Artificial Intelligence II: Deep learning methods

Ana Neacșu & Vlad Vasilescu

Lecture 4: Convolutional Neural Networks

National University of Science and Technology POLITEHNICA Bucharest, Romania
BIOSINF Master Program

March 2024

Overview

1. Introduction
2. 10 years of CNN revolution
3. Design principles of CNNs
4. We need data, but we don't have!
5. Model Debugging

Introduction

Why Convolutions?

- Convolutions can be used to extract features on data that have a spatial structure (locally correlated), e.g. images.
- They can also be used for temporal series that have a structure in time.

Original image

Discrete laplacian

Gaussian blur

Pattern matching

Tracking a Spaceship – A Convolutional Perspective

Scenario: You're tracking a spaceship flying through space, receiving signal readings over time from a sensor.

- The sensor emits a time-varying signal $x(t)$, representing noisy measurements.
- We use a **weighting function** $w(a)$ to average past values — where a represents the age of each measurement.
- This weighted averaging results in a new signal $s(t)$, combining current and historical measurements.

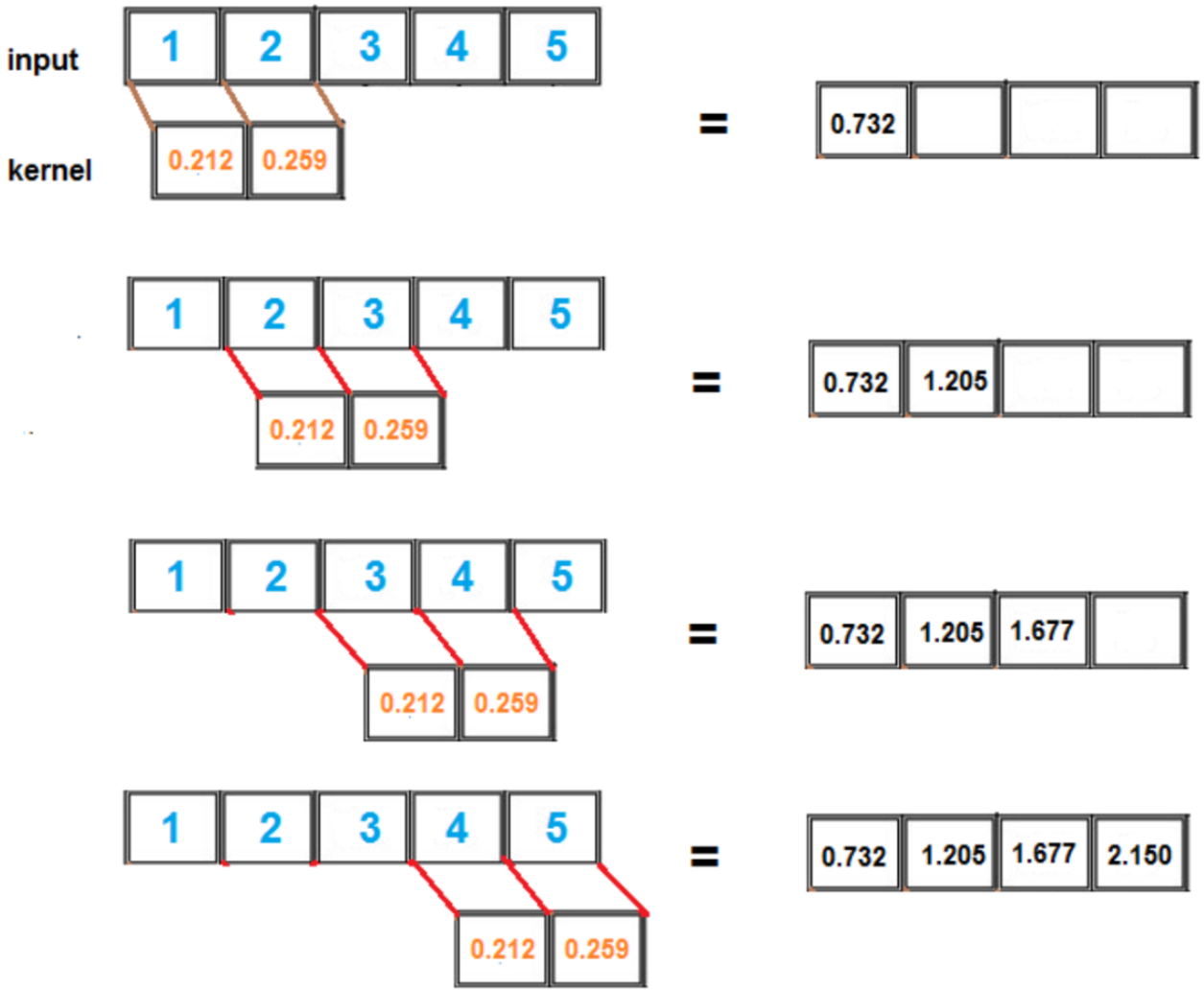
Mathematically:

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

$$s(t) = (x * w)(t)$$

Adapted from [**Goodfellow-et-al-2016**]

Convolution layers – 1D



Convolution vs Cross-Correlation

- **2D convolution:**

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

- **Cross-correlation (as actually used in CNNs):**

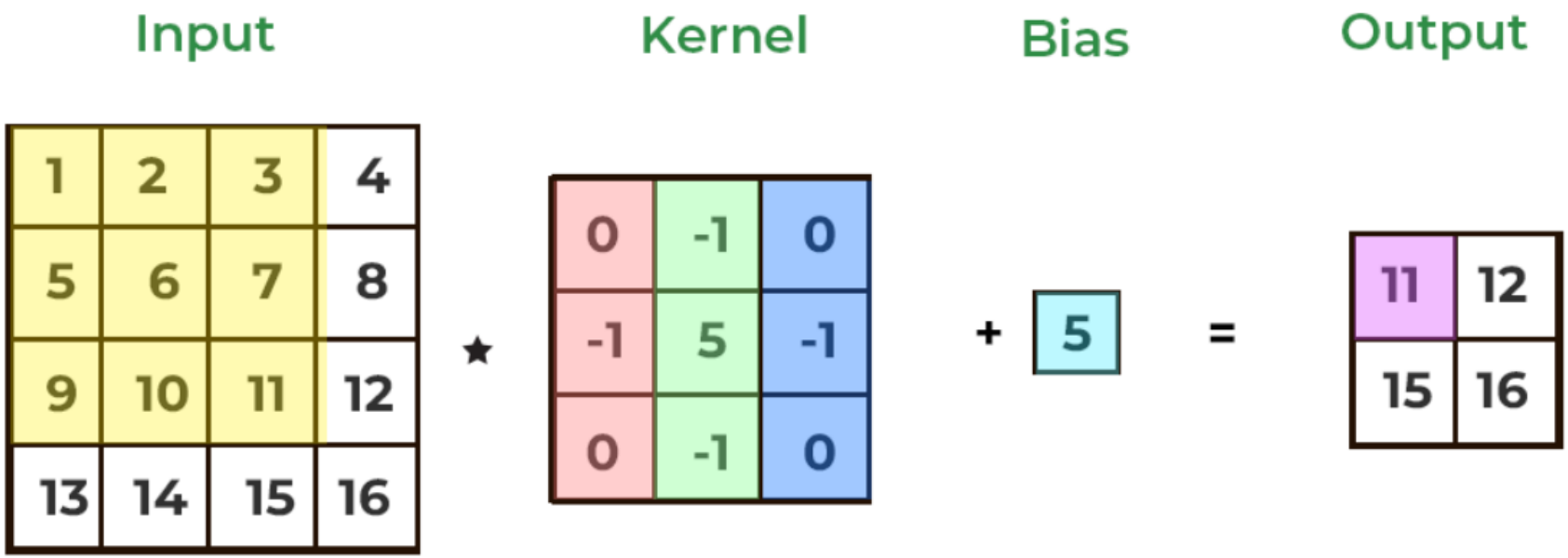
$$S(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

where I is the input tensor and K is the kernel.

- In practice, CNNs compute cross-correlation — not true mathematical convolution

Convolution layers – 2D

- Check this guide for info about how to compute output size.



Convolution as Matrix Multiplication

Convolution can be expressed as matrix multiplication using a structured matrix called a Toeplitz matrix.

- The sliding window operation can be reformulated as a matrix product.
- This is mostly used for mathematical analysis, not for efficient computation.
- The kernel is transformed into a **Toeplitz-like matrix**, which encodes all shifted versions of the kernel.
- As input size grows and kernel size remains fixed, the matrix becomes **very sparse**.
- Input: $[x_1, x_2, x_3, x_4]$
- Kernel: $[k_1, k_2, k_3]$

$$\begin{bmatrix} k_1 & k_2 & k_3 & 0 \\ 0 & k_1 & k_2 & k_3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Convolutional Layers – Sparse and Efficient

Sobel filters applied to a 512×512 image: one for vertical edges (G_x), one for horizontal (G_y).

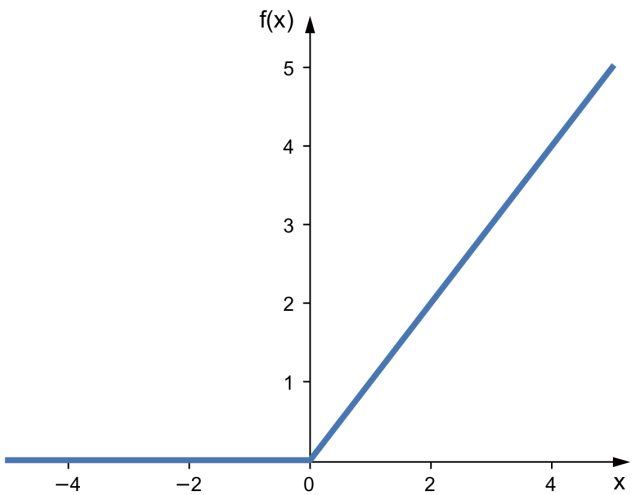
- **Sparse connectivity:** only local regions of the input are connected to each neuron.
- **Parameter sharing:** the same filter is reused across all spatial positions.
- Applying two 3×3 Sobel filters
- A fully connected layer for this task would require:

$$512 \times 512 \times 510 \times 510 \approx \text{millions of weights}$$

Activation Functions

- Activation functions introduce **non-linearity** into the network, allowing it to model complex functions.
- They are applied element-wise after convolutions or fully connected layers.
- A widely used example is the **ReLU (Rectified Linear Unit)**:

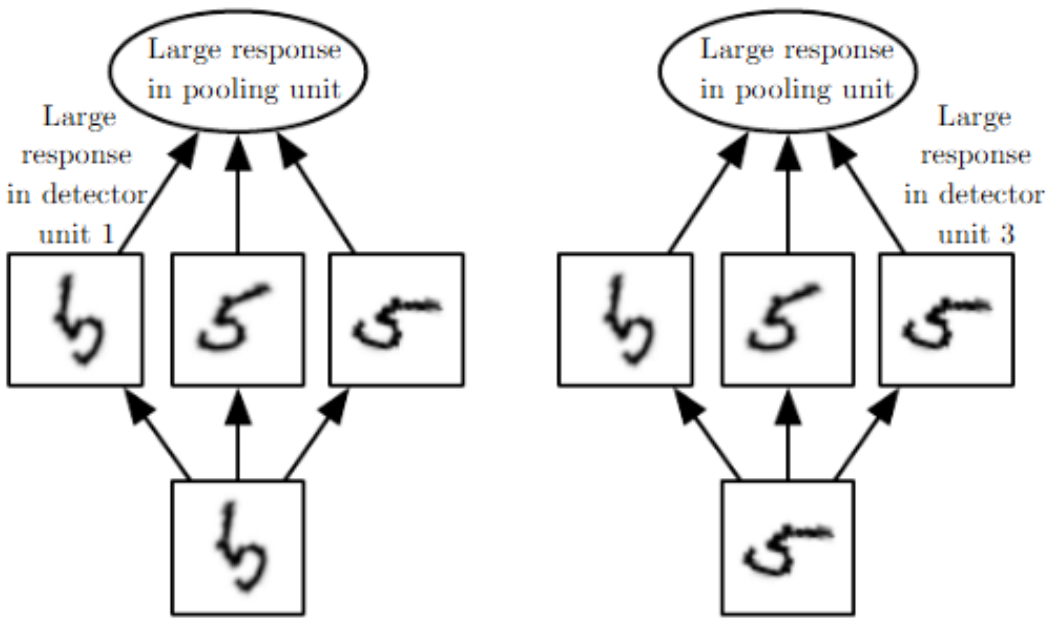
$$\text{ReLU}(x) = \max(0, x)$$



ReLU activation function: zero for negative inputs, linear for positive ones.

Pooling Layers

- Pooling layers perform a form of **sub-sampling**, typically using operations like **MaxPooling** or **AveragePooling**.
- These layers contain **no learnable parameters**.
- Pooling introduces a degree of **spatial invariance**, making the network more robust to small shifts and rotations in the input.
- Key role is spacial reductions



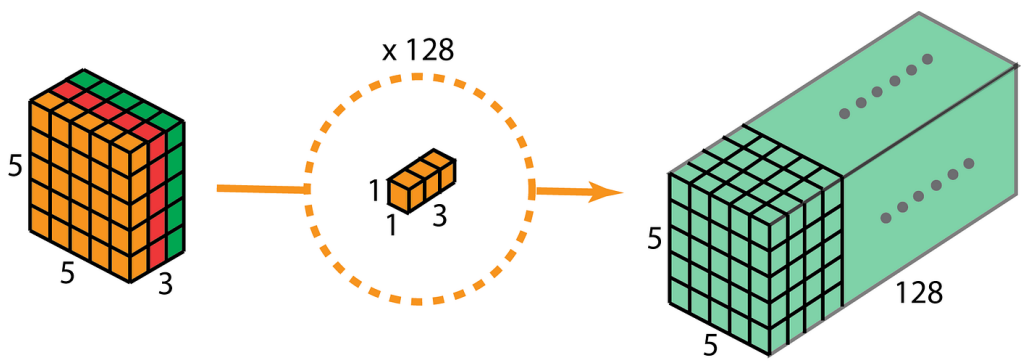
Example of spatial downsampling using max pooling.

Optimization Process

- The kernels (filters) in a convolutional layer are composed of **learnable weights**.
- During training, each weight is updated to reduce the loss using **gradient-based optimization**.
- Gradients are computed via **backpropagation**, which calculates how the loss changes with respect to each parameter.

Kernel Shapes and Output Depth

- The **depth of the kernel** must match the **depth of the input** tensor.
- Each kernel produces **one output channel**;
- More kernels = more learnable filters = ability to capture a wider variety of features.



Example: 3D kernel applied over depth slices of input to produce one output channel.

Learning higher level features

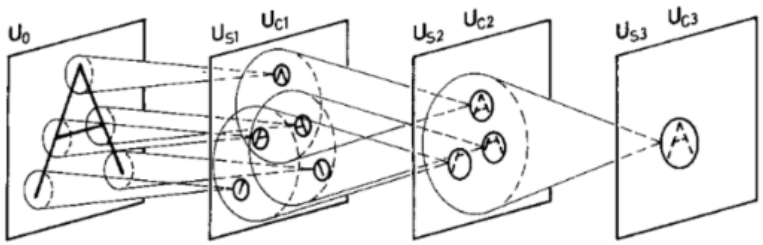
Idea : Local features can be combined to learn higher level features.
Example: Build a house detector

Architectural concepts

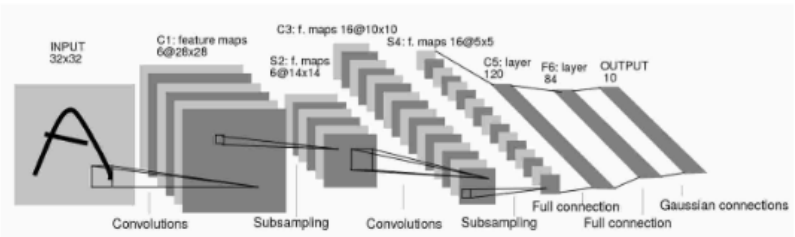
Main Idea : Using the structure of the inputs to limit the number of parameters without limiting the expressiveness of the network.

- 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 to 0
  - a lot of weights shared across positions

$\rightarrow$ strongly regularized!



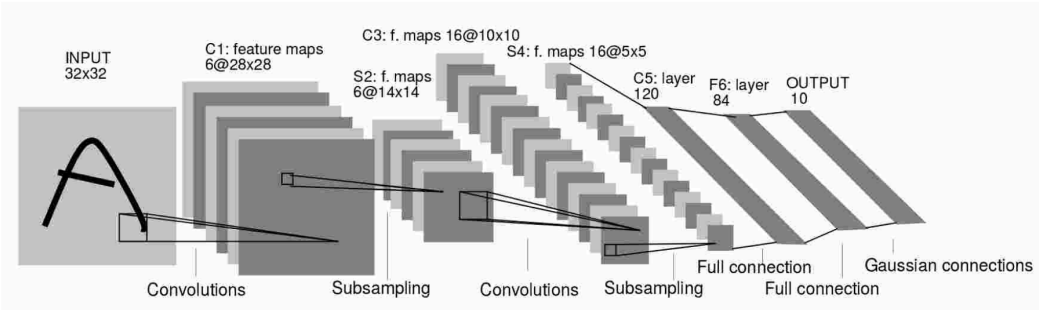
Neocognitron (Fukushima, 1980)



LeNet5 (LeCun et al., 1989)

LeNet5 architecture

- The architecture of LeNet-5, proposed by LeCun in 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.

LeNet5 (LeCun et al., 1989)

Architecture :

Two main parts:

- Convolutional part
- Fully connected part

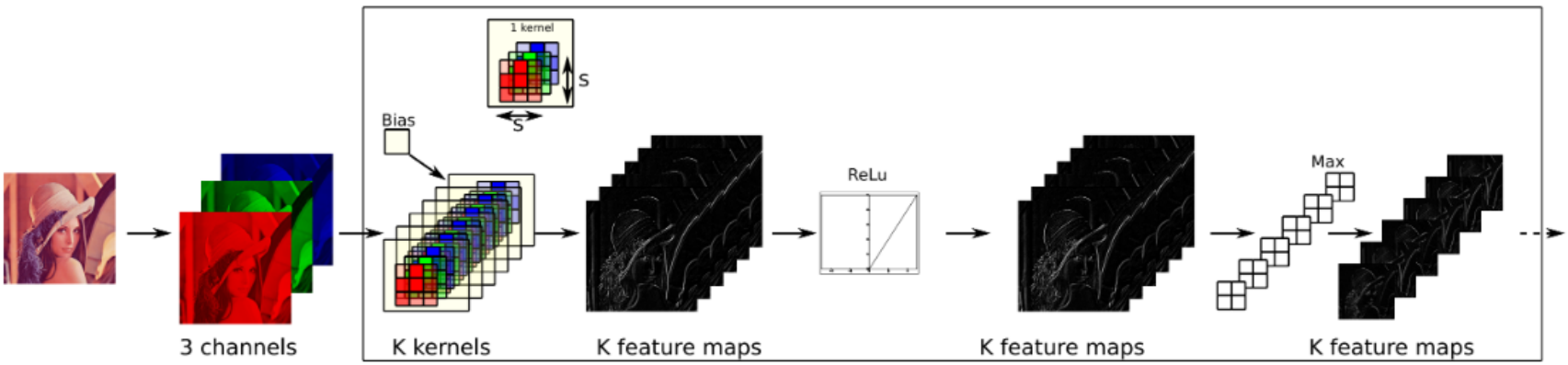
Specificities:

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

Number of parameters :

C1	156
S2	12
C3	1.516
S4	32
C5	48.120
F6	10.164

CNN Definitions



The building blocks of the convolutional part of a vanilla CNN

Convolution:

- size (e.g. 3×3, 5×5)
- 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)

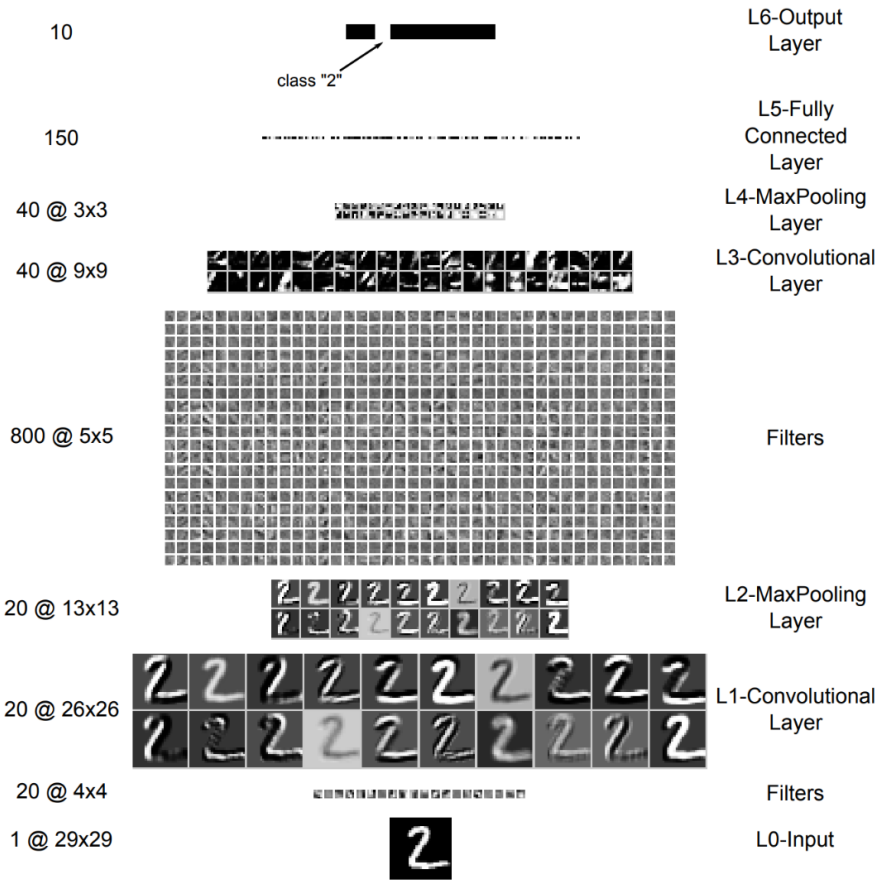
We work with 4D tensors for 2D images, 3D tensors for nD temporal series (e.g. multiple simultaneous recordings), 2D tensors for 1D temporal series.

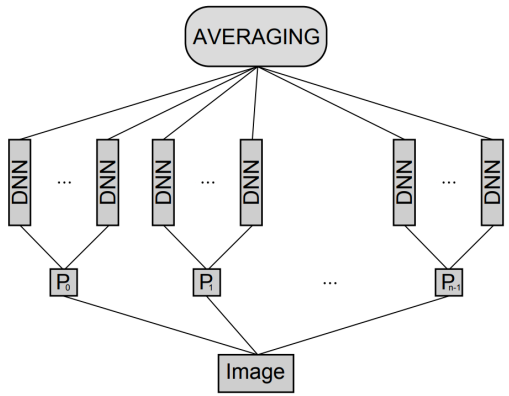
In Pytorch, the tensors follow the Batch-Channel-Height-Width (**BCHW**, channel-first) convention. Other frameworks, like TensorFlow or CNTK, use **BHWC** (channel-last)

10 years of CNN revolution

Multi-Column CDNN

Introduced by Cireașan et. al. in 2012, first ensemble of CNNs trained with dataset augmentation.

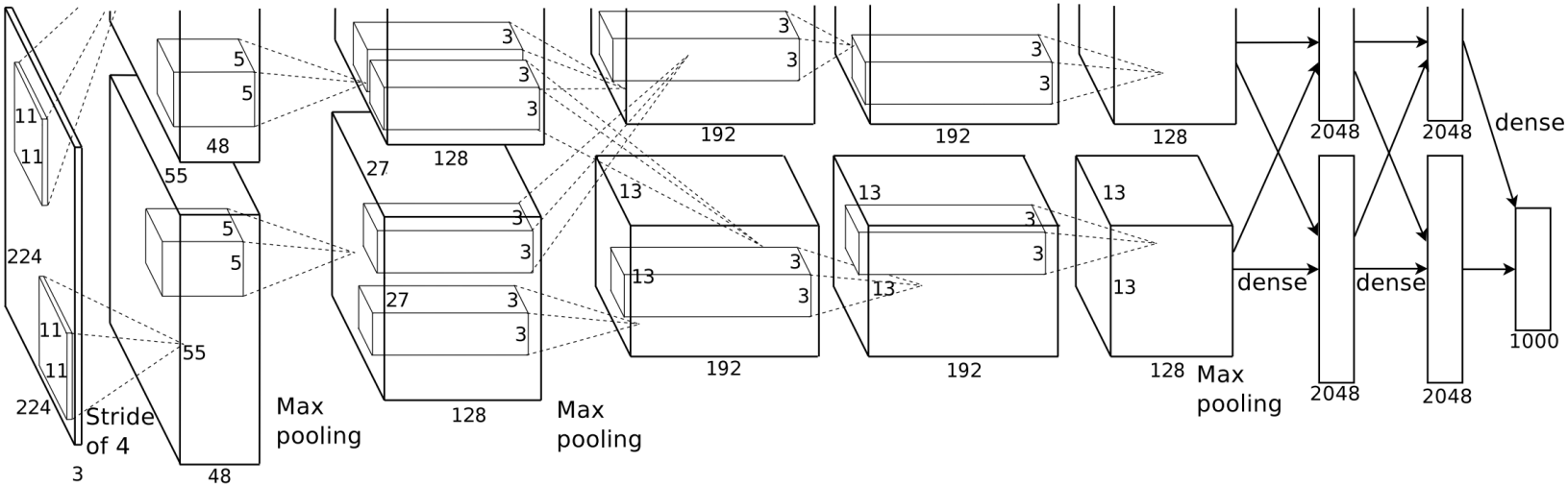




- 1.5 million of parameters/
- 0.23% test misclassification on MNIST.

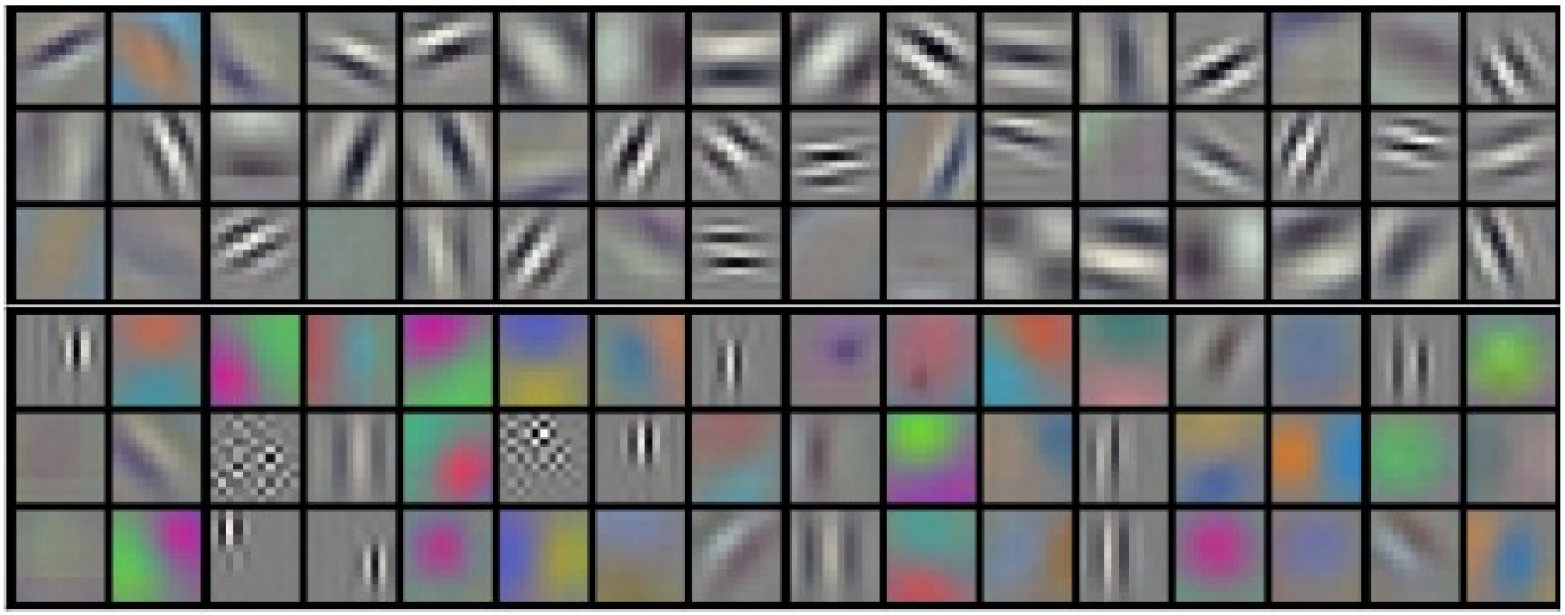
AlexNet – Supervision

Introduced by Krizhevsky et. al. in 2012, considered 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 0.001 at divided by 10 when validation error stalls
- At test time, avg probabilities on crops + reflections

Supervision



Heatmaps showing how filters from the first layer look like.

VGG Net

Introduced by Simonyan and Zisserman in 2015, the winner of ILSVRC'14.

- 16 layers : 13 convolutive, 3 FC
- 3×3 convolutions, 2×2 pooling
- Stacked 3×3 convolutions $\equiv 5 \times 5$ convolution receptive field
  - If $c_{in} = K$, $c_{out} = K$, 5×5 conv. $\rightarrow 25K^2$ parameters
  - If $c_{in} = K$, $c_{out} = K$, 2 stacked 3×3 conv. $\rightarrow 18K^2$ param.
- 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. A random from $\mathcal{N}(0, 10^{-2}, b = 0)$
- can cope with variable input size changing the FC layers to conv 7×7, conv 1×1.

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

Inception Net

Introduced by Szegedy et. al. in 2014, first very deep convolutional model that use 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 inference : polyack average and ensemble of models
- auxiliary heads to mitigate vanishing gradient.

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

Introduced by He et. al. in 2016, the winner of ILSVRC'16.

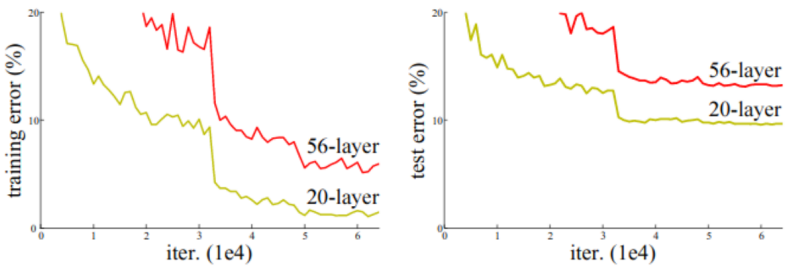
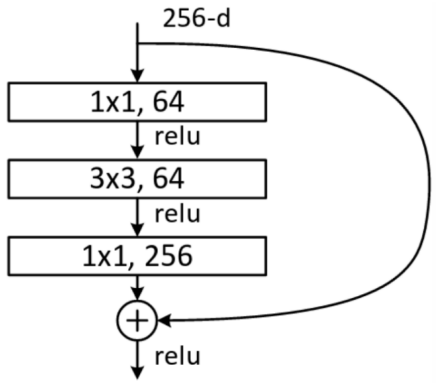


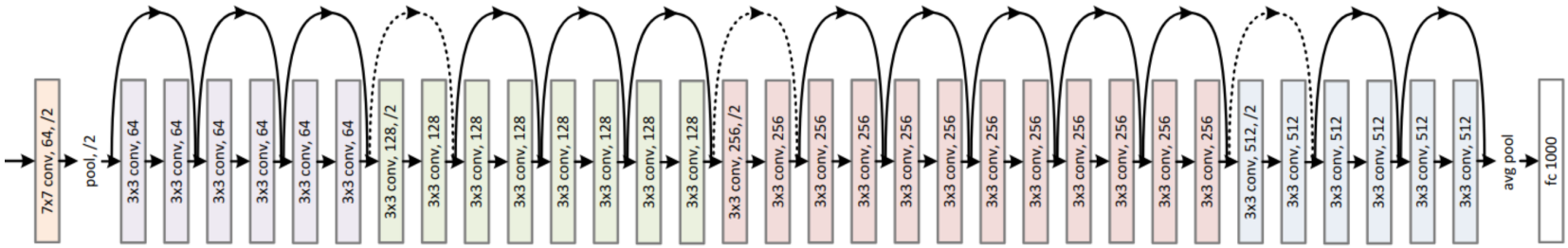
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Deeper is worse ?!



Residual block

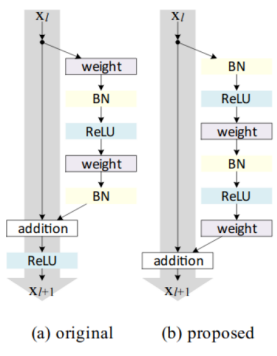
Residual Networks



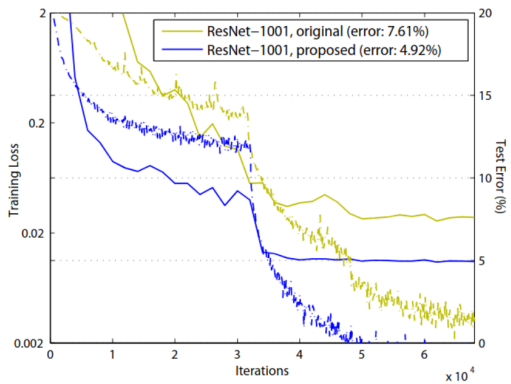
ResNet34. Dotted shortcuts and conv"/2" are stride 2 to match the spatial dimensions. Dotted shortcuts use 1×1 conv to match the depth. 0.46M parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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)

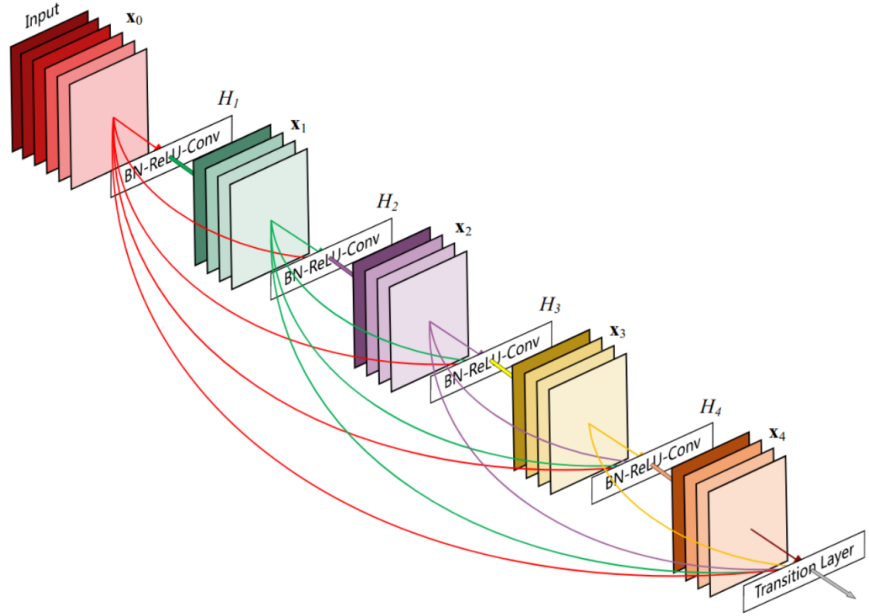
Other variants of skip connections

DenseNets, introduced by Huang et. al. in 2019.

Introduced by Srivastava et. al. in 2015, the so called **Highway Networks.** Uses "Gates" (as in LSTM):

- Transform gate $T(x) = \sigma(W_T x + b_T)$
- Carry gate $C(x) = \sigma(\rho_c(x))$

$$y = T(x) \cdot H(x) + C(x) \cdot x$$



Design principles of CNNs

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

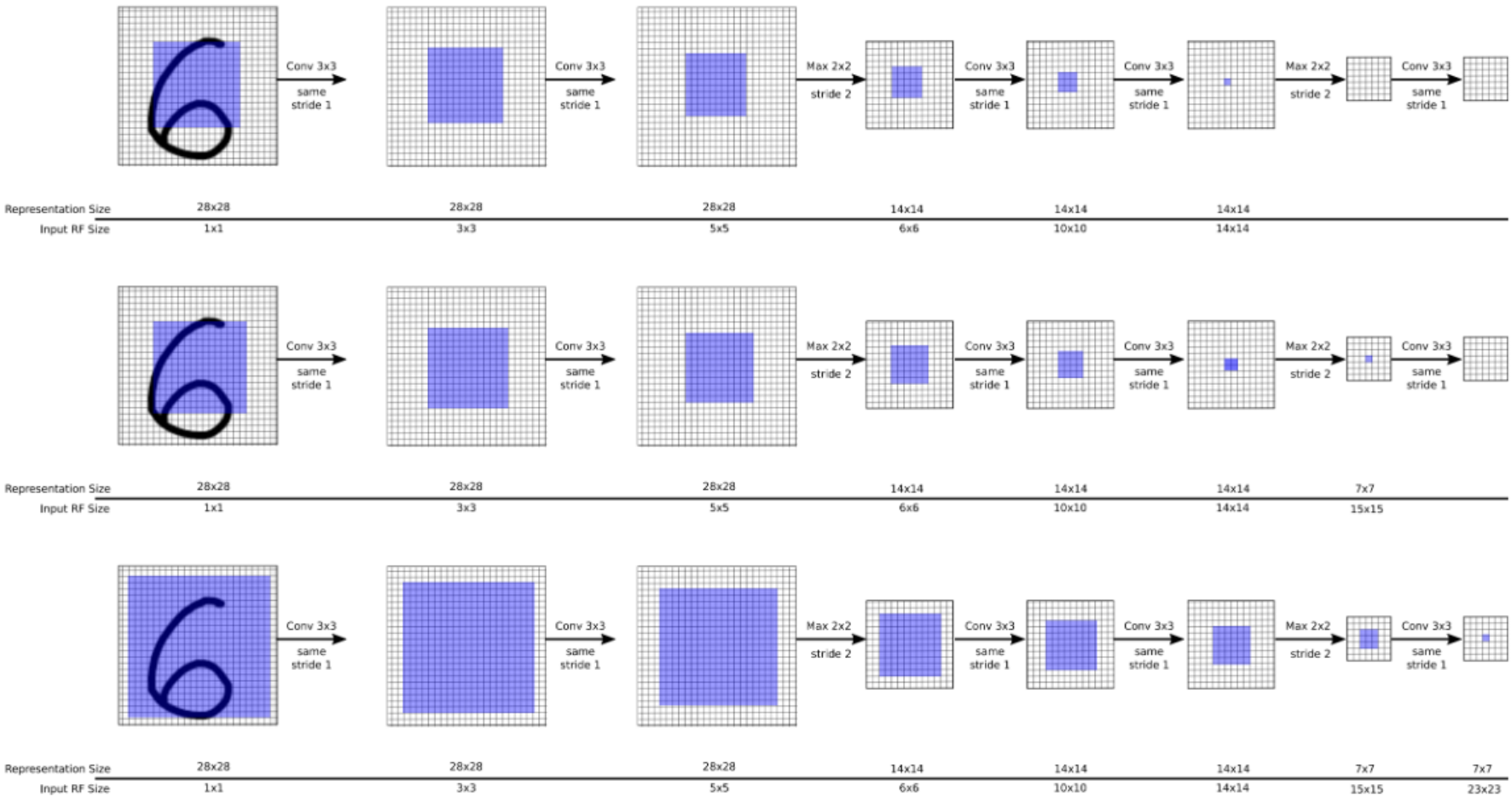
Some examples:

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

Receptive field

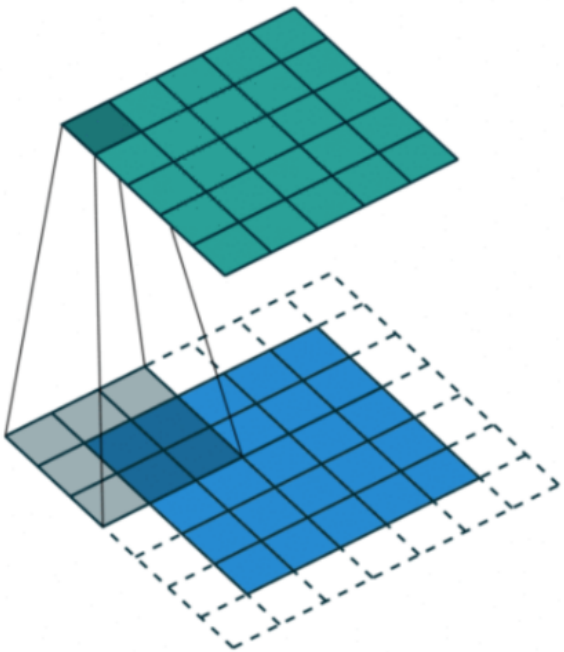
- The receptive field refers to the area of the input image that a particular neuron in the network "sees".
- It represents the region in the input space that contributes to the activation of a given neuron.
- Receptive fields can be local or global, depending on the architecture and layers of the neural network.
- Understanding the receptive field is crucial for interpreting the behavior and features learned by different layers in a convolutional neural network.
- It plays a key role in tasks like object detection, where the network must recognize objects at different scales and positions within an image.
- For calculating the effective receptive field size, see for exeample this calculator.

Effective receptive field

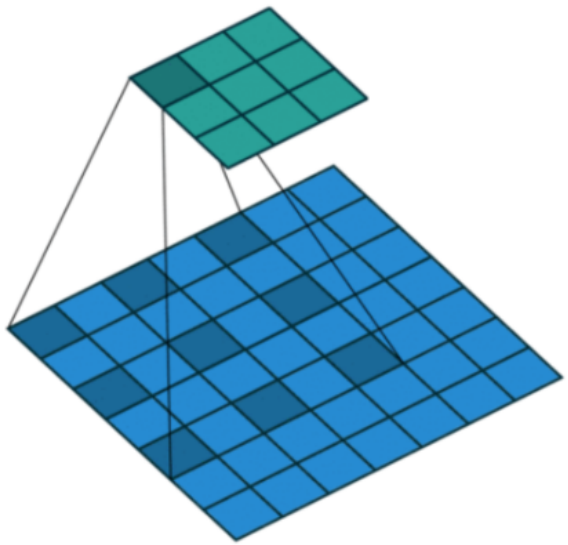


A-TROU Convolutions

- The effective receptive field can grow faster with a-trou convolutions (or dilated convolutions).
- Introduced by Yu and Koltun in 2016.



Conv 3, Pad 1, Stride 1



Conv 3, Pad 0, Stride 1, Dilated 1

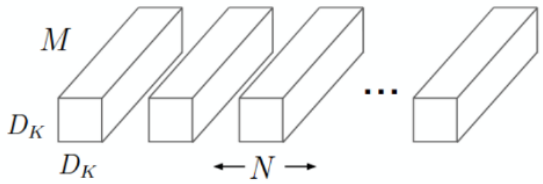
Depthwise separable convolutions

.

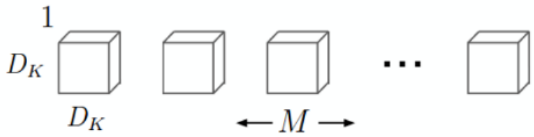
Idea: reduce computational complexity while maintaining performance.

Seaparte:

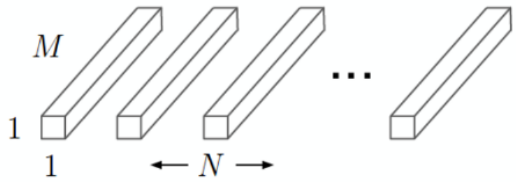
- feature extraction in each channel, in space : depthwise convolution
- feature combination between channels : pointwise convolution 1×1



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



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

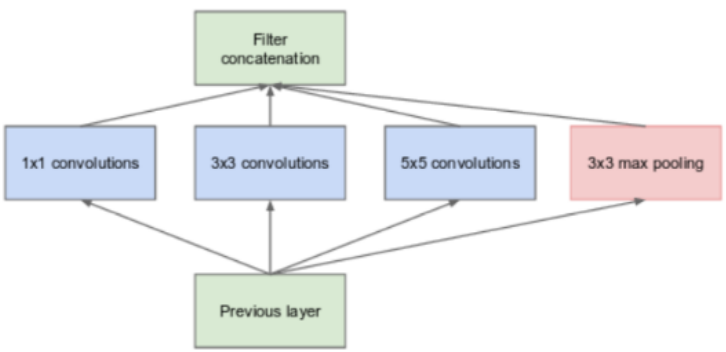
Depthwise and pointwise convolutions (Howard et al., 2017)

Pointwise and Depthwise Convolutions

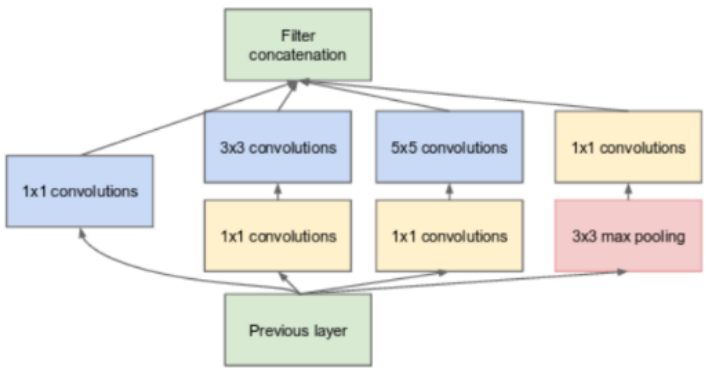
- **Pointwise convolution:** uses $1 \times 1 \times C$ kernels to **adjust the depth** of the input without changing spatial dimensions.
- **Depthwise convolution:** applies separate $H_k \times W_k \times 1$ kernels to each input channel independently.
- These operations can be combined to reduce computational cost while preserving model capacity — a key idea in efficient architectures like MobileNet.

Multi-Scale Feature extraction

Idea: capture information at different scales within an image. It involves analyzing the image at multiple resolutions or levels of detail to extract relevant features.
See Feature Pyramid Network introduced by Lin et. al. for details.



(a) Inception module, naïve version



(b) Inception module with dimension reductions

We need data, but we don't have!

Using pre-trained models

- All modern frameworks provide a **model zoo** of pre-trained networks. See for example torchvision documentation whre you can find pretrained models for classification, detection, segmentation ...
- What is common practice is to cut the head and finetune the softmax only.
- Do not forget about input normalization!

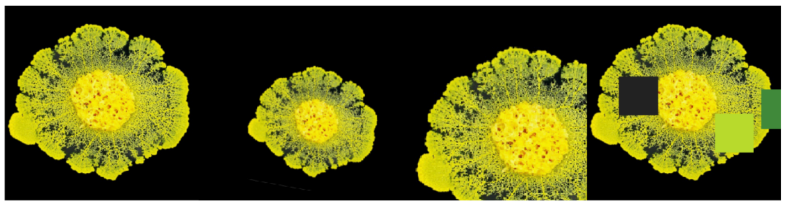
```
import torchvision.models as models

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)
```

Dataset Augmentation

Idea: oversample around the training samples by applying transforms (e.g. color jittering, translations, reflections, rotations, PCA, . . .) on the inputs that make predictable changes on the targets.

Examples

- Use imgaug library



- The images represent all *physarum polycephalum*, and was taken from CNRS
- Another interesting work around this subject is mixup.
- Make sure that the augmentation transforms make sense.

Example on CIFAR-100

- Has 100 object classes, each having 600 images per class.
- Size: $32 \times 32 \times 3$
- The training set has 500×100 images, and the test set has 100×100 images

Model Architecture

Operator	Res.	RF Size	#Ch
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
No. of. param: $\approx 2M$

Settings:

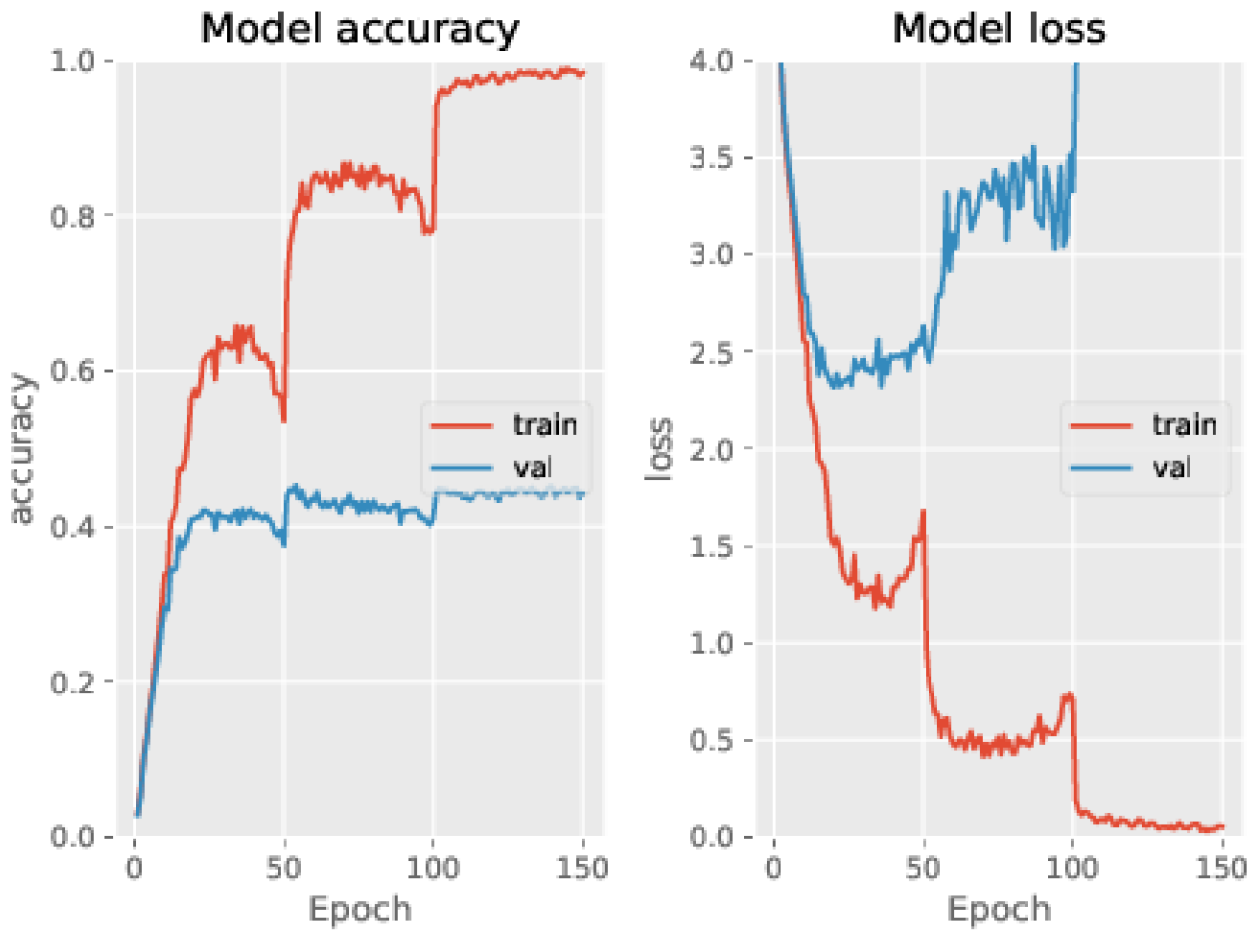
- BatchSize(32)
- SGD(lrate=0.01) with momentum(0.9)
- learning rate halved every 50 epochs
- validation on 20%, early stopping on the val loss

Other configurations:

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

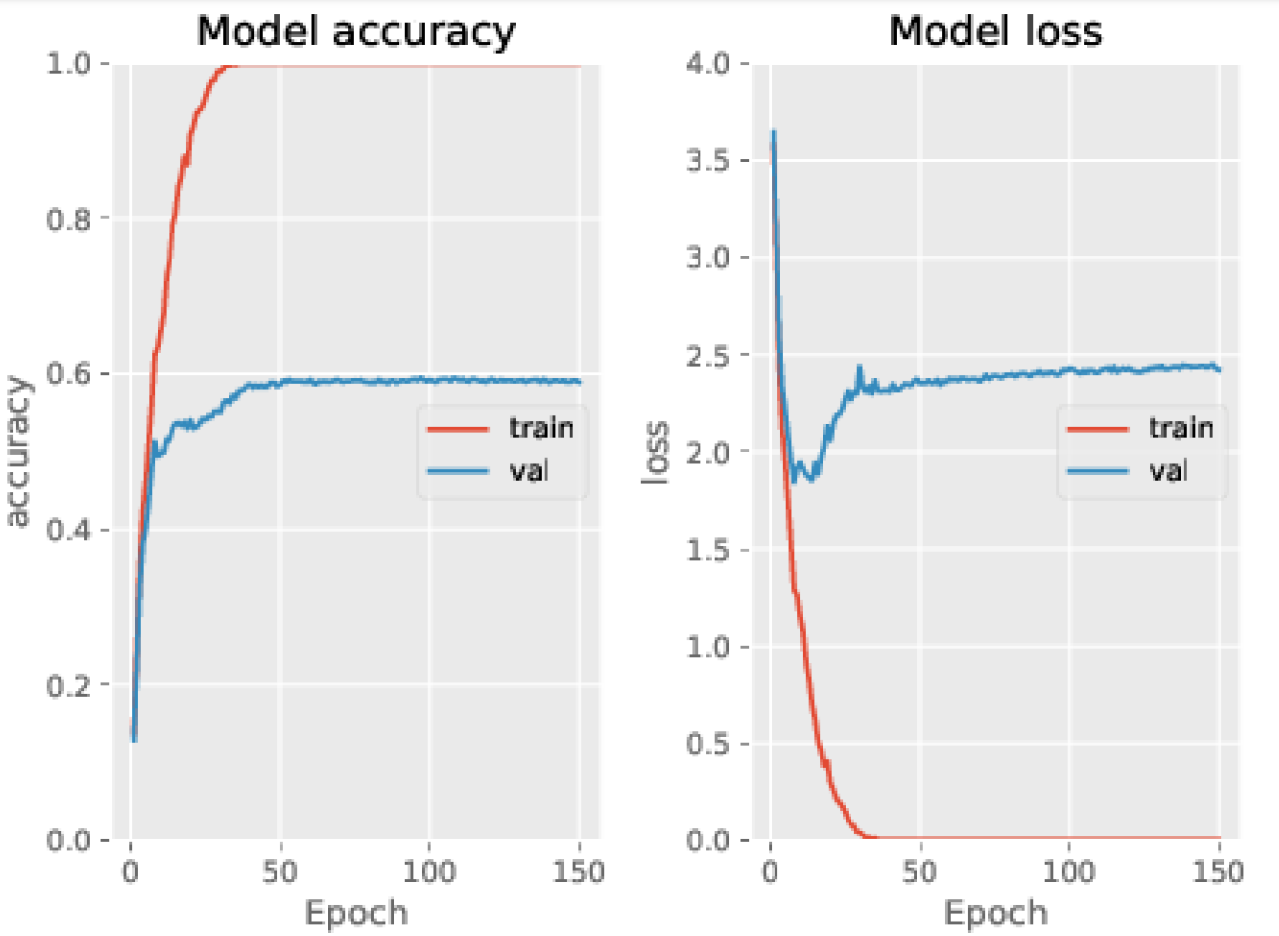
Baseline

- No regularization, no BatchNorm



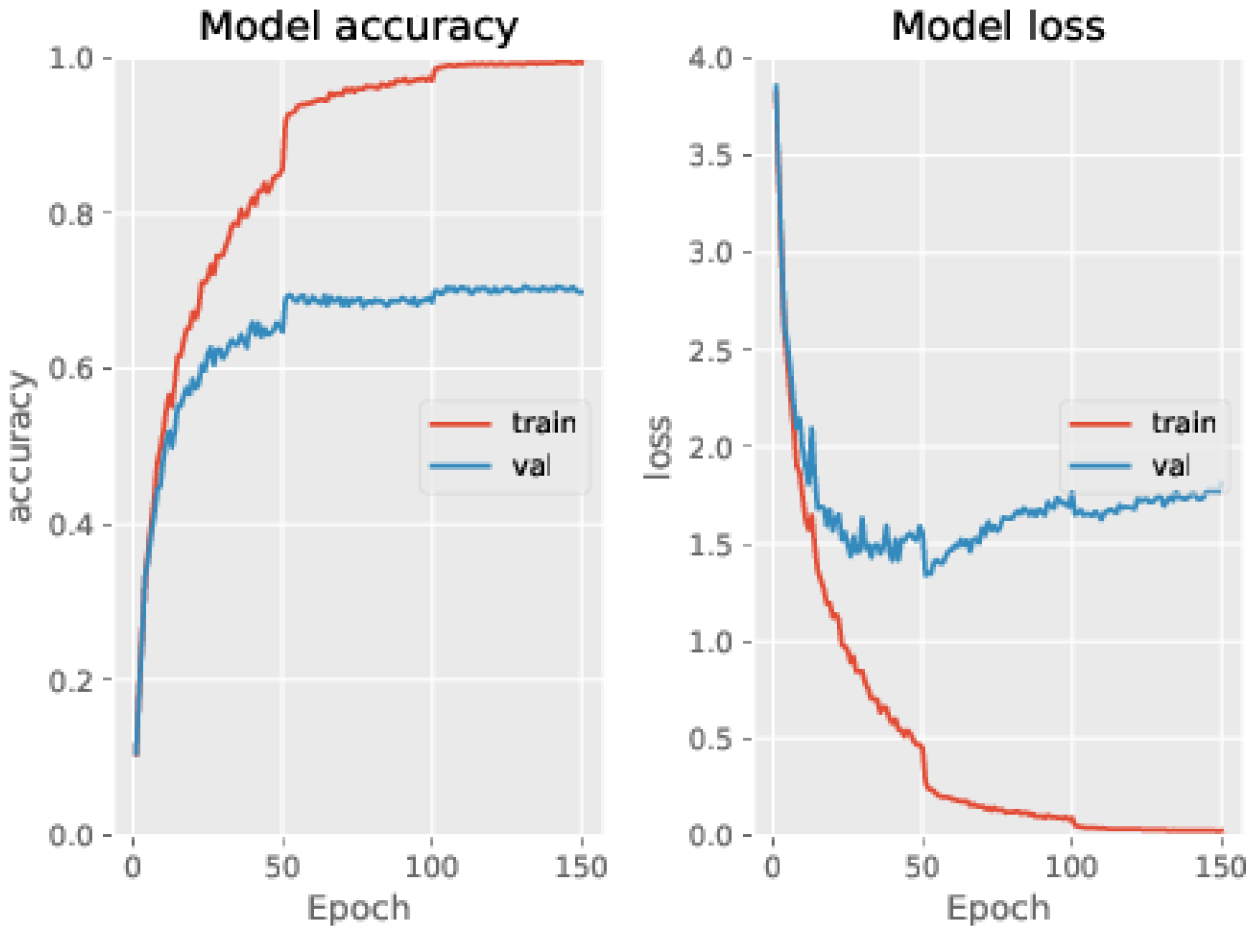
with BatchNorm

- with BatchNorm after each convolution



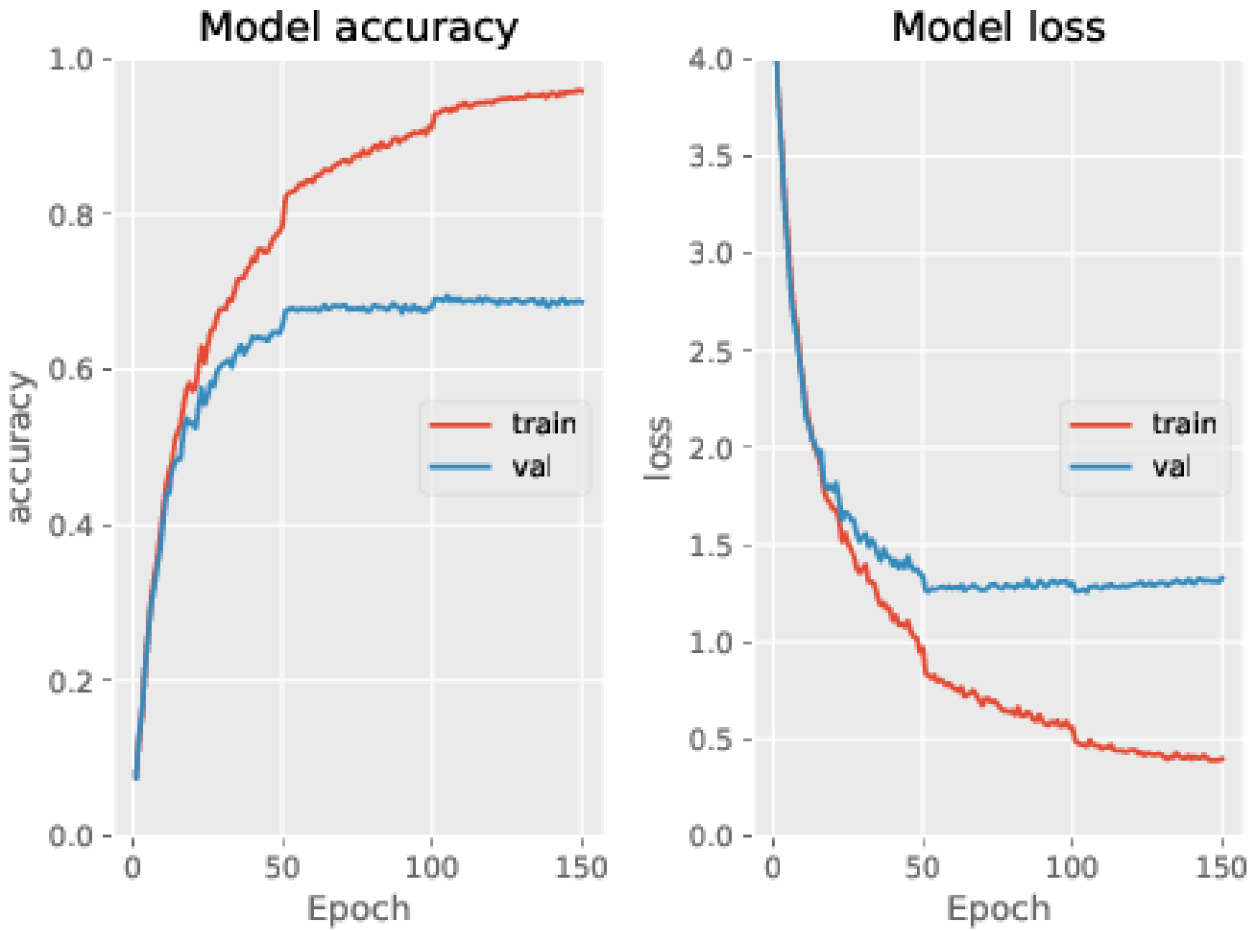
with Augmentation

- with input augmentation: HFlip, Scale, Trans



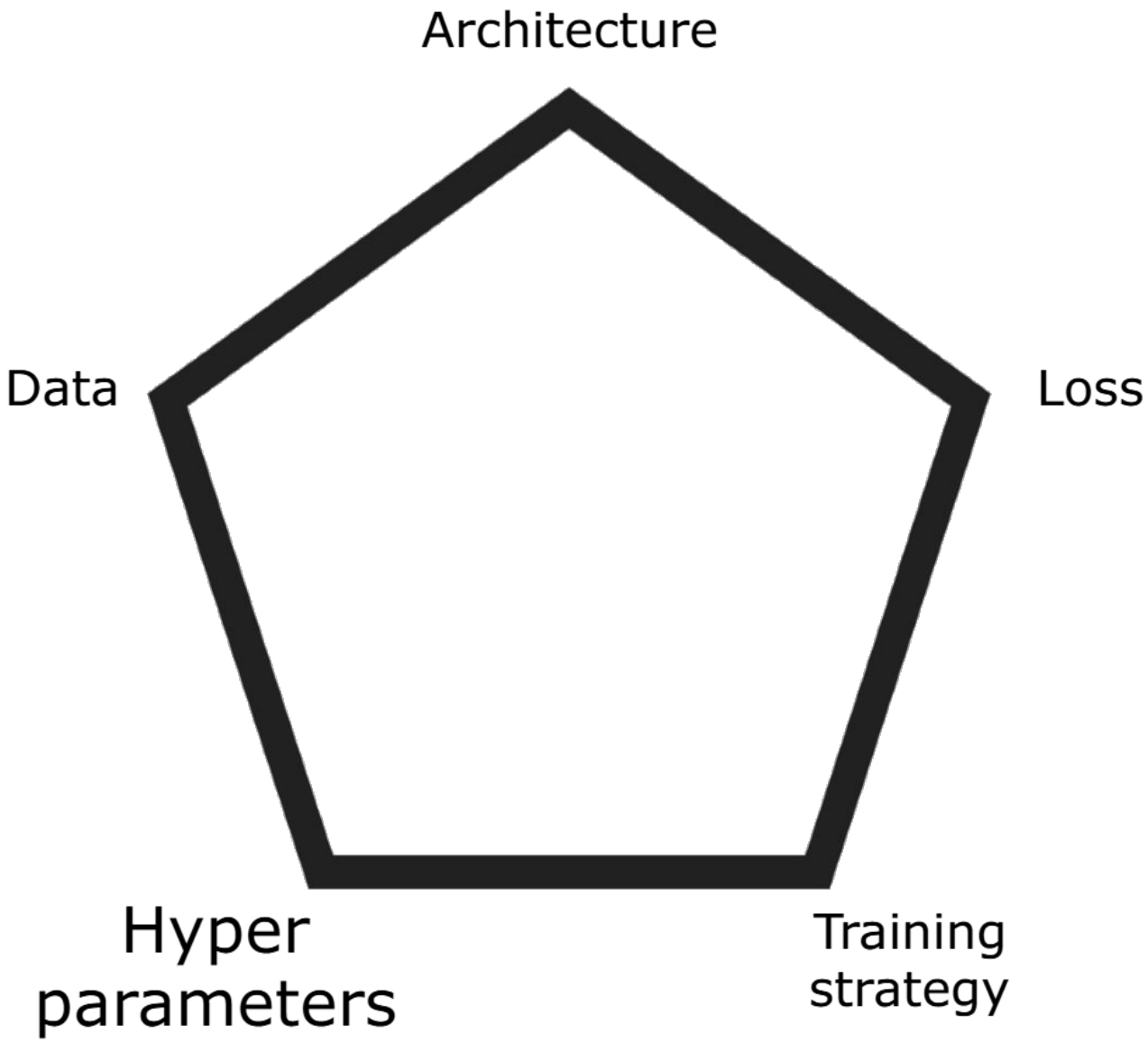
with Regularization

- L2 (0.0025), Dropout(0.5), Label smoothing(0.1)



Model Debugging

Model Debugging



Architecture Design

- **Depth vs Width:** Find the right trade-off between model complexity and performance.
- Too deep? Risk of overfitting. Too shallow? May underfit complex data.
- **Task-specific modules:** Many architectures are built for specific goals:
  - Autoencoders: compression, denoising, generation
  - GANs: high-fidelity image generation
  - UNet: segmentation
- Multi-branch networks help extract diverse features and handle data more flexibly.

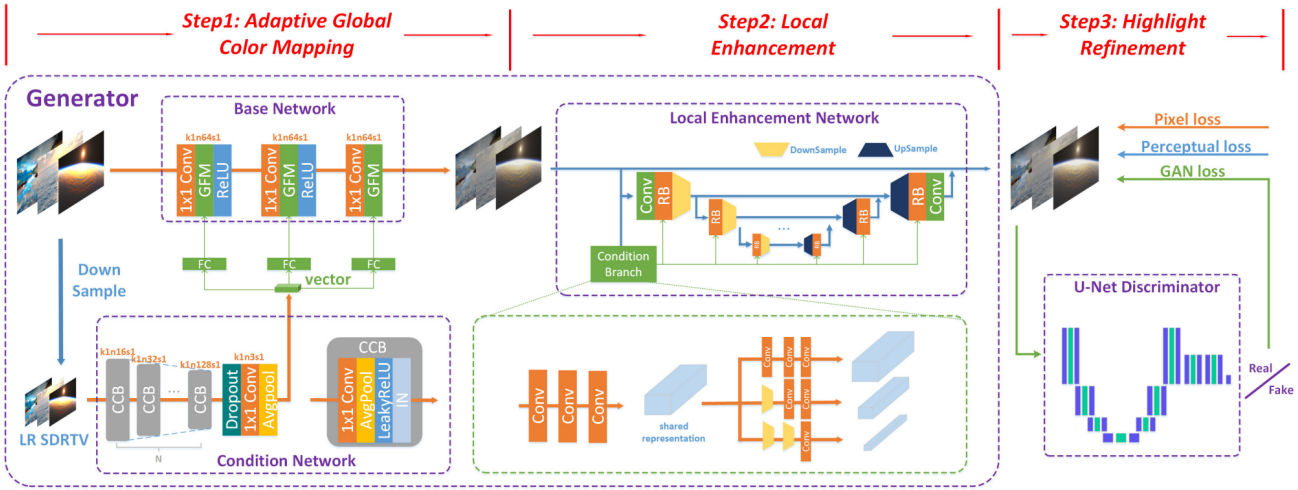
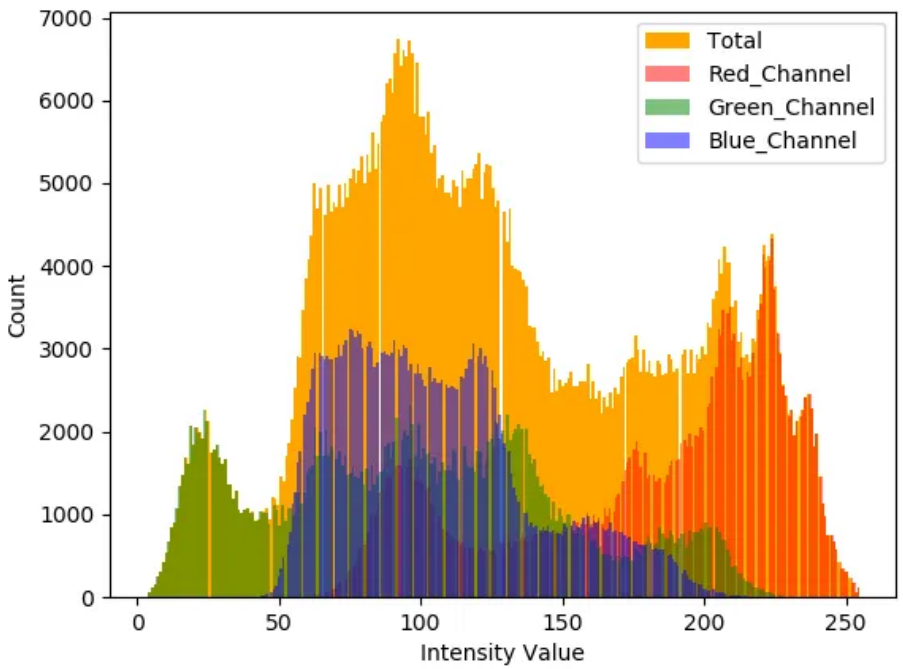


Figura: Visualization from Chen et al., *Towards Efficient SDRTV-to-HDRTV by Learning from Image Formation*, arXiv:2309.04084 (2023).

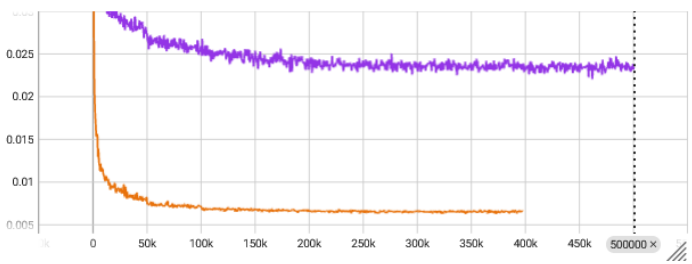
Data Quality and Distribution

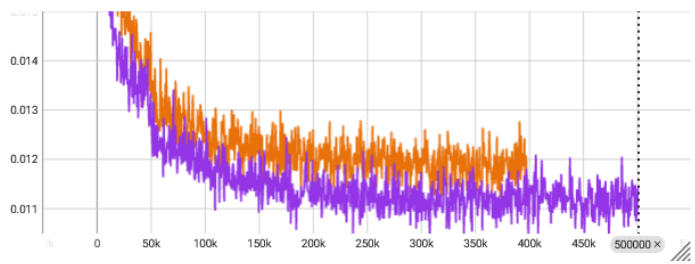
- Data quality is often overlooked — garbage in, garbage out.
- Datasets are split into: train, validation, and test sets.
- **Why both validation and test?** Validation is used during training, test is held out for final evaluation.
- Ideally: all datasets are IID (independent and identically distributed) — often not the case!
- **Histogram analysis** helps uncover distribution mismatches.
- Generalization vs Specialization: open this up for discussion/examples.



Loss Function Design

- The loss function defines the task: classification, segmentation, generation, etc.
- In real projects, the loss is usually a combination of multiple terms.
- **In theory:** each loss term supervises a different aspect of learning.
- **In practice:** one loss can dominate — masking model flaws.
- Plot individual loss components and tune weights accordingly.



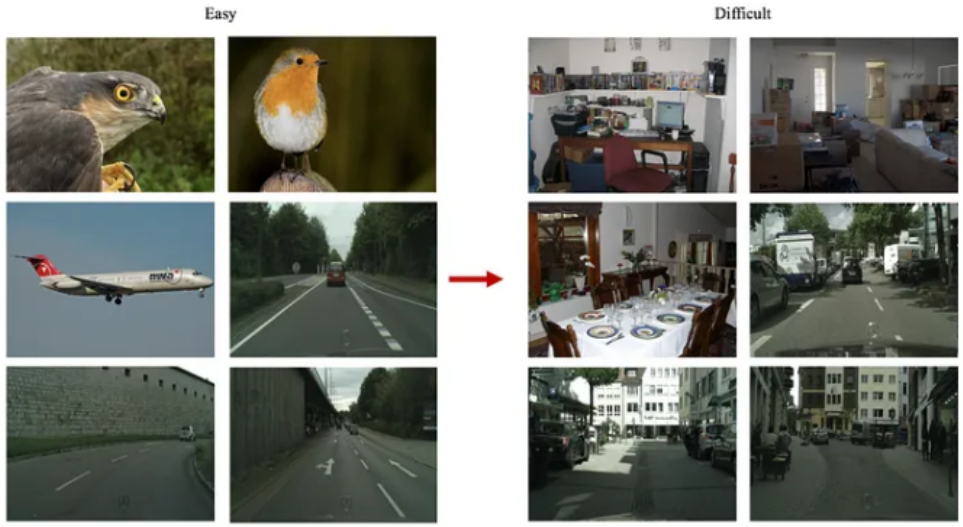


Hyperparameters and Optimization

- **Key hyperparameters:**
  - Learning Rate: most sensitive; use schedulers like cosine annealing, OneCycle.
  - Batch Size: smaller batches = better generalization, more noise.
  - Weight Decay: important for regularizing optimizers like AdamW.
- **Optimizers:** SGD, Adam, AdamW — each with trade-offs.
- **Tuning tools:** Grid search, Random search, Optuna (Bayesian).

Training Strategies

- **Curriculum Learning:** Start with easier samples, progress to harder ones.
- **Early Stopping:** Stops training when validation performance stops improving.
- **Mixed Precision Training:** Faster training, allows larger batch sizes.
- **Warmup Phase:** Gradually increase learning rate — helps with stability in early training.



Ablation Studies

- Start with a strong baseline model — known to work reliably.
- Add or remove one element at a time: layer, loss, augmentation, etc.
- Run isolated experiments to understand the impact of each change.
- Helps pinpoint what's helping (or hurting) performance.

Component Changed	**Performance**	**Impact**
Baseline (full)	92.5%	—
No dropout	88.3%	Overfitting increases
No skip connections	86.7%	Training becomes unstable
No pretraining	81.2%	Poor convergence
Freeze backbone	89.1%	Slight drop, faster training