Introduction to Convolution & Convolutional Neural Networks

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

LMH
Lady Margaret Hall

July 25, 2025

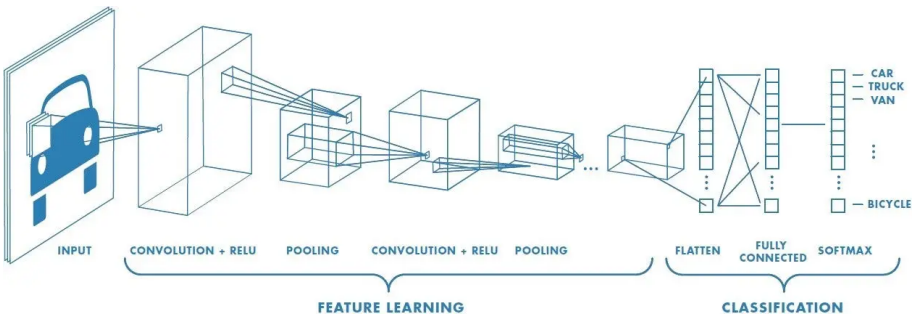

CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

Table of Contents

Table of Contents (cont.)

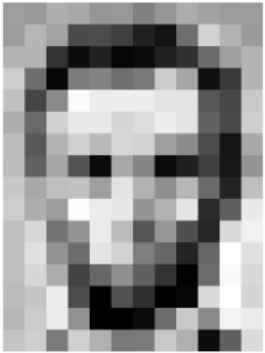
Welcome to CNNs!

Topic: Introduction to Convolution and CNNs

What we'll learn today:

- How computers see images
- What is convolution
- How CNNs work
- Implementing CNNs in PyTorch

Motivation

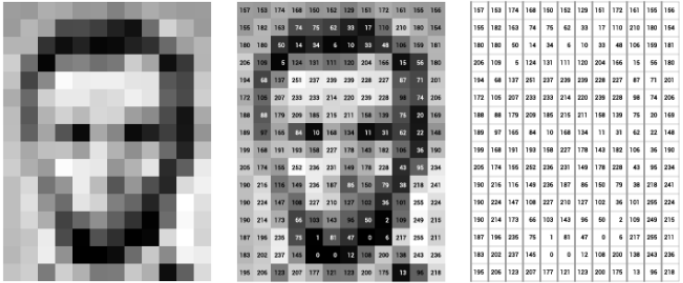


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

- Humans are amazing at understanding images: recognizing faces, reading signs, identifying objects.
- Computers? Not so much. They need help!

Motivation (cont.)



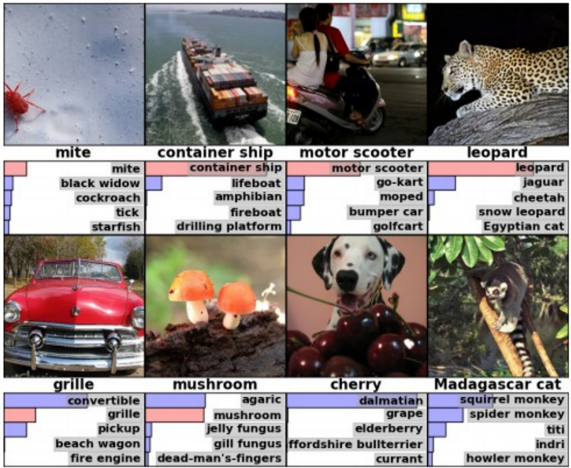
- What we human see as a simple image is just a grid (or matrix) of numbers to a computer.
- Each pixel has a value (0-255 for grayscale, 3 values for RGB).
- Computers need to learn patterns in these numbers to understand images.

Motivation (cont.)

- Convolution and CNNs are the backbone of modern computer vision.
- Used in self-driving cars, medical diagnosis, facial recognition, etc.

Some Applications

Image Classification

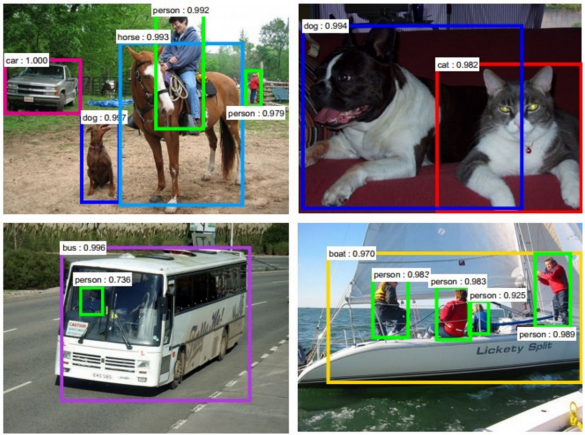


Some Applications (cont.)

Image Retrieval

Some Applications (cont.)

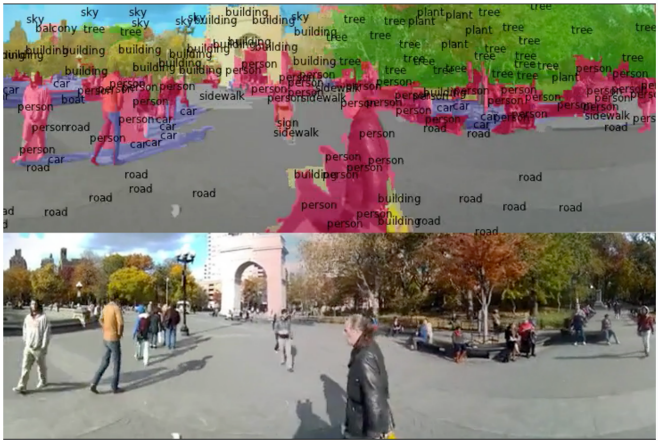
Object Detection



Ren, He, Girshick, and Sun, 2015

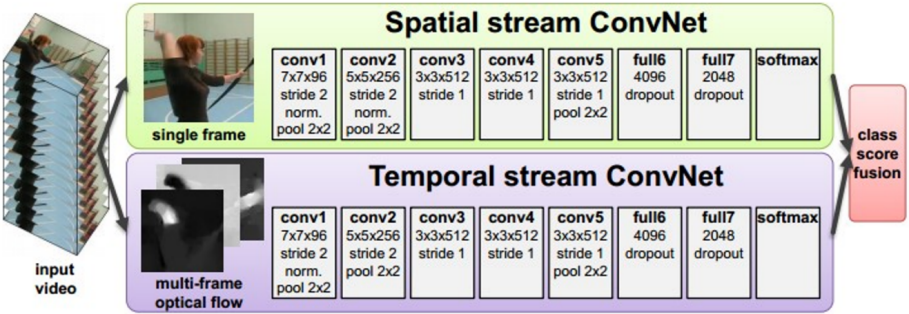
Some Applications (cont.)

Image Segmentation



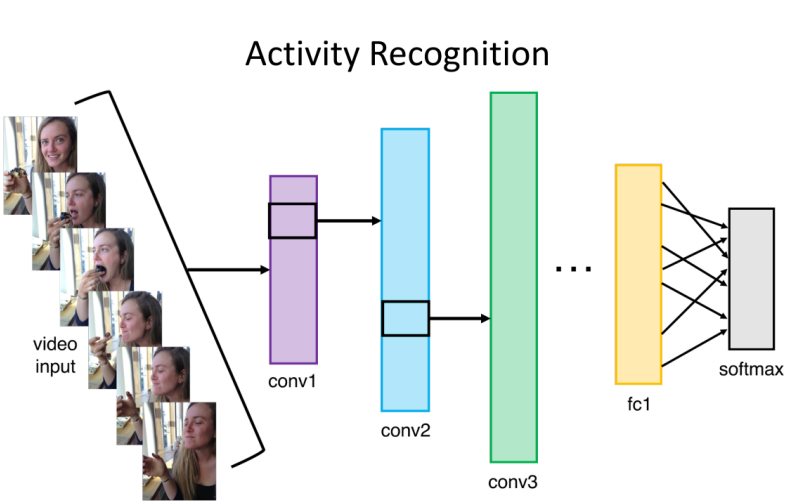
Fabaret et al, 2012

Video Classification



Simonyan et al, 2014

Some Applications (cont.)

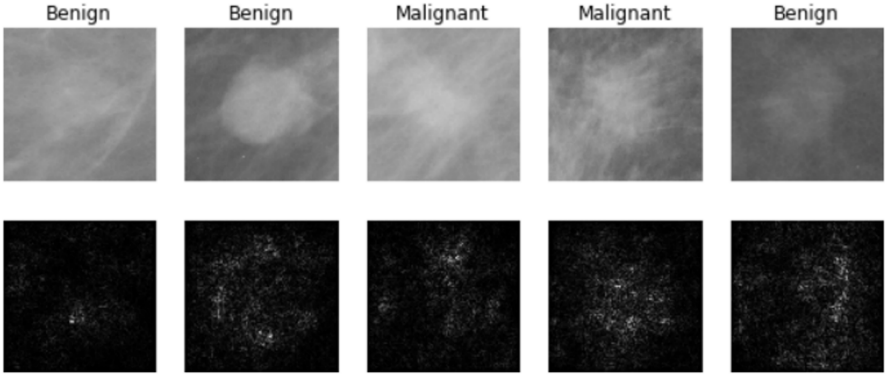


Some Applications (cont.)

Pose Recognition (Toshev and Szegedy, 2014)

Some Applications (cont.)

Medical Imaging



Some Applications (cont.)

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

Image Captioning

Vinyals et al, 2015
Karpathy and Fei-Fei, 2015

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Some Applications (cont.)

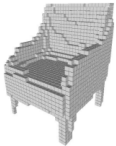
"Teddy bears working on new AI research underwater with 1990s technology"

DALL-E 2

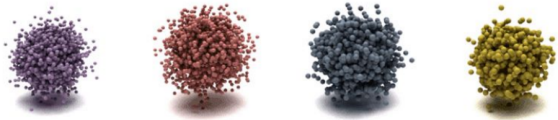
Some Applications (cont.)

Style Transfer

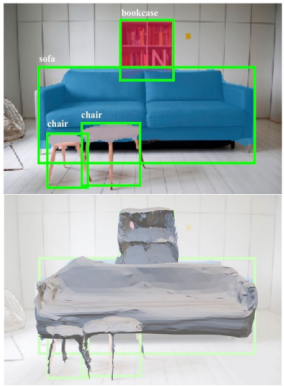
Some Applications (cont.)



Choy et al., 3D-R2N2: Recurrent Reconstruction Neural Network (2016)



Zhou et al., 3D Shape Generation and Completion through Point-Voxel Diffusion (2021)



Gkioxari et al., "Mesh R-CNN", ICCV 2019

Motivation

- **Transformative Impact:** Convolutional Neural Networks (CNNs) have revolutionized computer vision and deep learning.
- **Wide Applications:** Power tasks such as image classification, object detection, facial recognition, and medical image analysis.
- **Automatic Feature Learning:** CNNs learn hierarchical features directly from raw pixel data, reducing the need for manual feature engineering.
- **Demystifying CNNs:** This presentation breaks down core components and traces the evolution of key architectures.
- **Practical Relevance:** Understanding CNNs enables better design, optimization, and application to real-world problems.

⇒ Learning Outcomes

By the end of this session, you will be able to:

- Understand how images are represented in computers
- **Define CNNs:** Understand what a Convolutional Neural Network is and its role in deep learning.
- **Explain Key Components:**
  - Convolutional layers
  - Activation functions
  - Pooling layers
  - Padding and strides
  - Batch normalization
  - Dropout regularization
  - Flattening and fully connected layers

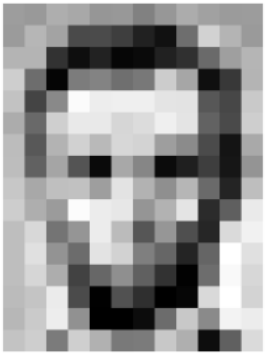
⇒ Learning Outcomes (cont.)

- **Differentiate Training Techniques:** Compare loss functions and optimization methods used in CNNs.
- Write and train a simple CNN using PyTorch

CNN: **Introduction to Convolutional Neural Networks**

Why Do We Need Special Models for Images?

Images are grids/matrix of numbers!



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Images are **BIG** (e.g., $256 \times 256 \times 3 \approx 200{,}000$ numbers)

Why Do We Need Special Models for Images? (cont)

Fully Connected Neural Networks:

- Too many parameters
- Miss spatial information

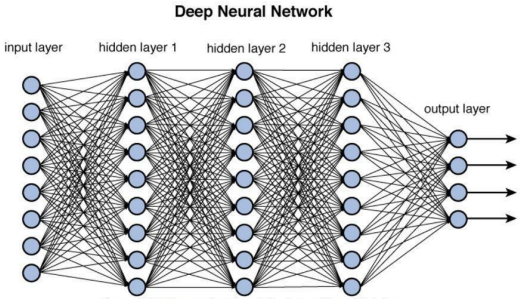


Figure 12.2 Deep network architecture with multiple layers.

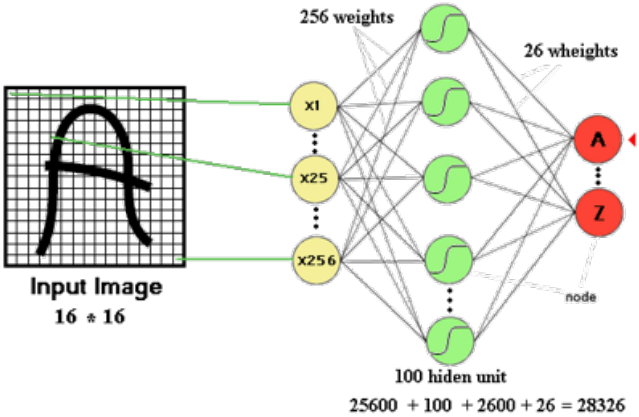
$$z = W_1 x_1 + W_2 x_2 + \cdots + W_n x_n + b$$

[a]

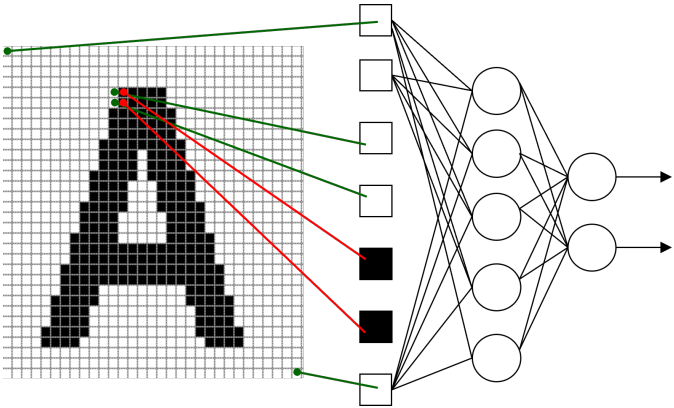
[a] https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Drawbacks of Fully-Connected Neural Networks

- The number of trainable parameters becomes extremely large

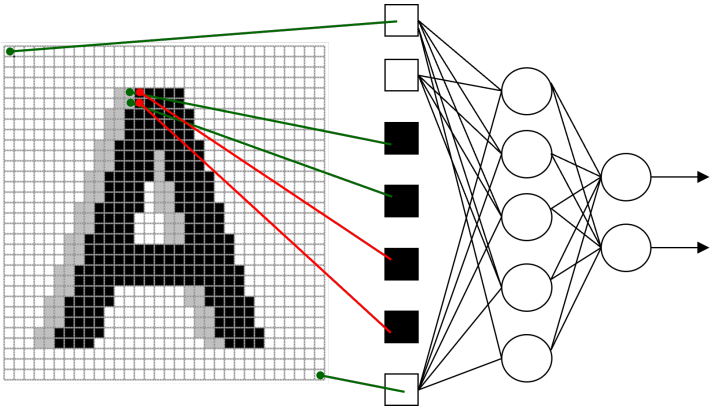


- Little or no invariance to shifting, scaling, and other forms of distortion



Drawbacks of Fully-Connected Neural Networks (cont.)

- Little or no invariance to shifting, scaling, and other forms of distortion

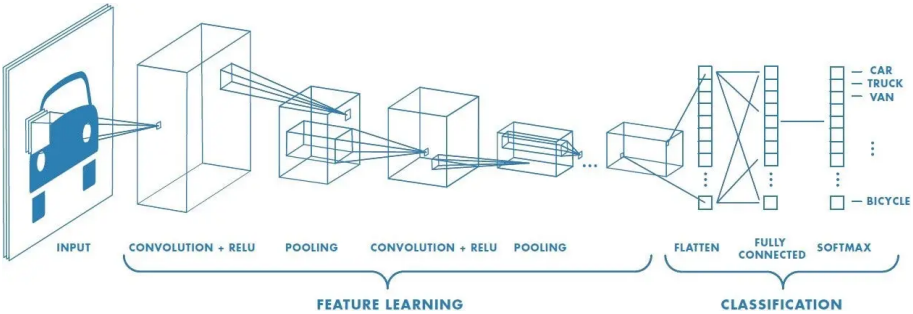


Drawbacks of Fully-Connected Neural Networks (cont.)

- The topology of the input data is completely ignored
- For a 32×32 image, we have
  - Black and white patterns: $2^{32*32} = 2^{1024}$
  - Grayscale patterns: $256^{32*32} = 256^{1024}$

We need something smarter:

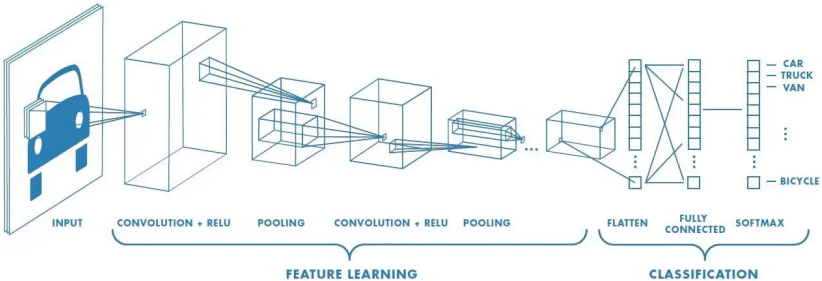
Convolution!



$$z = W * x_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} W_{ab} x_{(i+a)(j+b)}$$

What is Convolution?

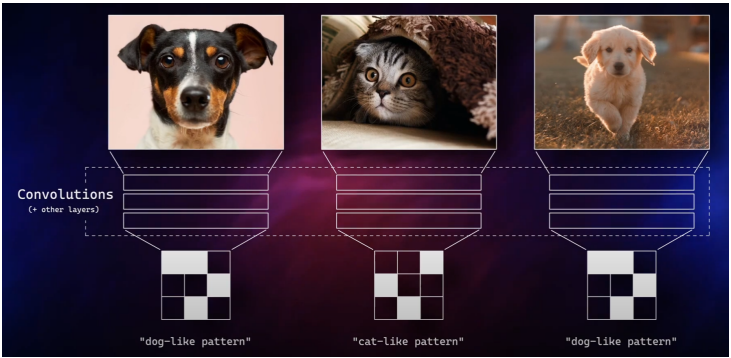
- A way to process local areas of the image
- Slide a small filter (kernel) over the image
- At each step, do element-wise multiply and sum
- Outputs a feature map (like a filtered image)



CNN: **Architecture**

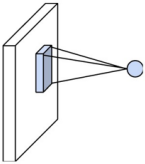
What makes a Convolutional Neural Network?

Characterised by "Convolutional Layer" – they are able to detect "abstract features" and "almost ideas within the image"

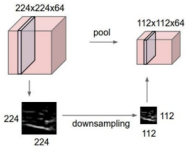


Components of a CNN

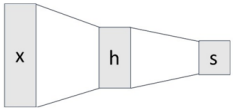
Convolution Layers



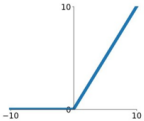
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

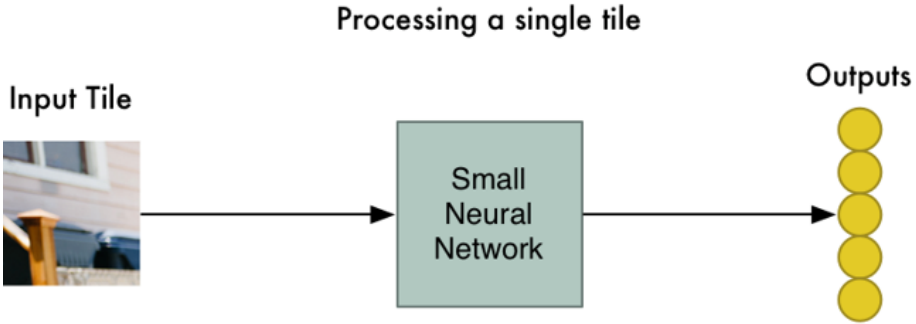
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

CNN: **How Does Convolution Work?**

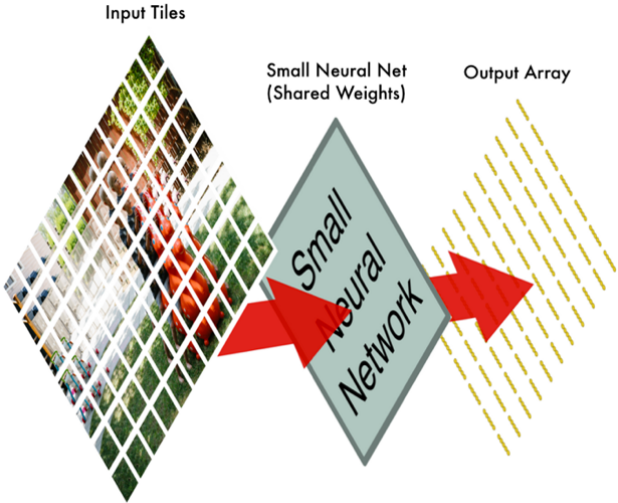
How Does Convolution Work?

How Does Convolution Work? (cont.)

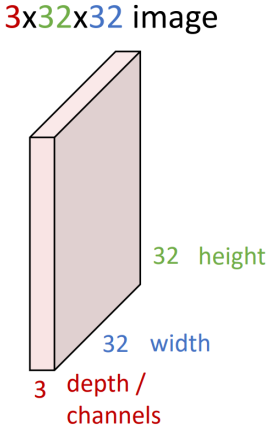
How Does Convolution Work? (cont.)

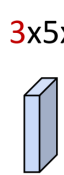


How Does Convolution Work? (cont.)



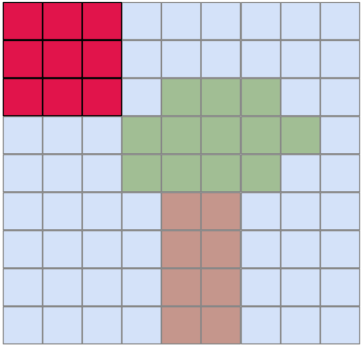
How Does Convolution Work? (cont.)

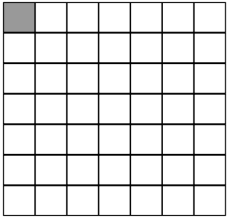




Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

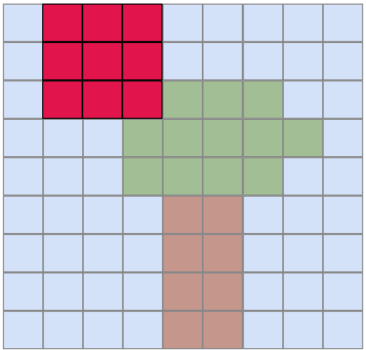
How Does Convolution Work? (cont.)

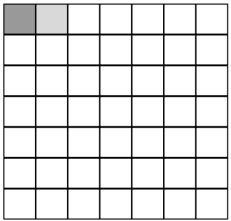




The **kernel** slides across the image and produces an output value at each position

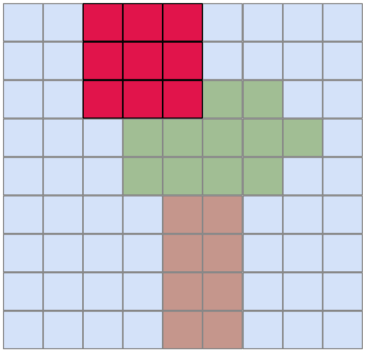
How Does Convolution Work? (cont.)

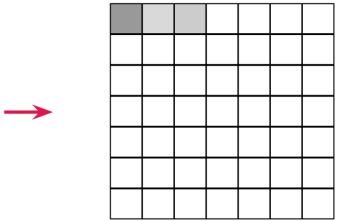




The **kernel** slides across the image and produces an output value at each position

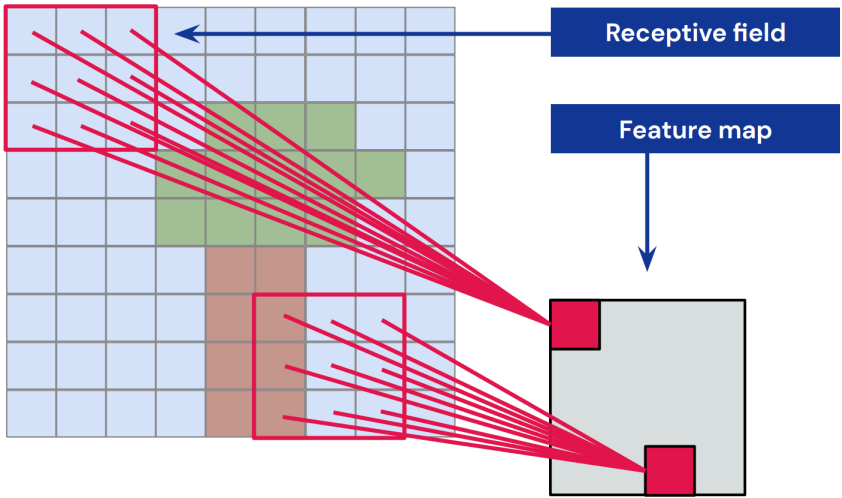
How Does Convolution Work? (cont.)



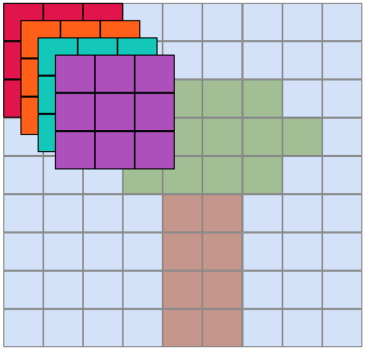


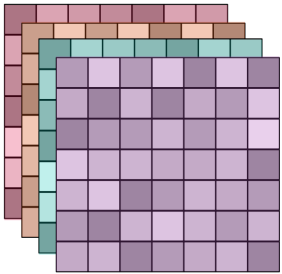
The **kernel** slides across the image and produces an output value at each position

How Does Convolution Work? (cont.)



How Does Convolution Work? (cont.)





We convolve multiple kernels and obtain multiple feature maps or **channels**

How Does Convolution Work? (cont.)

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

CNN: **Layers**

Layers of a CNN

- Convolutional layers
- Pooling layers
- Fully connected layers

CNN Layer: **Convolutional Layer**

Convolutional Layer

Operation:

- Element-wise multiply filter with image patch and sum → feature map.

Hyperparameters:

- kernel size
- number of filters
- stride
- padding

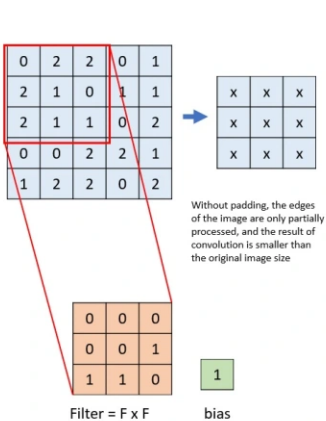
Listing 1: Code snippet (PyTorch)

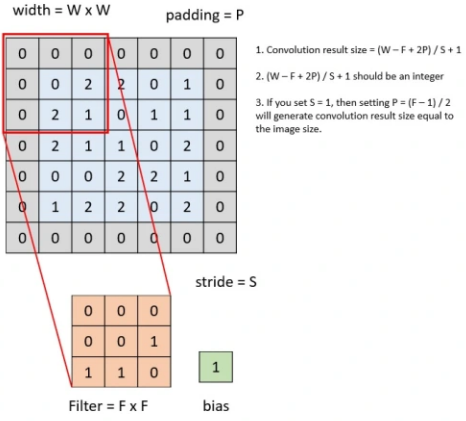
```
import torch.nn as nn

conv = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3,
    stride=1, padding=1)

output = conv(input_tensor)  # input_tensor: [batch_size, 3, H, W]
```

CNN - Convolutional Layer

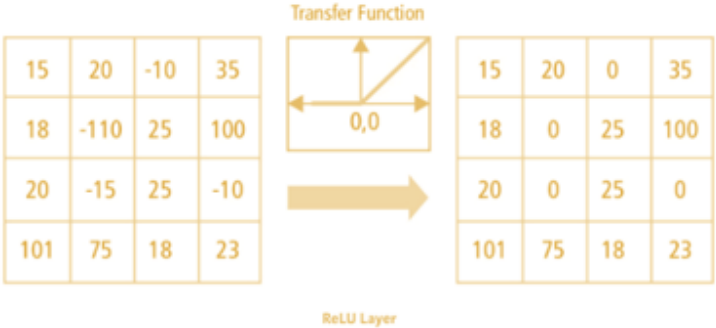




Interactive demo: cs231n.github.io/convolutional-networks

Activation

- Just like Fully-Connected Neural Networks, we can apply an activation over convolutional layer outputs
- It helps break linearity
- For example, Rectified Linear Unit (ReLU): $\sigma(x) = max(0, x)$



Activation Functions

Role:

- Introduce non-linearity so multiple conv layers can learn complex mappings.

Common:

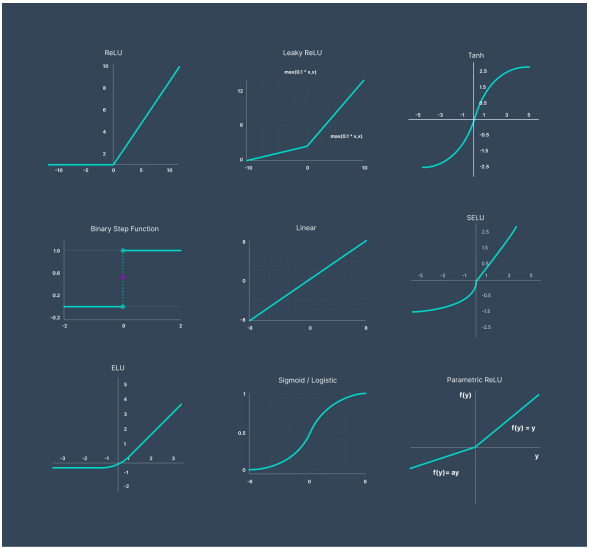
- ReLU (Rectified Linear Unit)
- Leaky ReLU
- Sigmoid
- Tanh

Listing 2: Code snippet (PyTorch)

```
import torch.nn.functional as F

x = conv(input_tensor)
x = F.relu(x)                # ReLU
x = F.leaky_relu(x, 0.1)     # Leaky ReLU
```

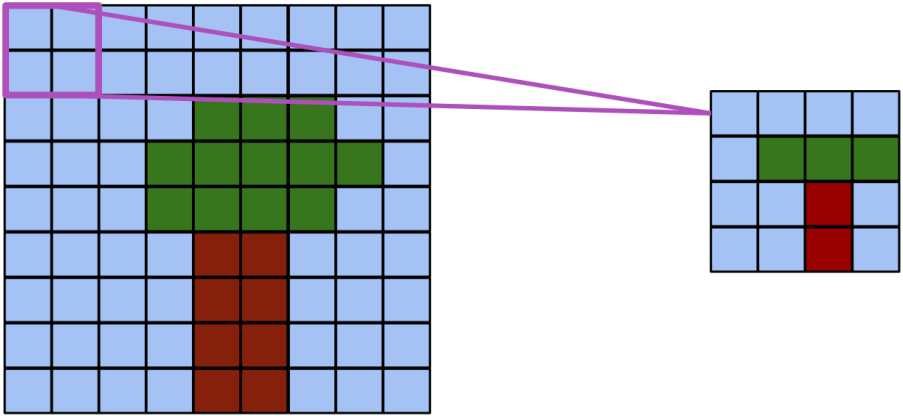
Activation Functions



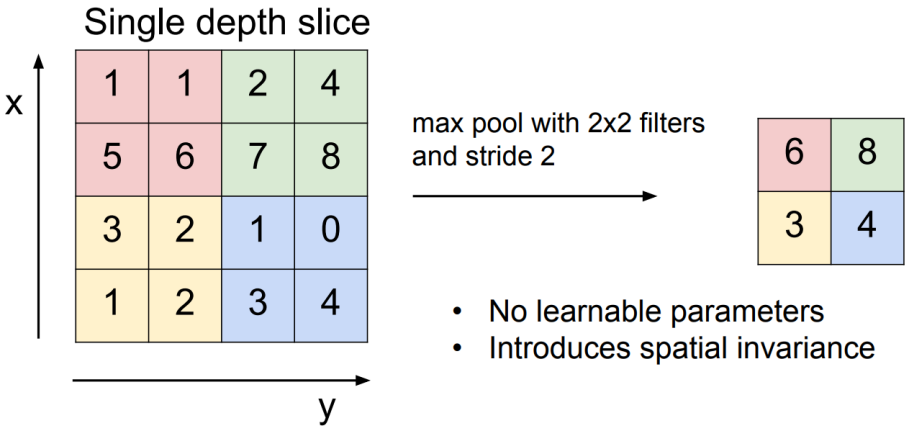
CNN Layer: **Pooling Layers**

Pooling

- Compute mean or max over small windows to reduce resolution



Pooling (cont.)



- No learnable parameters
- Introduces spatial invariance

Pooling Layers

Purpose:

- Downsample feature maps, reduce spatial dims and parameters, add invariance.

Types:

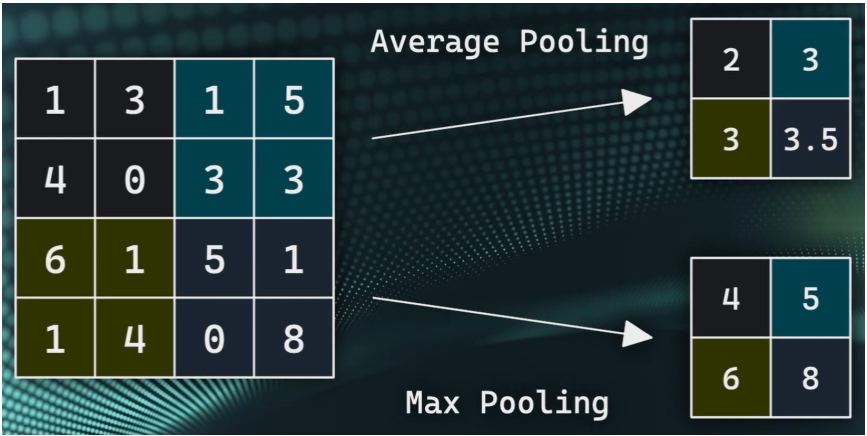
- Max Pooling
- Average Pooling
- Global Average Pooling
- Global Max Pooling

Listing 3: Code snippet (PyTorch)

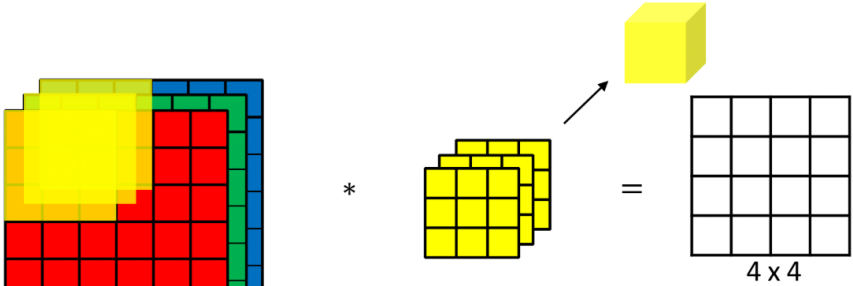
```
import torch.nn as nn

pool = nn.MaxPool2d(kernel_size=2, stride=2)
pooled = pool(x)  # halves H and W
```

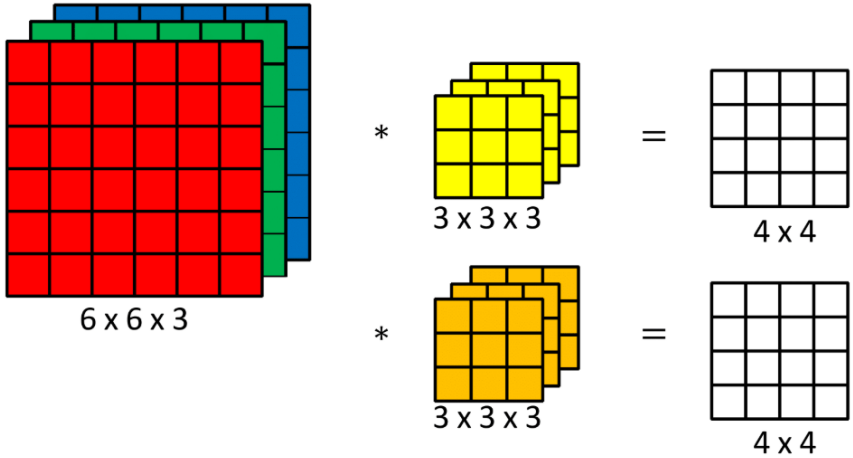
Pooling Layers



Convolutions on RGB images



Multiple Channels & Multiple Filters



Number of Parameters?

If you have 10 filters of size $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?

Each filter has size $3 \times 3 \times 3 = 27$ parameters (weights).

With 10 filters: $10 \times 27 = 270$ weights.

If each filter also has a bias term, add 10 more parameters: $270 + 10 = 280$ parameters in total.

CNN Layer: **Flattening Fully Connected Layers**

Flattening Fully Connected Layers

Flatten:

- Flattening is the process of converting a multi-dimensional tensor into a one-dimensional tensor.
- Flattening is typically done before passing the data to fully connected layers.

FC Layer:

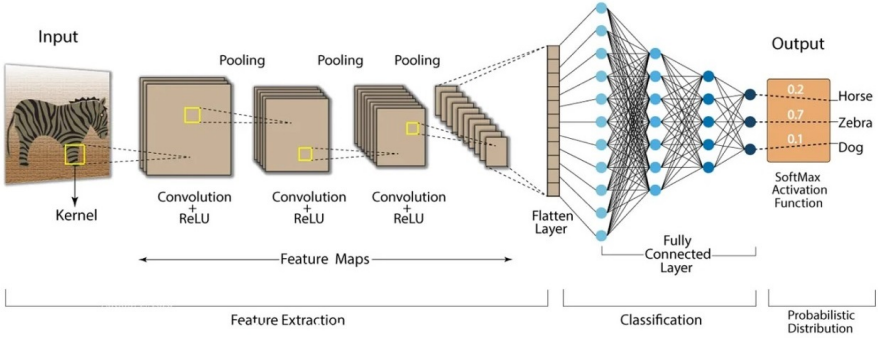
- Used to learn complex relationships between features.
- Typically used at the end of the network to make predictions.
- Take the flattened output from the previous layer and produce a vector of class scores.
- Are often followed by activation functions like ReLU or softmax.

Listing 4: Code snippet (PyTorch)

```
import torch.nn as nn
import torch

x = torch.flatten(x, 1)  # preserve batch dim
fc = nn.Linear(in_features=16*8*8, out_features=10)
out = fc(x)
```

CNN - Flattening Fully Connected Layers



CNN: **Key Terms - Hyperparameters in CNNs**

Key Terms in CNNs

- Kernels Filters
- Padding
- Strides
- Feature Maps

CNN Hyperparameter: **Kernels and Filters**

Kernels and Filters

- Kernels and filters are the same thing in CNNs.
- They are small matrices used to extract features from input data.
- The kernel **slides over** the input image, performing **element-wise** multiplication and summation.
- This process is known as **convolution**.

Kernels and Filters (cont.)

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

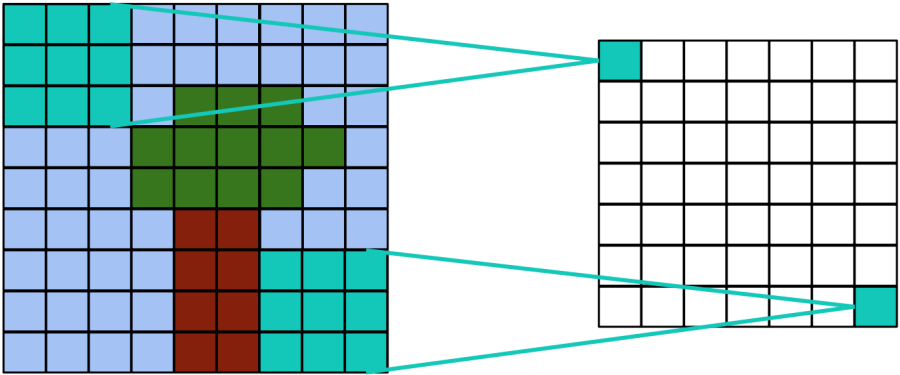
Kernels and Filters (cont.)

- The kernel size is typically smaller than the input image (e.g., 3x3, 5x5).
- Multiple kernels can be used in a single convolutional layer to extract different features.
- Each kernel produces a feature map, which is a transformed version of the input image highlighting specific features.
- The number of filters (kernels) in a layer determines the depth of the output feature map.

CNN Hyperparameter: **Padding**

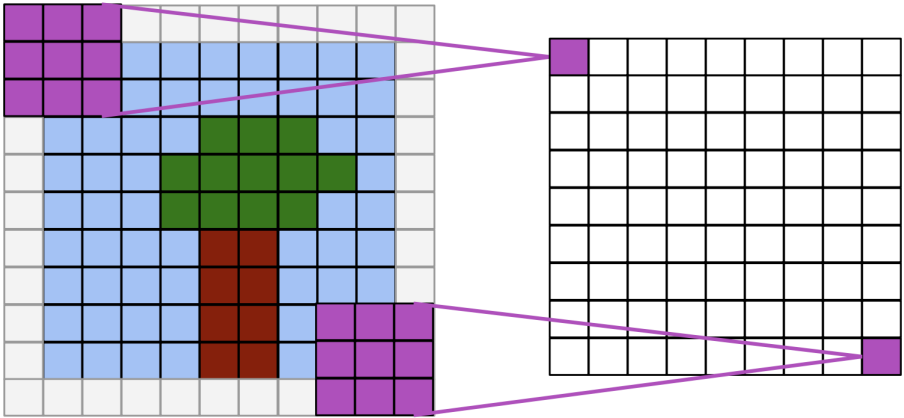
Padding

- Applying Convolution as such reduces the size of the borders.
- Sometimes this is not desirable.
- We can pad the border with zeros.



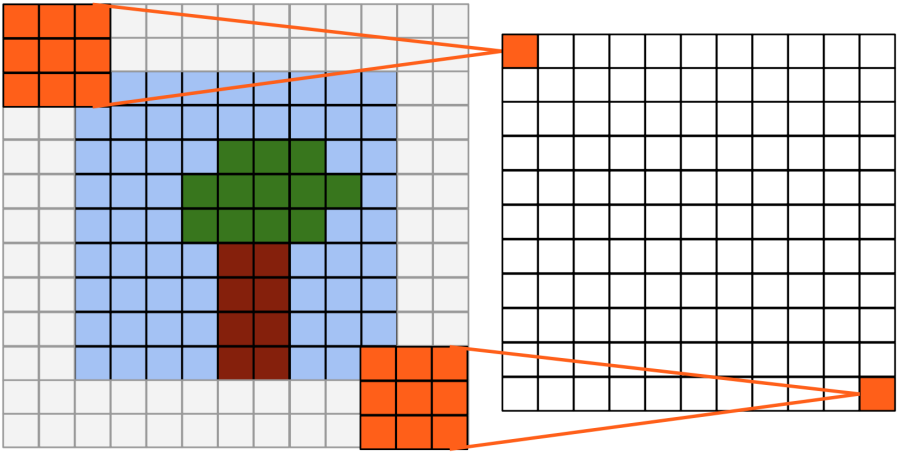
Padding (cont.)

- Same Convolution: Output is the same size as input



Padding (cont.)

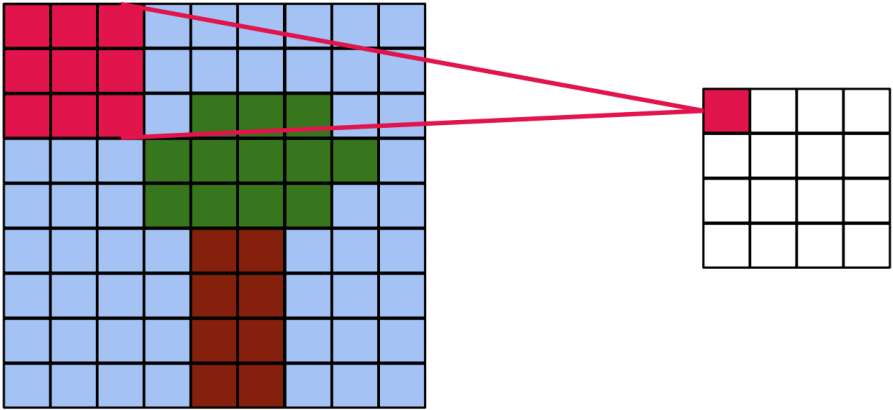
- Full Convolution: output size = input size + kernel size - 1



CNN Hyperparameter: **Strides**

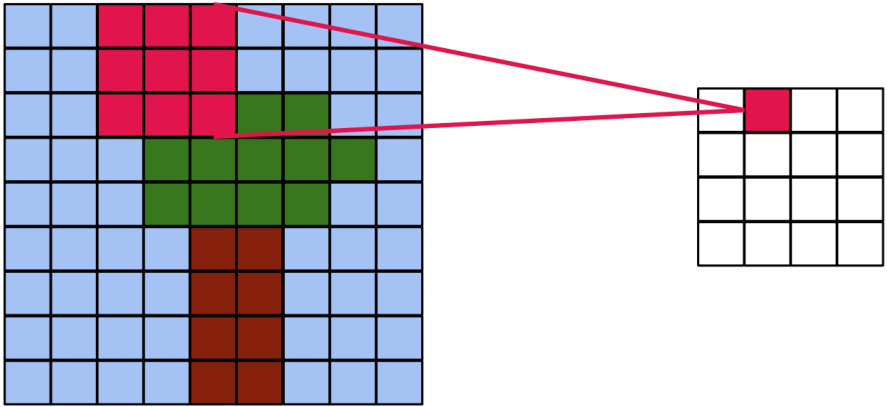
Strided Convolution

- Kernel slides along the image with a step > 1



Strided Convolution (cont.)

- Kernel slides along the image with a step > 1



CNN Hyperparameter: **Padding & Strides Together**

Padding & Strides

Padding:

- "same" preserves spatial size by adding zeros around input;
- "valid" reduces spatial size by not adding any padding.
- Padding is used to control the spatial size of the output feature map.
- Padding is added to the input image before applying the convolution operation.

Strides:

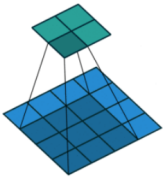
- Strides control how much the filter moves across the input image.
- Strides can be set independently for height and width.
- Strides are used to control the spatial size of the output feature map.
- Strides are set in the convolutional layer.

Listing 5: Code snippet (PyTorch)

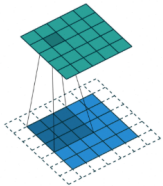
```
import torch.nn as nn

conv_valid = nn.Conv2d(3, 16, 3, stride=2, padding=0) #
       Valid    : no padding (padding=0)
conv_same  = nn.Conv2d(3, 16, 3, stride=1, padding=1) # Same
```

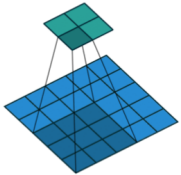
Padding & Strides



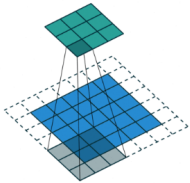
padding = 0, stride = 1



padding = 1, stride = 1



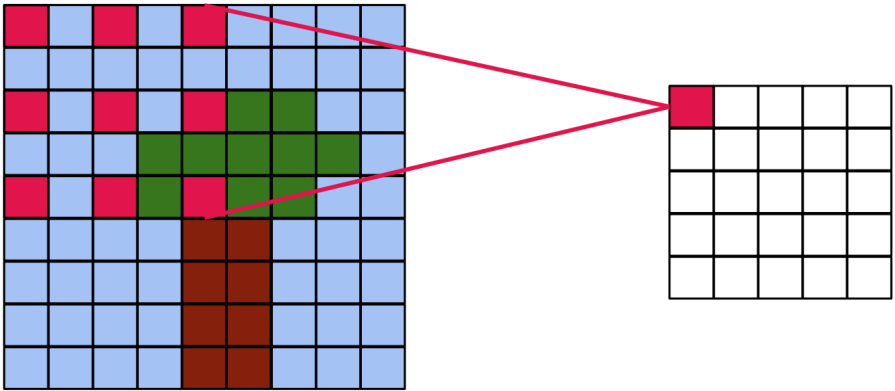
padding = 0, stride = 2



padding = 1, stride = 2

Dilated Convolution

- Kernel is spread out, step > 1 between kernel elements



Output Shape

- Output shape can be calculated as:

$$O = \left\lfloor \frac{I + 2P - K}{S} + 1 \right\rfloor$$

where:

- O = output size
- I = input size
- P = padding
- K = kernel size
- S = stride

CNN: **Flattening Fully Connected Layers**

Flattening Fully Connected Layers

Flatten:

- Flattening is the process of converting a multi-dimensional tensor into a one-dimensional tensor.
- Flattening is typically done before passing the data to fully connected layers.

FC Layer:

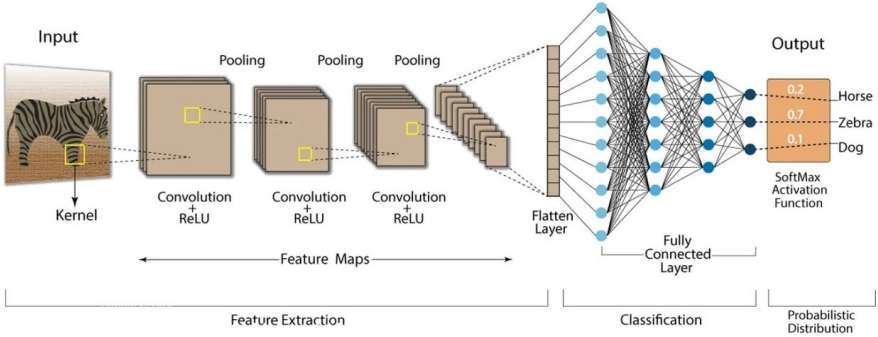
- Used to learn complex relationships between features.
- Typically used at the end of the network to make predictions.
- Take the flattened output from the previous layer and produce a vector of class scores.
- Are often followed by activation functions like ReLU or softmax.

Listing 6: Code snippet (PyTorch)

```
import torch.nn as nn
import torch

x = torch.flatten(x, 1)  # preserve batch dim
fc = nn.Linear(in_features=16*8*8, out_features=10)
out = fc(x)
```

Flattening Fully Connected Layers



CNN: **Implementing CNNs in PyTorch**

PyTorch Implementation of CNNs


```
import torch
import torch.nn as nn
import torch.nn.functional as F
class SimpleCNN(nn.Module):
   def __init__(self):
     super(SimpleCNN, self).__init__()
     self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
     self.pool = nn.MaxPool2d(2, 2)
     self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
     self.fc1 = nn.Linear(64 * 8 * 8, 128)
     self.fc2 = nn.Linear(128, 10) # assuming 10 classes
   def forward(self, x):
     x = self.pool(F.relu(self.conv1(x)))
     x = self.pool(F.relu(self.conv2(x)))
     x = x.view(-1, 64 * 8 * 8)
     x = F.relu(self.fc1(x))
     x = self.fc2(x)
     return x
```

PyTorch Implementation of CNNs (cont.)


```
import torch.optim as optim
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Training loop
for epoch in range(10):
   for images, labels in dataloader:
     optimizer.zero_grad()
     outputs = model(images)
     loss = criterion(outputs, labels)
     loss.backward()
     optimizer.step()
```

CNN: **Limitations**

Limitations of CNNs

- Require lots of labeled data
- Struggle with geometric transformations (rotation, scale)
- Can be fooled by adversarial examples
- Limited ability to capture global context

Alternatives or enhancements:

- Transformers for vision
- Capsule networks
- Data augmentation techniques

CNN: **Summary**

Summary of CNNs

- CNNs are specialized for image data
- They use convolutional layers to extract features
- Pooling layers reduce dimensionality
- Fully connected layers perform classification

Key Advantages:

- Translation invariance
- Reduced number of parameters
- Ability to learn hierarchical features

PyTorch makes it easy to build and train CNNs!

CNN: **References & Resources**

References & Resources

[1] Stanford CS231n Lecture Notes.
http://cs231n.stanford.edu/

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning, Chapter 9.

[3] PyTorch Official Documentation.
https://pytorch.org/docs/

[4] Jason Brownlee.
A Guide to CNNs.
https://machinelearningmastery.com/

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton.
"ImageNet Classification with Deep Convolutional Neural Networks."
NIPS 2012.

References & Resources (cont.)

[6] Visualizing CNNs.
https://distill.pub/2017/feature-visualization/

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.