResNet + cutout	$\mathbf{5.54 \pm 0.08}$	3.08 ± 0.16	$\mathbf{23.94 \pm 0.15}$	18.41 ± 0.27	$\mathbf{1.30 \pm 0.03}$
Shake-shake regularization [4]	-	2.86	-	15.85	-
Shake-shake regularization + cutout	-	$\mathbf{2.56 \pm 0.07}$	-	$\mathbf{15.20 \pm 0.21}$	-

Regularisation: Mixup

- Blend two images and labels

Zhang et. al 2018



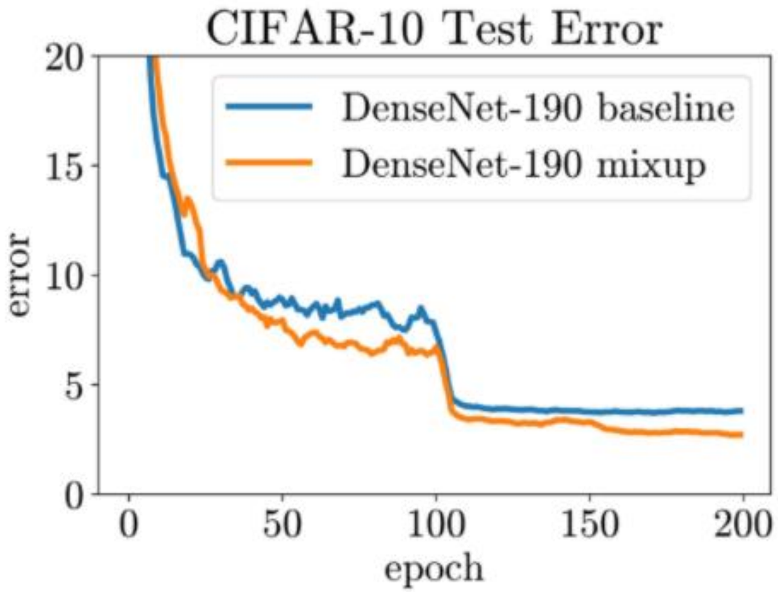
Target label:
cat: 0.4
dog: 0.6

Randomly blend the pixels of pairs of training images, e.g. 40% cat, 60% dog

$$\hat{x} = \lambda x_i + (1 - \lambda) x_j$$

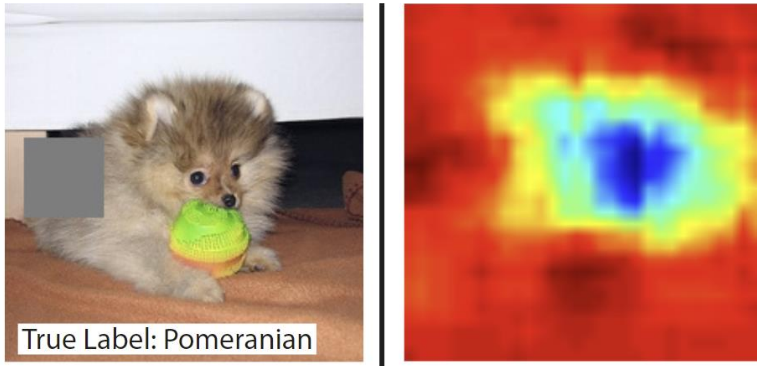
$$\hat{y} = \lambda y_i + (1 - \lambda) y_j$$

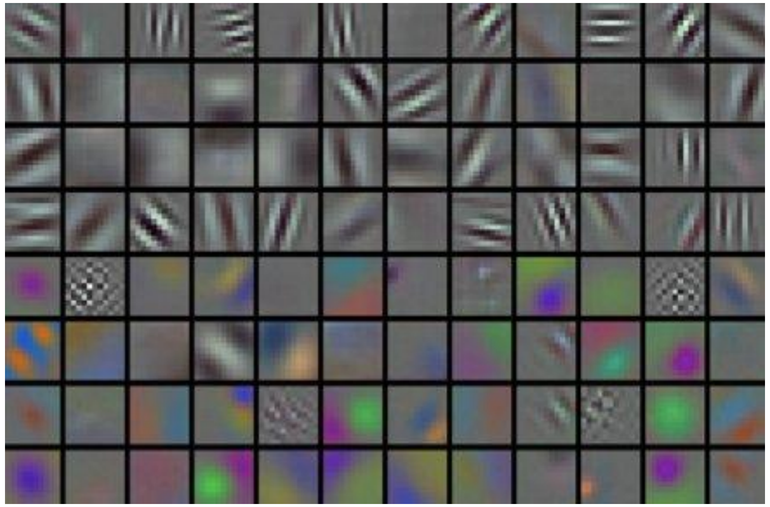
$$\lambda \sim \text{Beta}(\alpha = 0.2)$$

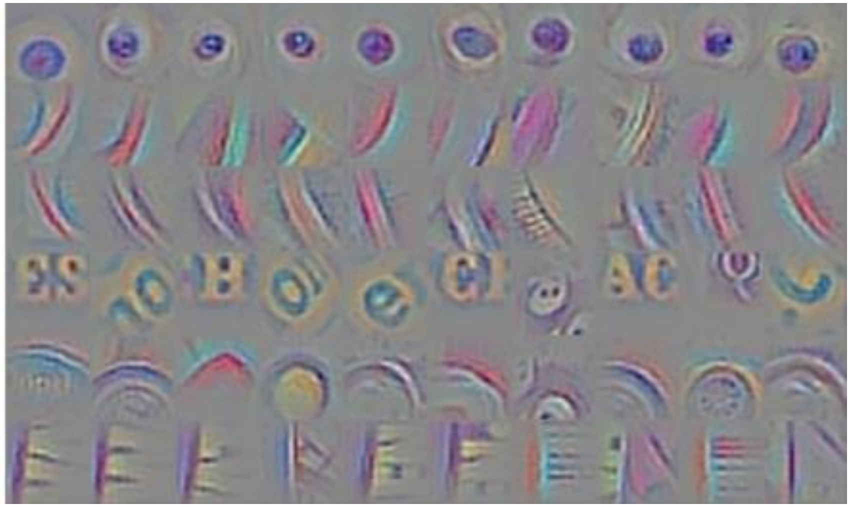


Explainability of CNNs

- Visualising filters and semantically similar images
- Dimensionality reduction
- Maximally activating patches
- Visualising activations
- Deconvolution
- Guided backpropagation
- Occlusion sensitivity, LIME
- Class activation maps and Grad-CAM

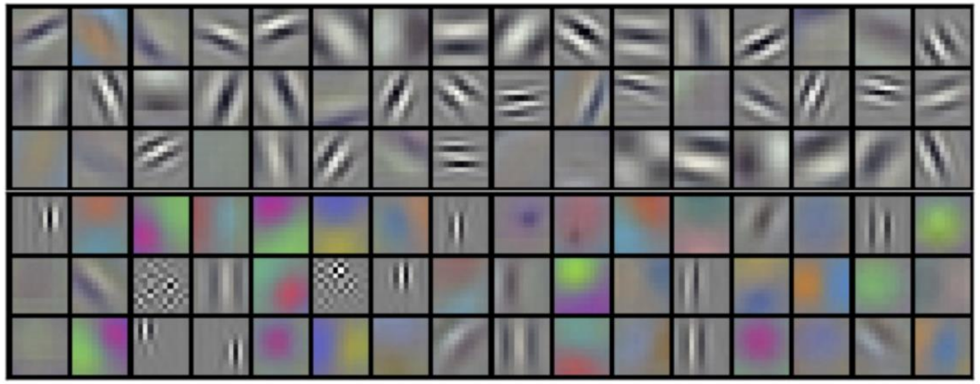






Visualising filters and images

- Visualising filters on the first layer



- Images with similar embedding (last layer features)

[Krizhevsky, 2012]

t-SNE visualization of CNN codes

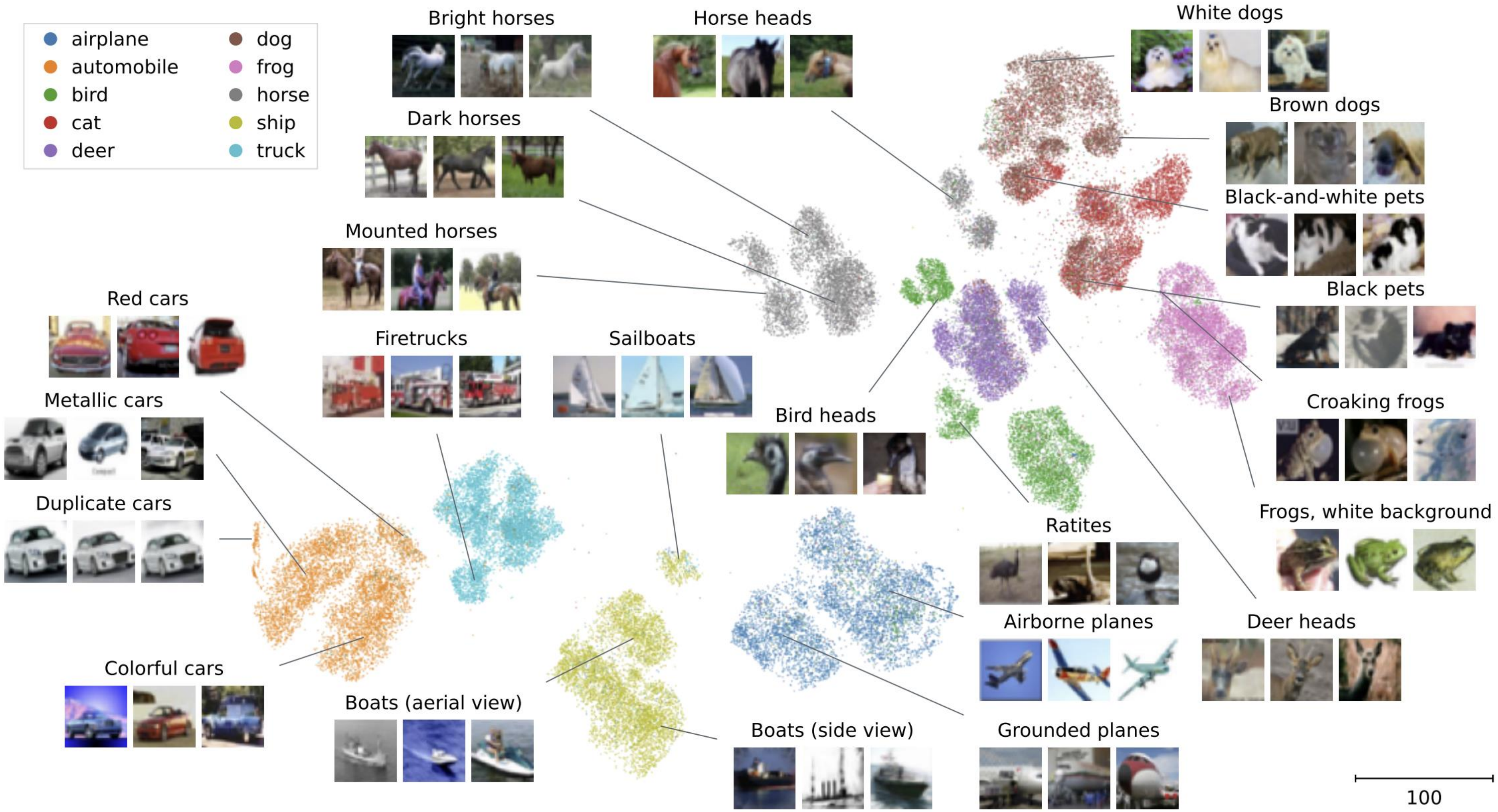
van der Maaten, 2013

Karpahty, 2014

t-SimCNE

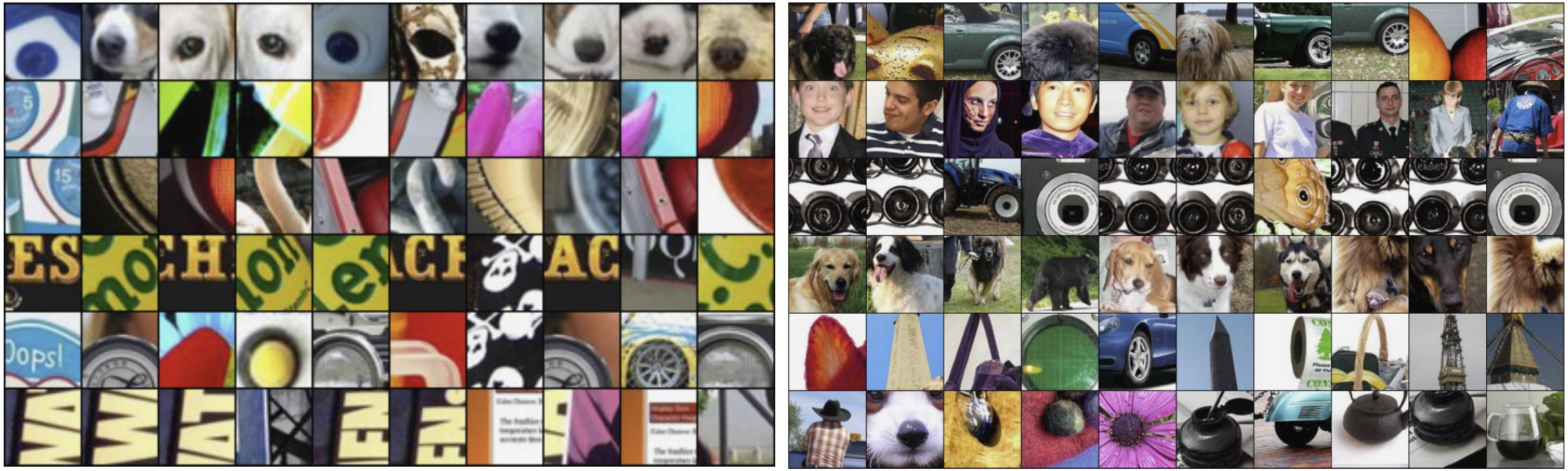
- Unsupervised visualisation using contrastive learning

[Bohm et al., 2023]



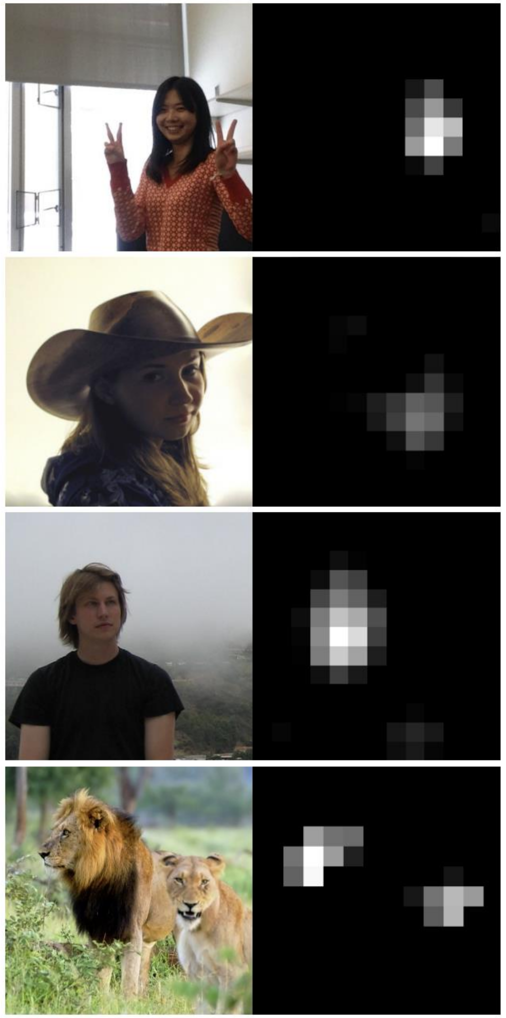
Maximally Activating Patches

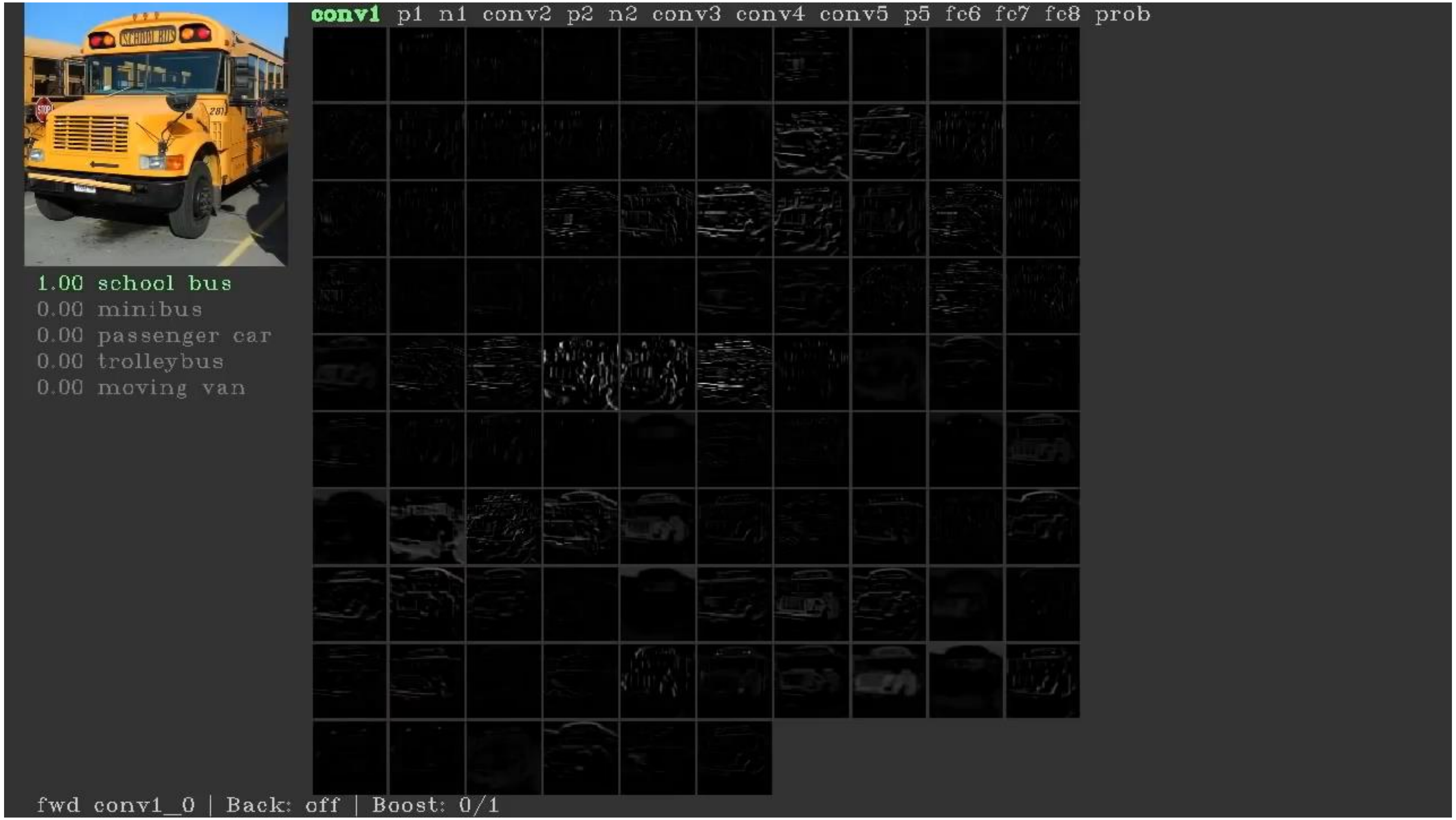
- Patches in the images that activate a paricular neuron at a particular layer most



Springenberg et al., 2015

Visualising activations

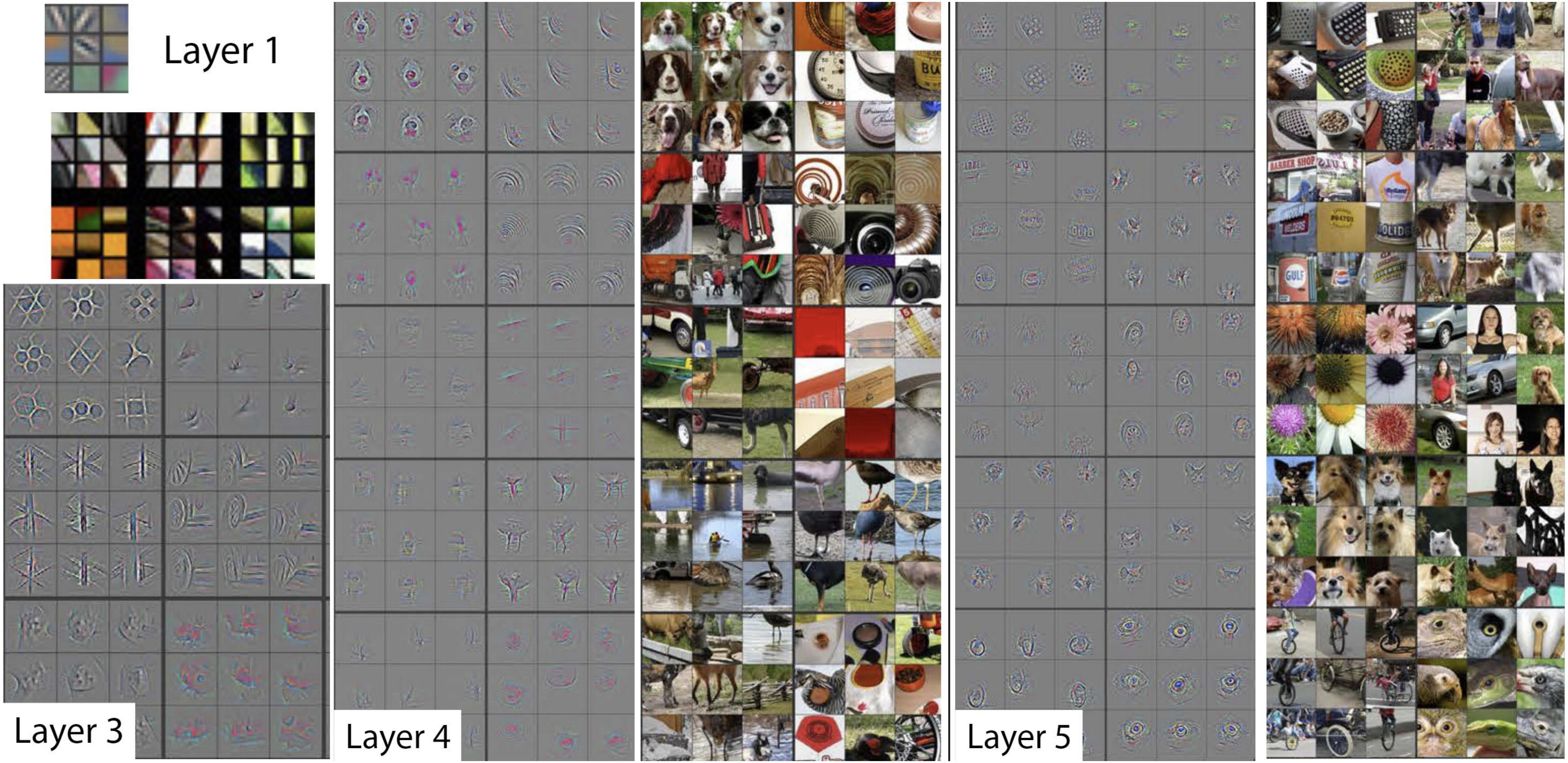




Yosinsky et al., 2015

Deconvolution

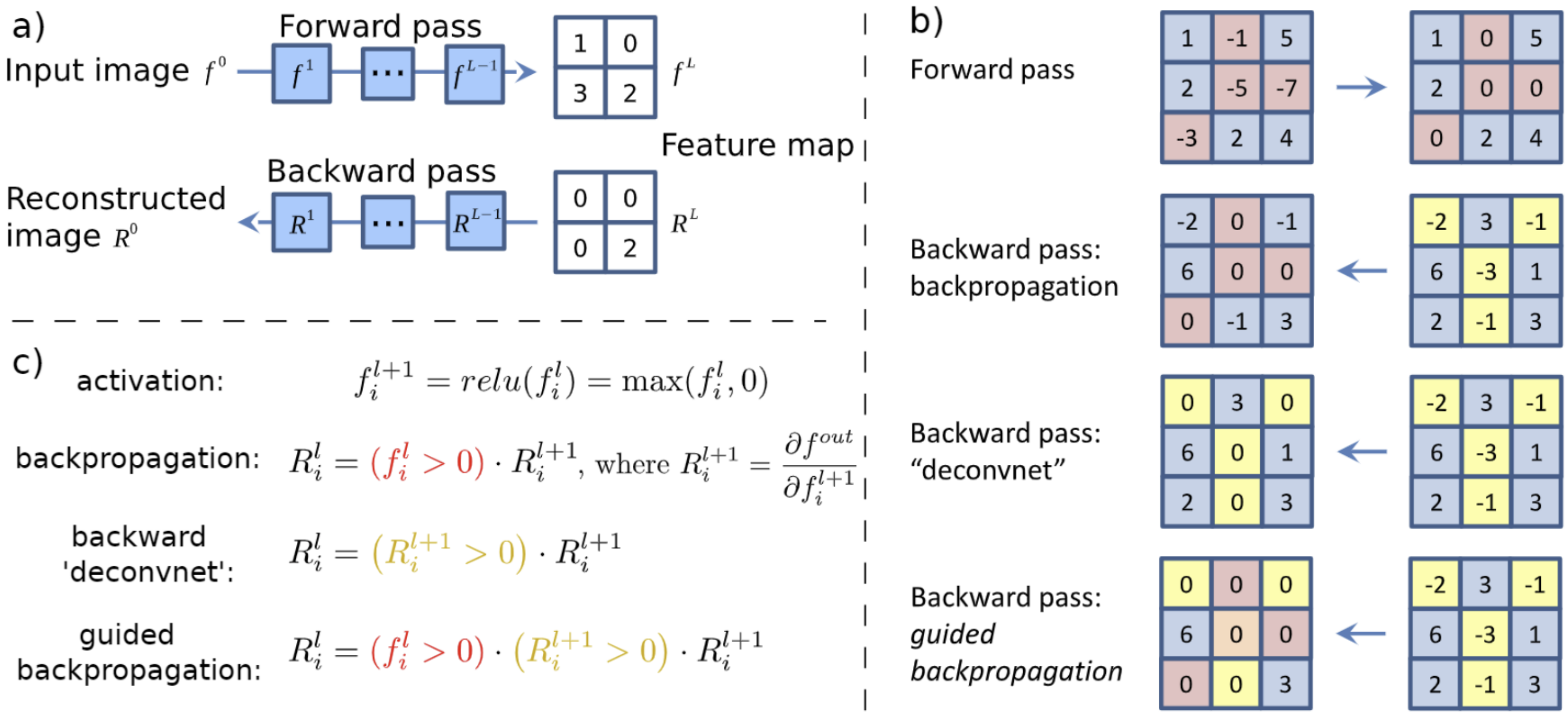
Zeiler & Fergus, 2013



Guided backpropagation

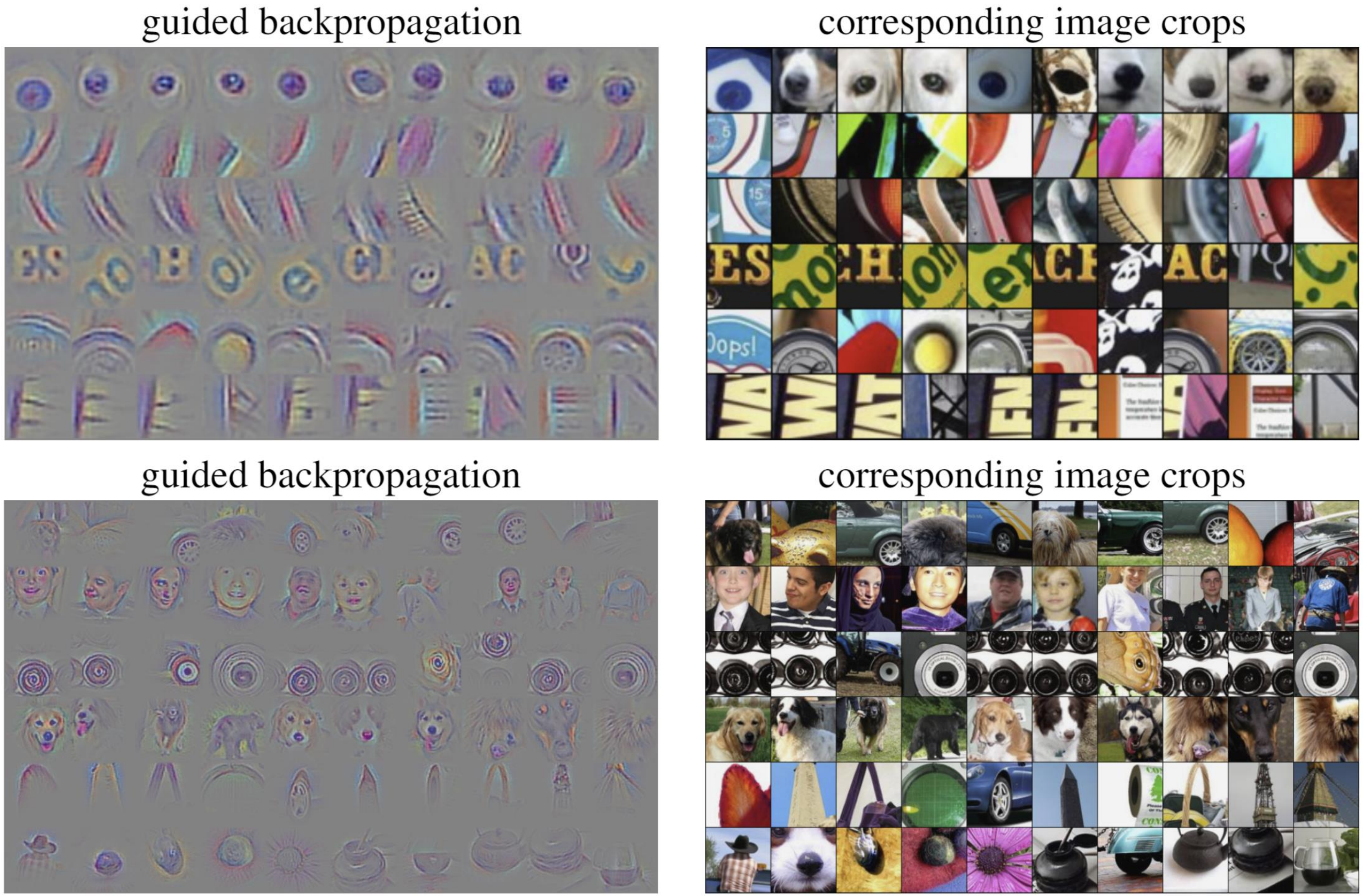
[Springenberg et al., 2015]

- Backpropagate to the image



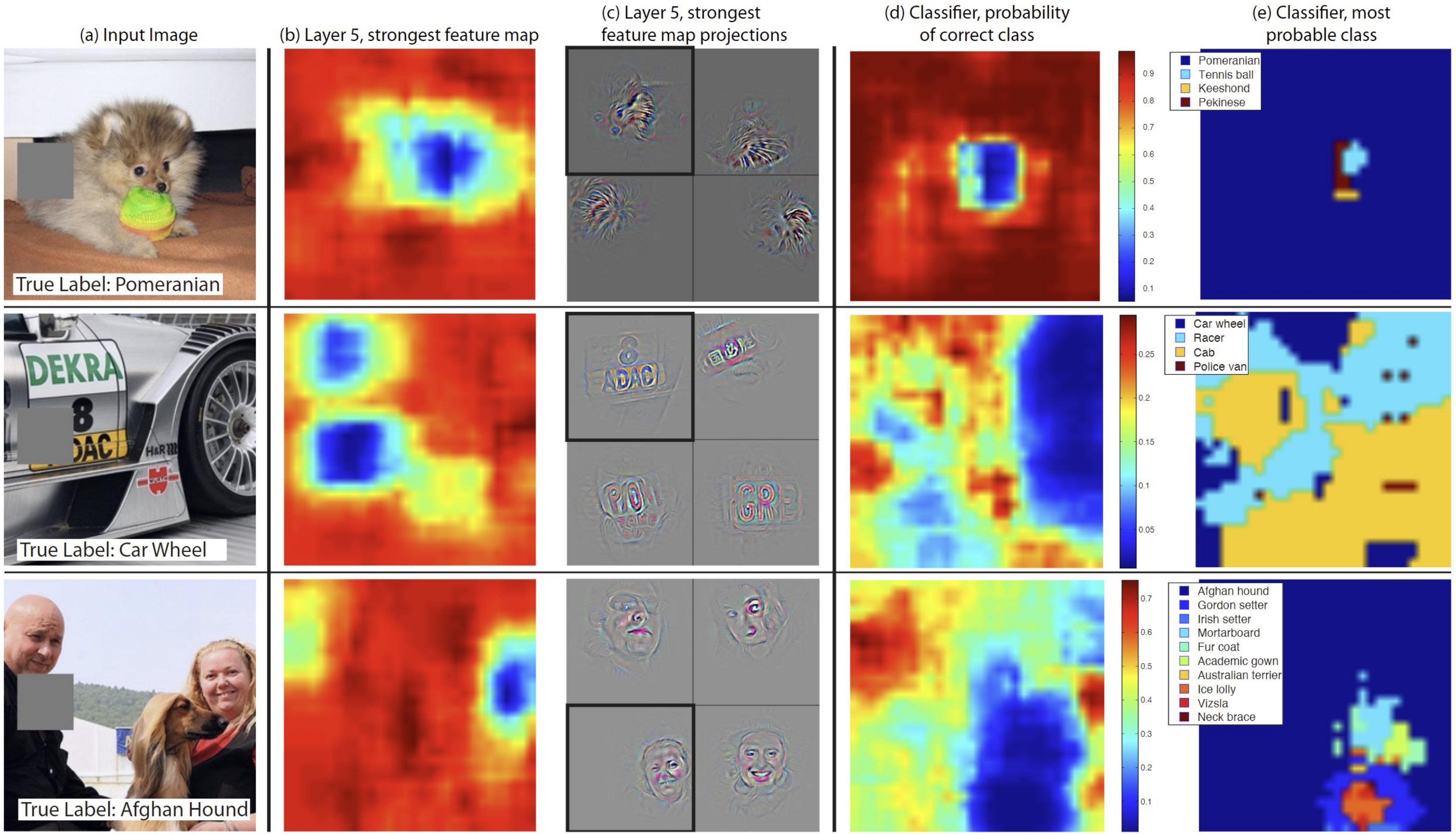
Guided backpropagation

Springenberg et al., 2015



Occlusion Sensitivity

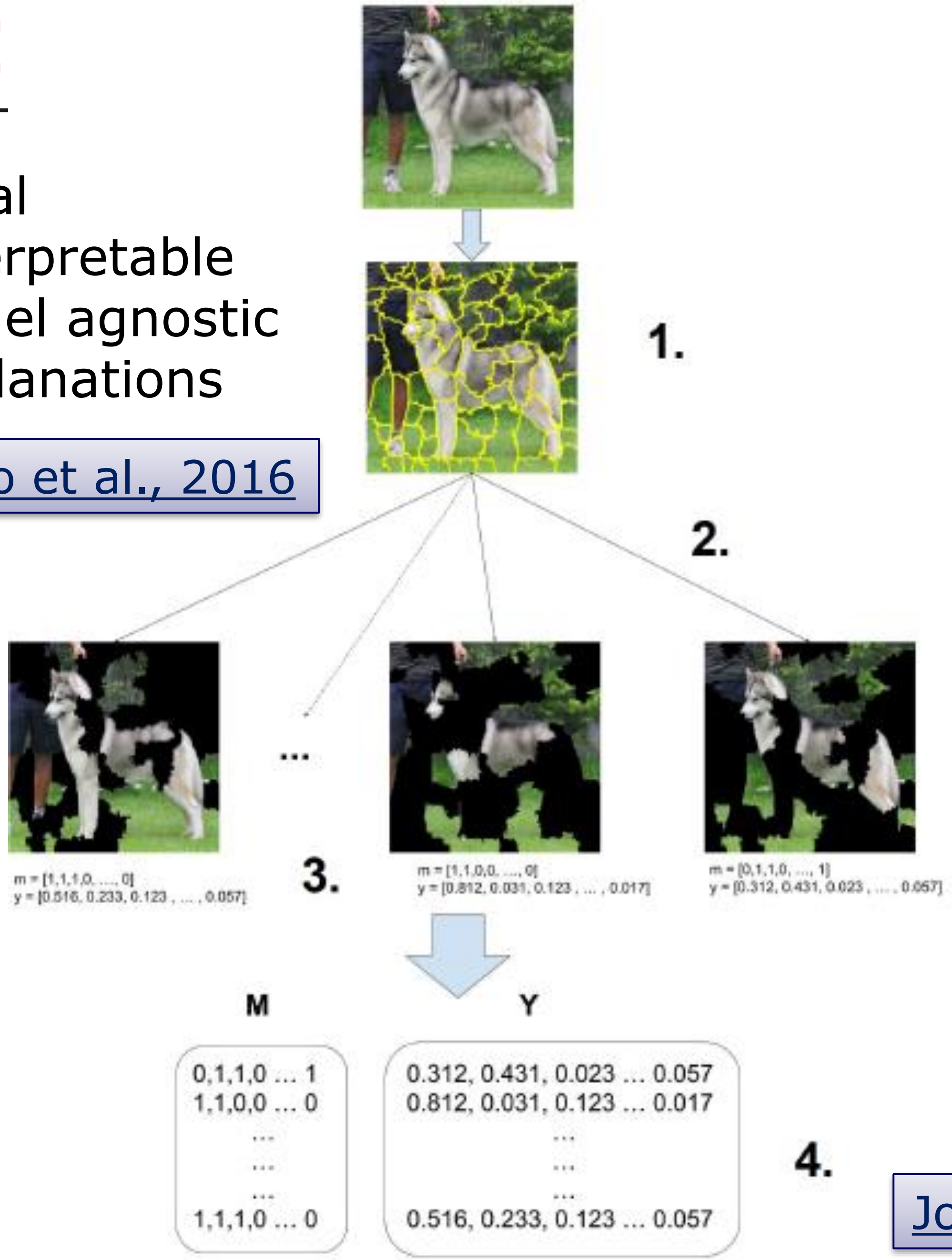
Zeiler & Fergus, 2013



LIME

- Local Interpretable Model agnostic Explanations

[Ribeiro et al., 2016]

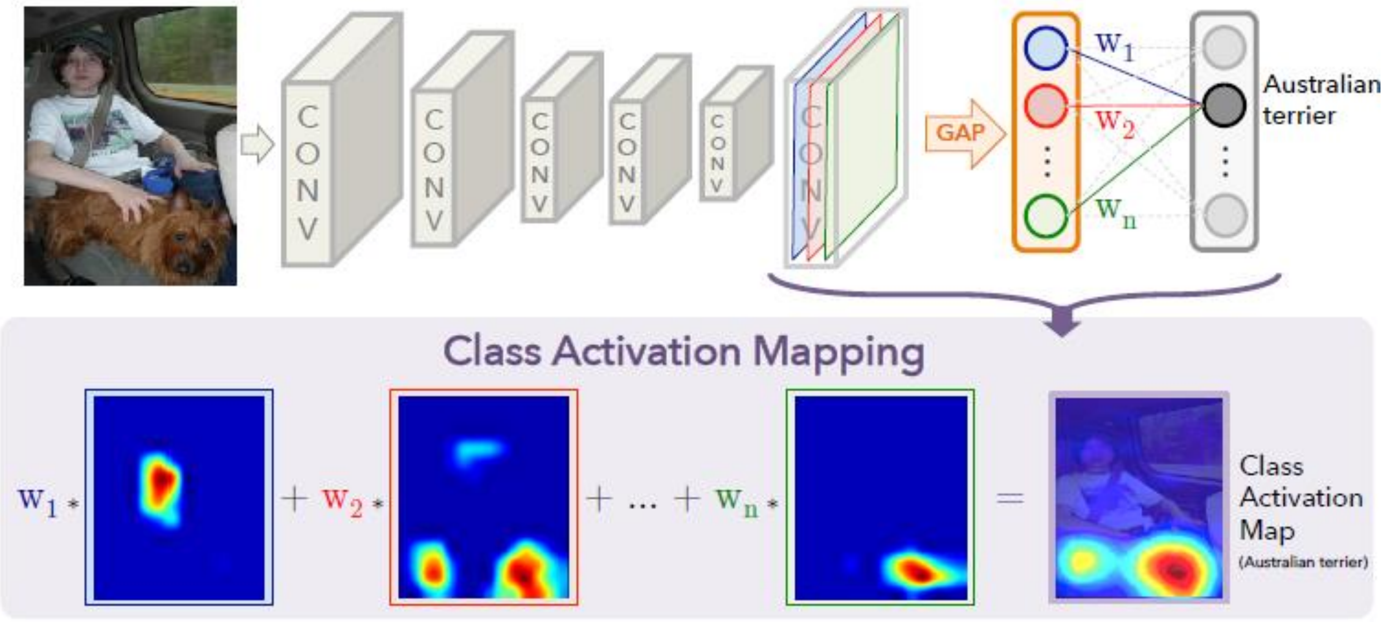


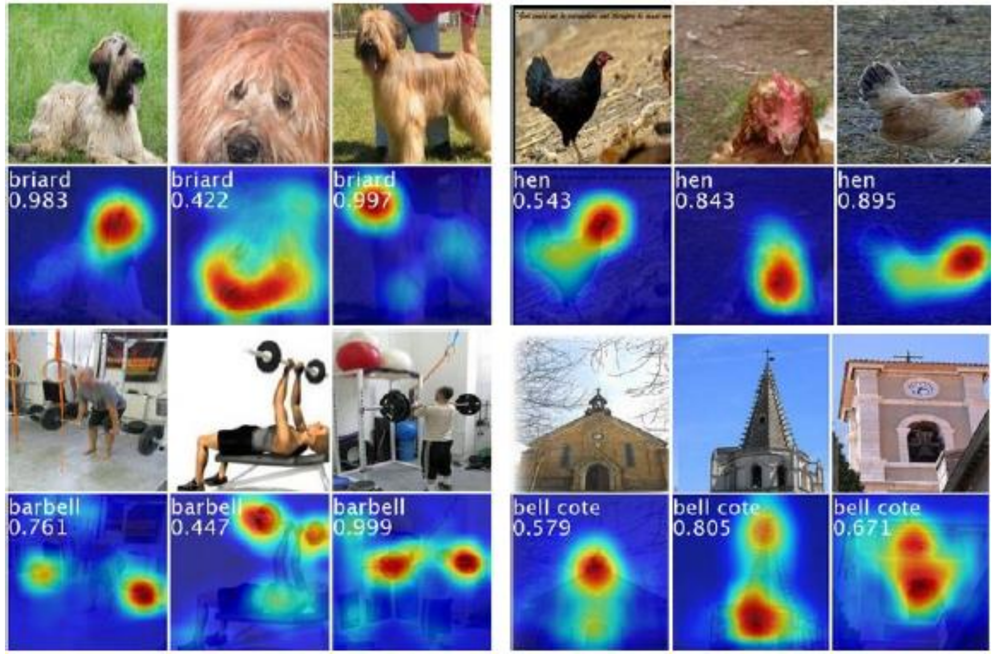

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

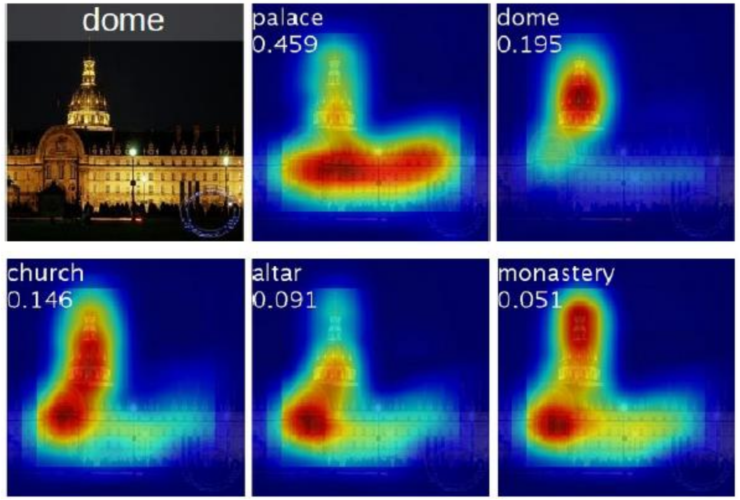
[Jonke, 2020]


(a) Husky classified as wolf (b) Explanation

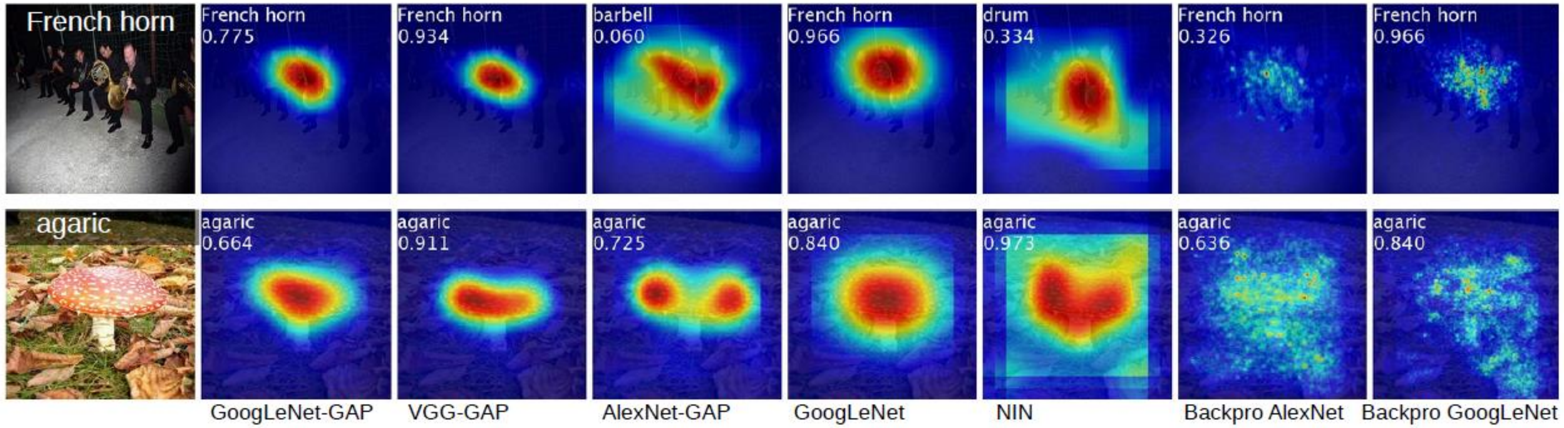
Class activation maps





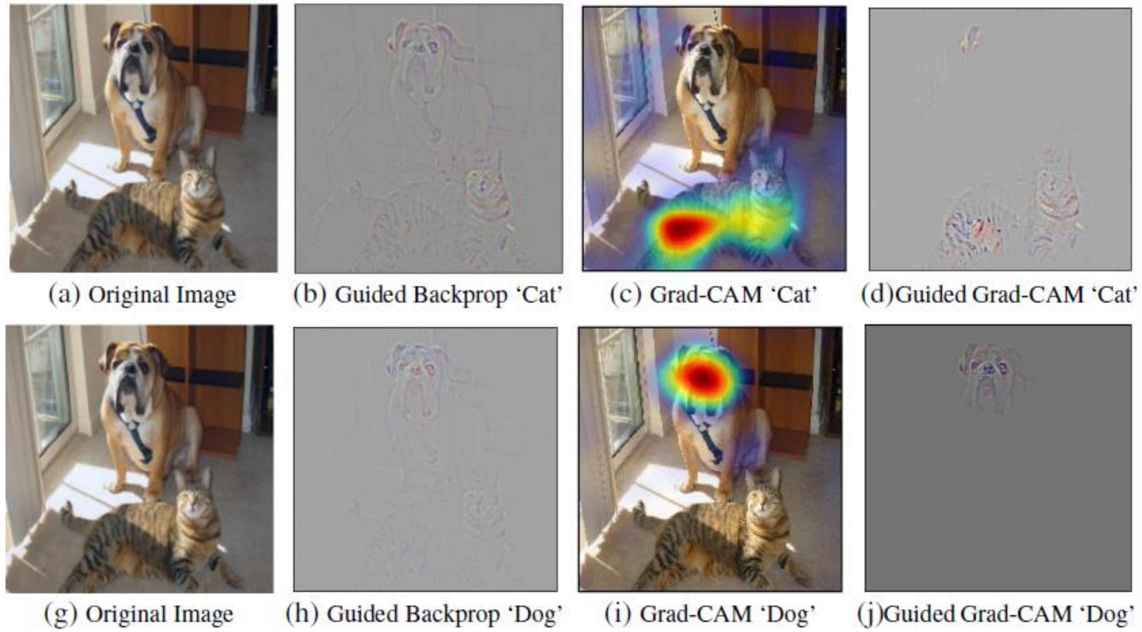


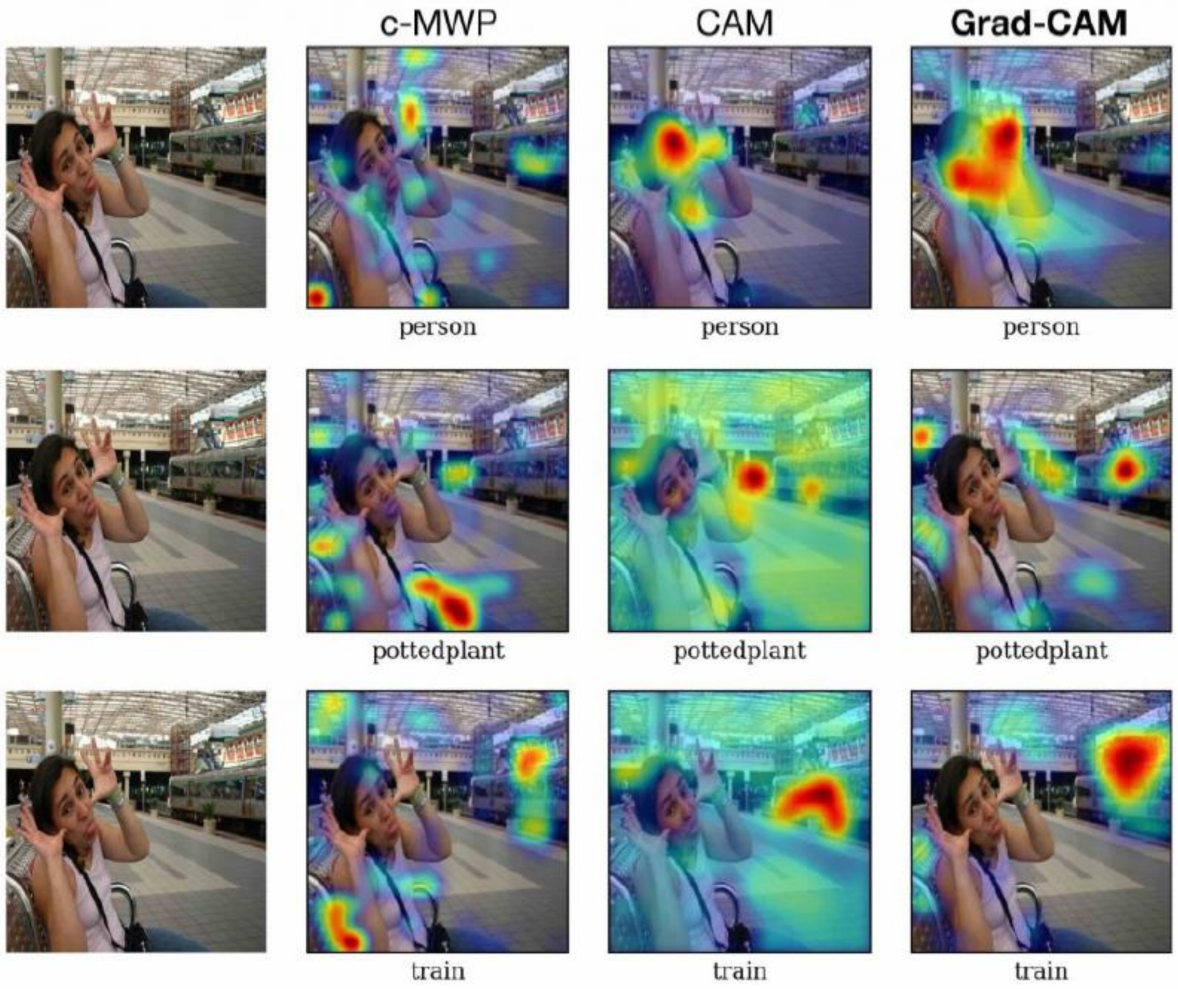
Zhoe et al., 2016

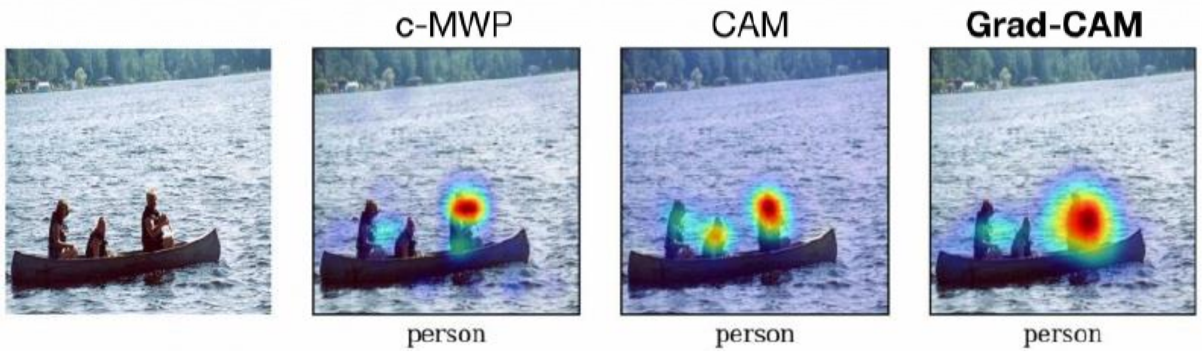


Grad-CAM

- Visual Explanations from Deep Networks via Gradient-based Localization



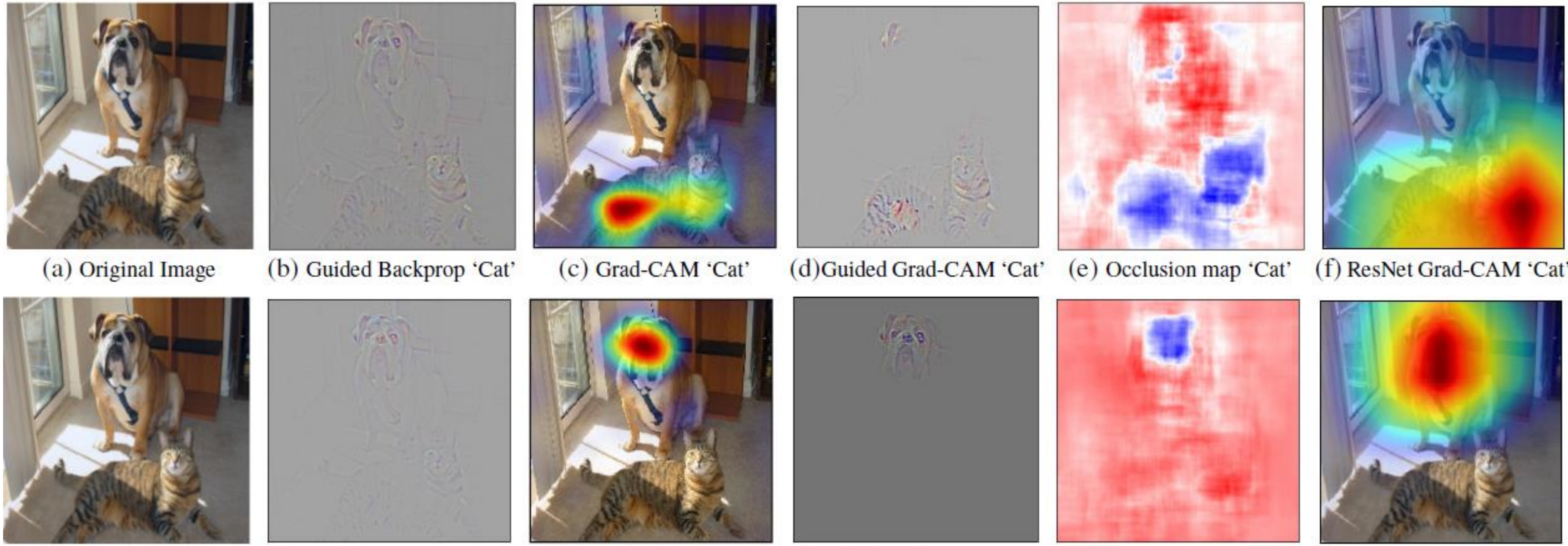




Selvaraju et al., 2017

Grad-CAM

- Visual Explanations from Deep Networks via Gradient-based Localization

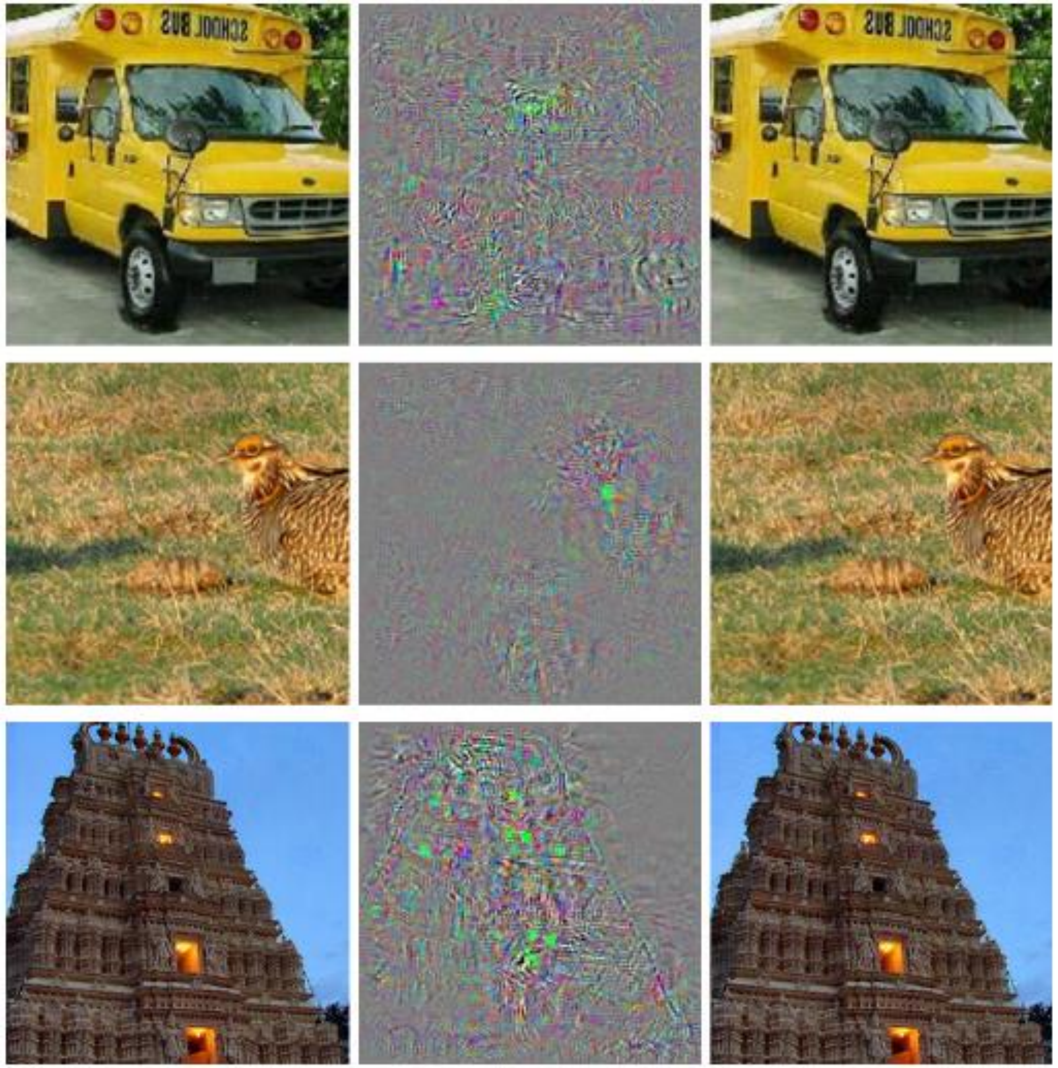


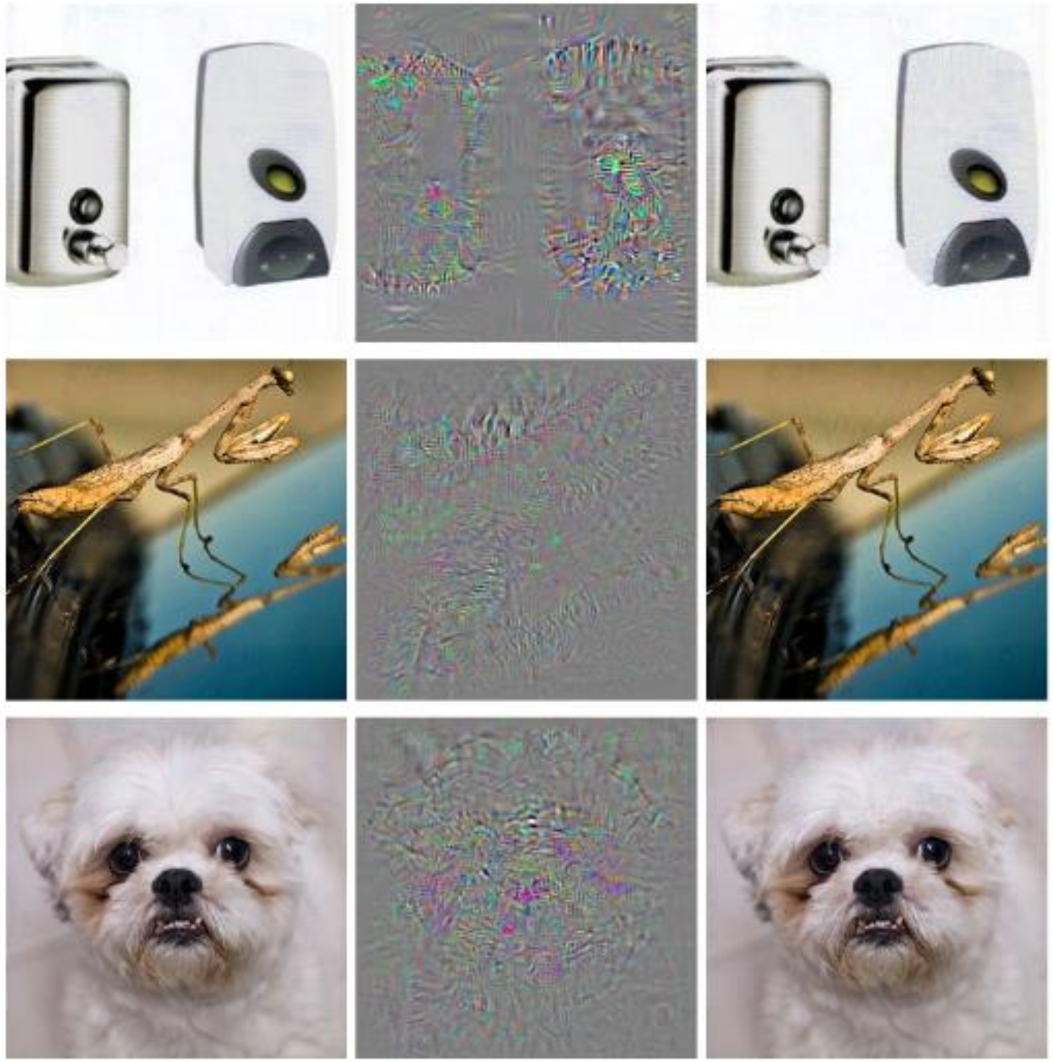
(a) Original Image (b) Guided Backprop ‘Cat’ (c) Grad-CAM ‘Cat’ (d)Guided Grad-CAM ‘Cat’ (e) Occlusion map ‘Cat’ (f) ResNet Grad-CAM ‘Cat’

(g) Original Image (h) Guided Backprop ‘Dog’ (i) Grad-CAM ‘Dog’ (j)Guided Grad-CAM ‘Dog’ (k) Occlusion map ‘Dog’ (l)ResNet Grad-CAM ‘Dog’

Selvaraju et al., 2017

Adversarial images





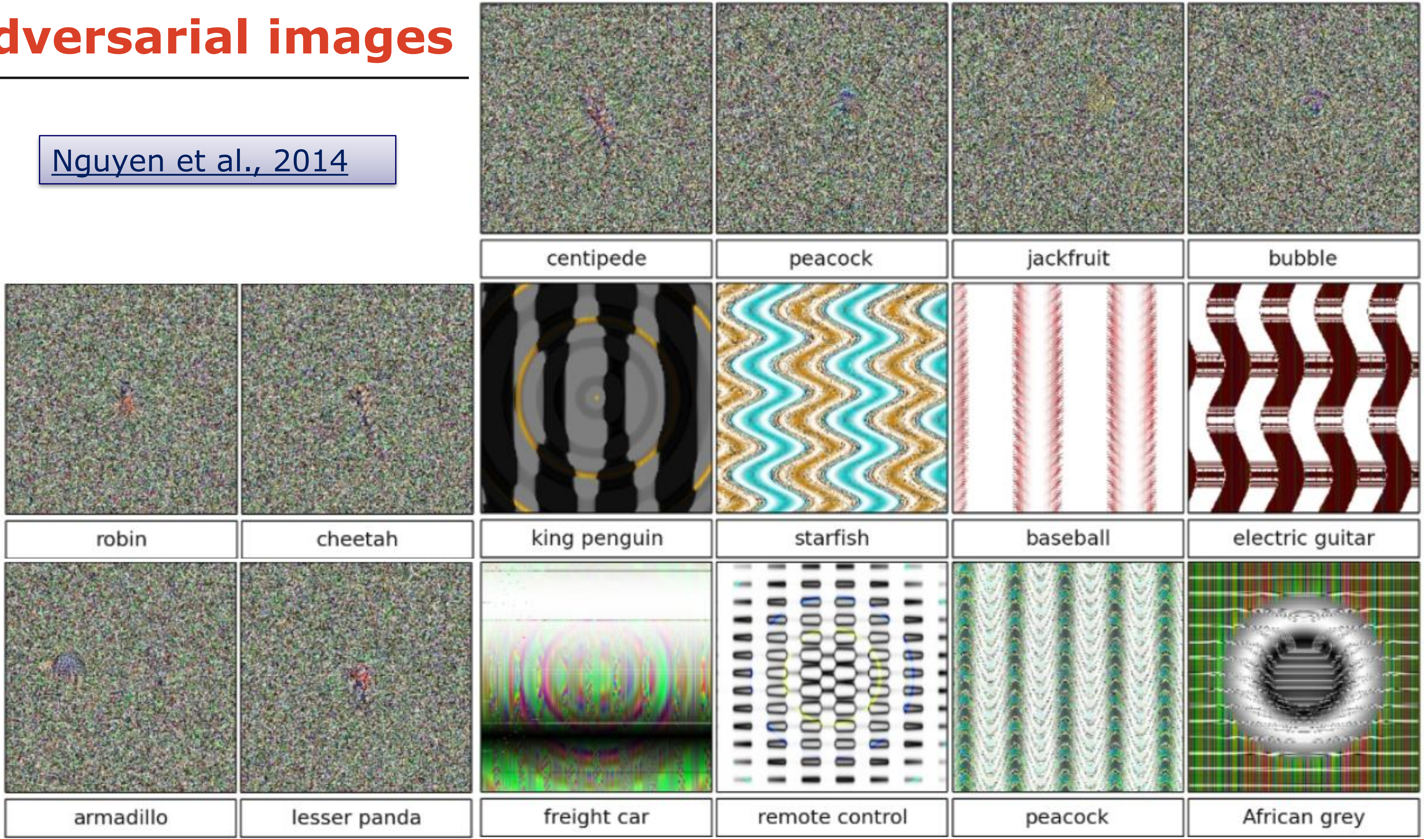
[Szegedy et al., 2013]

Ostrich!

Ostrich!

Adversarial images

Nguyen et al., 2014

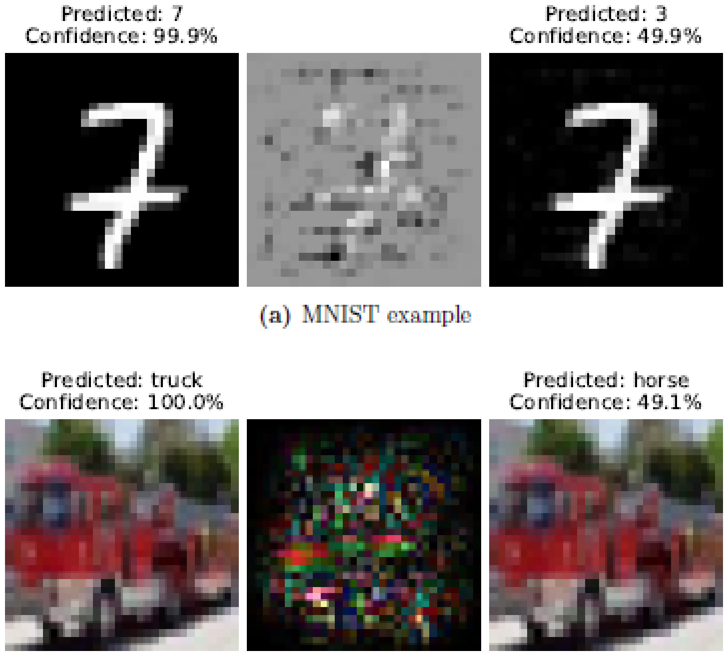


Adversarial example

- Find minimal perturbation that misleads the classifier
  - targeted
  - untargeted

$$\begin{aligned} \text{Minimize} \quad & \|\boldsymbol{\eta}\|_2 \\ \text{subject to} \quad & \mathcal{C}(\boldsymbol{x}+\boldsymbol{\eta}) = l \quad \text{where} \quad l \neq \mathcal{C}^*(\boldsymbol{x}) \\ & \boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^n \end{aligned}$$

$$\begin{aligned} \text{Minimize} \quad & c\|\boldsymbol{\eta}\|_2^2 + J(\boldsymbol{x}+\boldsymbol{\eta}, l) \\ \text{subject to} \quad & \boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^n \end{aligned}$$



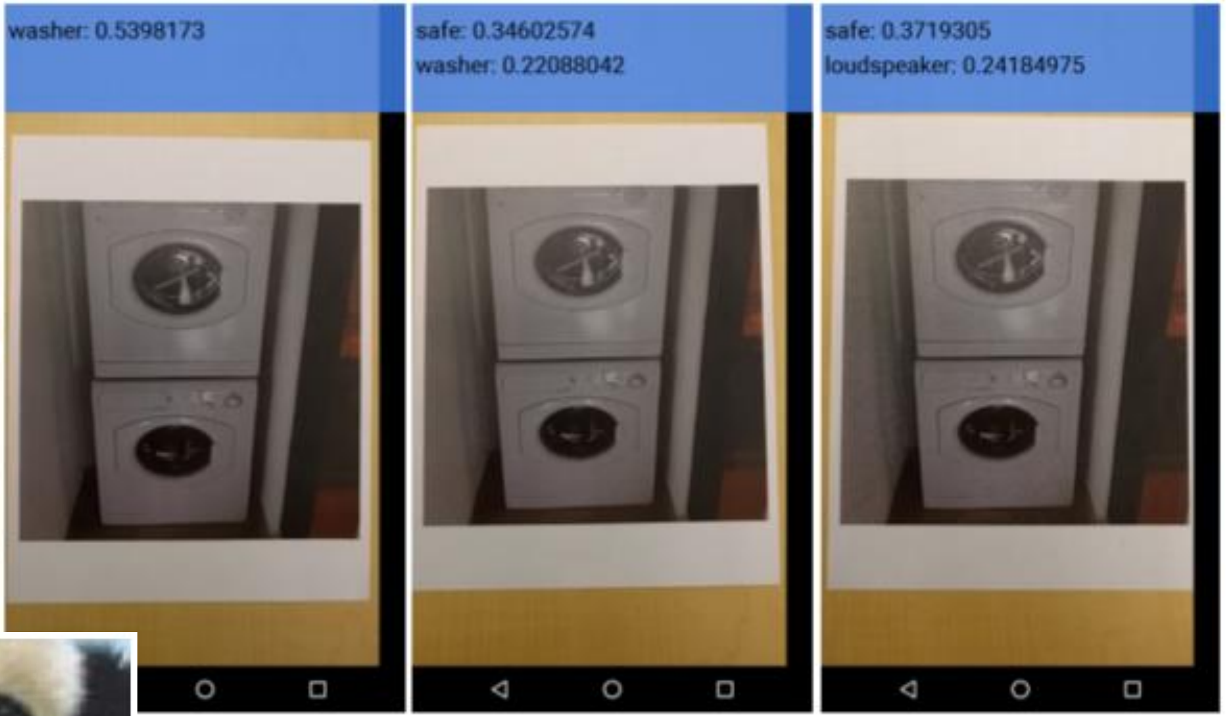
(a) MNIST example
(b) CIFAR10 example

FGSM and BIM

- Fast Gradient Sign Method
- Basic Iterative Method

$$\eta = \varepsilon \, \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

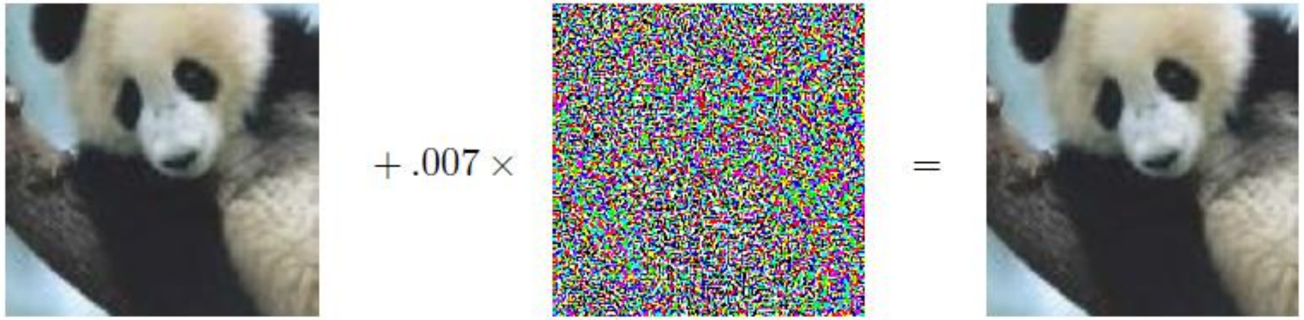
[Goodfellow et al., 2015]



Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

gn$(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}_i, y))$

$+ \eta_i\}$



$\boldsymbol{x}$ "panda" 57.7% confidence

$+ .007 \times$

$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ "nematode" 8.2% confidence

$=$

$\boldsymbol{x} + \epsilon \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ "gibbon" 99.3 % confidence

0

[Kurakin et al., 2017]

DeepFool

$$\Delta(\boldsymbol{x};\mathcal{C}) \equiv \min_{\boldsymbol{\eta}} \|\boldsymbol{\eta}\|_2 \text{ subject to } \mathcal{C}(\boldsymbol{x}+\boldsymbol{\eta}) \neq \mathcal{C}(\boldsymbol{x})$$

$$\rho_{\text{adv}}(\mathcal{C}) = \mathbb{E}_{\boldsymbol{x}} \frac{\Delta(\boldsymbol{x};\mathcal{C})}{\|\boldsymbol{x}\|_2}$$

Moosavi-Dezfooli et al., 2016

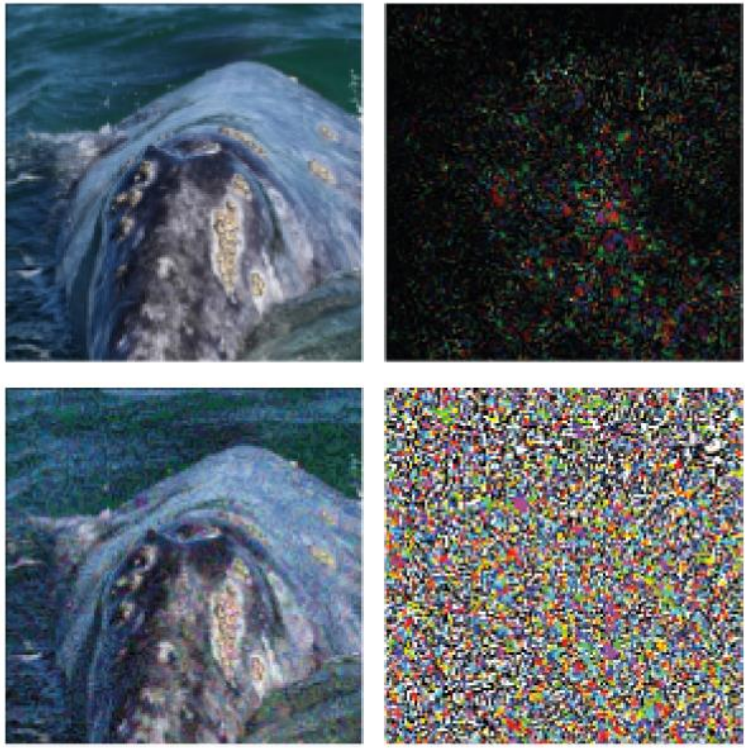
Algorithm 3 The DeepFool algorithm for a multiclass general classifier

input: Image $\boldsymbol{x}$, classifier $\mathcal{C}$, output of the classifier's final layer F, maximum number of iterations `max_iter`, parameter `overshoot`

output: Perturbation $\hat{\boldsymbol{\eta}}$

```
Initialize x_0 ← x, i ← 0
while C(x_i) = C(x_0) and i < max_iter do
    for l ≠ C(x_0) do
        w'_k ← ∇_x F_k(x_i) − ∇_x F_{C(x_0)}(x_i)
        F'_k ← F_k(x_i) − F_{C(x_0)}(x_i)
    end for
    l̂ ← arg min_{l≠C(x_0)} |F'_k| / ||w'_k||_2
    η_i ← |F'_l̂| / ||w'_l̂||_2 · w'_l̂
    x_{i+1} ← x_i + η_i
    i ← i + 1
end while
return η̂ = (1 + overshoot) Σ_i η_i
```

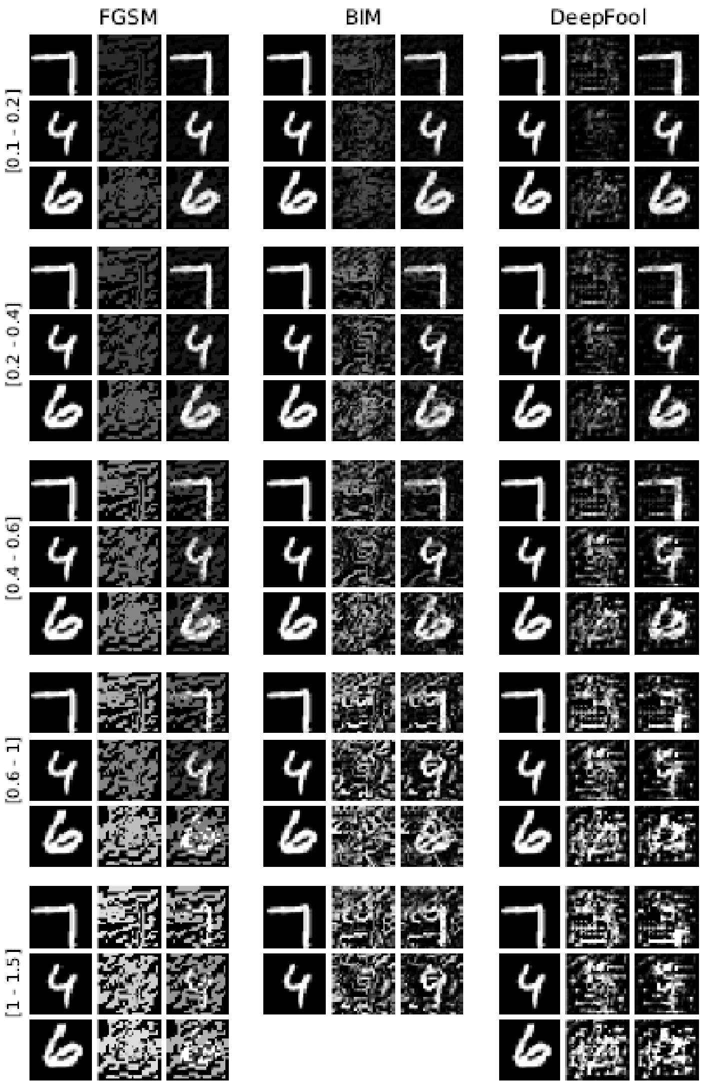
„Whale“

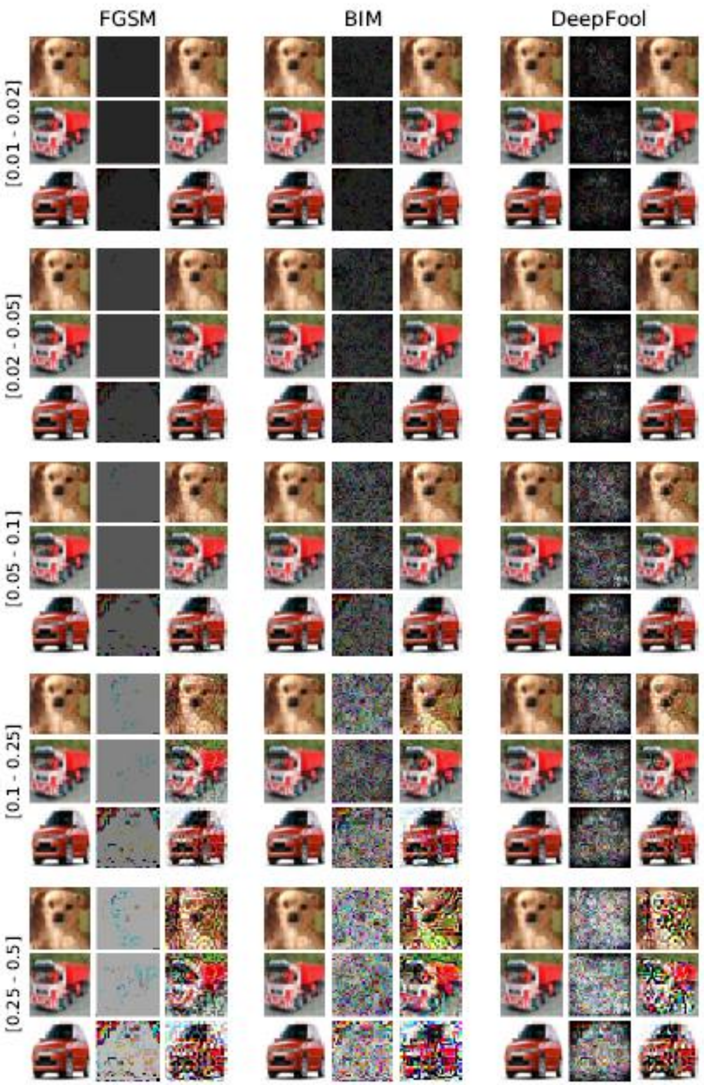


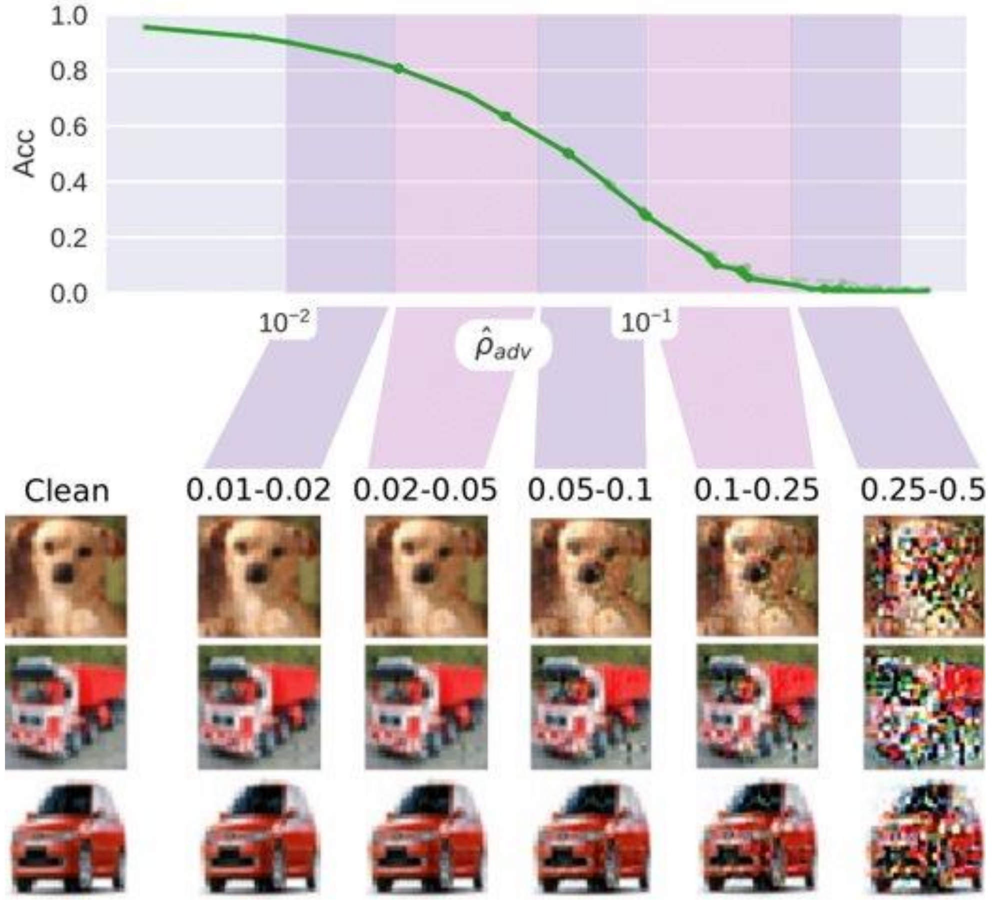
„Turtle“
DeepFool

„Turtle“
FGSM

Accuracy-perturbation curves



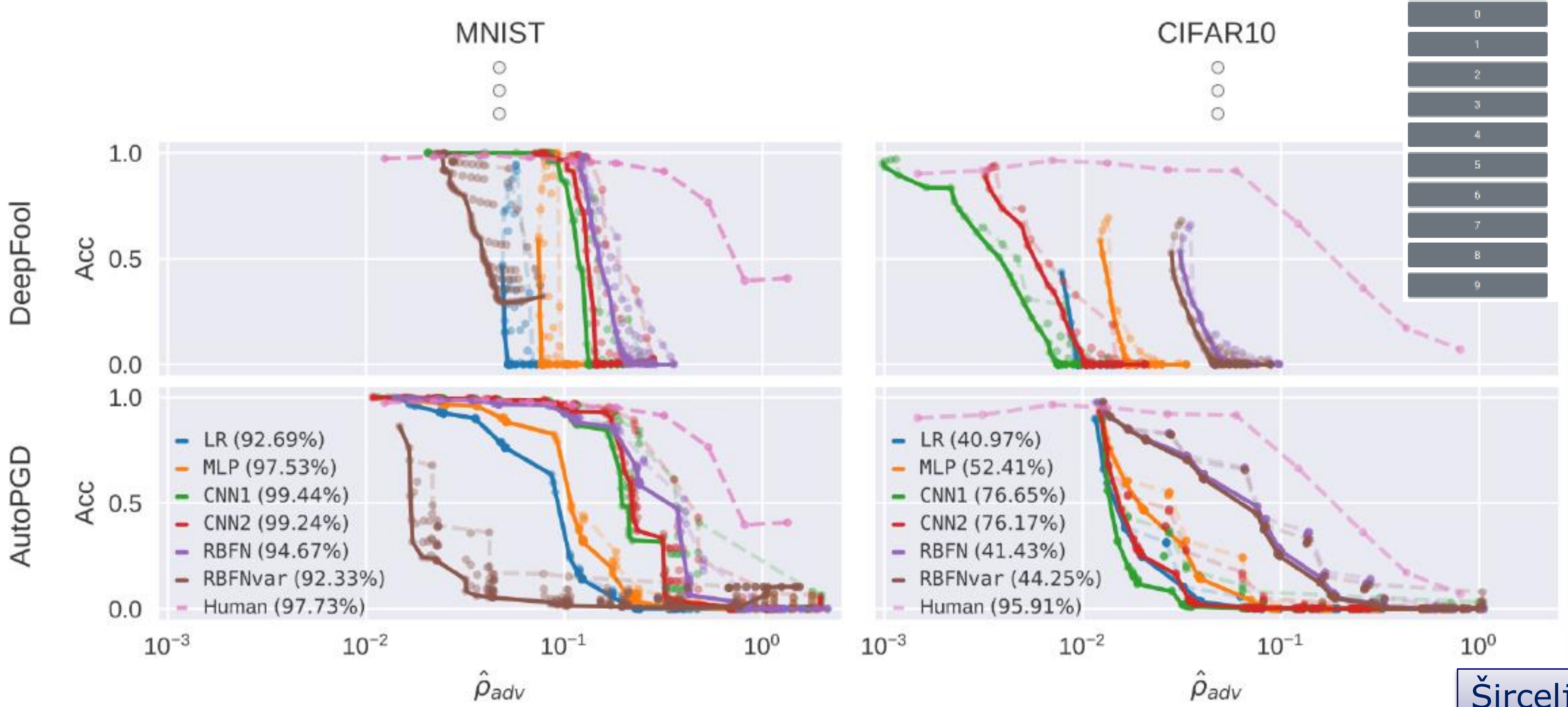




$$\hat{\rho}_{\mathrm{adv}}(\mathcal{C}) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{\|\eta\|_2}{\|x\|_2}$$

Šircelj, 2020

Evaluating adversarial attacks



Šircelj, 2019

Universal perturbations

- Universal perturbations
  - wrt. images
  - wrt. architectures

input: Data points X, classifier $\hat{k}$, desired ℓ_p norm of the perturbation ξ, desired accuracy on perturbed samples δ.
output: Universal perturbation vector v.
Initialize $v \leftarrow 0$.
while $\text{Err}(X_v) \leq 1 - \delta$ **do**
  for each datapoint $x_i \in X$ **do**
    if $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
      Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

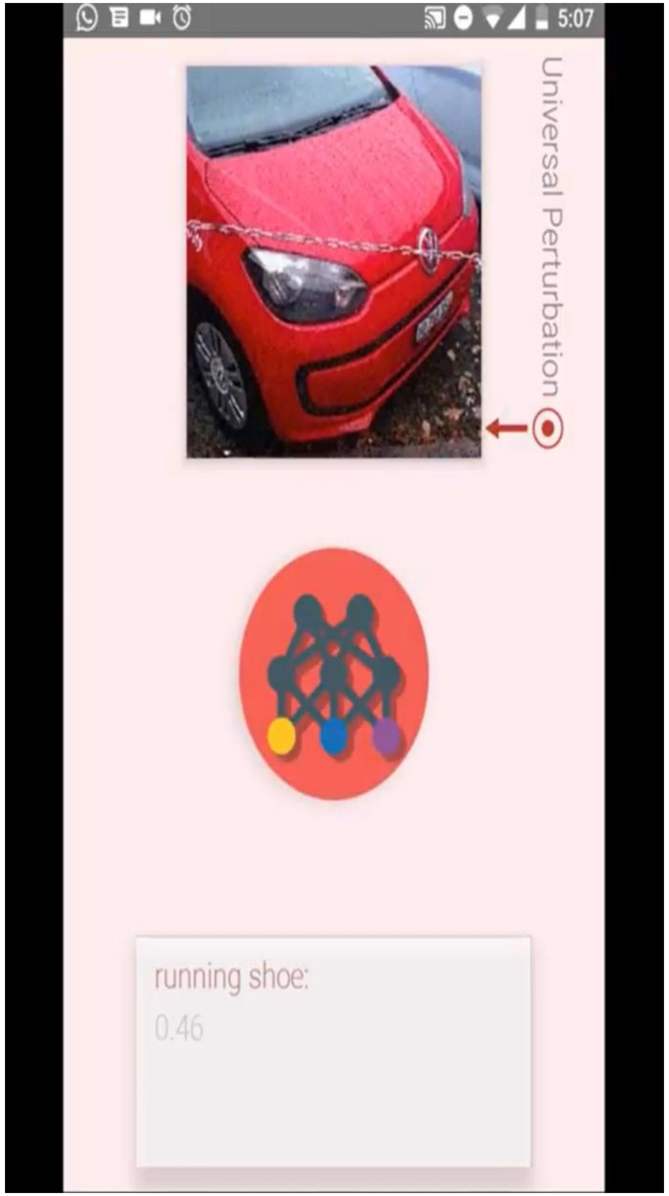
$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

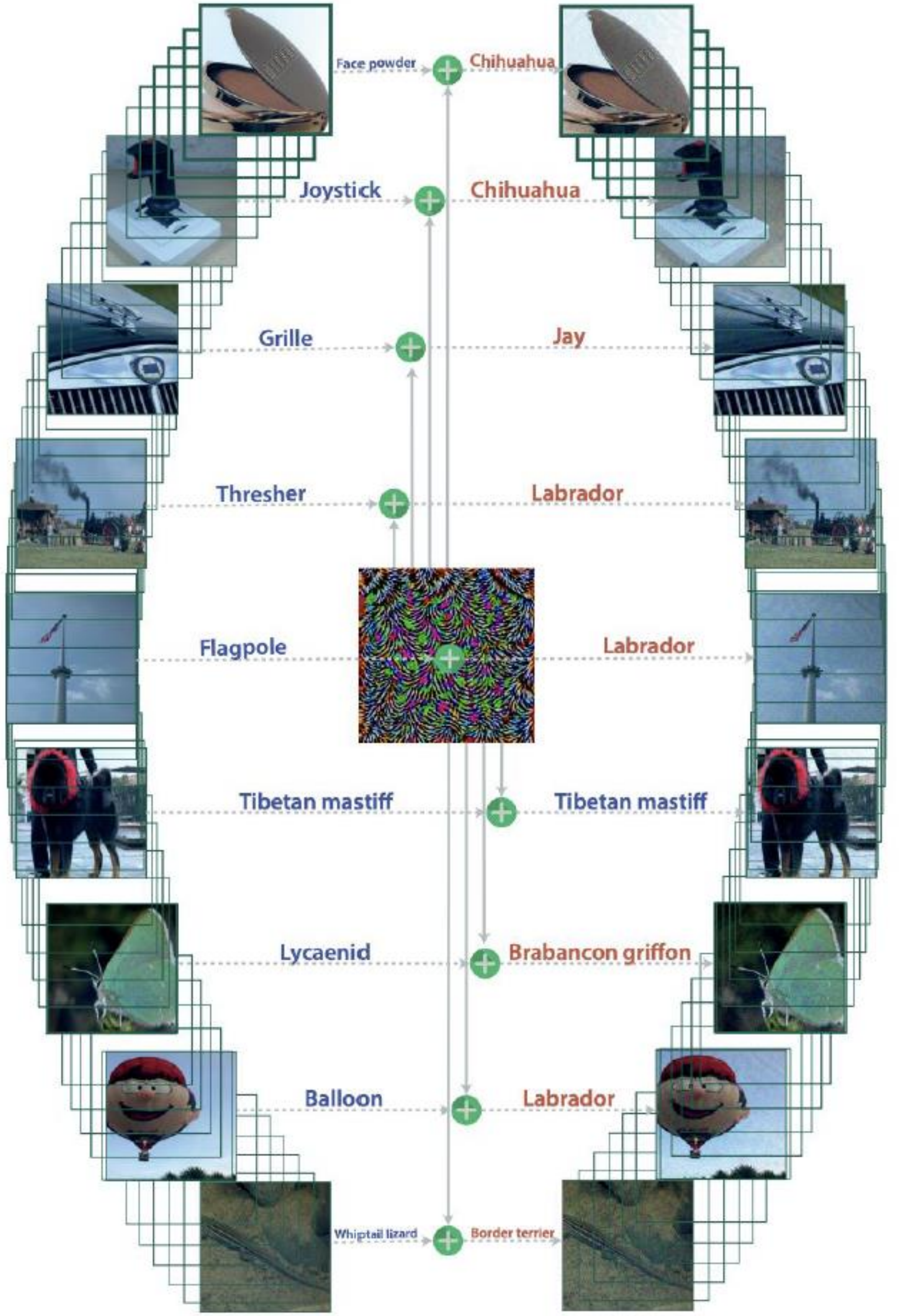
      Update the perturbation:

$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

    end if
  end for
end while

[Moosavi-Dezfooli et al., 2017]

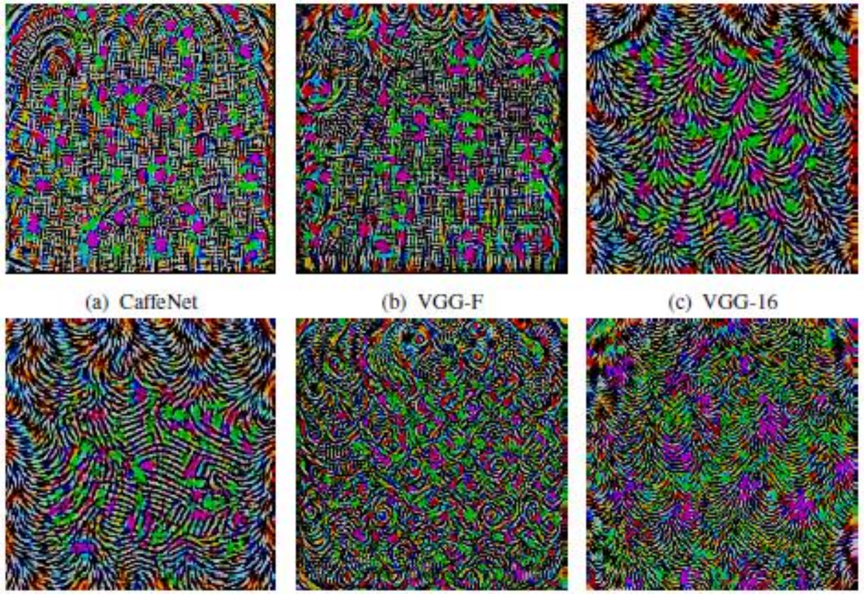




Universal perturbations

- Universal perturbations
  - wrt. images
  - wrt. architectures

	VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19	ResNet-152
VGG-F	**93.7%**	71.8%	48.4%	42.1%	42.1%	47.4 %
CaffeNet	74.0%	**93.3%**	47.7%	39.9%	39.9%	48.0%
GoogLeNet	46.2%	43.8%	**78.9%**	39.2%	39.8%	45.5%
VGG-16	63.4%	55.8%	56.5%	**78.3%**	73.1%	63.4%
VGG-19	64.0%	57.2%	53.6%	73.5%	**77.8%**	58.0%
ResNet-152	46.3%	46.3%	50.5%	47.0%	45.5%	**84.0%**



(a) CaffeNet (b) VGG-F (c) VGG-16
(d) VGG-19 (e) GoogLeNet (f) ResNet-152

		CaffeNet [8]	VGG-F [2]	VGG-16 [17]	VGG-19 [17]	GoogLeNet [18]	ResNet-152 [6]
ℓ_2	X	85.4%	85.9%	90.7%	86.9%	82.9%	89.7%
	Val.	85.6	87.0%	90.3%	84.5%	82.0%	88.5%
ℓ_∞	X	93.1%	93.8%	78.5%	77.8%	80.8%	85.4%
	Val.	93.3%	93.7%	78.3%	77.8%	78.9%	84.0%

Moosavi-Dezfooli et al., 2017

SparseFool

- Sparse perturbations

Modas et al., 2019

Adversarial examples in real world

- Synthesizing Robust Adversarial Examples – adversarial objects
- Adversarial Generative Nets – adversarial glasses
- Robust Physical Perturbations (RP2) – adversarial stickers

classified as turtle   classified as rifle
classified as other

Athalye et al., 2018

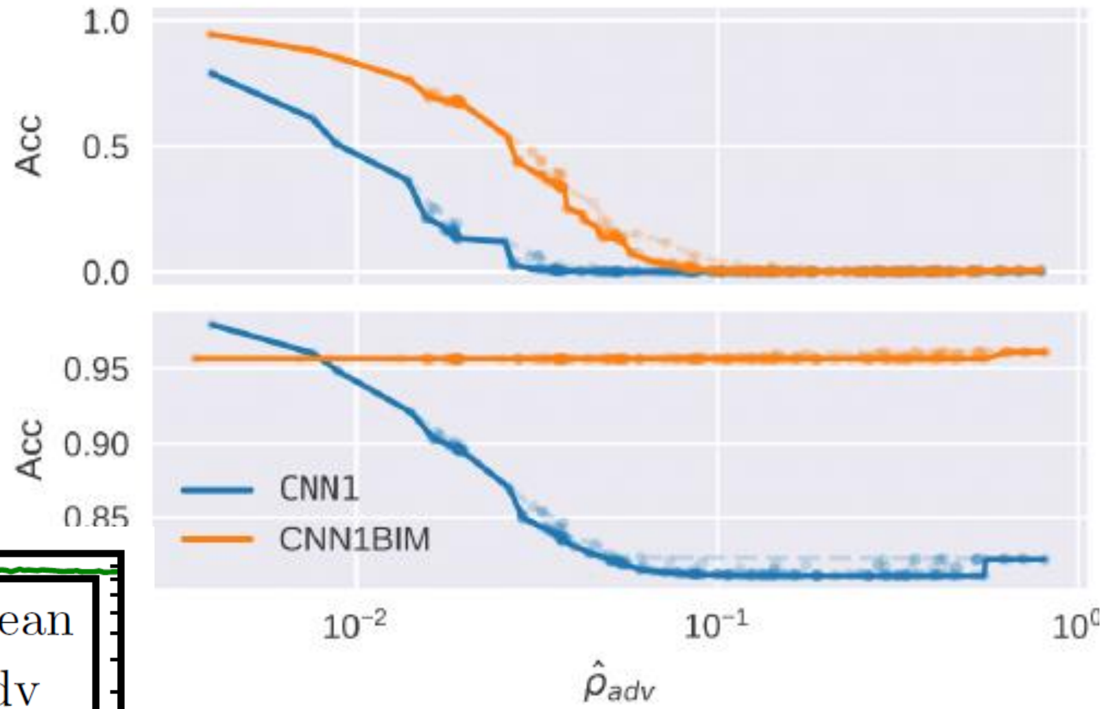
Sharif et al., 2019

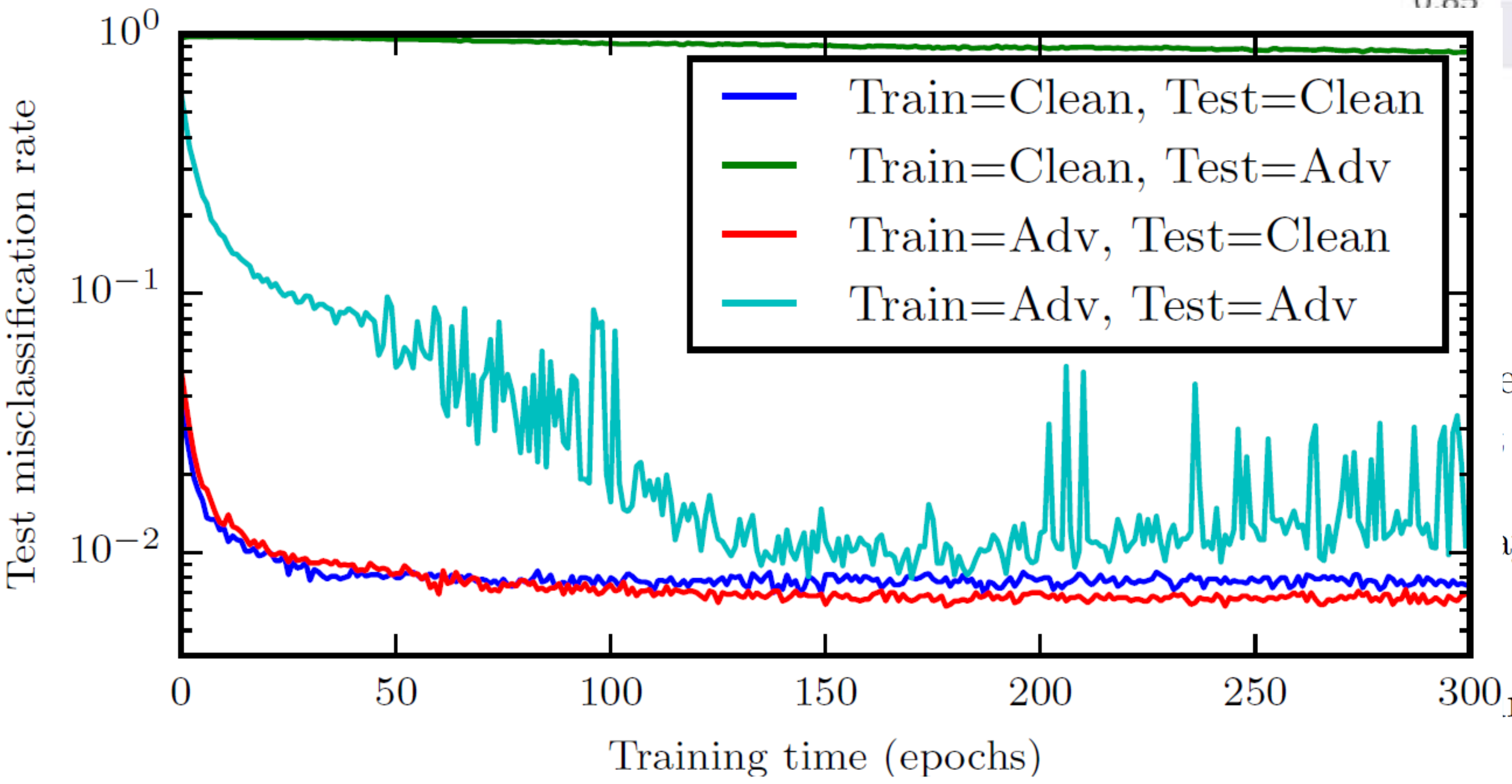
Eykholt et al., 2018

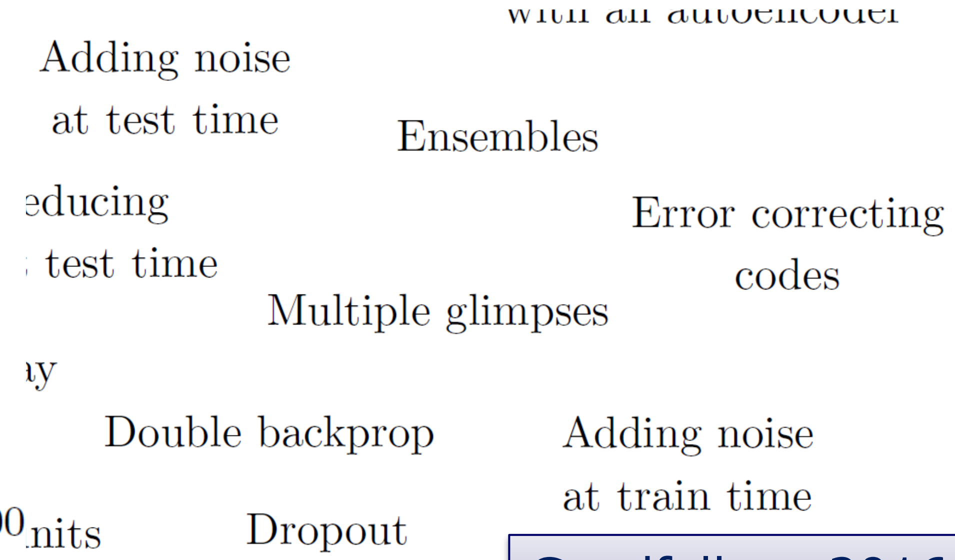
Adversarial training

- Training on adversarial examples
- Works also as a regulizer

Šircelj, 2019







Goodfellow, 2016