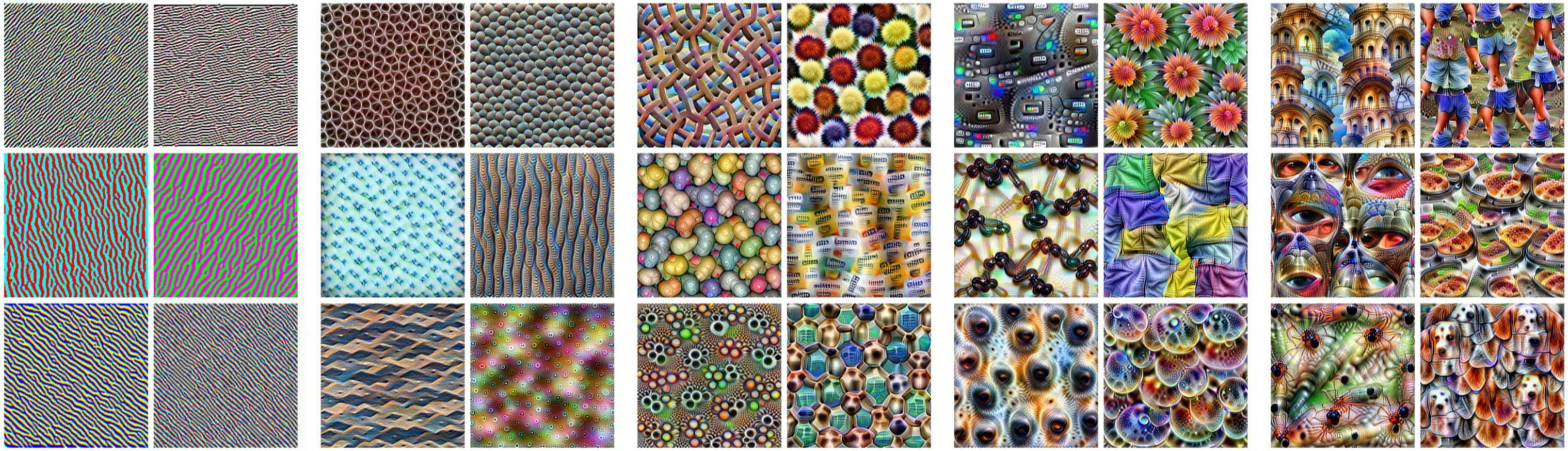
Data Science & Artificial Intelligence

Summer Term 2022

L09 Convolutional Neural Networks Part 1

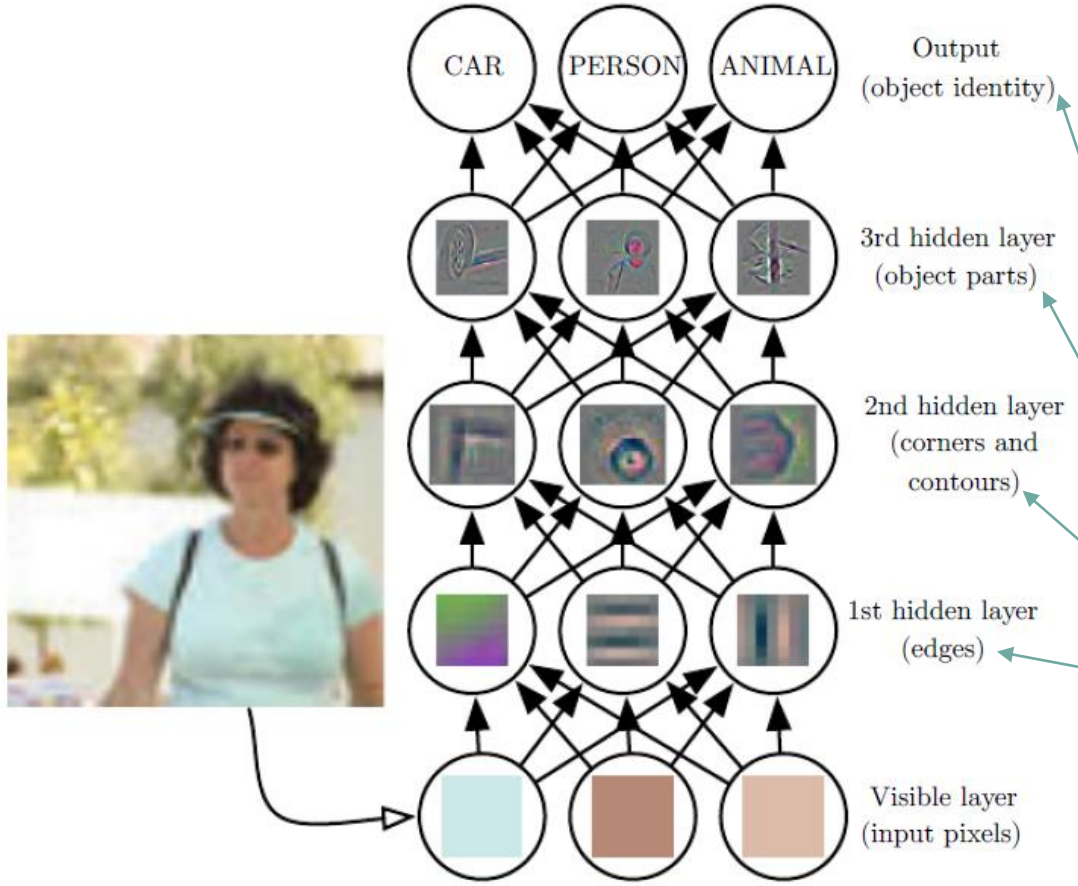
B. Stuhr, J. Haselberger, D. Schneider



Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

L09.1 Visual Features

What are visual Features?



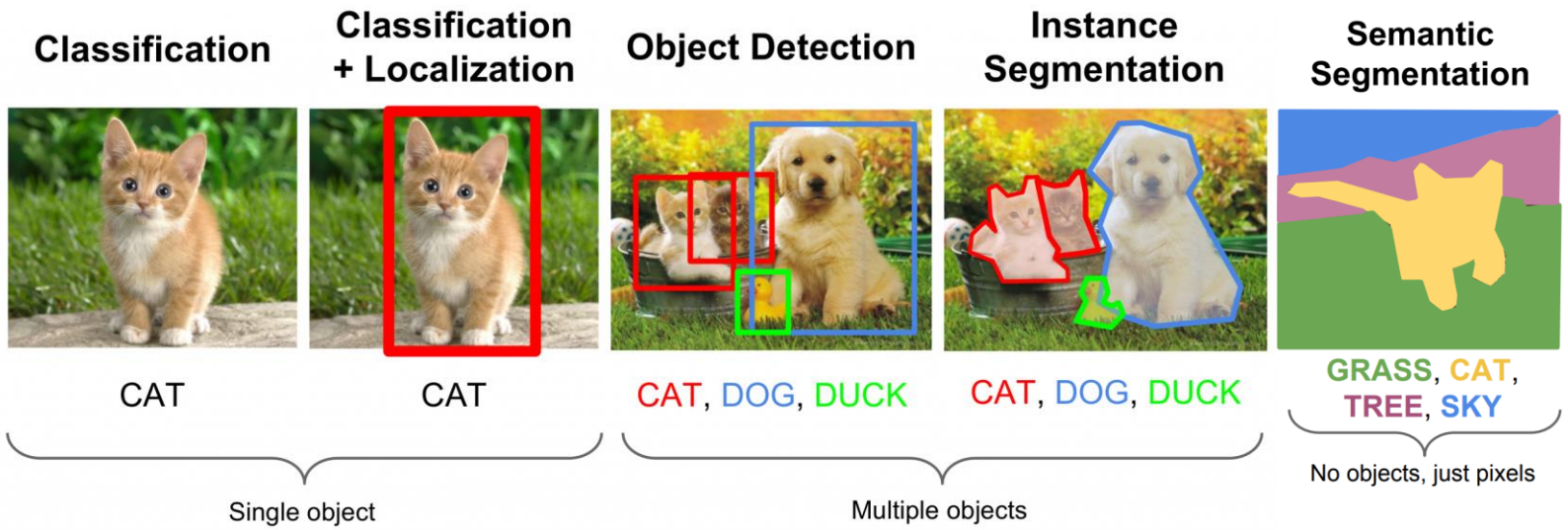
[Goodfellow2016]

❖ Remember:

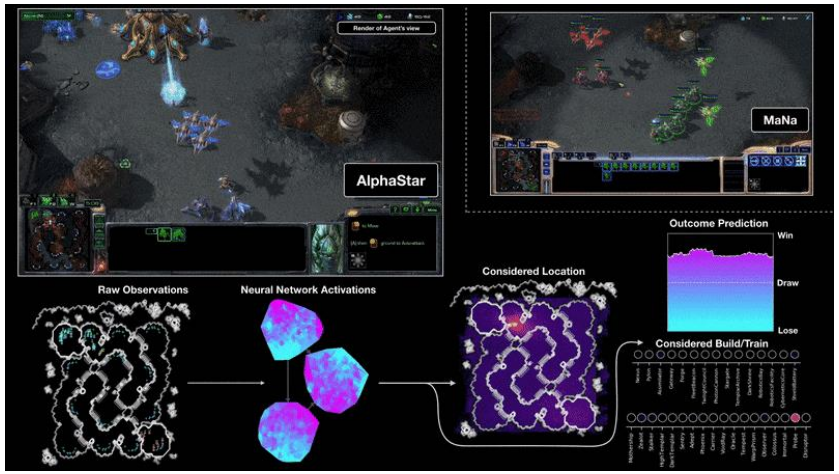
A *feature* is generally a recognizable property, that distinguishes one person, thing, or abstract context from another. It is each piece of distinct information included in the representation.

A *visual feature* is a recognizable, distinguishable property of a visual input.

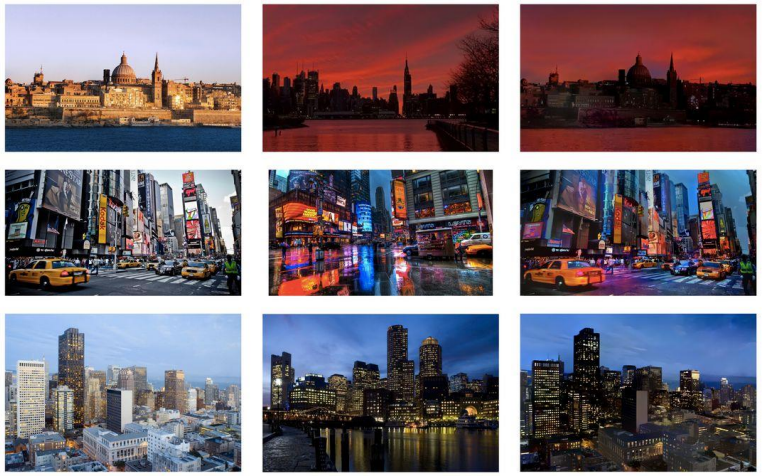
Motivation: Why do we need to learn visual Features?



Pose Tracking in crowded scenes



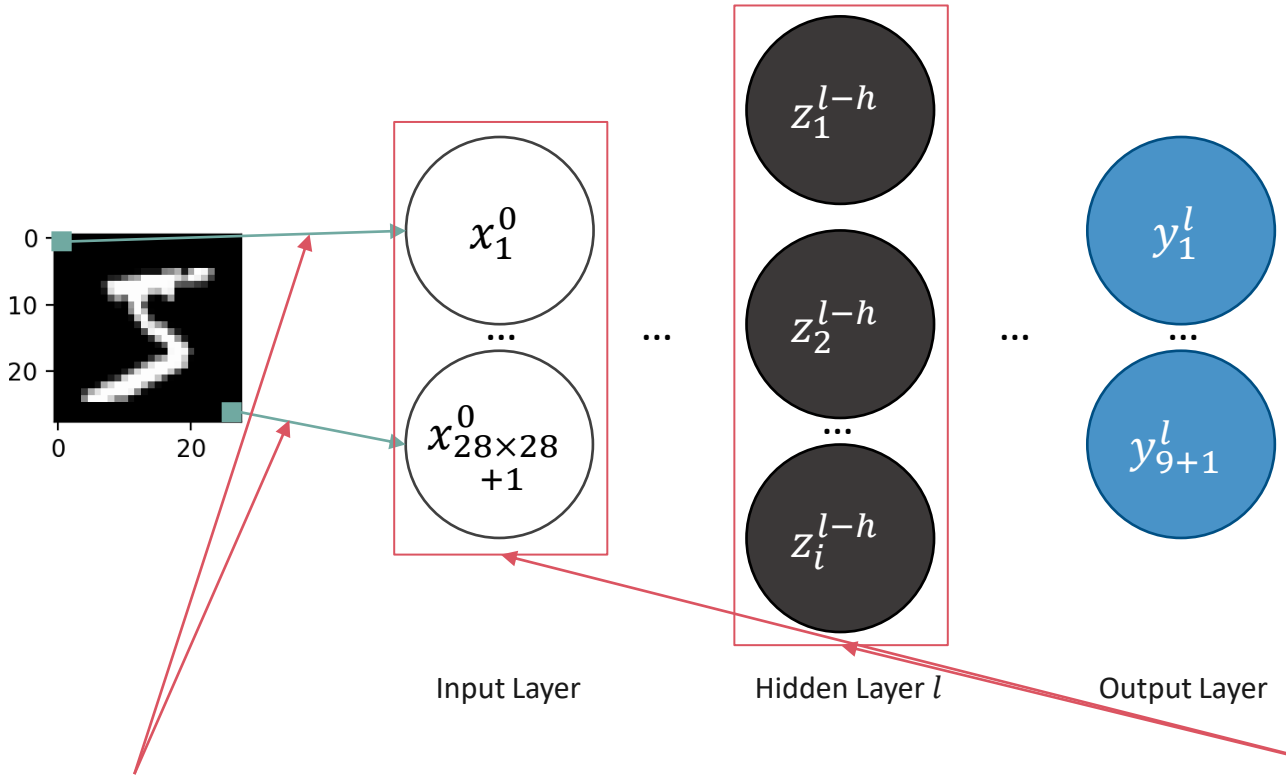
AlphaStar plays Starcraft



Original photo | Reference photo | Result

- And other slides from L07!
- And way more: Basically, every application that requires understanding of visual inputs
- Visual features and methods to learn them are widely used
- Methods are transferred to areas, which are not vision-based

Limitations of MLPs for visual Feature Learning



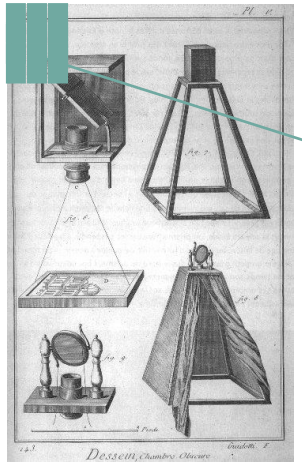
Each pixel is seen by every neuron in layer l

- This leads to a huge number of parameters for large images
  - For a 28×28 image there will be $784 \times i$ weights between input and hidden layer, but for a 256×256 already $65.536 \times i$

The input of each layer is just a flattened vector!

- It is harder to utilize spatial information in the image

Examples of a Classic Method for Edge Detection: Sobel operator

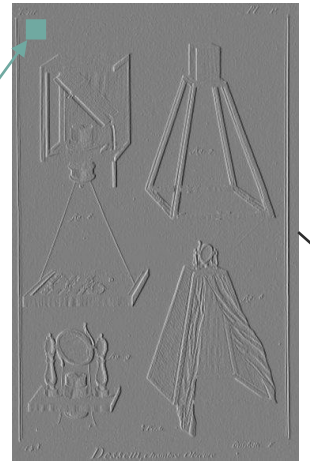


$\mathbf{A}$

$$\sum \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = z$$

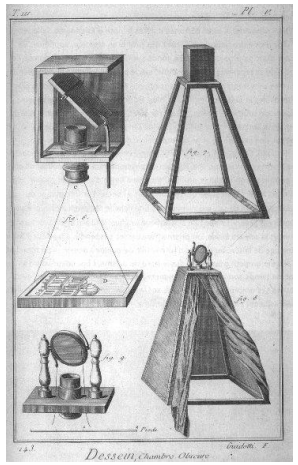
$\mathbf{S}_x$

Filter or Kernel



$\mathbf{G}_x = \mathbf{S}_x * \mathbf{A}$

Convolution Operator!

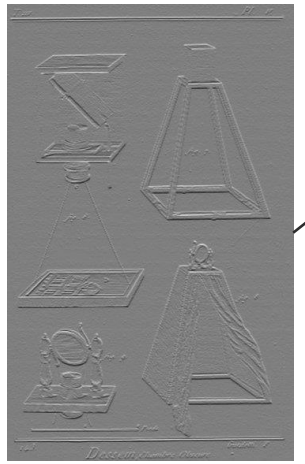


$\mathbf{A}$

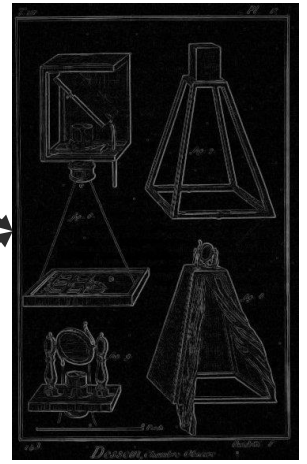
$$* \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} =$$

$\mathbf{S}_y$

More info here: Link

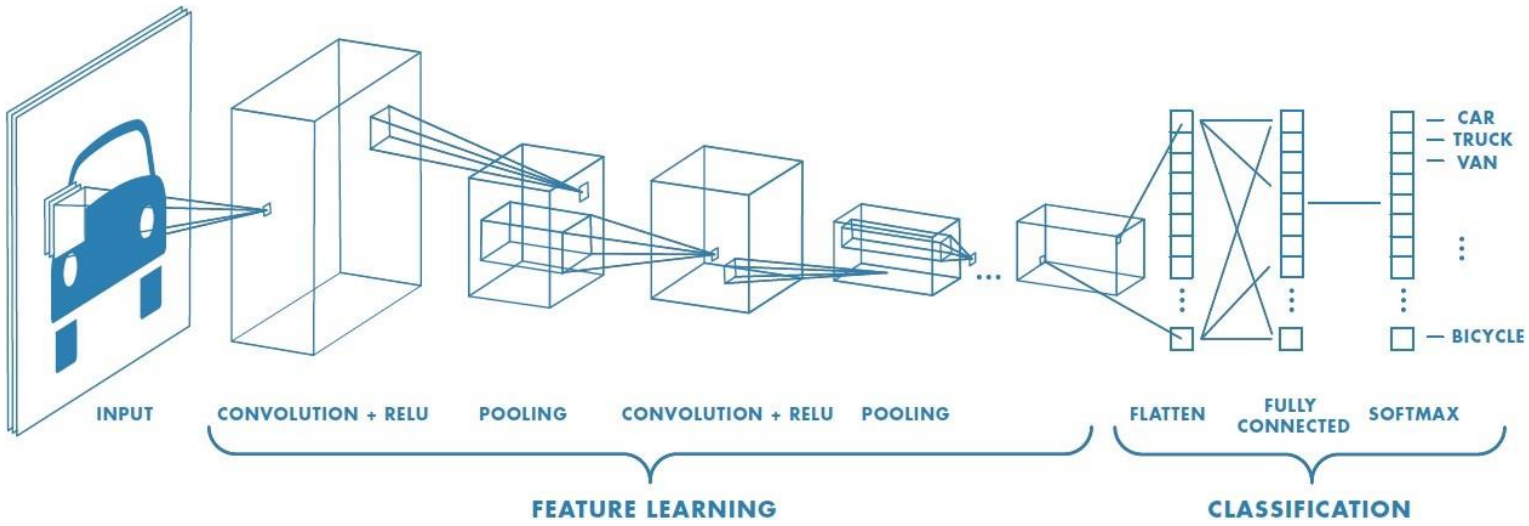


$\mathbf{G}_y = \mathbf{S}_y * \mathbf{A}$



$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

- Only a few parameters are required for the filter!



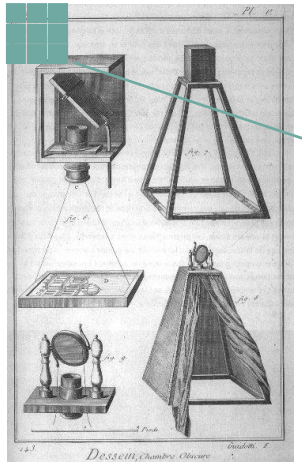
Feature Extractor

Head

in this case a MLP for classification

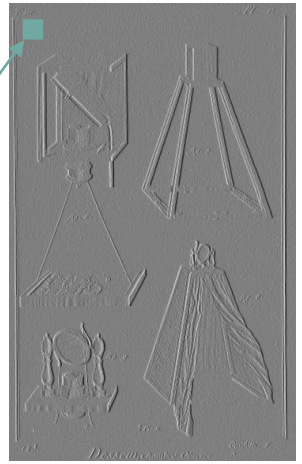
L09.2 Convolutional Neural Networks (CNNs)

From unlearned filter to learned filters with a Neuron



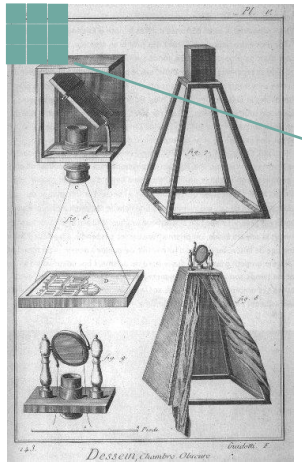
$\mathbf{A}$

$$\sum \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{bmatrix} \circ \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}}_{\mathbf{S}_x} = z$$



$\mathbf{G}_x = \mathbf{S}_x * \mathbf{A}$

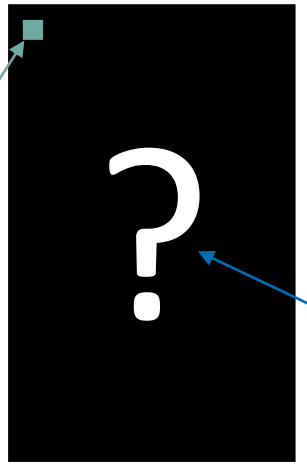
$$y_i^l = \sum_{j=1}^{n^l} w_{j,i}^l a_j^l$$



$\mathbf{A}$

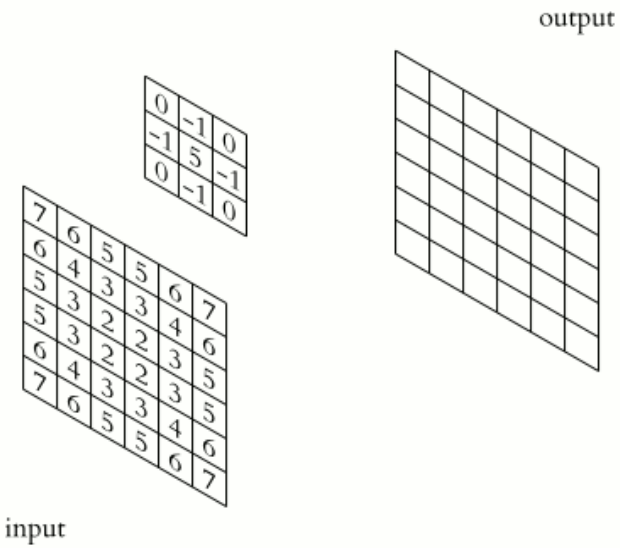
$$\sum \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{bmatrix} \circ \underbrace{\begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix}}_{\mathbf{W}} = z$$

This is a neuron without bias an activation function



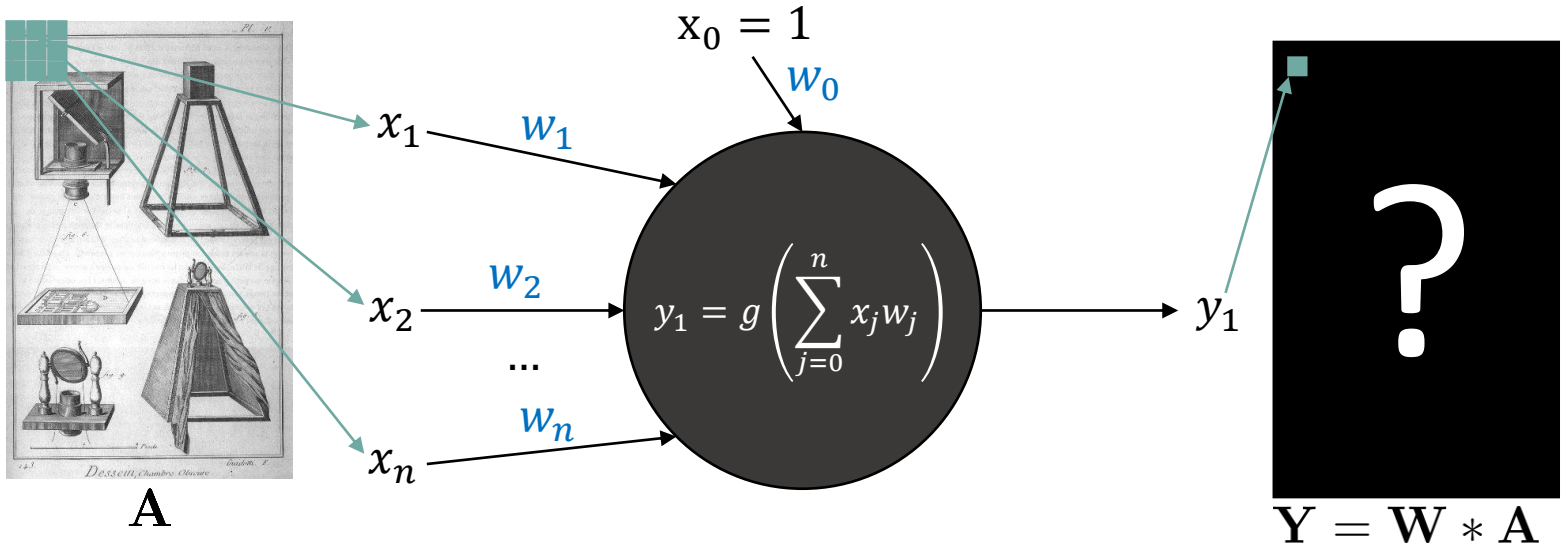
$\mathbf{Y} = \mathbf{W} * \mathbf{A}$

We do not know what is learned through the filter



Convolution Operation Example [Link]

From unlearned filter to learned filters with a Neuron

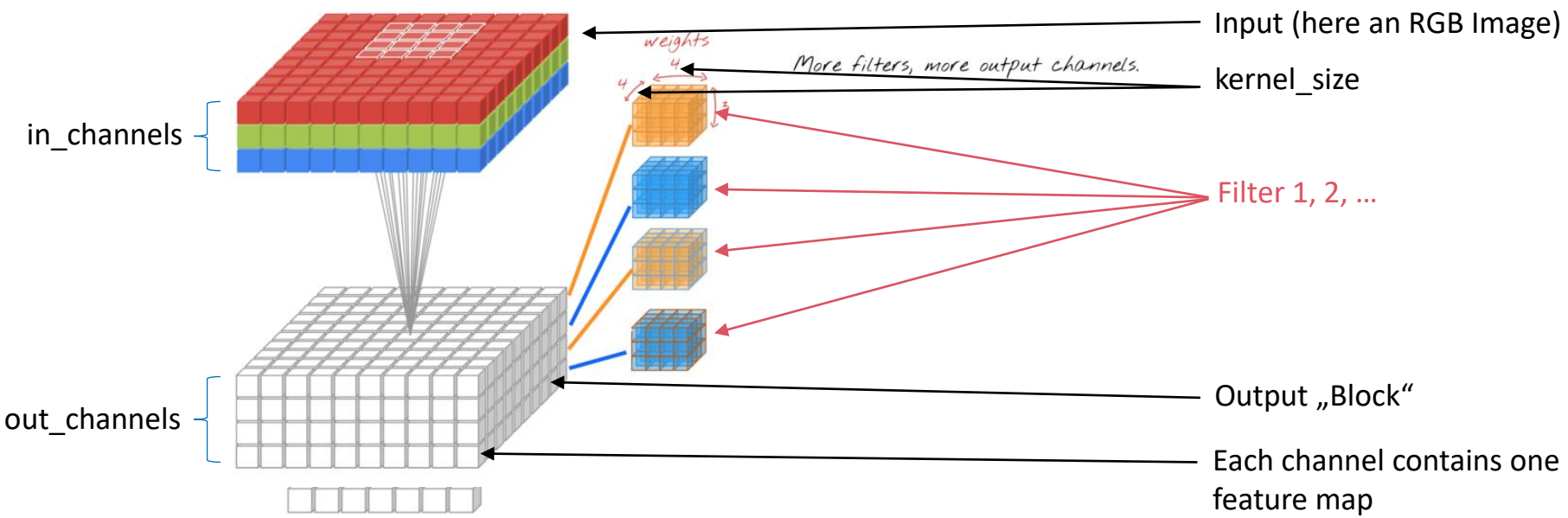


We can add a bias and activation function to our learned filter.

- In this way we achieve non-linearity and arrive by a **neuron**

If we use **multiple neurons in one convolutional layer** (Multioutput Perceptron) and slide them over the image, we learn **multiple filters**

The convolutional Layer: kernel_size , in_channels, out_channels

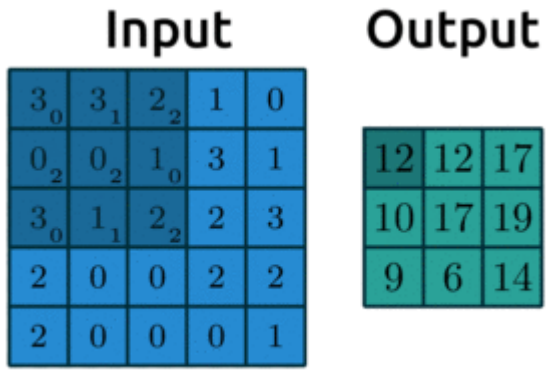


- kernel_size: describes the width and height of each filter
  - E.g., the number of pixels a filter/neuron in the first layer sees at each position
- in_channels: “depth” of the input and therefore of each filter
  - For a rgb image in_channels is 3
  - For the output of a previous convolutional layer in_channels is the number of filters from the last layer
- out_channels: number of filters in this layer

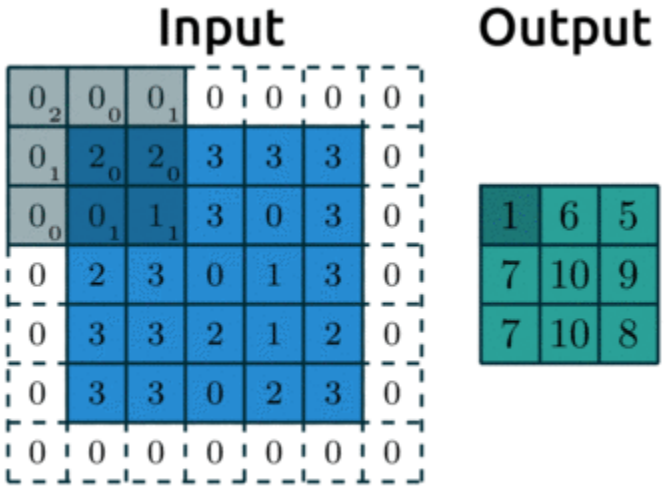
```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
                dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Images from [Link]

The convolutional Layer: stride, padding, padding_mode



no stride, no padding



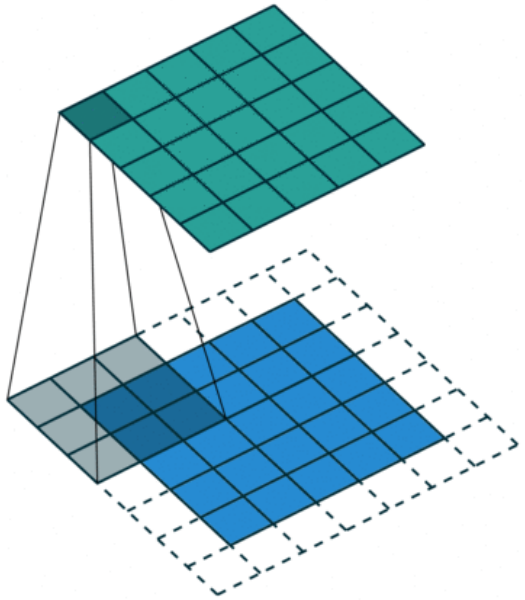
stride=2, padding=1
padding_mode= ‘zeros‘

- stride: the number of steps the filter is moved to reach the next position
- padding: used to control the width and height of the output block
  - Extends the width and height input block by the defined number
- padding_mode: used to define the values of the extended part of the block (e.g., zeros)

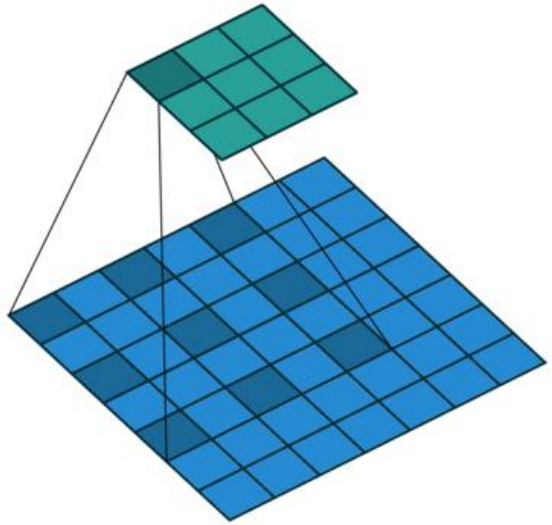
```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
                dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Images from [Link]

The convolutional Layer: other Variants

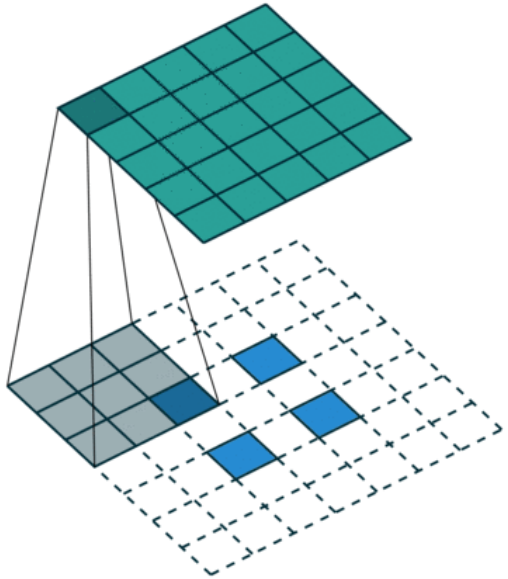


Convolution



Dilated/Atrous Convolution

- Deliver a wider field of view at the same computational cost
- Popular in the field of real-time segmentation
- dialation parameter in torch.nn.Conv2d



Transposed/De-/Up-/fractionally strided Convolution

Images from [Link]

Pooling Layers: Max Pooling, Average Pooling

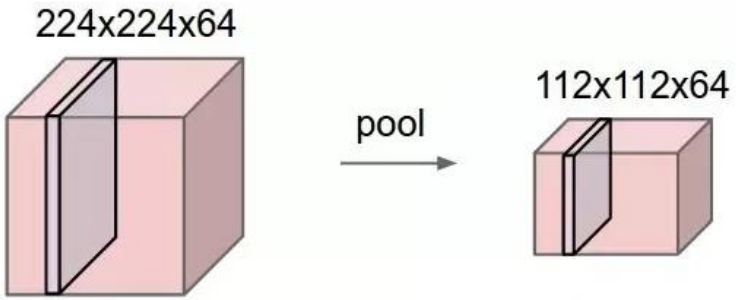
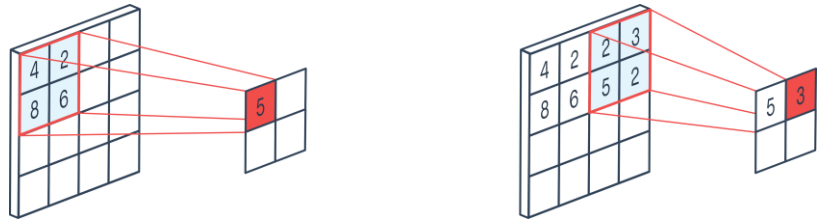
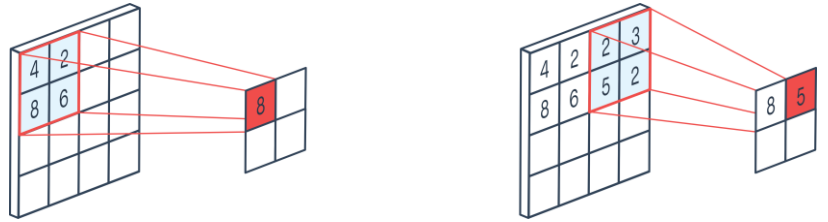


Image adapted from [Link]

Average Pooling:



Max Pooling:



Images from [Link]

There are two common pooling operations:
- **Max Pooling**: Selecting the maximal value in the kernel
- **Average Pooling**: Calculating the average value in the kernel

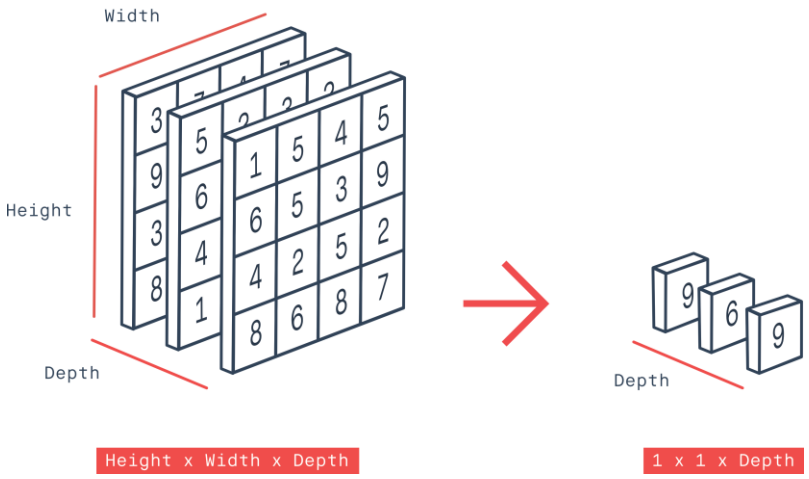
Pooling reduces the spatial dimensions
- The depth dimension (number of filters in a CNN) is not reduced

Pooling operations are applied the same way a convolution operation is applied
- With stride, kerne_size, padding, …

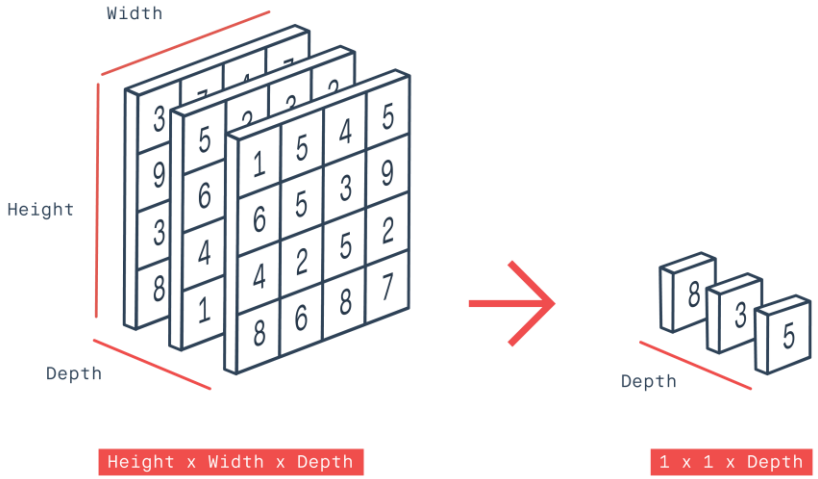
```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False,
                   ceil_mode=False)
```

```
torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True,
                   divisor_override=None)
```

Pooling Layers: Global Max Pooling, Global Average Pooling



Global Max Pooling



Global Average Pooling

Pooling operations can be applied globally to summarize the entire block into a single vector:

- **Global Max Pooling**: Selecting the maximum value in the entire feature map
- **Global Average Pooling**: Calculating the average value from the entire feature map

This is helpful if the output should always have the same spatial dimension when the input images have different sizes

- When training a CNN with a MLP head, the MLP input vector must always be the same size

Images from [Link]

Pooling Layers: Advantages and Disadvantages

- ✓ Less parameters in the network:
  - ➢ Network is more efficient
  - ➢ Reduces Overfitting
- ✓ Makes to model invariant to some changes in the input
  - Especially Translation Invariance
    - Translational invariance means that the system produces the same response regardless of how its input is shifted
  - Especially when using max pooling
- ✖ Loss of information
  - Especially when global pooling is used
  - Modern CNNs use pooling very cautiously

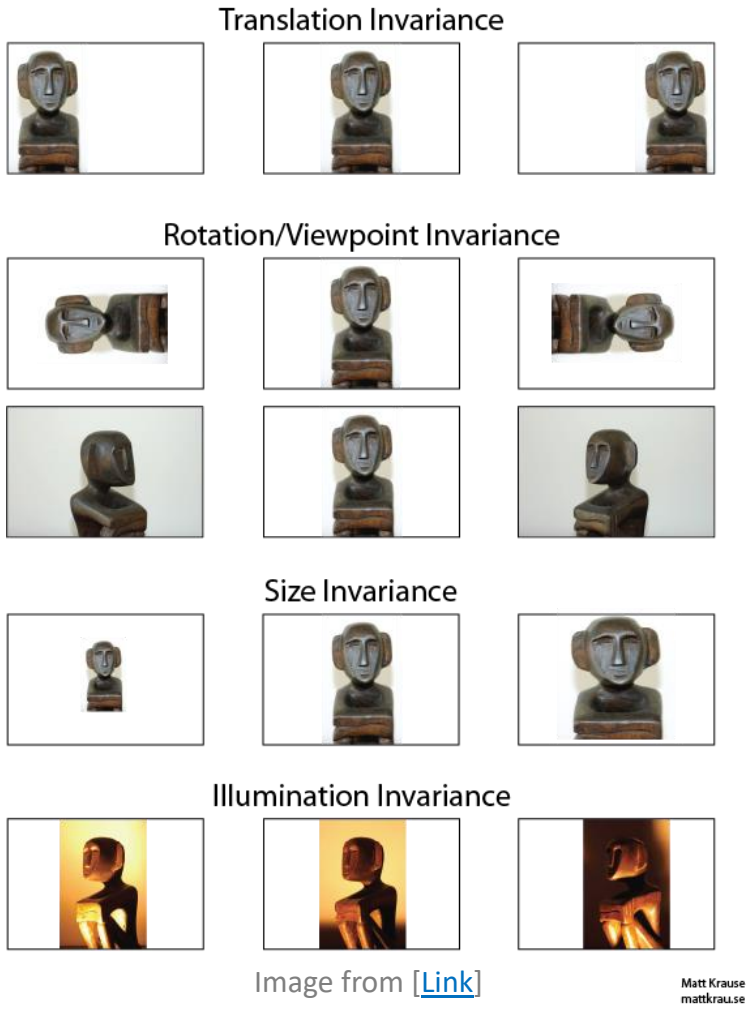
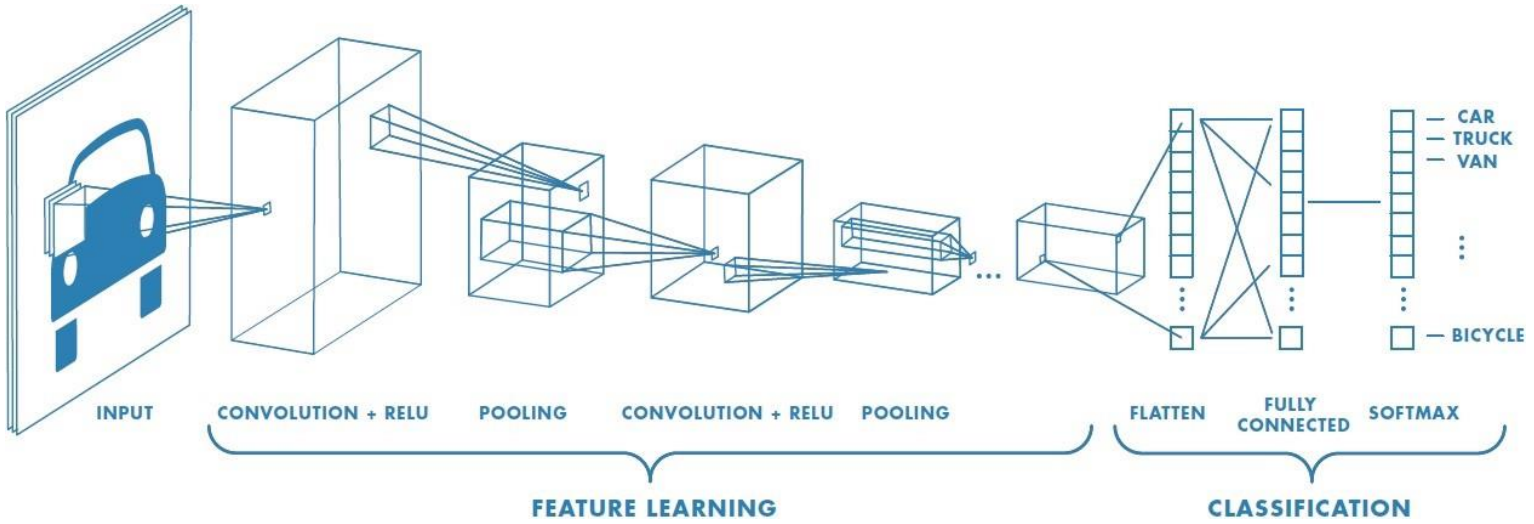


Image from [Link]

Architecture of a common CNN



Feature Extractor

Head

in this case a MLP for classification

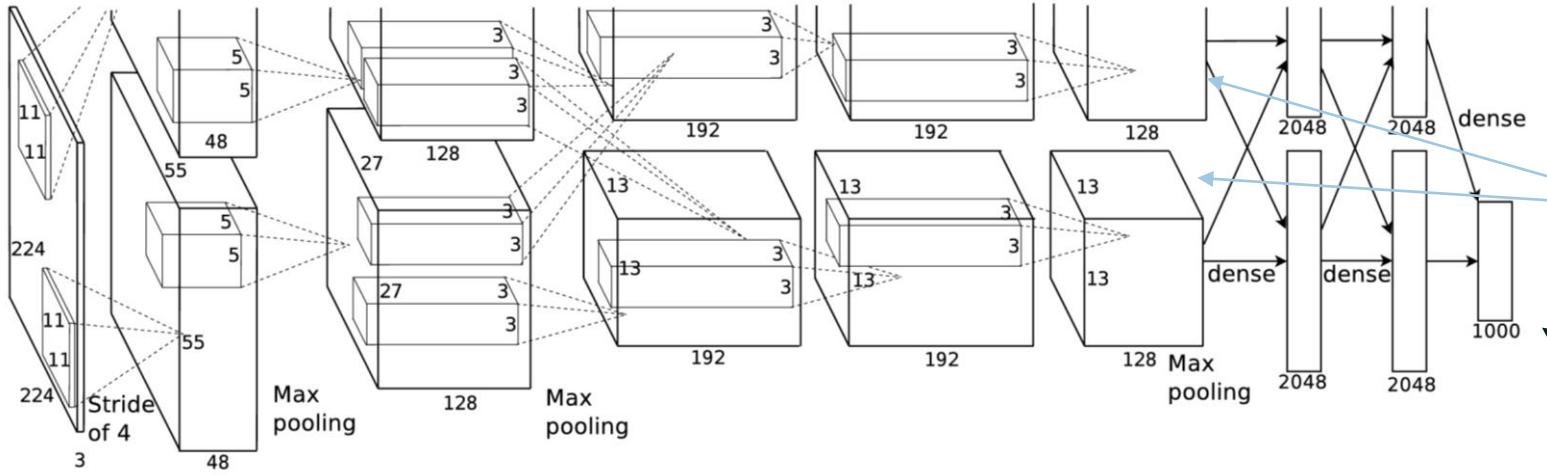
! With convolutional layers we are able to extract features from an input

For the final decision we often need one or more heads which use the learned features as input for the final decision.

- E.g., a MLP for classification or regression

Image from [Link]

Full CNN Example (AlexNet) and Code



The CNN in split into two parts, to train in on 2 GPUs, because it was, memory expensive

1000 classes

Input: Image	Feature Extractor: Consists of convolutional and max pooling layers	Head: MLP

!

AlexNet won the ImageNet Large Scale Visual Recognition Challenge (2012)

- With 15.3% top-5 error, around 11% lower than second place

➢ **This kicked of the Deep Learning trend**

- Designed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton (Ph.D. advisor) [Krizhevsky2012]

AlexNet modern Code

Simple implementation of the AlexNet model:

```
class AlexNet(nn.Module):

    def __init__(self):
        super(AlexNet, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11, stride=4, padding=0)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2)
        self.conv2 = nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=1, padding=2)
        self.conv3 = nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride=1, padding=1)
        self.conv5 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(in_features=9216, out_features=4096)
        self.fc2 = nn.Linear(in_features=4096, out_features=4096)
        self.fc3 = nn.Linear(in_features=4096, out_features=1000)

    def forward(self,x):
        x = F.relu(self.conv1(x))
        x = self.maxpool(x)
        x = F.relu(self.conv2(x))
        x = self.maxpool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = self.maxpool(x)
        x = x.reshape(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

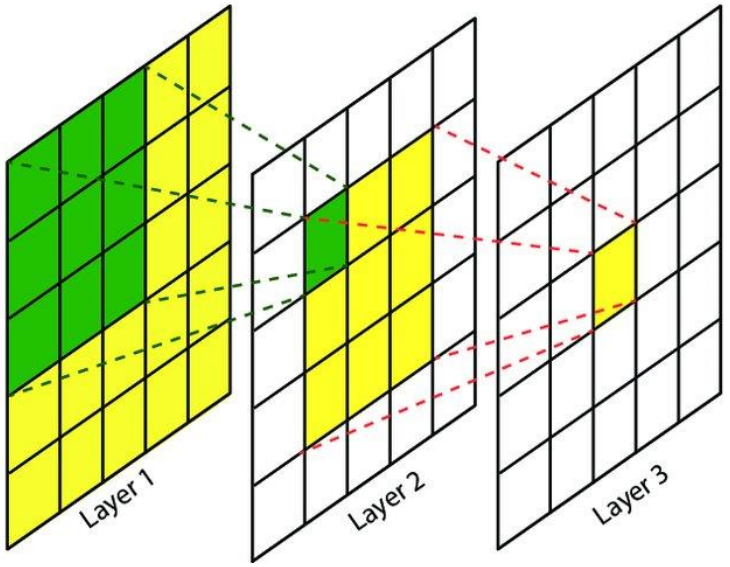
Code adapted from [Link]

Use a pretrained model from the Model Zoo

```
import torchvision.models as models
model = models.alexnet(pretrained=True)
model.eval()
```

By changing the model, other, already trained models can be loaded

Receptive Field



Simple Example of a Receptive Field [Link]

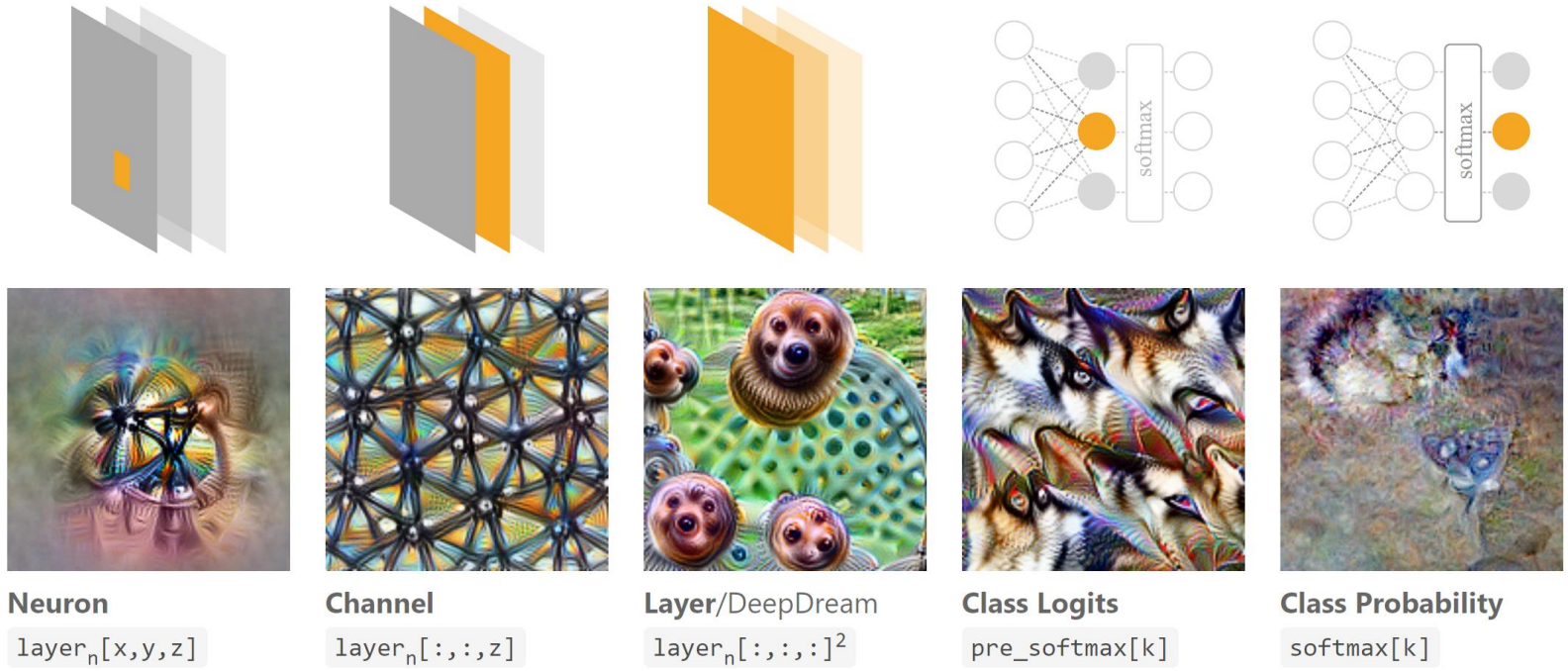
! Describes the size of the area a neuron in a layer sees from the input

- In a CNN, the neurons in the deeper layers usually have a larger or equal receptive field than the layer in before of them
- The term “receptive field” comes form neuroscience
  - “The receptive field, or sensory space, is a delimited medium where some physiological stimuli can evoke a sensory neuronal response in specific organisms.” [Link]

Visual Features learned by a Convolutional Neural Network (CNN)

Different **optimization objectives** show what different parts of a network are looking for.

`n` layer index
`x,y` spatial position
`z` channel index
`k` class index



More examples and a detailed explanation can be found here.

Visual Features learned by a Convolutional Neural Network (CNN)

Dataset Examples show us what neurons respond to in practice

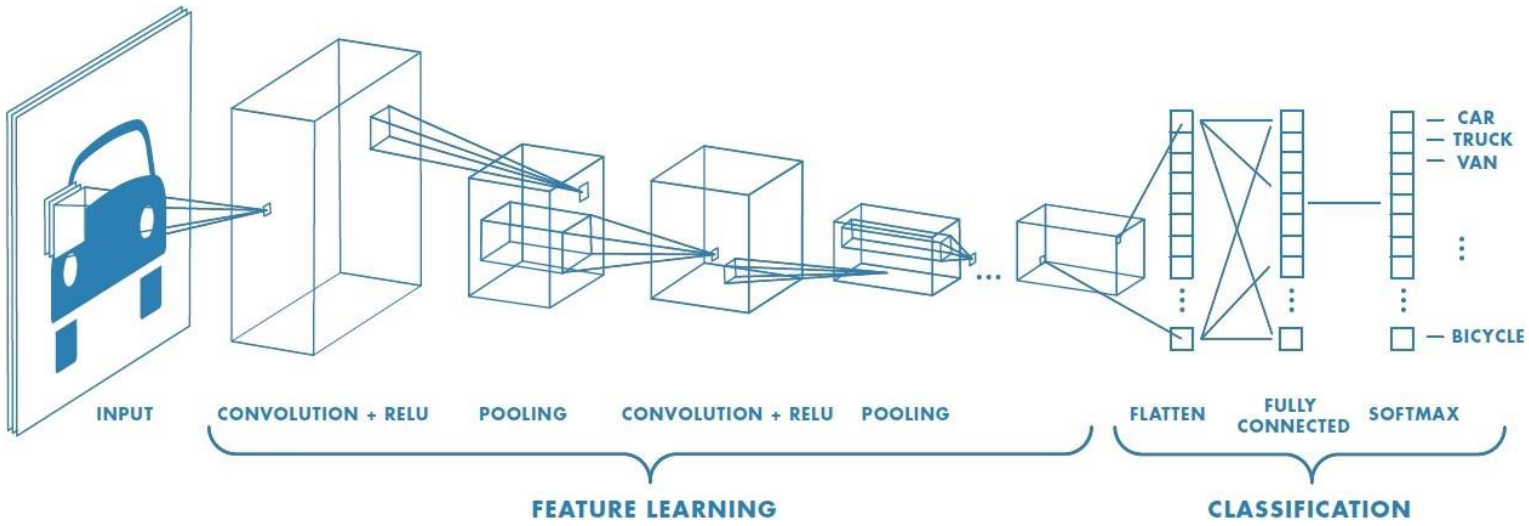
Optimization isolates the causes of behavior from mere correlations. A neuron may not be detecting what you initially thought.

Baseball—or stripes?
mixed4a, Unit 6

Animal faces—or snouts?
mixed4a, Unit 240

Clouds—or fluffiness?
mixed4a, Unit 453

Buildings—or sky?
mixed4a, Unit 492

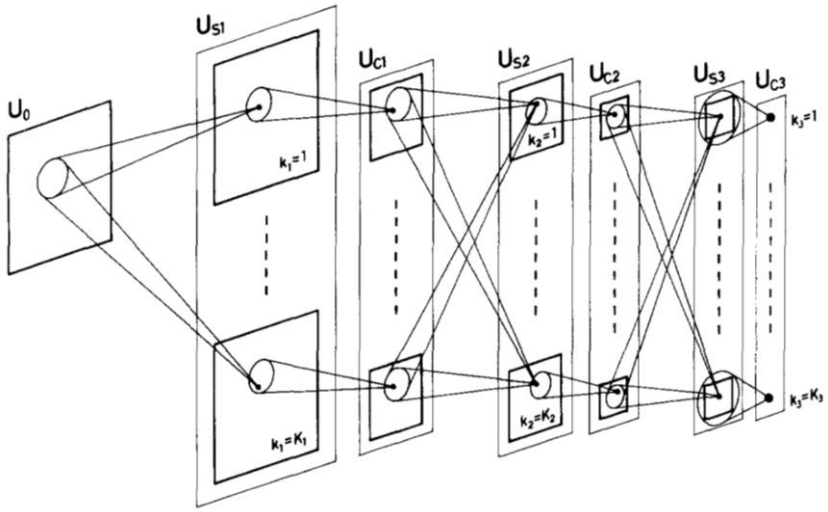


Feature Extractor

Head
in this case a MLP for classification

L09.3 Some important CNN Architectures

Neocognitron (1979/1980): Inspiration for CNNs

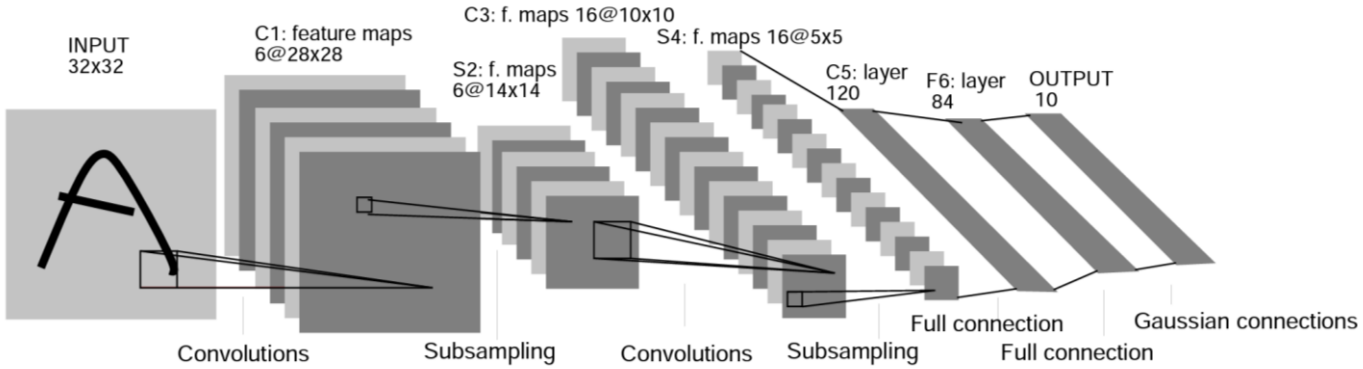


Is the inspiration of CNNs

- Local features in the input are integrated gradually and classified in the higher layers
- This idea is found in other models (e.g., CNNs)

- Proposed by Kunihiko Fukushima et al. in 1979/1980 [Fukushima1980]
- Hierarchical Model
- Biologically Inspired
- Consists of multiple types of cells
  - S-cells: extract local features
  - C-cells: tolerate features' deformations (e.g., local shifts)

LeNet Model Family (1989)

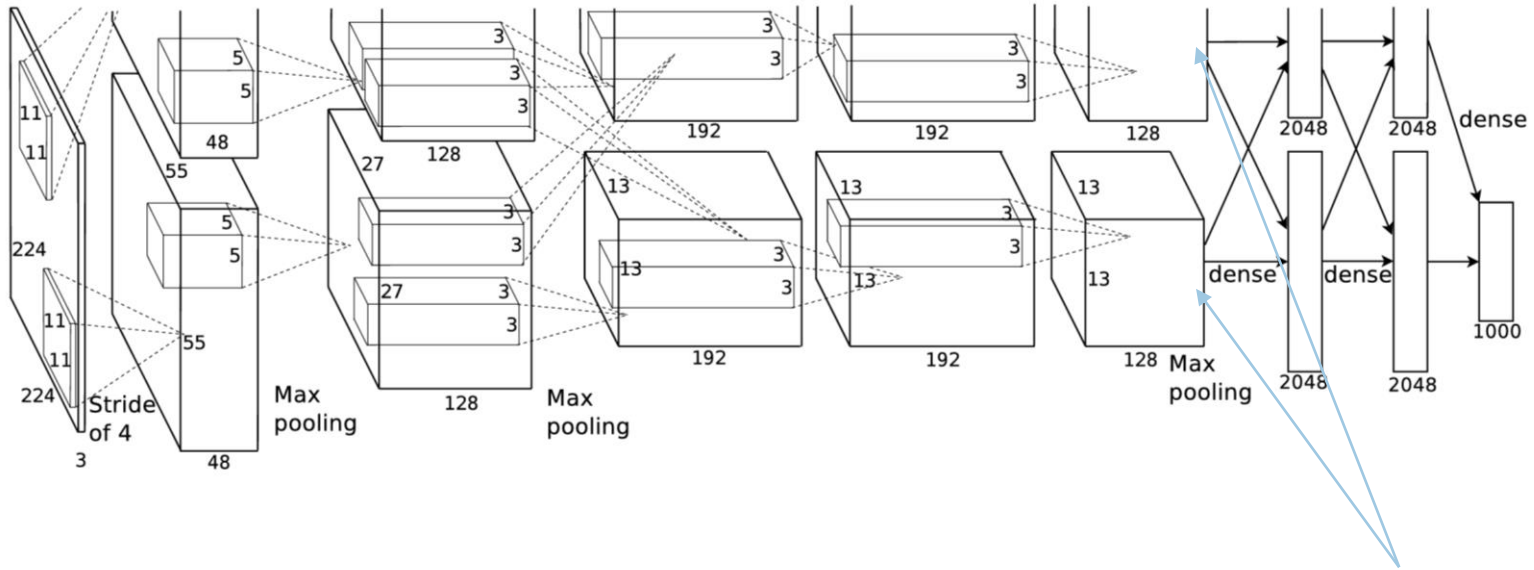


Model Family: Models with different configurations of the same underlying architecture

One of the earliest CNNs

- Proposed by Yann LeCun et al. in 1989 [LeCun1989]
- In general, LeNet refers to LeNet-5 (60K trainable parameters)
- Input is 32x32 instead of 28x28 (for MNISIT)
  - Features (e.g., stroke endpoints or corners) can appear in the center of the receptive field of the highest-level neurons
- Gaussian Connections at the last layer to increase performance (short reason)
  - Are not used in standard CNNs today

AlexNet (2012)



The CNN in split into two parts, to train in on 2 GPUs, because it was, memory expensive

AlexNet won the ImageNet Large Scale Visual Recognition Challenge (2012)

- With 15.3% top-5 error, around 11% lower than second place
- ➢ **This kicked of the Deep Learning trend**

Main contributions: depth of the model is essential for high performance

- Deep models were computationally expensive in 2012 (61M parameters)
- Was made feasible due to the utilization of GPUs during training

- Proposed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton (Ph.D. advisor) [Krizhevsky2012]
- Is still in use for some tasks today

VGG Model Family (2014)

Main contributions:

- Evaluation of networks of increasing depth using an architecture with very small (3×3) kernel size
- Improvement on the state-of-the-art with model depth of 16–19 weight layers

- Proposed by Karen Simonyan and Andrew Zisserman [Simonyan2014] from the **V**isual **G**eometry **G**roup from the University of Oxford (therefore VGG)
- Won the ImageNet localization challenge of ILSVRC-2014
- Configuration D and E are called VGG16 and VGG19 today and are still in use for some tasks

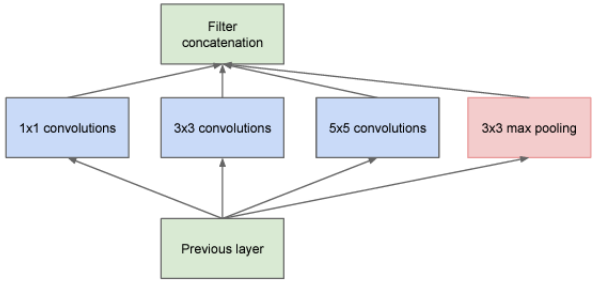
Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The 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					

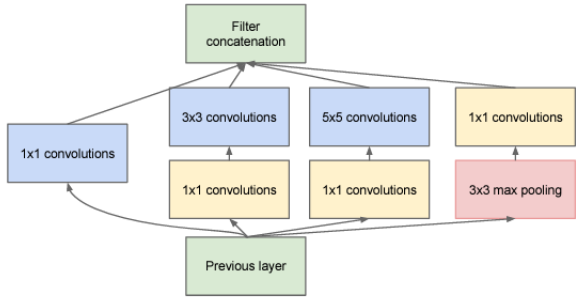
Table 2: **Number of parameters** (in millions).

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

GoogleNet (2014), Inceptionv3 (2015), Inceptionv4 (2016)



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Paper Goal: find the "optimal" local construction

- uses multiple convolutions with different kernel sizes and pooling in some layers
- Used 1x1 convolutions with fewer filters for efficiency. This reduces the depth of the input to the following convolutions

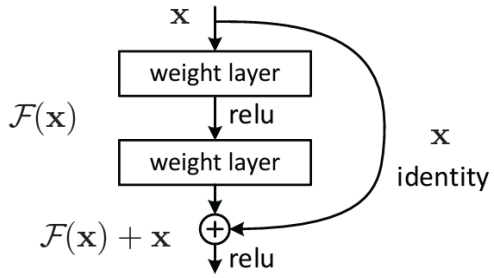
- Models proposed by Google [Szegedy2014] [Szegedy2015] [Szegedy2015]
- Named after the "we need to go deeper" meme (was cited in the paper)
- Inceptionv2-v4 incorporate various design improvements
  - E.g., Inceptionv4: Skip-Connections inspired by ResNet
- Inceptionv4 is still in use today for some tasks
- Parameters: GoogleNet (7M), Inceptionv3 (24M), Inceptionv4 (43M)

ResNet Model Family (2015)

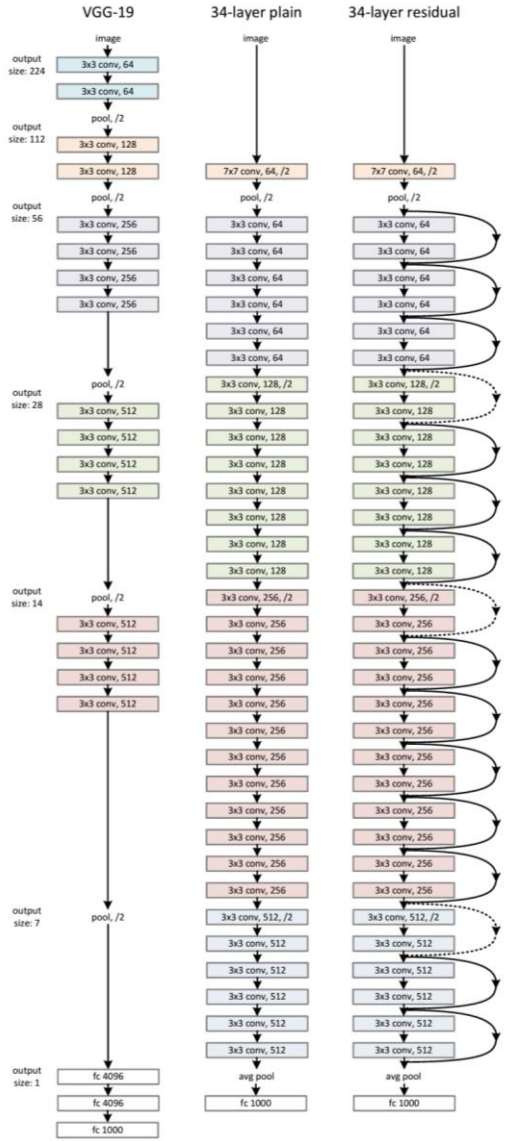
When going deeper with traditional CNNs problems arise:
- Vanishing Gradients
- Degradation (accuracy saturation) problem:
  - Adding more layers to a suitably deep model leads to higher training error

Main contribution: Skip Connections, to mitigate above problems:
- A connection between weight layers that skips some weight layers

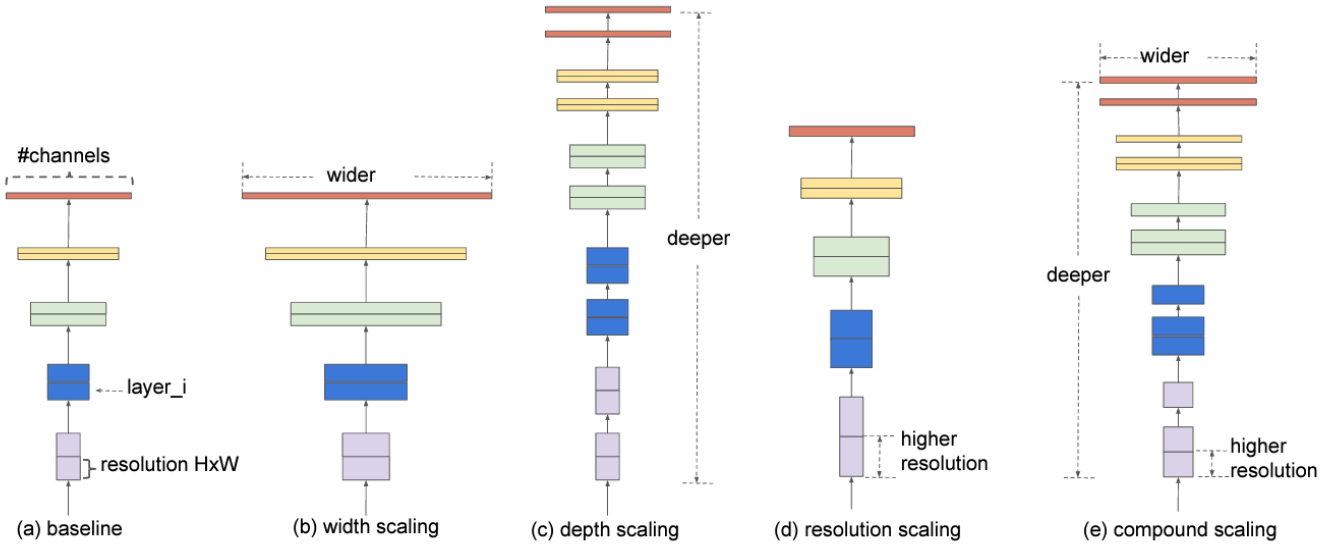
- Proposed by Kaiming He et al. from Microsoft in 2015 [He2015]
- Won the ImageNet Large Scale Visual Recognition Challenge (2016)
- Investigated models with up to 1202 Layers
- Most common models:
  - ResNet18 (11.4M params)
  - ResNet50 (23.9M params)
  - ResNet101 (42.8M params)
- ResNets are heavily used today

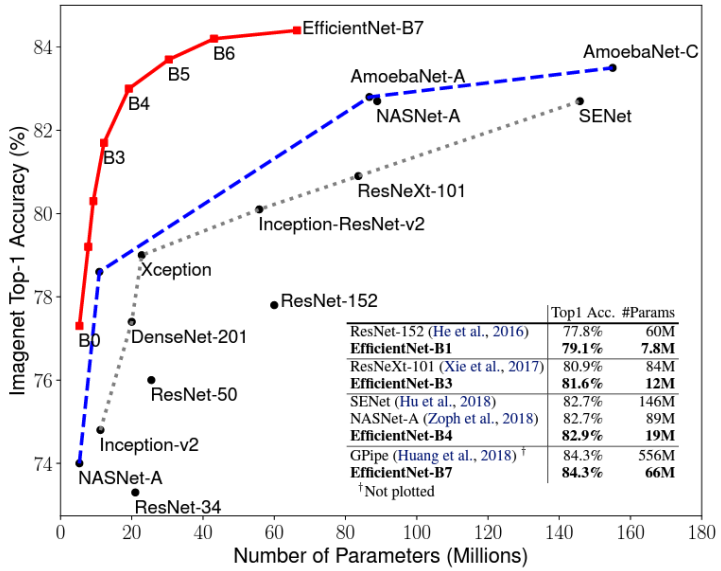


Skip Connection



EfficientNet Model Family (2019)





Main Contributions: Systematic study of model scaling

- Finds that careful tuning of model depth, width and resolution can lead to better performance
- New scaling method that scales depth/width/resolution uniformly by using a simple but very effective composite coefficient

- Proposed by in Mingxing Tan and Quoc V. Le from Google in 2019 [Tan2019]
- In use today

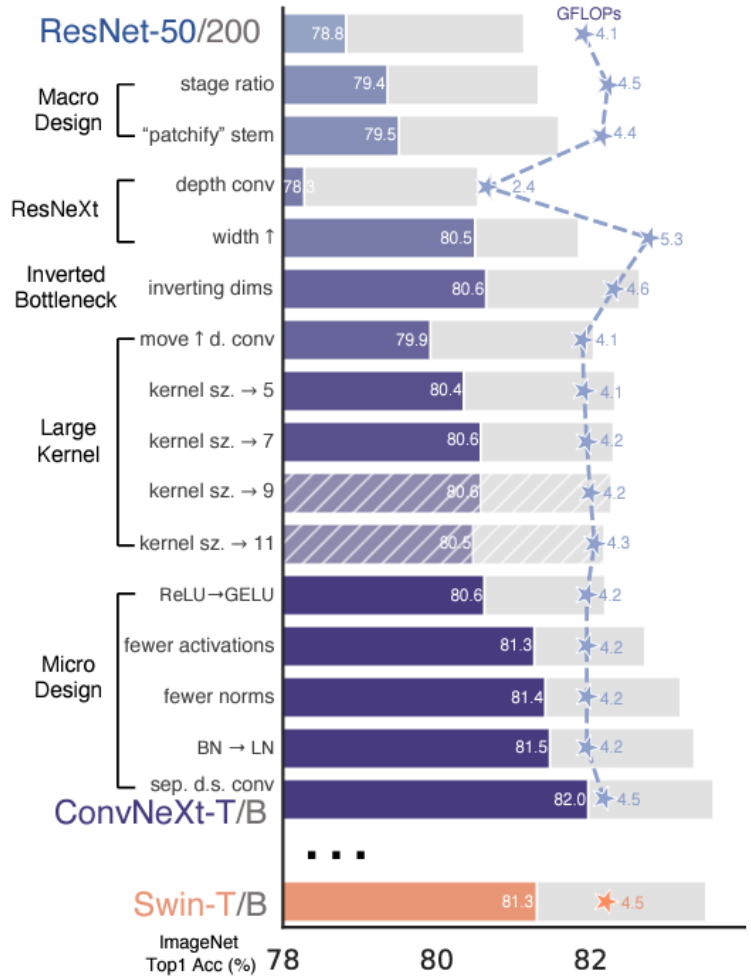
ConvNeXt (2022)

Main contribution: "Modernization" of a standard ResNet with design and training decisions introduced by Vision Transformers

➢ Discovers several key components of Transformers that increase the performance of ResNets

- Vision Transformers superseded CNNs for image classification in the 2020s
  - Vision Transformers will not be discussed in this lecture
- Proposed by Zhuang Liu et al. from Facebook in 2022 [Liu2022]
- In use today
- Broth design choices made in transformers to ResNets
  - Better augmentations for training
  - Change of Activation function
  - Change of kernel sizes
  - Patchify: Use 4x4 convolutions with stride 4 → first layers sees “patches” of the input
  - …

Image spitted in patches:



Conclusions

- First CNNs are a big step in learning visual features
- Most of the model architectures that follow LeNet and AlexNet try to find the optimal configuration for the network or improve the training procedure
- Over time, new modeling strategies, training strategies, activation functions, norms, layer types, etc. have been discovered
- The search for the optimal vision model is ongoing, but major advances are made almost every year

Further Reading

- Deep Learning Book: https://www.deeplearningbook.org/
- The newest Papers (there are way too many)
- YouTube Lectures of other Unis, e.g.:
  - MIT: https://www.youtube.com/watch?v=5tvmMX8r_OM&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI
  - Stanford: https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk
  - TUM: https://www.youtube.com/watch?v=QLOocPbztuc&list=PLQ8Y4kIIbzy_OaXv86lfbQwPHSomk2o2e
  - TUM (advanced): https://www.youtube.com/watch?v=Bt5O1HjT9cI&list=PLog3nOPCjKBkngkkF552-Hiwa5t_ZeDnh
  - Tübingen: https://www.youtube.com/watch?v=OCHbm88xUGU&list=PL05umP7R6ij3NTWIdtMbfvX7Z-4WEXRqD
  - ….

www.hs-kempten.de/ifm