DEEP LEARNING

An introduction to deep learning

Jeremy Fix

December 18, 2020

Slides made with slidemaker

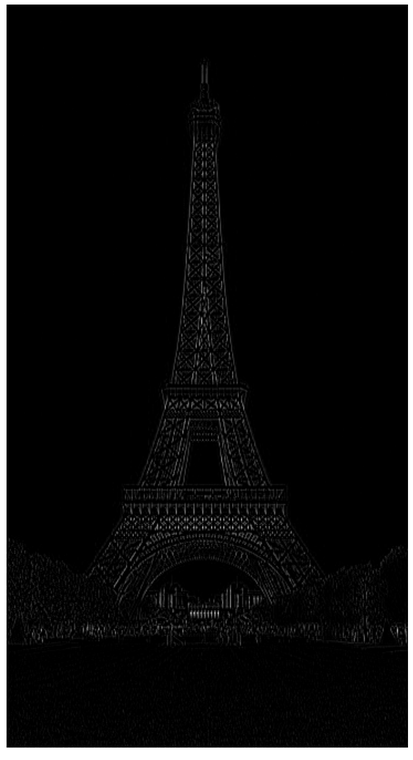
CONVOLUTIONAL NEURAL NETWORKS

EXTRACTING FEATURES WITH CONVOLUTIONS

From data that have a spatial structure (locally correlated), features can be extracted with convolutions.

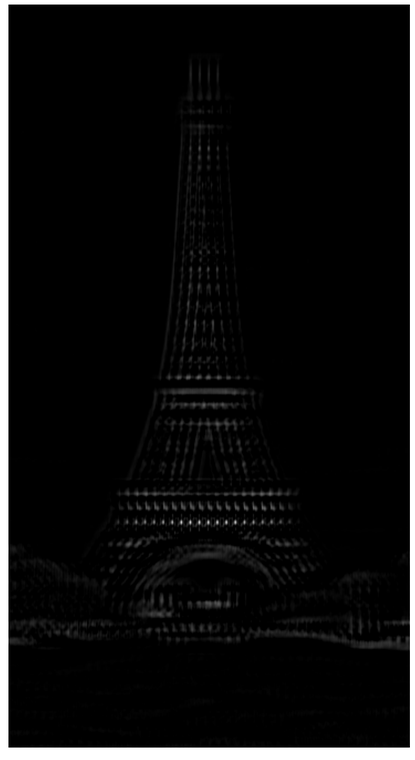
On **Images**

Original image



Discrete laplacian

Gaussian blur



Pattern matching

The pattern

That also makes sense for **temporal series** that have a structure in time.

A CONVOLUTION AS A SPARSE MATRIX MULTIPLY

What is a convolution : Example in 2D

Seen as a matrix multiplication

Given two 1D-vectors f, k, say $k = [c, b, a]$

$$(f * k) = \begin{bmatrix} b & c & 0 & 0 & \cdots & 0 & 0 \\ a & b & c & 0 & \cdots & 0 & 0 \\ 0 & a & b & c & \cdots & 0 & 0 \\ 0 & 0 & a & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & b & c \\ 0 & 0 & 0 & 0 & \cdots & a & b \end{bmatrix} \cdot \begin{bmatrix} \\ f \\ \\ \end{bmatrix}$$

COMPOSITION TO LEARN HIGHER LEVEL FEATURES

Local features can be combined to learn higher level features.

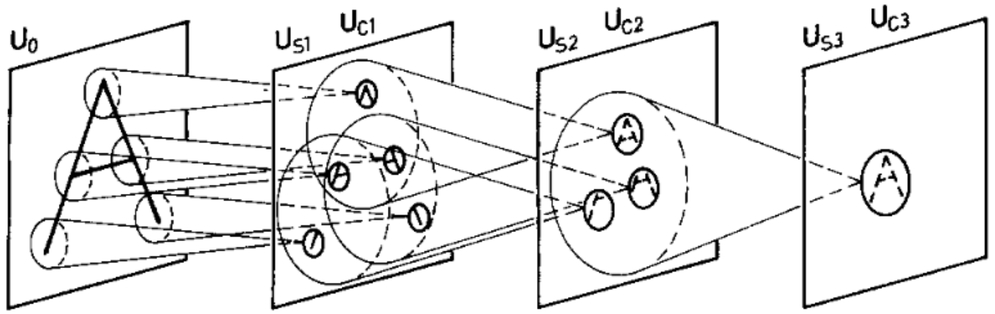
Let us build a house detector

ARCHITECTURE

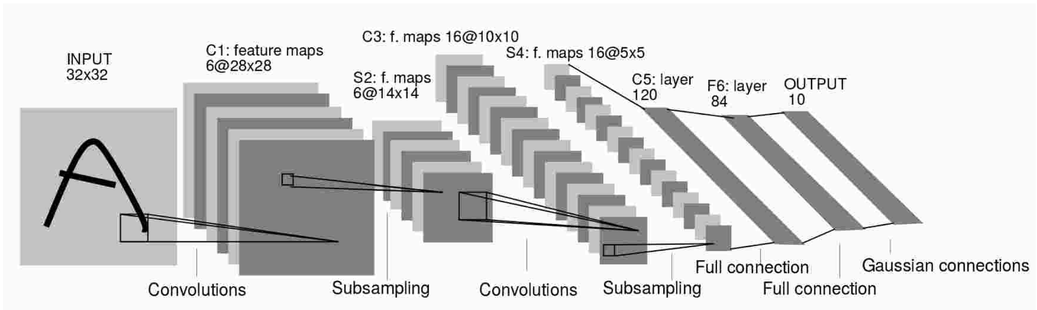
Ideas Using the structure of the inputs to limit the number of parameters without limiting the expressiveness of the network

- For inputs with spatial (or temporal) correlations, features can be extracted with **convolutions of local kernels**
- A convolution can be seen as a fully connected layer with :
  - a lot of weights set exactly to 0
  - a lot of weights **shared** across positions

$\longrightarrow$ strongly regularized !



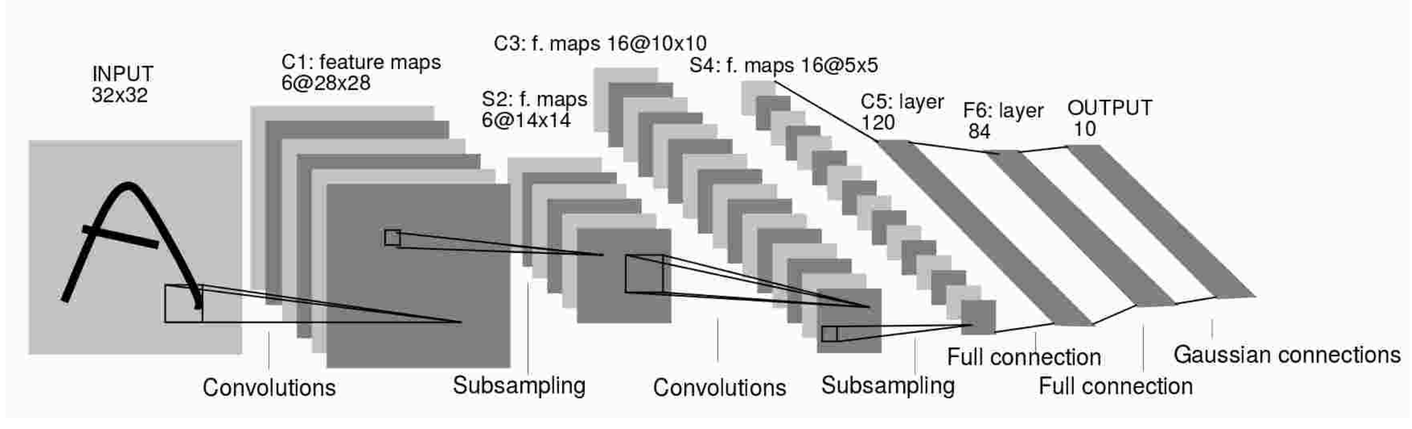
Neocognitron (Fukushima, 1980)



LeNet5 (LeCun et al., 1989)

VANILLA CNN OF LECUN

The architecture of LeNet-5 (LeCun et al., 1989), let's call it the **Vanilla CNN**



LeNet5 (LeCun et al., 1989)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Architecture

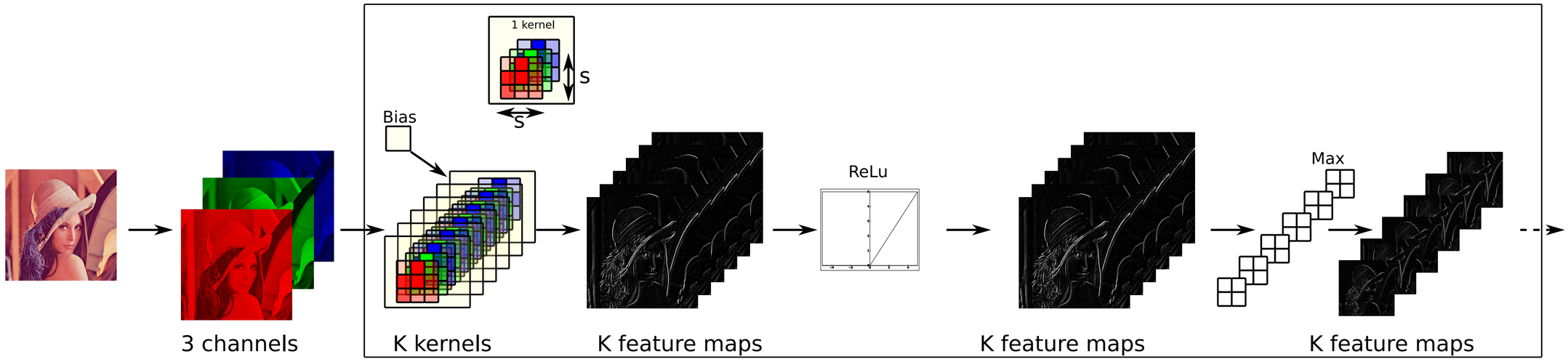
Two main parts :
- convolutional part : C1 -> C5 : convolution - non-linearity - subsampling
- fully connected part :

Specificities :
- Weighted sub-sampling
- Gaussian connections (RBF output layer)
- connectivity pattern $S_2 - C_3$ to reduce the number of weights

Number of parameters :

Layer	Parameters
C_1	156
S_2	12
C_3	1.516
S_4	32
C_5	48.120
F_6	10.164

CNN VOCABULARY



The building blocks of the convolutional part of a vanilla CNN

Convolution :
- size (e.g. $3 \times 3, 5 \times 5$)
- padding (e.g. $1, 2$)
- stride (e.g. 1)

Pooling (max/average):
- size (e.g. 2×2)
- padding (e.g. 0)
- stride (e.g. 2)

We work with 4D tensors for 2D images, 3D tensors for nD temporal series (e.g. multiple simultaneous recordings), 2D tensors for 1D temporal series

In Pytorch, the tensors follow the Batch-Channel-Height-Width (**BCHW**, channel-first) convention. Other frameworks, like TensorFlow or CNTK, use BHWC (channel-last).

CNN IN PRACTICE

Pytorch code for implementing a CNN : Conv1D Conv2D, MaxPool1D MaxPool2D, AveragePooling, etc…

```
conv_model  = nn.Sequential(
*conv_relu_maxpool(cin=3, cout=32,
                   csize=3, cstride=1,
    cpad=1,
                   msize=2, mstride=2,
    mpad=0),
```

```
def conv_relu_maxpool(cin, cout, csize,
        cstride, cpad, msize, mstride,
        mpad):
    return [nn.Conv2d(cin, cout, csize,
        cstride, cpad),
                nn.ReLU(inplace=True),
```

e.g. Conv2d(32, 64, 3, 1, 1)

```
fc_model = nn.Sequential(
    *linear_relu(output_size, 256),
    nn.Linear(256, num_classes)
)
```

How can I get the feature dimensions of conv_model output ?

```
dummy_output = conv_model(torch.zeros((1, 3, height, width)))
output_size = np.prod(dummy_output.shape[1:] )
```

CNN IN PRACTICE

All of these should fit into a nn.Module subclass :

```
class MyModel(torch.nn.Module):
    def __init__(self, ....):
        super(MyModel, self).__init__()
        self.conv_model = nn.Sequential(...)
        output_size = ...
        self.fc_model = nn.Sequential(...)

    def forward(self, inputs):
```

TRANSPOSED CONVOLUTION

Given two 1D-vectors x_1, k, say $k = [c, b, a]$

$$y_1 = (x_1 * k) = \begin{bmatrix} b & c & 0 & 0 & \cdots & 0 & 0 \\ a & b & c & 0 & \cdots & 0 & 0 \\ 0 & a & b & c & \cdots & 0 & 0 \\ 0 & 0 & a & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & b & c \\ 0 & 0 & 0 & 0 & \cdots & a & b \end{bmatrix} . \begin{bmatrix} \\ x_1 \\ \\ \end{bmatrix} = W_k x_1$$

If we compute the gradient of the loss, in denominator layout:

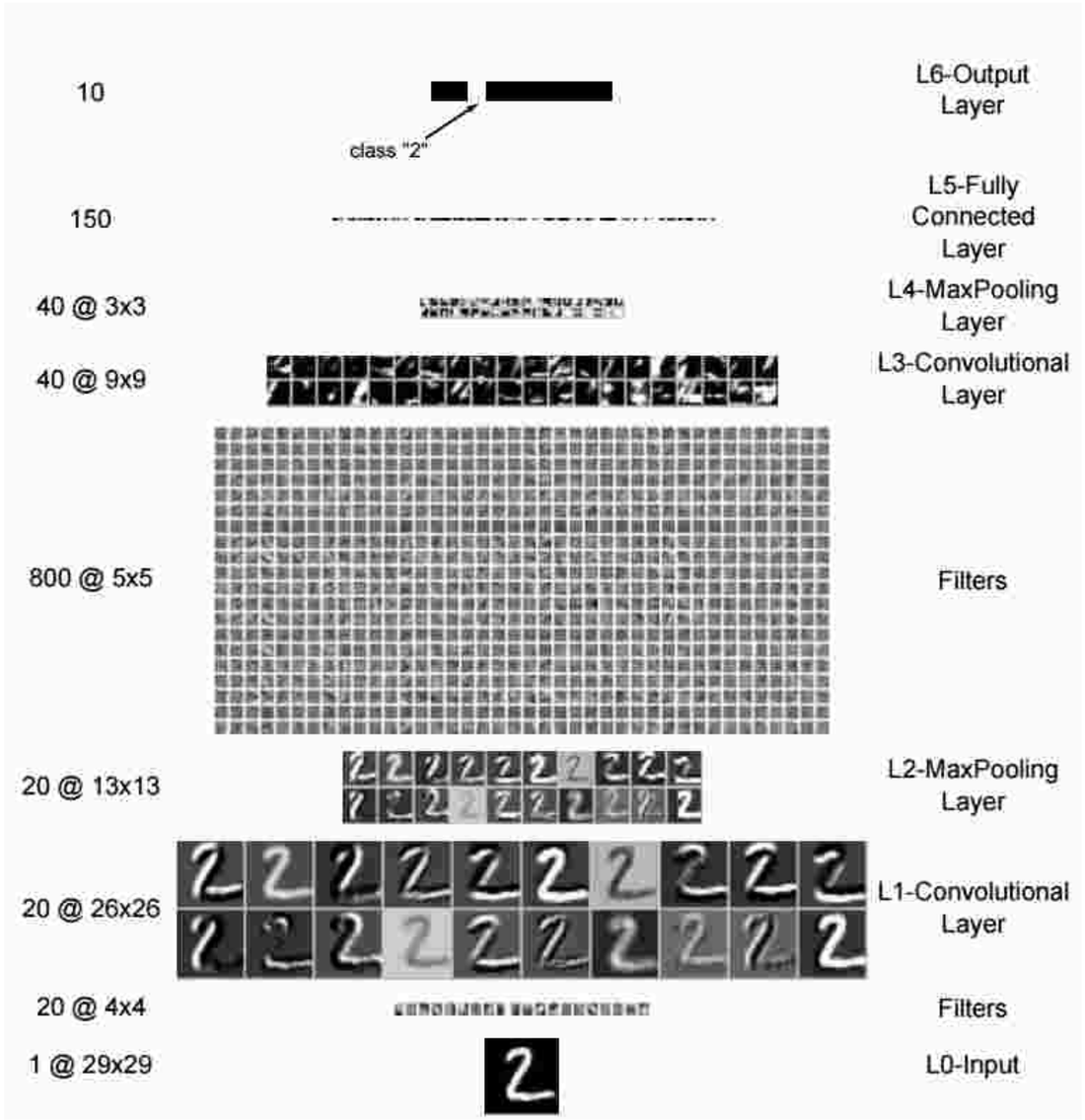
$$\frac{\partial L}{\partial x_1} = \frac{\partial y_1}{\partial x_1} \frac{\partial L}{\partial y_1} = W_k^T \frac{\partial L}{\partial y_1}$$

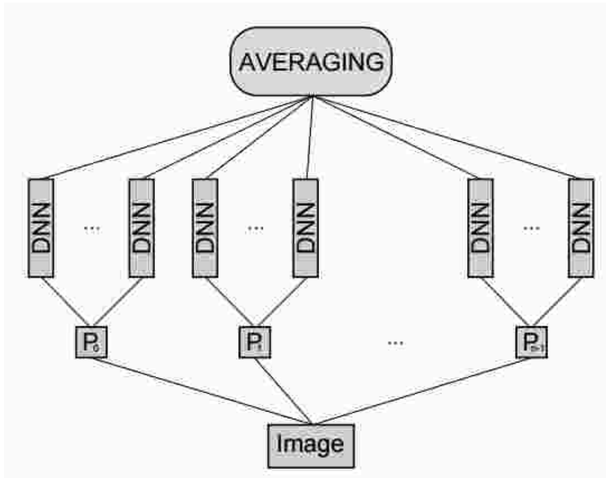
Hence, it is coined the term **transposed convolution** or **backward convolution**. This will pop up again when speaking about **deconvolution**.

10 YEARS OF CNN REVOLUTION

MULTICOLUMN CDNN

Introduced in (Ciresan, Meier, & Schmidhuber, 2012), ensemble of CNNs trained with dataset augmentation

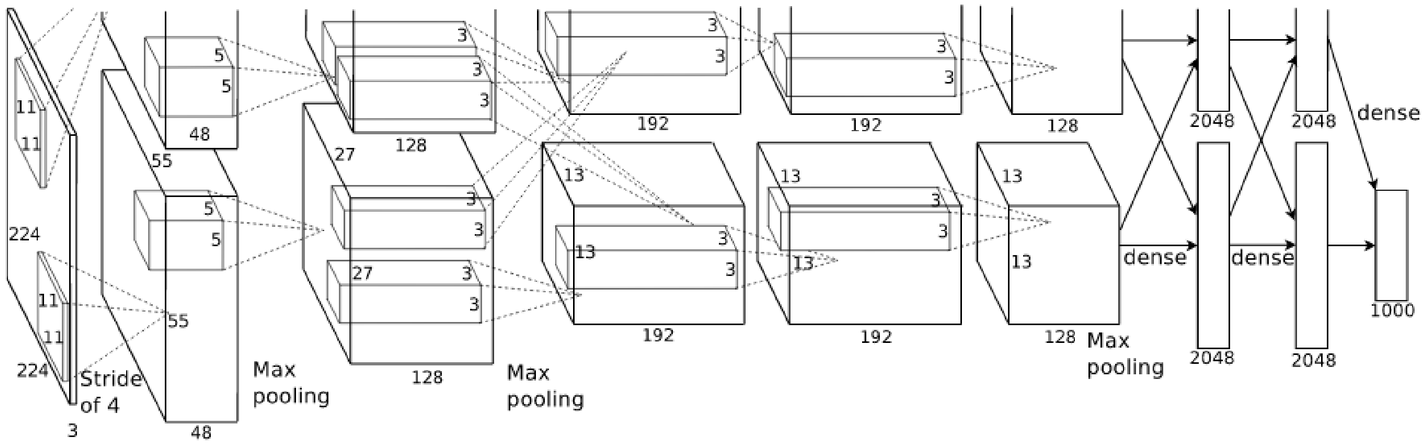


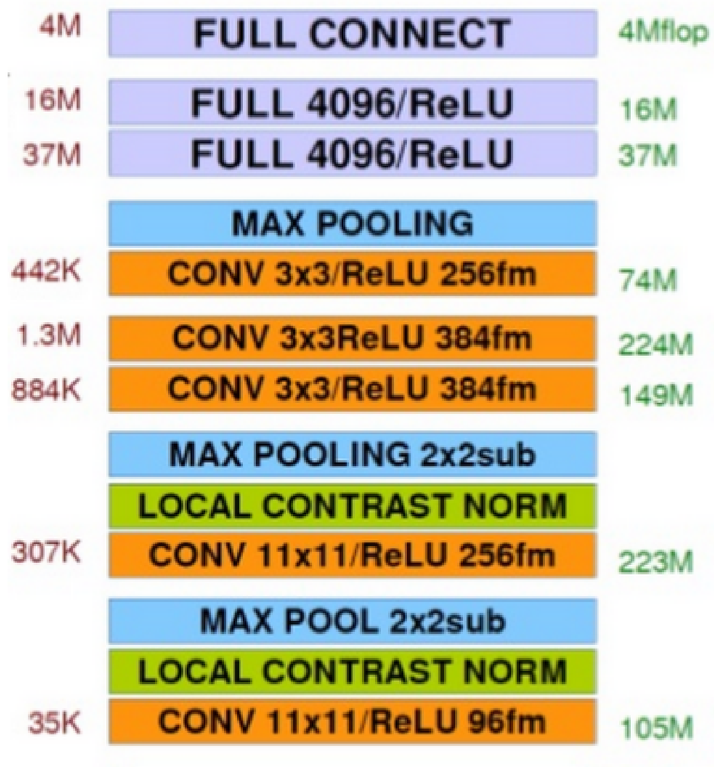


- 0.23% test misclassification on MNIST.
- 1.5 million of parameters

SUPERVISION

Introduced in (Krizhevsky, Sutskever, & Hinton, 2012), the “spark” giving birth to the revival of neural networks.

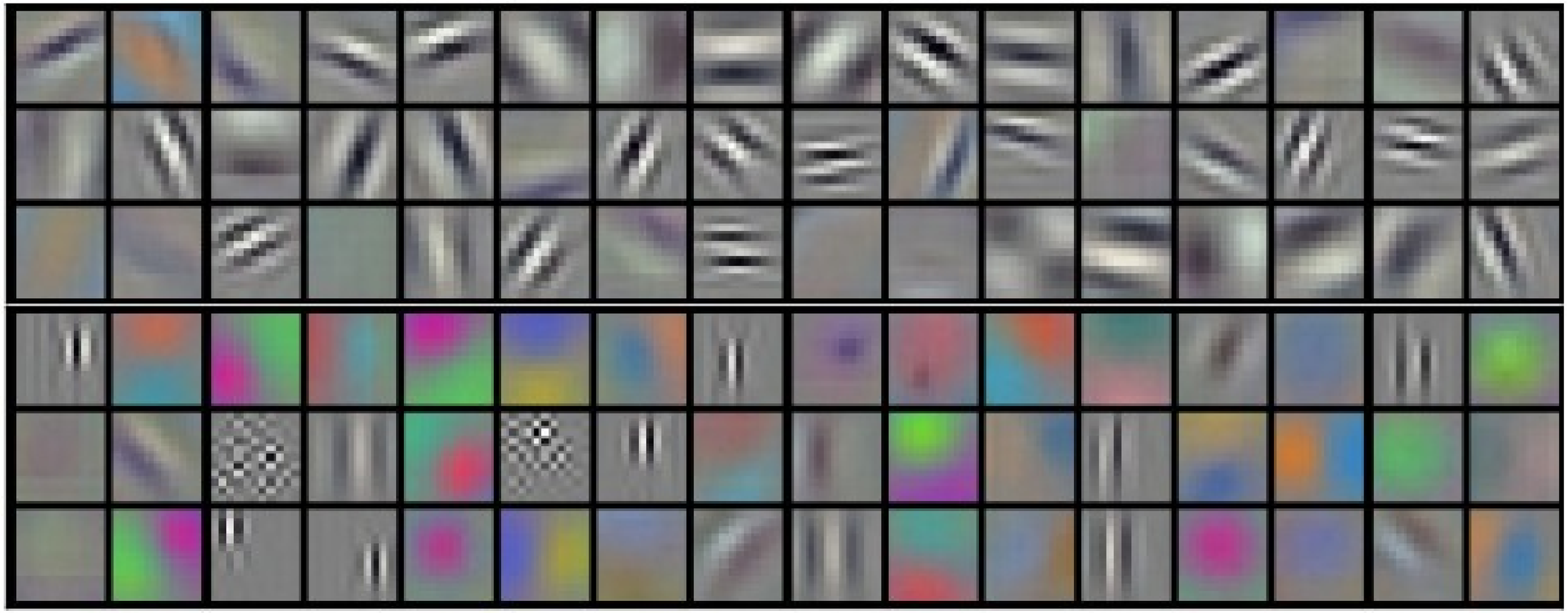




- Top 5 error of 16%, runner-up at 26%
- several convolutions stacked before pooling
- trained on 2 GPUs, for a week on ImageNet (resized to $256 \times 256 \times 3$), 1M images. (now it’s 18 minutes)
- 60 Million parameters, dropout, momentum, L2 penalty, dataset augmentation (rand crop 224×224, translation, reflections, PCA)
- Learning rate at 0.01 divided by 10 when validation error stalls
- at test time, avg probabilities on 5 crops + reflections
- The conv layers are cheap but super important

SUPERVISION

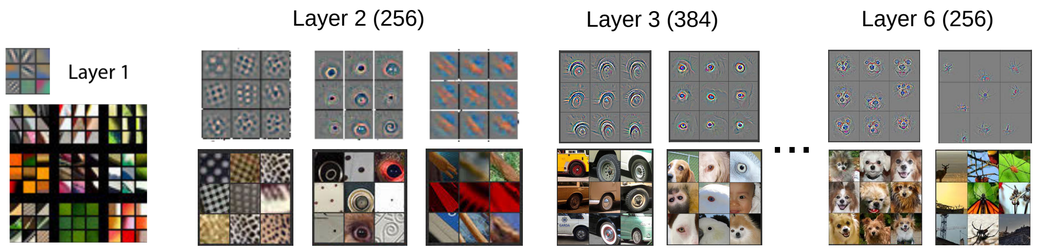
The first layer learned to extract meaningful features



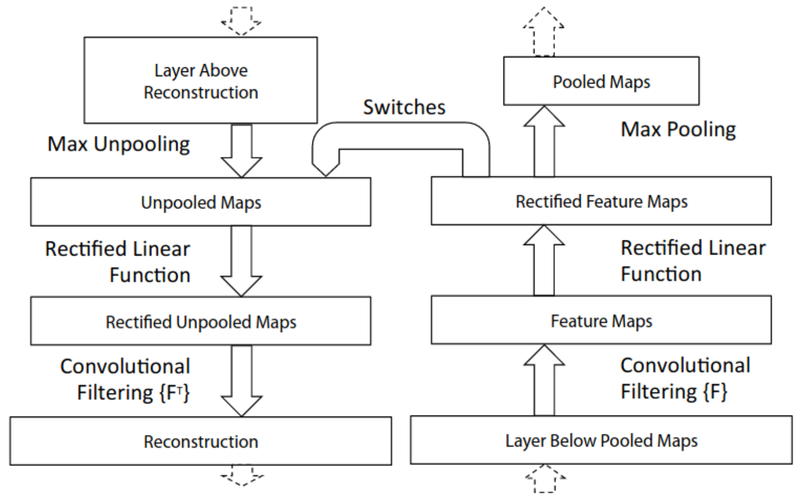
ZFNET

ILSVRC'13 winner. Introduced in (Zeiler & Fergus, 2014)

- Introduced visualization techniques to inspect which features are learned.



Some inputs got by deconvolution



Deconvnet

- Ablation studies on AlexNet : the FC layers are not that important
- Introduced the idea of supervised pretraining (pretraining on ImageNet, finetune the softmax for Caltech-101, Caltech-256, Pascal 2012)
- SGD minibatch(128), momentum(0.9), learning rate (0.01) manual schedule,

Deconvnet computes approximately the gradient of the loss w.r.t. the input (Simonyan, Vedaldi, & Zisserman, 2014). It differs in the way the ReLu is integrated.

VGG

ILSVRC'14 1st runner up. Introduced by (Simonyan & Zisserman, 2015).

- 16 layers : 13 convolutive, 3 fully connected
- Only 3×3 convolution, 2×2 pooling
- **Stacked 3×3 convolutions** $\equiv 5 \times 5$ convolution receptive field with less parameters
  - If $c_{in} = K, c_{out} = K, 5 \times 5$ convolution $\rightarrow$ $25K^2$ parameters
  - If $c_{in} = K, c_{out} = K$, 2 stacked 3×3 convolution $\rightarrow 18K^2$ parameters
- **140 million parameters**, batch size(256), Momentum(0.9), Weight decay(0.0005), Dropout(0.5) in FC, learning rate(0.01) divided 3 times by 10
- Initialization of B, C, D, E from trained A. Init of A random $\mathcal{N}(0, 10^{-2}), b = 0$. Noticed (Glorot & Bengio, 2010) after submission.
- can cope with **variable input size** changing the FC layers to conv 7×7, conv1×1.

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

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).

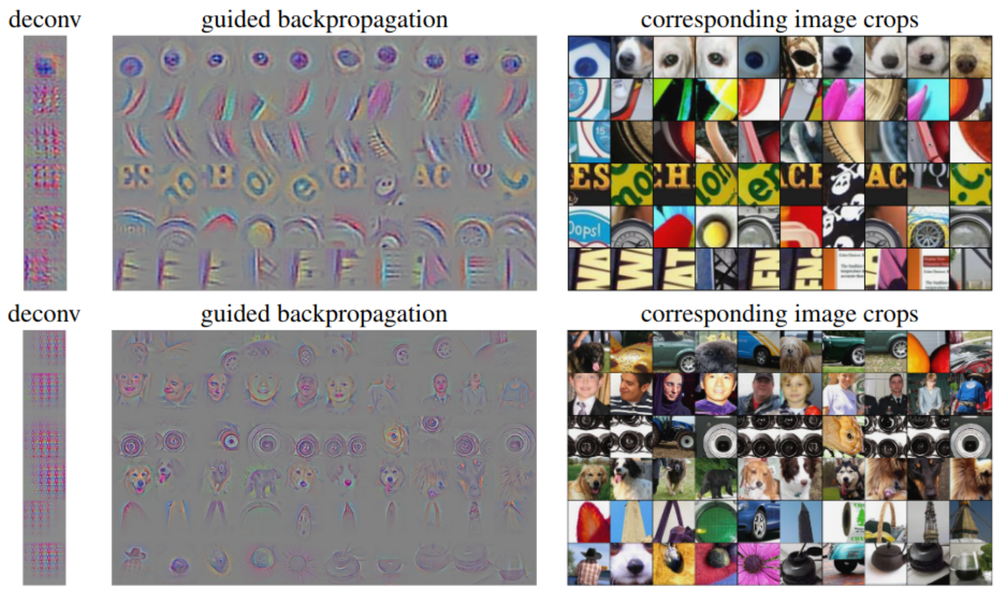
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

The VGG architectures

STRIVING FOR SIMPLICITY

Introduced in (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015).

- uses only convolutions, with various strides, no max pooling
- introduces “guided backpropagation” visualization



Guided backpropagation examples

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

Architectures

GOOGLENET (INCEPTION V1)

ILSVR'14 winner. Introduced by (Szegedy et al., 2014).

Idea Multi-scale feature detection and dimensionality reduction

- 22 layers, 6.8M parameters
- trained in parallel , asynchronous SGD, momentum(0.9), learning rate schedule (4% every 8 epochs)
- at test : polyack average and ensemble of 7 models
- auxiliary heads to mitigate vanishing gradient

GoogleNet last layers

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

RESIDUAL NETWORKS (RESNET)

ILSVRC'15 winner. Introduced in (He et al., 2016a)

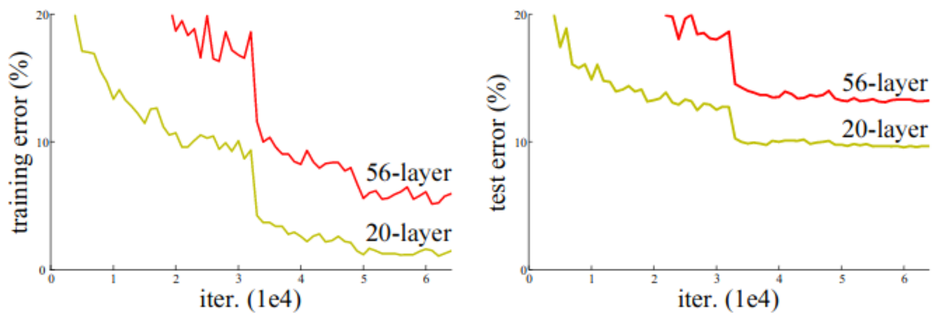
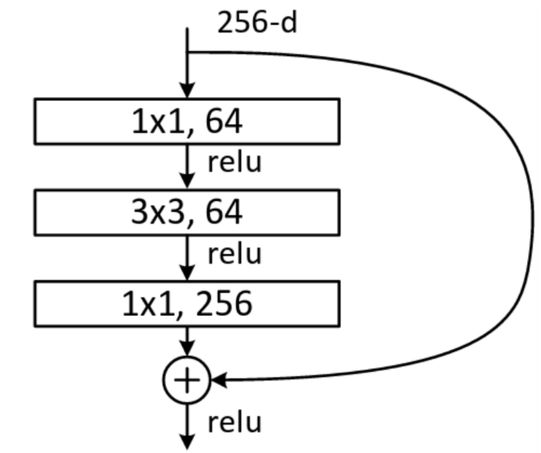


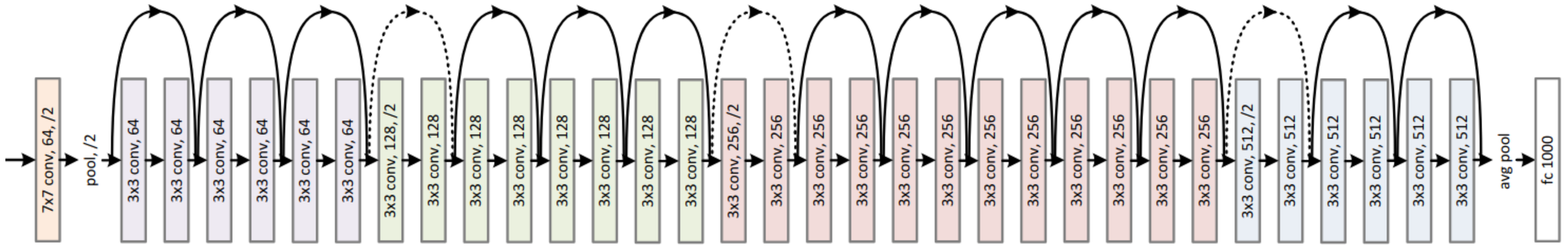
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Deeper is worse ?!



Residual block

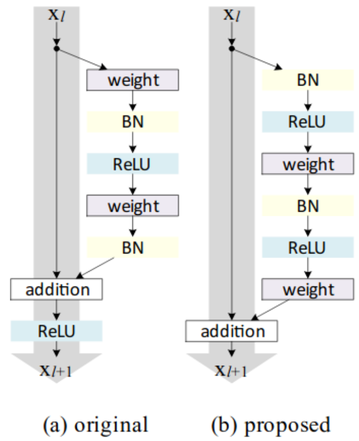
RESIDUAL NETWORKS (RESNET)



ResNet34. Dotted shortcuts and conv"/2" are stride 2 to match the spatial dimensions. Dotted shortcuts use 1×1 conv to match the depth. 0.46M parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Resnet architectures. Conv are "Conv-BN-Relu". ResNet-50 has 23M parameters.



(a) original (b) proposed

Shortcut variations (He et al., 2016b)

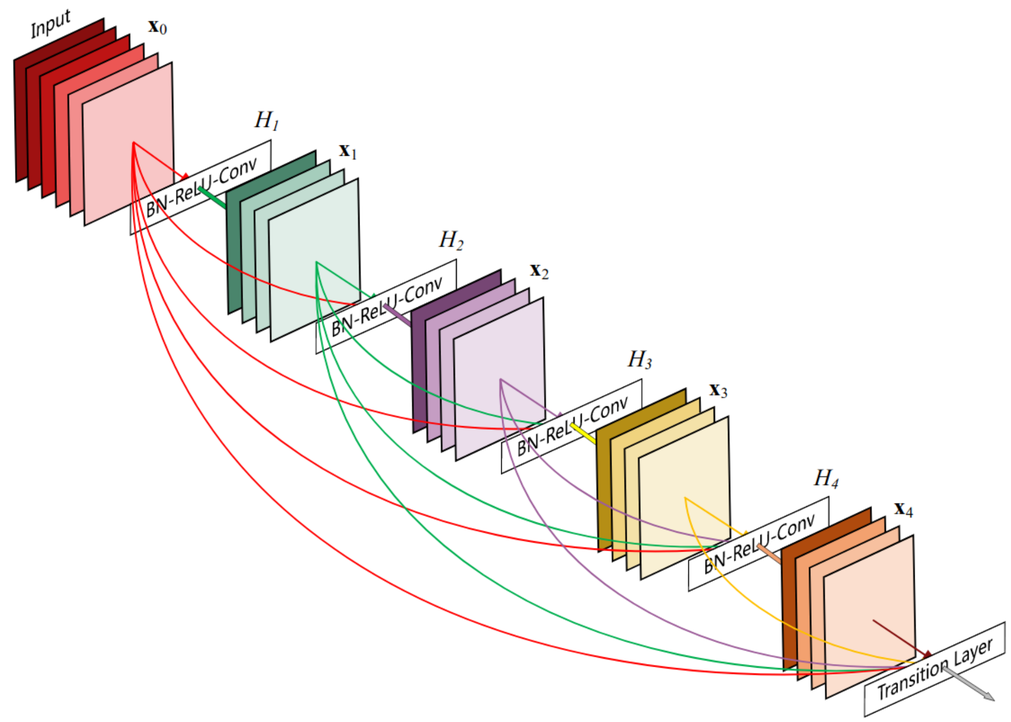
VARIATIONS AROUND SKIP LAYER CONNECTIONS

Highway Networks (Srivastava, Greff, & Schmidhuber, 2015)

- Uses “gates” (as in LSTM, see lectures on RNN) :
  - Transform gate $T(x) = \sigma(W_T x + b_T)$
  - Carry gate $C(x) = \sigma(f_c(x))$

$$y = T(x).\, H(x) + C(x).\, x$$

DenseNets



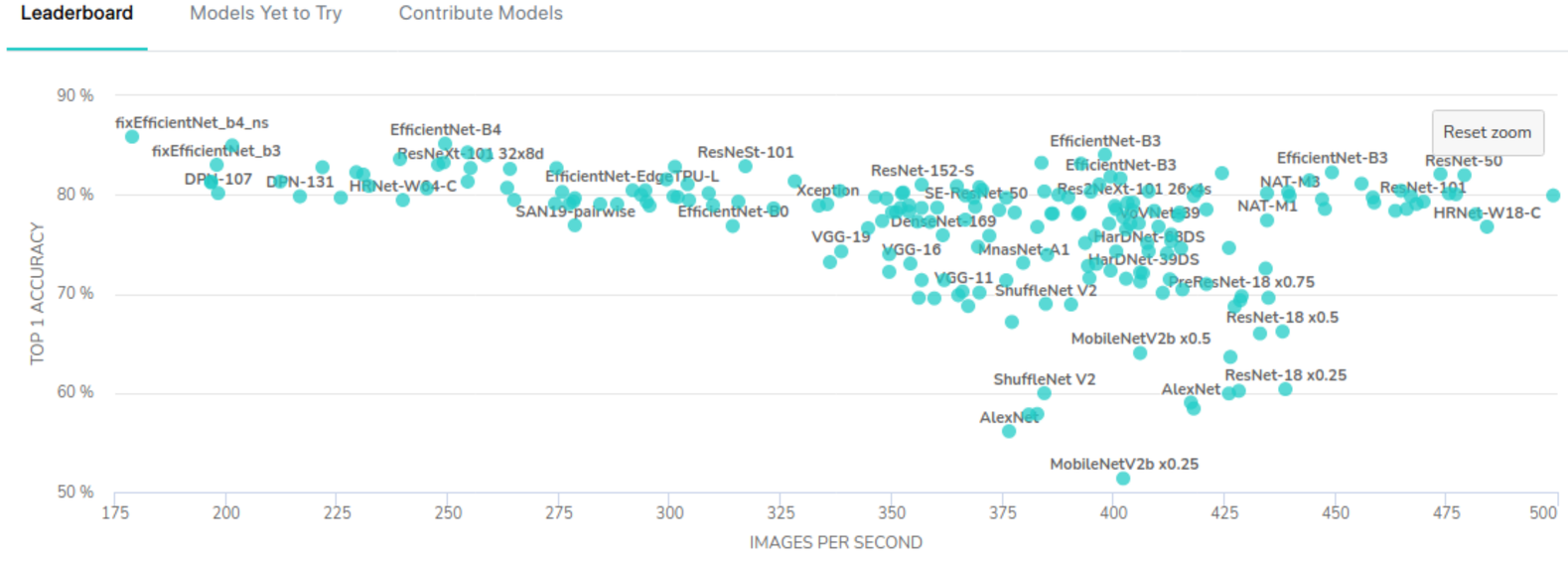
Densenets (Huang, Liu, Maaten, & Weinberger, 2018)

OTHER NETWORKS

- Fitnet [Romero(2015)]
- Wideresnet(2017)
- Mobilenetv1, v2, v3 [Howard(2019)] : searching for the best architecture
- EfficientNet (Tan & Le, 2020)

See also :

- Papers with code
- SOTA bench



#	MODEL	REPOSITORY	TOP 1 ACCURACY	TOP 5 ACCURACY	SPEED ⓘ	PAPER	ε-REPRODUCES PAPER ⓘ
1	**FixEfficientNet_L2** ⓘ	rwightman / **pytorch-image-models**	88.5%	98.7%	11.1		--
2	**EfficientNet-L2** ⓘ	rwightman / **pytorch-image-models**	88.3%	98.7%	6.0		--

CNN DESIGN PRINCIPLES

NUMBER OF FILTERS

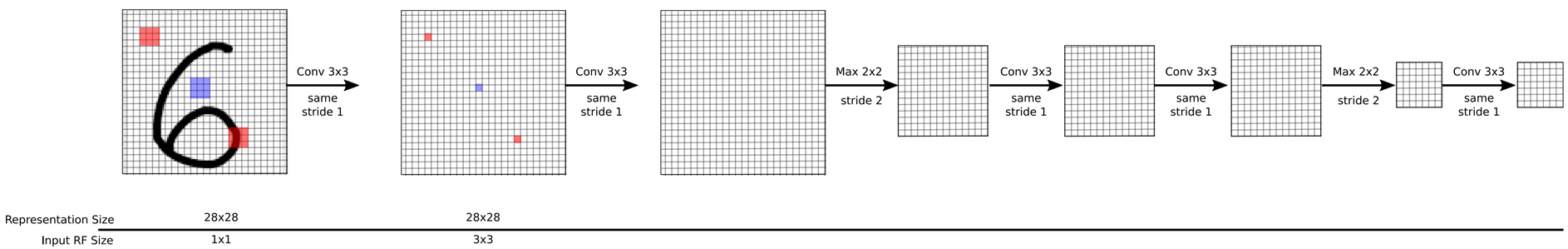
You should increase the number of filters throughout the network :

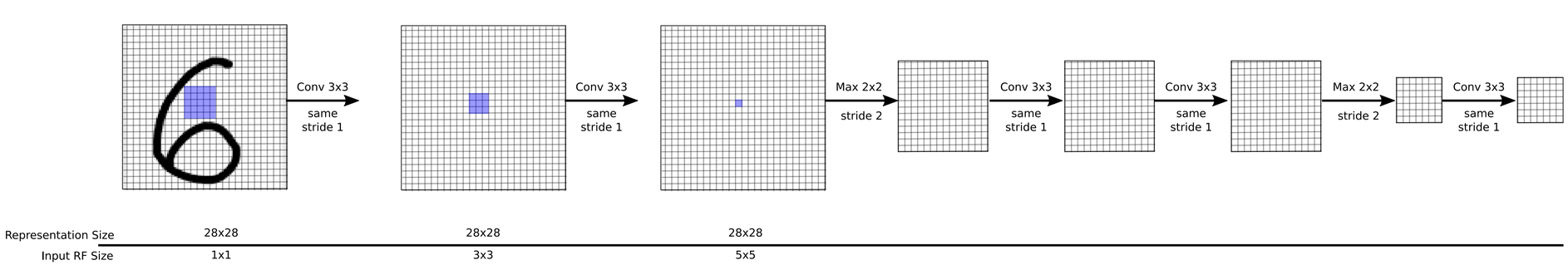
- the first layer extracts low level features
- the higher layers **compose** on the lower layer dictionary of features

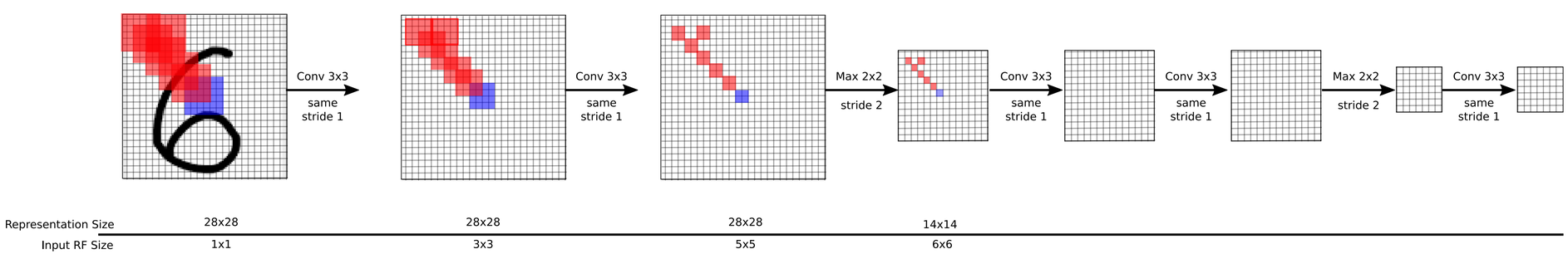
Examples :

- LeNet-5 (1998) : $65 \times 5, 165 \times 5$
- AlexNet (2012) : $9611 \times 11, 2565 \times 5, 2 \times (3843 \times 3), 2563 \times 3$
- VGG (2014) : $64 - 128 - 256 - 512$, all 3×3
- ResNet (2015) : $64 - 128 - 256 - 512$, all 3×3
- Inception (2015) : $32 - 64 - 80 - 192 - 288 - 768 - 1280 - 2048, 1 \times 1, 3 \times 3, ”\ 5 \times 5\ ”$

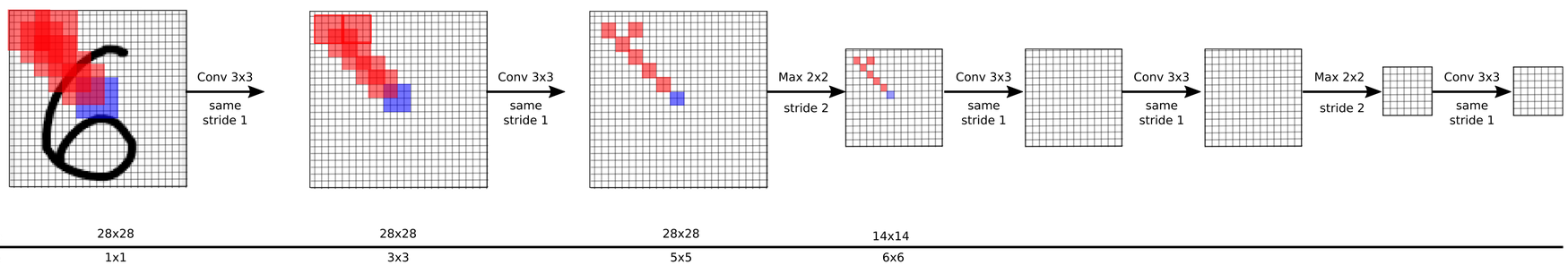
EFFECTIVE RECEPTIVE FIELD (1/3)

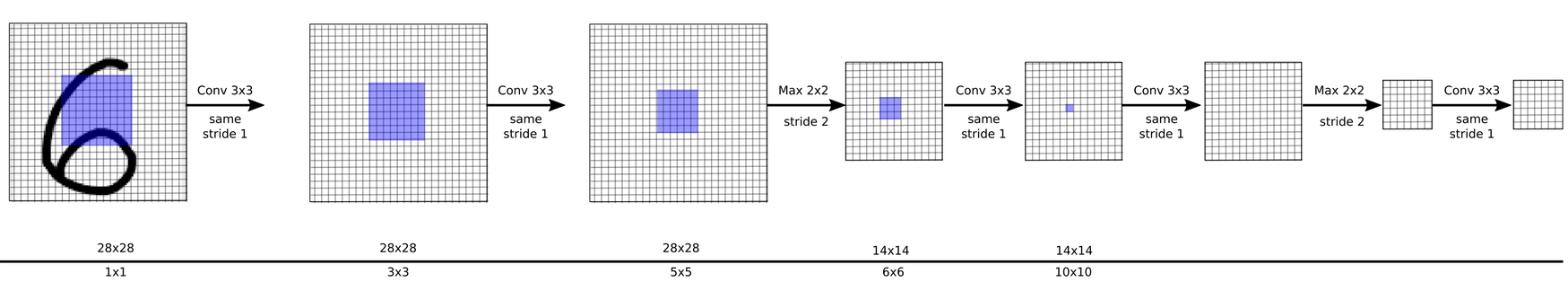


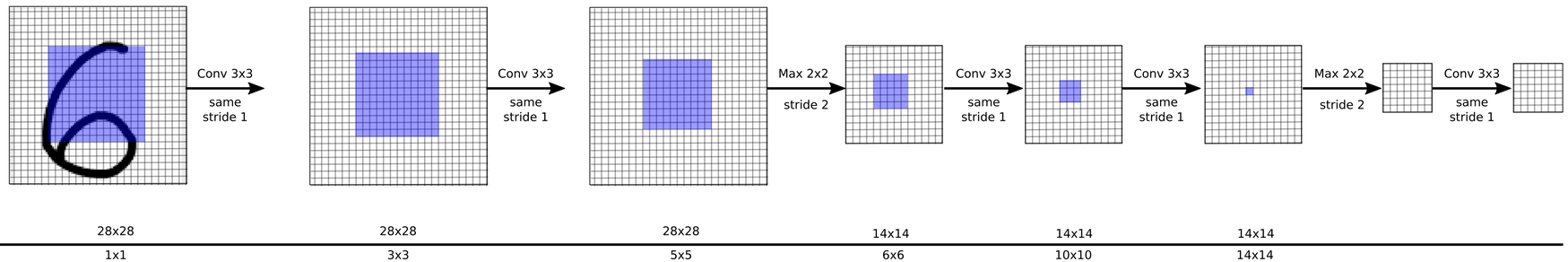




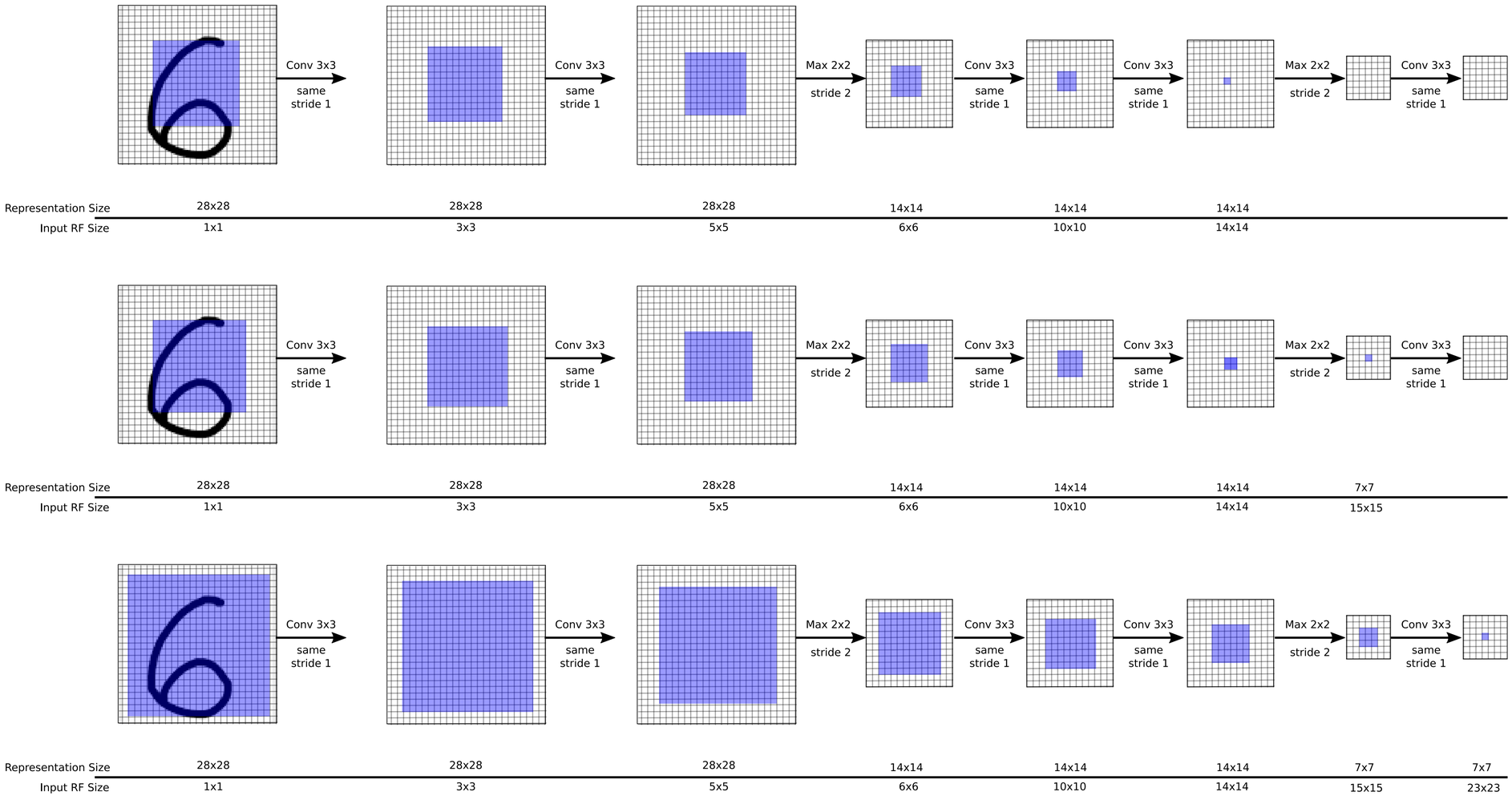
EFFECTIVE RECEPTIVE FIELD (2/3)







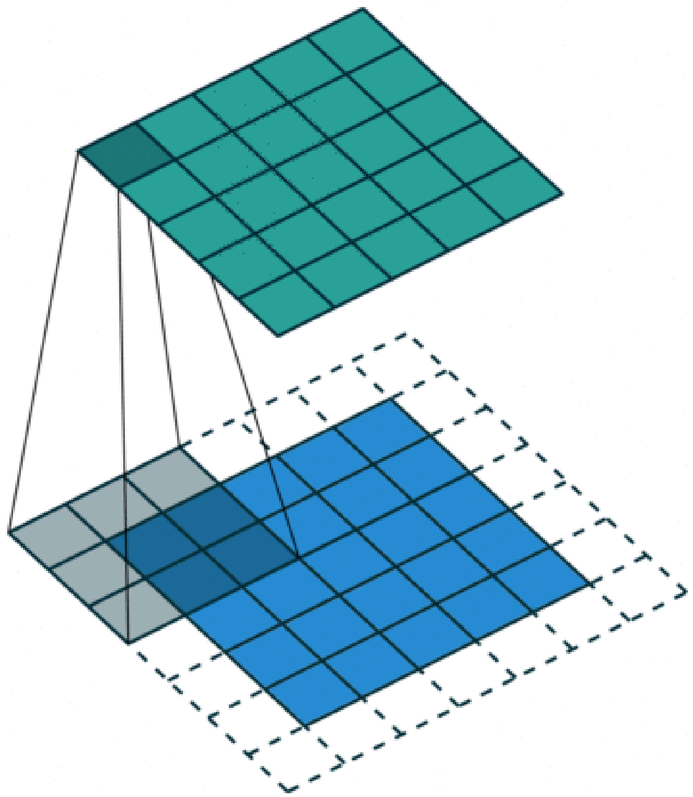
EFFECTIVE RECEPTIVE FIELD (3/3)



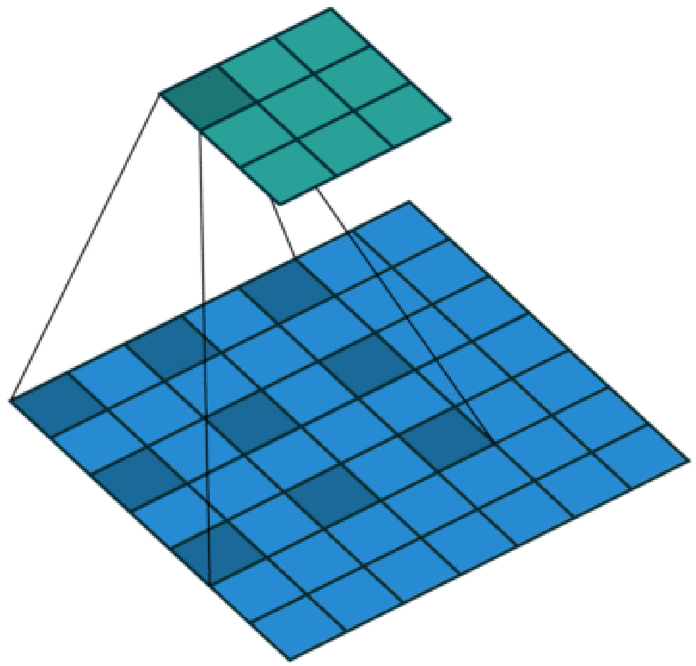
For calculating the effective receptive field size, see for example this calculator or this guide on conv arithmetic.

A-TROU CONVOLUTIONS

Your effective receptive field can grow faster with *a-trou* convolutions (or dilated convolutions) (Yu & Koltun, 2016):



Conv 3, Pad 1, Stride 1



Conv 3, Pad 0, Stride 1, Dilated 1

Illustrations from this guide on conv arithmetic. The Conv2D object's constructor accepts a *dilation* argument.

STACKING AND FACTORIZING SMALL KERNELS

Introduced in Inception v3 (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2015)

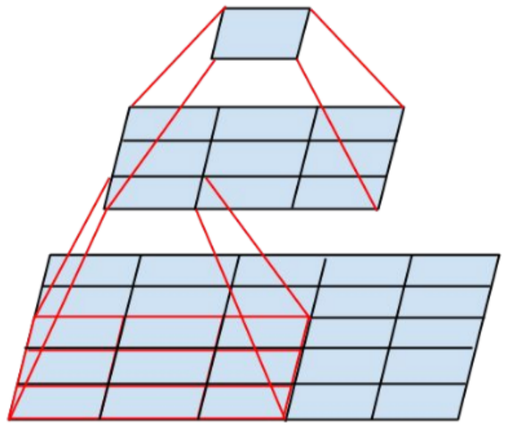


Figure 1. Mini-network replacing the 5×5 convolutions.

Stacking $2(3 \times 3)$ conv

n input filters,αn output filters :

- $(\alpha n, 5 \times 5)$ conv : $25\alpha n^2$ params
- $(\sqrt{\alpha} n, 3 \times 3)$- $(\alpha n, 3 \times 3)$: $9\sqrt{\alpha} n^2 + 9\sqrt{\alpha}\alpha n^2$ params;

$\alpha = 2 \Rightarrow -24\%$

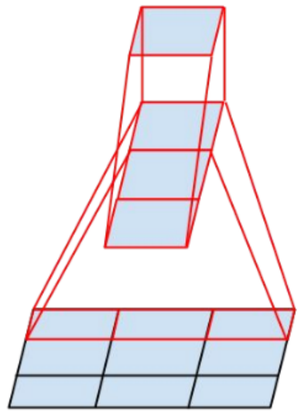


Figure 3. Mini-network replacing the 3×3 convolutions. The lower layer of this network consists of a 3×1 convolution with 3 output units.

1×3 and 3×1 conv

n input filters,αn output filters :

- $(\alpha n, 3 \times 3)$ conv : $9\alpha n^2$ params
- $(\sqrt{\alpha} n, 1 \times 3)$ - $(\alpha n, 3 \times 1)$: $3\sqrt{\alpha} n^2 + 3\alpha\sqrt{\alpha} n^2$ params

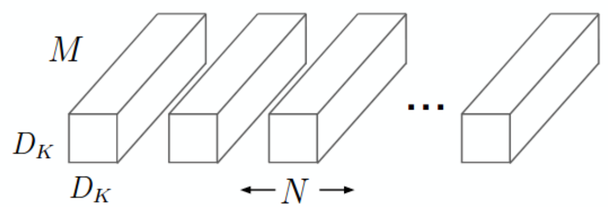
$\alpha = 2 \Rightarrow -30\%$

See also the recent work on “Rethinking Model scaling for convolutional neural networks” (Tan & Le, 2020)

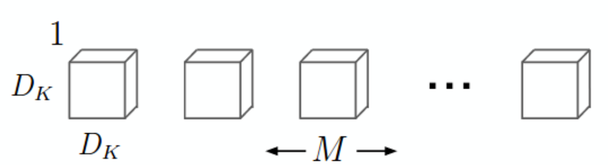
DEPTHWISE SEPARABABLE CONVOLUTIONS

Inception and Xception, Mobilnets. It separates :

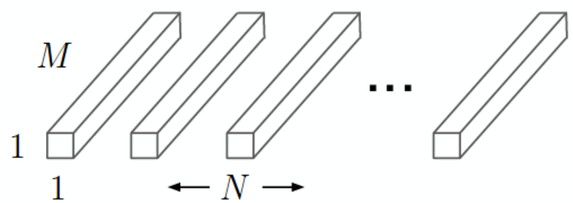
- feature extraction in each channel, in space : depthwise convolution
- feature combination between channels : pointwise convolution 1×1



(a) Standard Convolution Filters



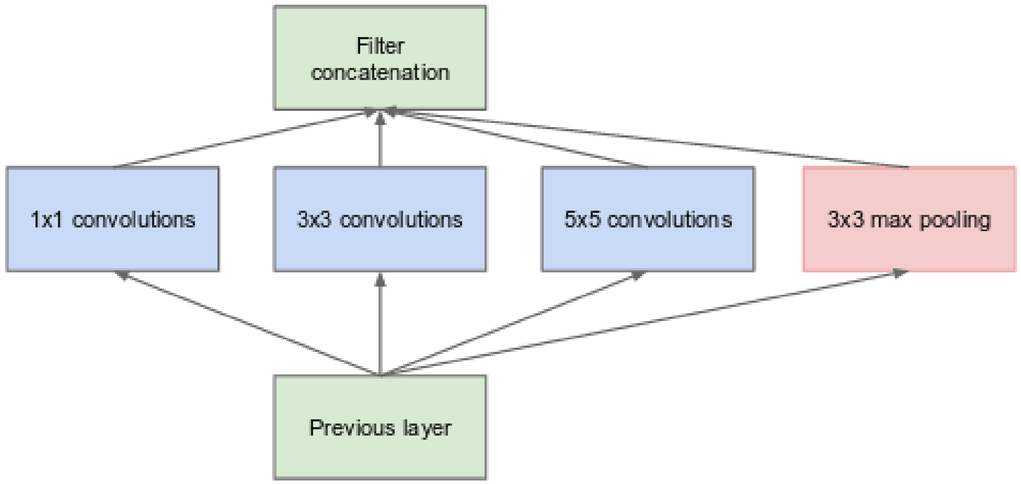
(b) Depthwise Convolutional Filters



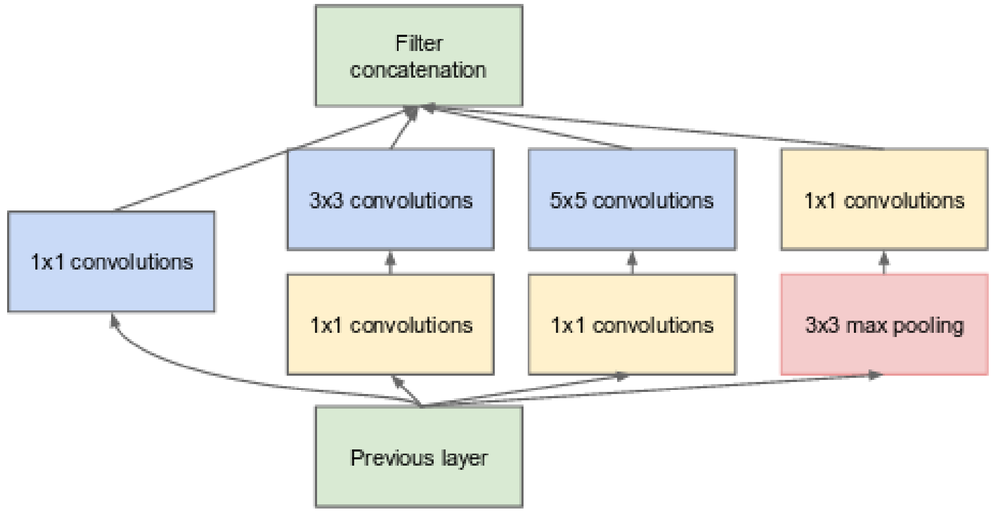
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Depthwise and pointwise convolutions (Howard et al., 2017)

MULTI-SCALE FEATURE EXTRACTION



(a) Inception module, naïve version

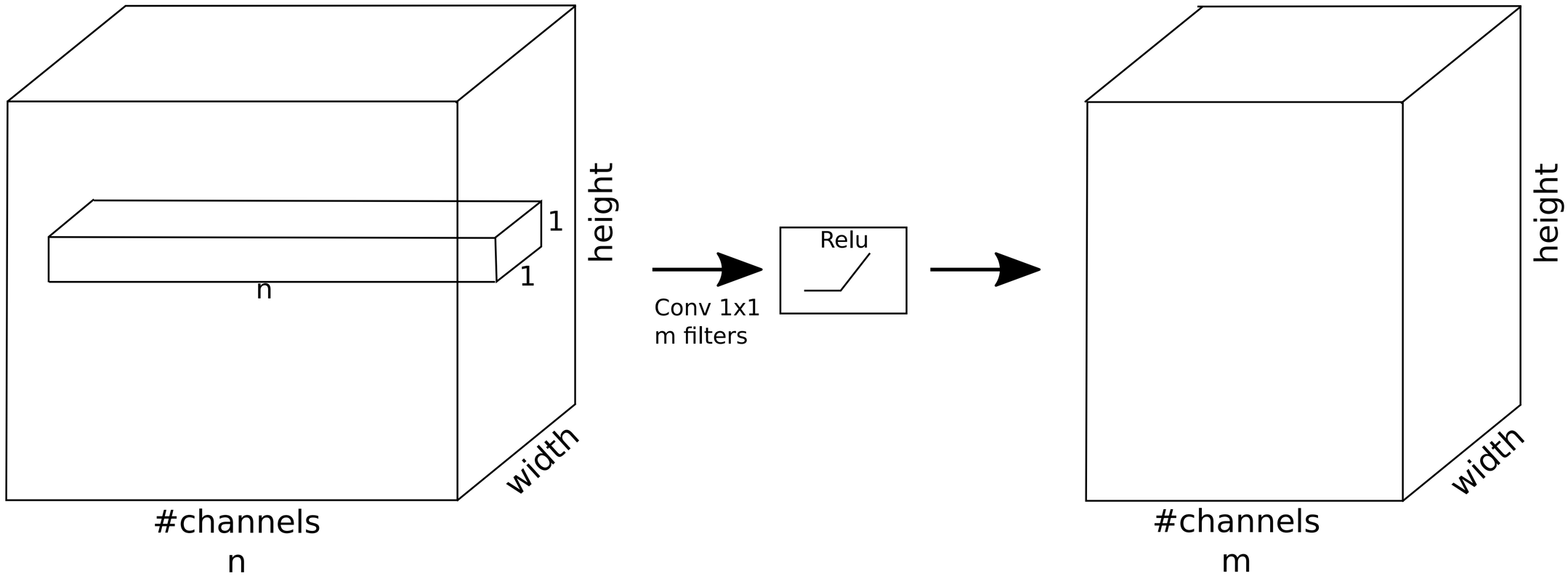


(b) Inception module with dimension reductions

Extract features at multiple scales

See also the Feature Pyramid Networks for multi-scale features.

DIMENSIONALITY REDUCTION



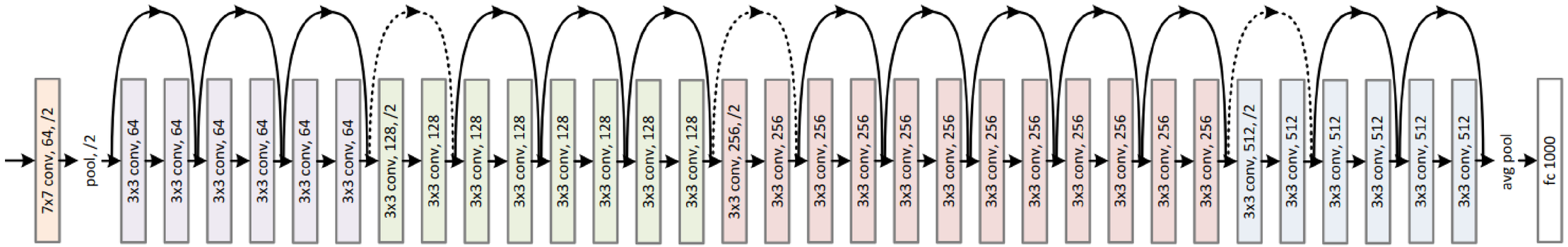
Dimensionality reduction with 1×1 conv

Trainable non-linear transformation of the channels. Network in network (Lin, Chen, & Yan, 2014)

EASING THE GRADIENT FLOW

You can check the norm of the gradient w.r.t. the first layers' parameters to diagnose **vanishing gradients**

- Shortcut connections (e.g. ResNet, DenseNet, Highway)



- auxiliary heads (e.g. GoogleNet)

DO WE NEED MAX POOLING ?

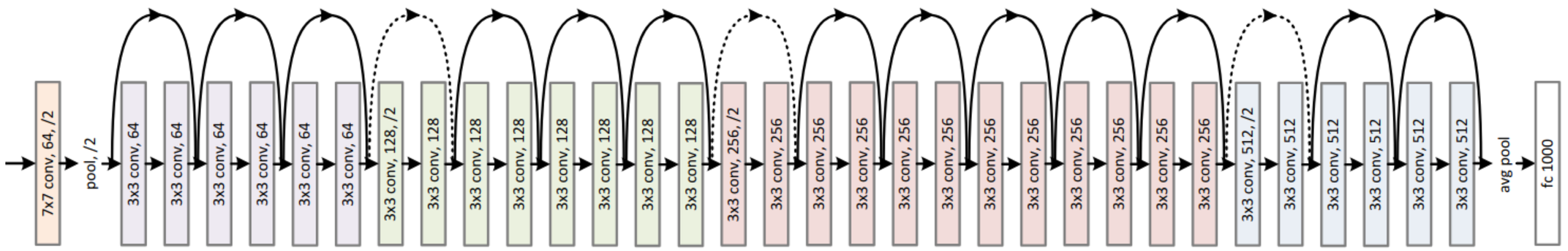
Recent architectures remove the max pooling layers and replace them by conv(stride=2) for downsampling

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

MobileNetv2 (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018). Bottleneck used also in EfficientNet(2019)

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

Striving for simplicity (Springenberg et al., 2015)



ResNet

MODEL AND WEIGHT AVERAGING

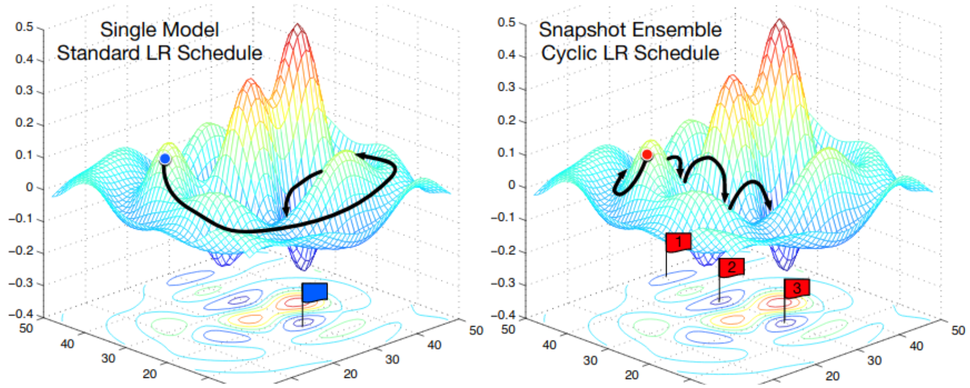
All the competitors in ImageNet do perform model averaging.

Model averaging

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Model averaging performance on ImageNet'12 with multiple models and multiple crops-scale-flips

Weight averaging



Snapshot ensembles (Huang et al., 2017)

If you worry about the increased computational complexity, see knowledge distillation (Hinton, Vinyals, & Dean, 2015).

WE NEED DATA !

USING PRE-TRAINED MODELS

All the frameworks provide you with a **model zoo** of pre-trained networks. E.g. in PyTorch, for image classification. You can cut the head and finetune the softmax only.

```
import torchvision.models as models

resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
mobilenet = models.mobilenet_v2(pretrained=True)
```

Do not forget the input normalization !

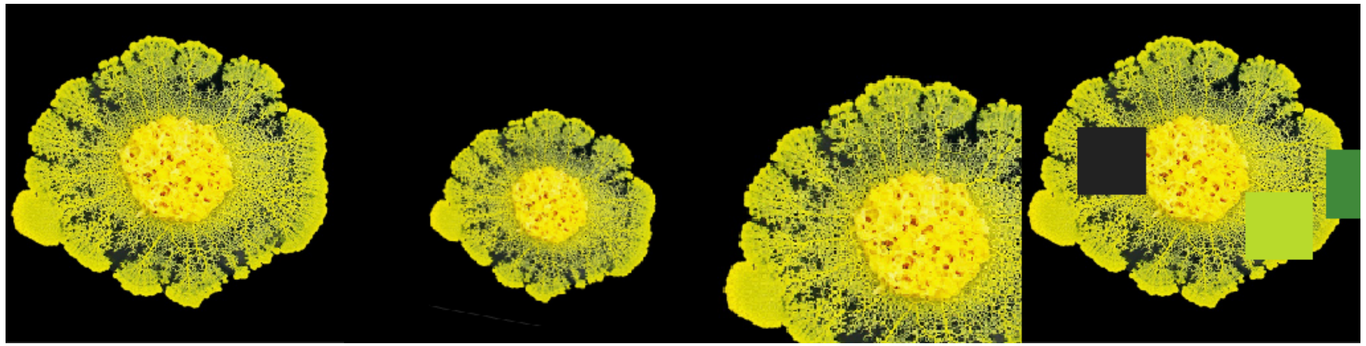
```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

Have a look in the torchvision doc, there are pretrained for classification, detection, segmentation … See also pytorch hub

DATASET AUGMENTATION

You can oversample around your training samples by applying transforms on the inputs that make predictable changes on the targets.

- color jittering, translations, reflections, rotations, PCA, …



Some images generated with imgaug. They are all *physarum polycephalum*, right ? Source image from the CNRS

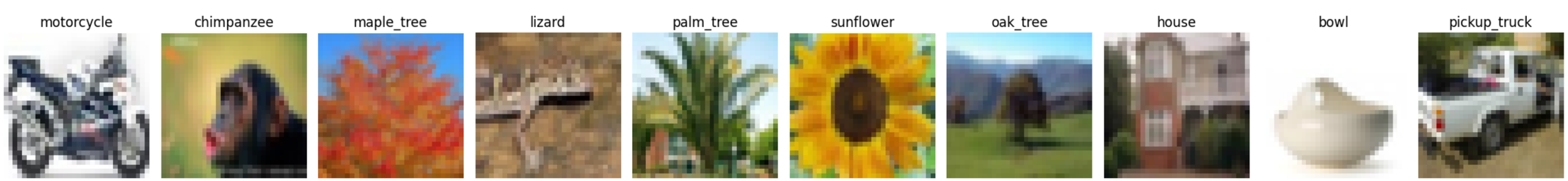
⚠ Your augmentation transforms must be well calibrated !

See also mixup: Beyond empirical risk minimization

EXAMPLE CNN

CIFAR-100 DATASET

- The CIFAR-100 dataset is made of 100 classes with 600 images per class.
- The images are 32×32 RGB



Extract from CIFAR-100

- The training set has 500×100 images, and the test set has 100×100 images.

MODEL ARCHITECTURE AND OPTIMIZATION SETUP

Operator	Resolution	RF size	#Channels
ConvBlock	32×32	5×5	32
ConvBlock	32×32	9×9	32
Sub	16×16	15×15	32
ConvBlock	16×16	15×15	128
ConvBlock	16×16	23×23	128
Sub	8×8	31×31	128
AvgPool	1×1		128
Linear	100		

ConvBlock: 2x [Conv(1x3)-(BN)-Relu-Conv(3x1)-(BN)-Relu]
Sub : Conv(3x3, stride=2)-(BN)-Relu

Number of parameters: $\simeq 2M$
Time per epoch (1080Ti) : 17s. , 42min training time

Common settings :

- BatchSize(32),
- SGD(lrate=0.01) with momentum(0.9)
- learning rate halved every 50 epochs
- validation on 20%, early stopping on the val loss

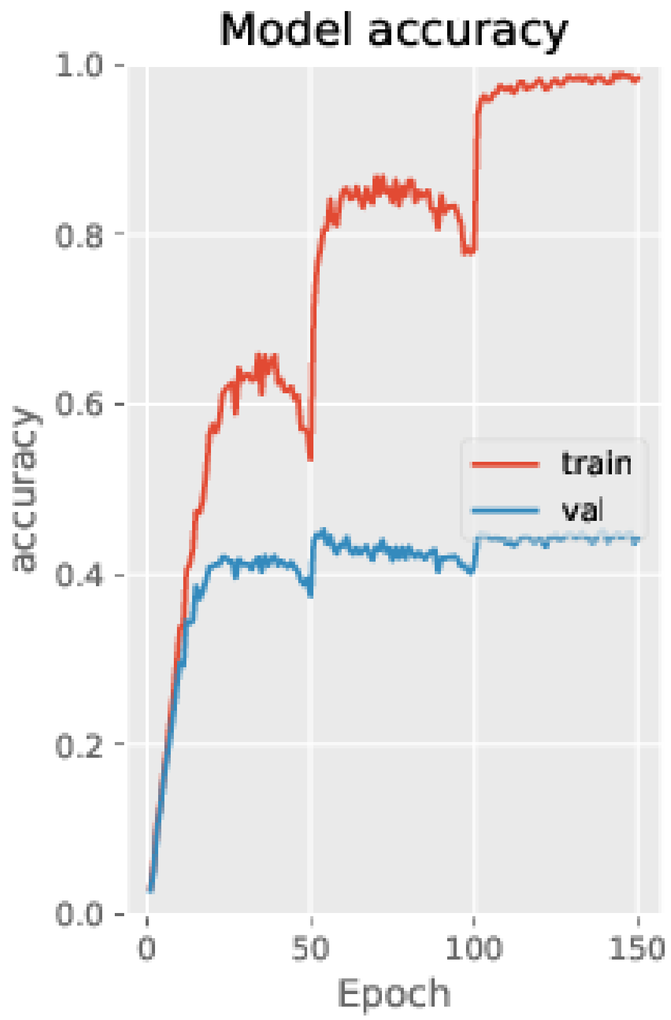
Different configurations :

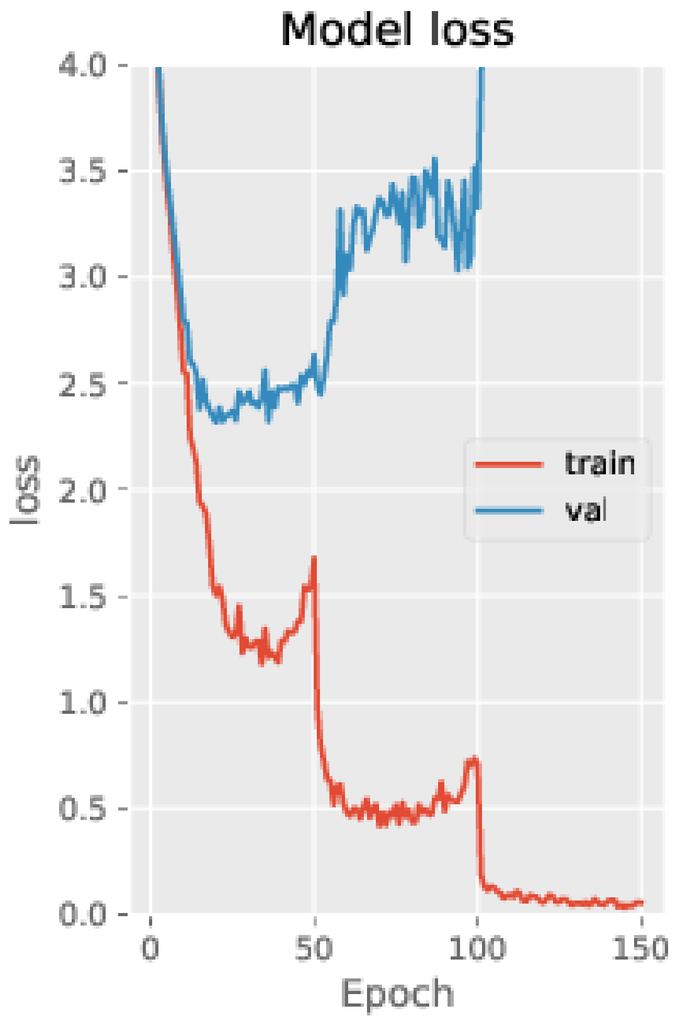
- base
- Conv-BN-Relu or Conv-Relu
- dataset augmentation (HFlip, Trans(5pix), Scale(0.8,1.2)), CenterCrop(32)
- Dropout, L2, label smoothing

If applied, only the weights of the convolution and linear layers are regularized (not the bias, nor the coefficients of the Batch Norm)

BASELINE

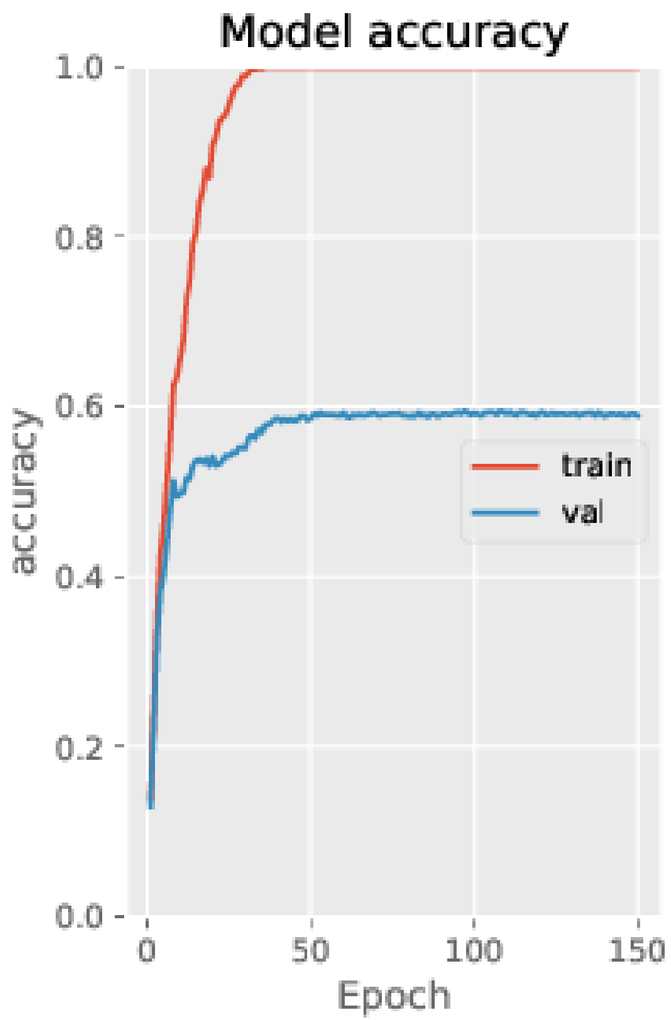
No regularization (either L2, Dropout, Label smoothing, data augmentation), No BatchNorm

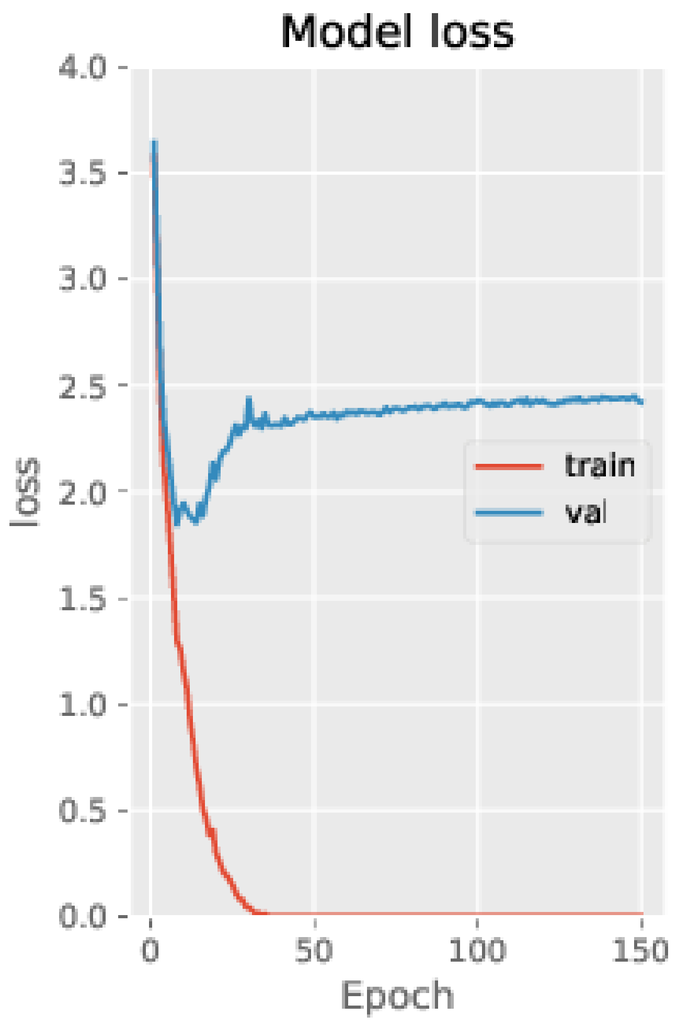




WITH BATCHNORM

With batchnorm after every convolution

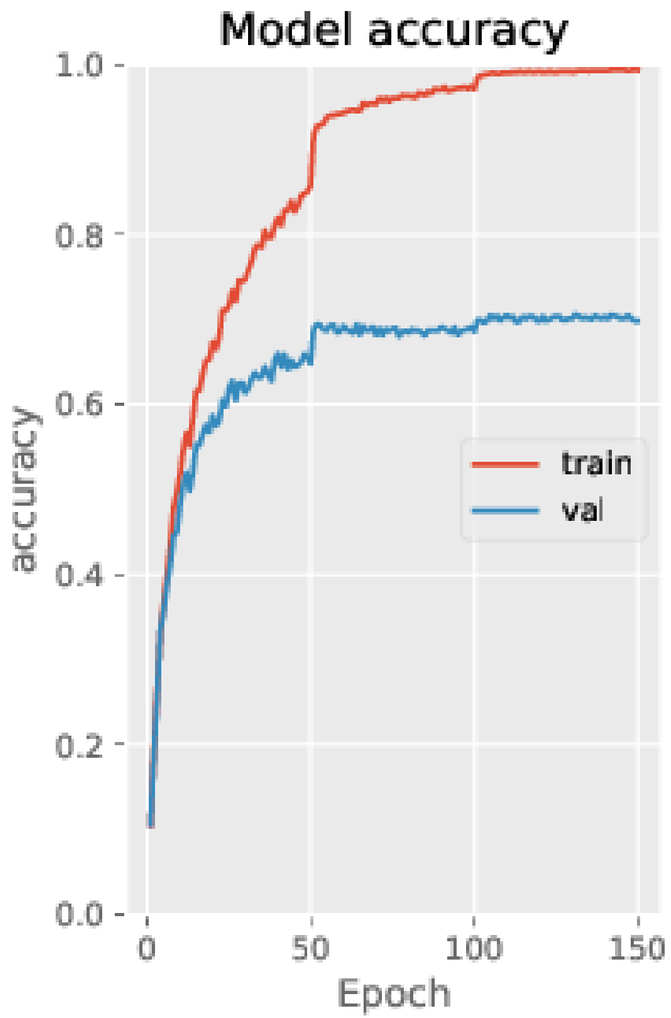


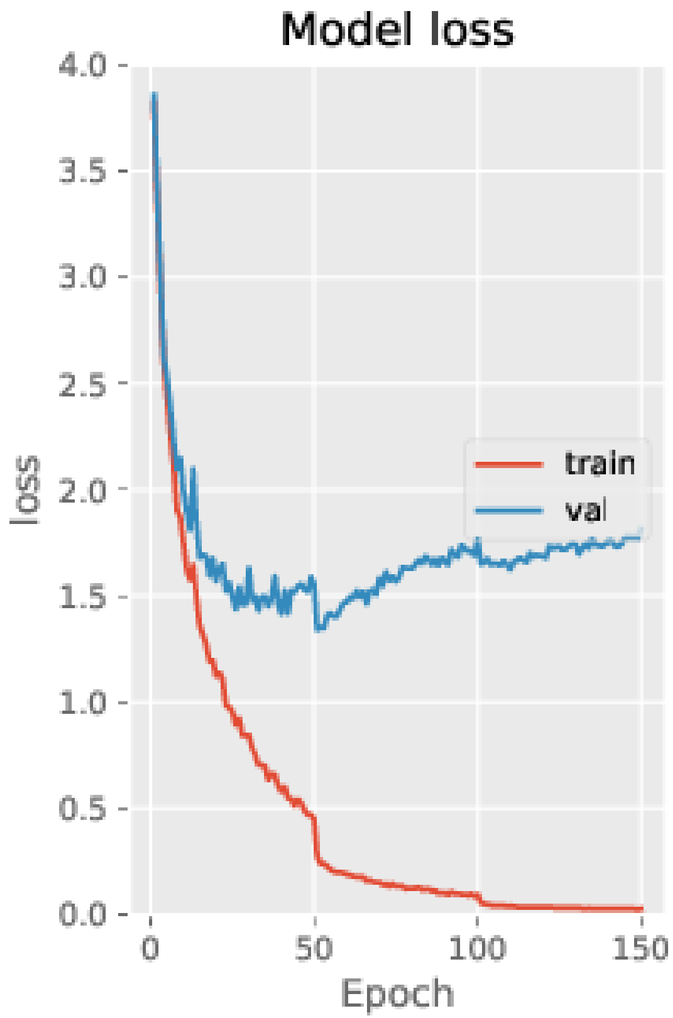


Note it is also regularizing the network.

WITH DATA AUGMENTATION

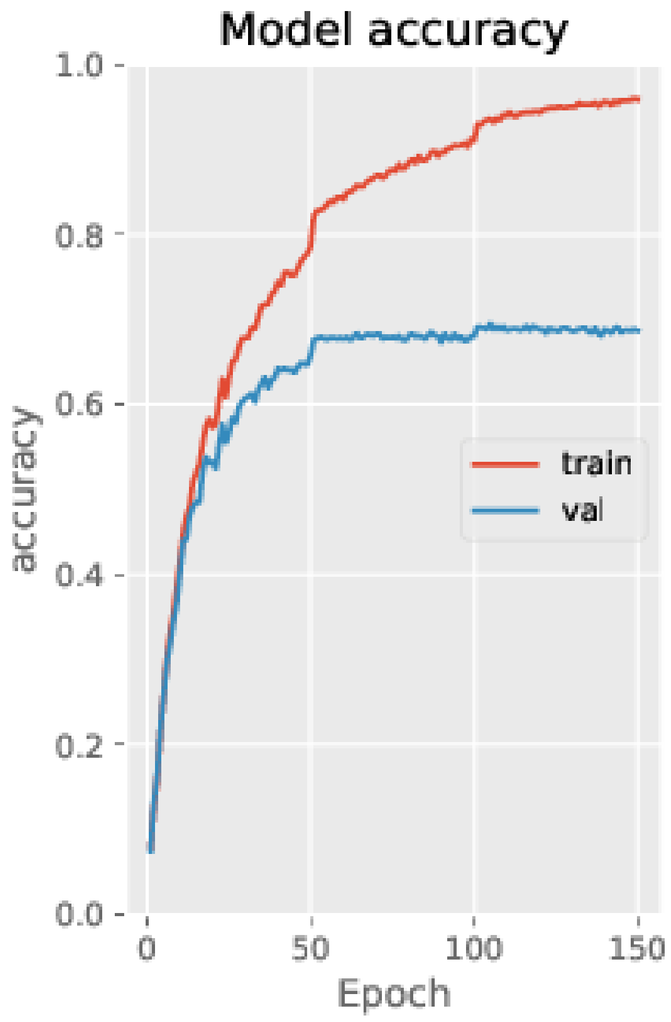
With dataset augmentation (HFlip, Scale, Trans)

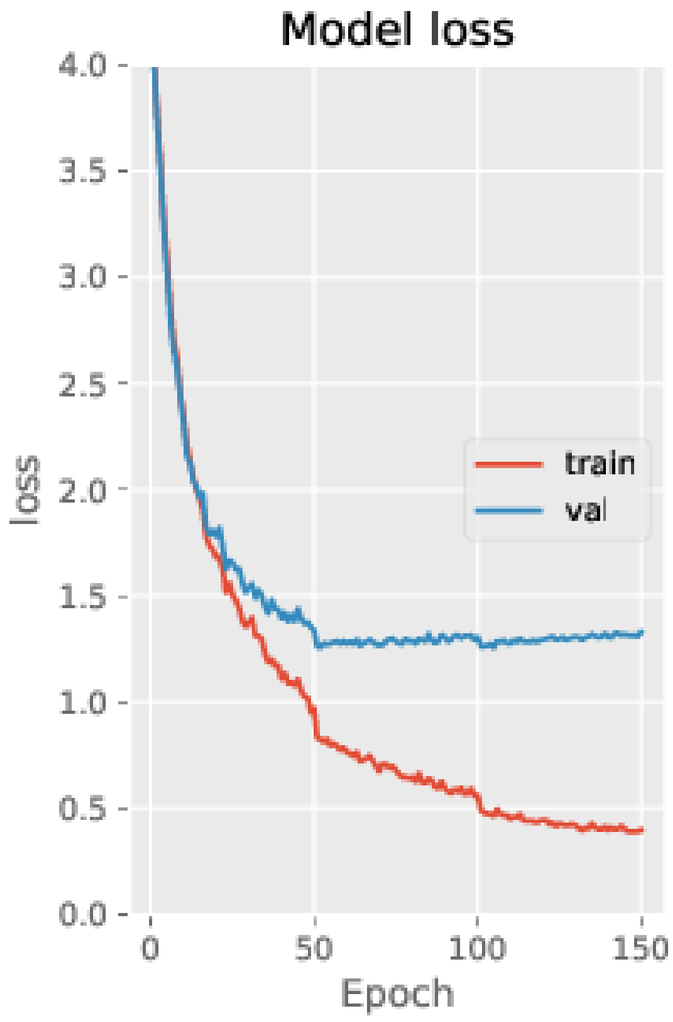




WITH REGULARIZATION

With regularization : L2 (0.0025), Dropout(0.5), Label smoothing(0.1)





BIBLIOGRAPHY

REFERENCES

Rather check the full online document references.pdf

Ciresan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3642–3649). Providence, RI: IEEE. https://doi.org/10.1109/CVPR.2012.6248110

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/BF00344251

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Roceedings of the13thInternational Conferenceon Artificial Intelligence and Statistics (AISTATS) 2010* (p. 8).

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). Las Vegas, NV, USA: IEEE. https://doi.org/10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity Mappings in Deep Residual Networks. *arXiv:1603.05027 [Cs]*. Retrieved from http://arxiv.org/abs/1603.05027

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1503.02531

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [Cs]*. Retrieved from http://arxiv.org/abs/1704.04861

Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., & Weinberger, K. Q. (2017). Snapshot ensembles: Train 1, get m for free. In (p. 14).

Huang, G., Liu, Z., Maaten, L. van der, & Weinberger, K. Q. (2018). Densely Connected Convolutional Networks. *arXiv:1608.06993 [Cs]*. Retrieved from http://arxiv.org/abs/1608.06993

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Lin, M., Chen, Q., & Yan, S. (2014). Network In Network. *arXiv:1312.4400 [Cs]*. Retrieved from http://arxiv.org/abs/1312.4400

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). Salt Lake City, UT: IEEE. https://doi.org/10.1109/CVPR.2018.00474

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [Cs]*. Retrieved from http://arxiv.org/abs/1312.6034

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *arXiv:1409.1556 [cs]*. Retrieved from http://arxiv.org/abs/1409.1556

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [Cs]*. Retrieved from http://arxiv.org/abs/1412.6806

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training Very Deep Networks, 9.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv:1409.4842 [Cs]*. Retrieved from http://arxiv.org/abs/1409.4842

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [Cs]*. Retrieved from http://arxiv.org/abs/1512.00567

Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1905.11946

Yu, F., & Koltun, V. (2016). Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [Cs]*. Retrieved from http://arxiv.org/abs/1511.07122

Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (pp. 818–833). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-10590-1_53