Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.

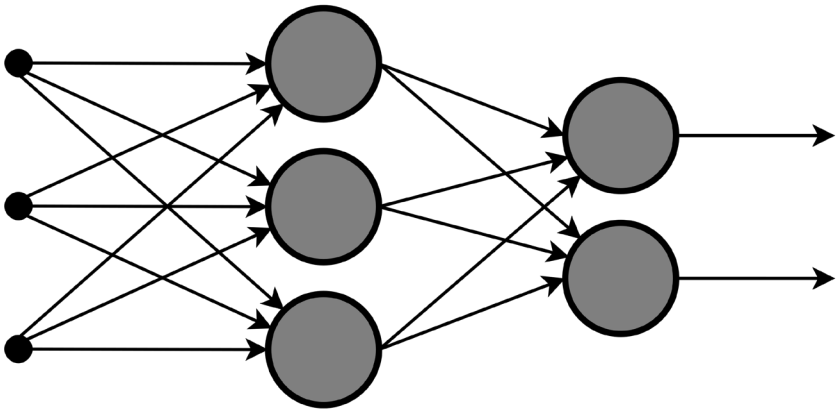
Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class. Releasing these recordings is strictly prohibited.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

SEAS 8520 Lecture 5 - CNNs

Walid Hassan, M.B.A, D.Eng.



Neural Networks

Early Stopping

Early Sopping - Example-FMNIST

patience=3, min_delta=0.001

Epochs 1-6	**Epochs 7-12**
Epoch 1: 0.3787	Epoch 7: 0.2650
Epoch 2: 0.3228	Epoch 8: 0.2589
Epoch 3: 0.3017	Epoch 9: 0.2480
Epoch 4: 0.2636	Epoch 10: 0.2608
Epoch 5: 0.2770	Epoch 11: 0.2560
Epoch 6: 0.2520	Epoch 12: 0.2567

Analysis of Early Stopping Logic

- Continuous Improvement Until Epoch 6:
  - From Epoch 1 to 6, there is a consistent decrease in validation loss, indicating continuous improvement in the model's performance.
  - The lowest validation loss at this stage is at Epoch 6 (0.2520).

- Increase and Fluctuations in Validation Loss (Epochs 7-9):
  - The validation loss slightly increases at Epochs 7 and 8, but then decreases again at Epoch 9 to 0.2480, which is lower than any previous epoch.
  - This decrease at Epoch 9 resets the early stopping counter as it is a significant improvement.

Analysis of Early Stopping Logic

- Evaluating Epochs 10-12:
  - Post-Epoch 9, we look for an improvement in the validation loss for the next 3 epochs.
  - The losses at Epochs 10 (0.2608), 11 (0.2560), and 12 (0.2567) do not improve beyond the loss at Epoch 9.

- Triggering Early Stopping:
  - Since the validation loss does not decrease by at least 0.001 for 3 consecutive epochs (10, 11, 12) after the reset at Epoch 9, the early stopping mechanism is activated.
  - The training stops after Epoch 12 to prevent overfitting and save computational resources.

Adam

Adam is an Adaptive learning rate that adjusts the learning rate for each parameter individually, taking into account the historical gradients. This means that parameters with large gradients (indicating steep loss landscape) will have their learning rate reduced to prevent overshooting, while parameters with small gradients (indicating a flat loss landscape) will have their learning rate increased to speed up convergence.

Use Cases: Effective across various applications, notably in settings with sparse gradients like NLP and Computer Vision.

Adam: Adaptive Moment Estimation

Normalize:

A straightforward approach is to normalize the gradients so that we move a fixed distance (governed by the learning rate) in each direction. To do this, we first measure the gradient $\mathbf{m}_{t+1}$ and the pointwise squared gradient $\mathbf{v}_{t+1}$:

$$\begin{aligned} \mathbf{m}_{t+1} &\leftarrow \frac{\partial L[\phi_t]}{\partial \phi} \\ \mathbf{v}_{t+1} &\leftarrow \left(\frac{\partial L[\phi_t]}{\partial \phi}\right)^2. \end{aligned} \tag{6.13}$$

Then we apply the update rule:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}, \tag{6.14}$$

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Adam

Add Momentum:

$$\begin{aligned}\mathbf{m}_{t+1} &\leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}} \\ \mathbf{v}_{t+1} &\leftarrow \gamma \cdot \mathbf{v}_t + (1-\gamma)\left(\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}\right)^2, \end{aligned} \tag{6.15}$$

where β and γ are the momentum coefficients for the two statistics.

Using momentum is equivalent to taking a weighted average over the history of each of these statistics. At the start of the procedure, all the previous measurements are effectively zero, resulting in unrealistically small estimates. Consequently, we modify these statistics using the rule:

$$\begin{aligned}\tilde{\mathbf{m}}_{t+1} &\leftarrow \frac{\mathbf{m}_{t+1}}{1-\beta^{t+1}} \\ \tilde{\mathbf{v}}_{t+1} &\leftarrow \frac{\mathbf{v}_{t+1}}{1-\gamma^{t+1}}. \end{aligned} \tag{6.16}$$

Since β and γ are in the range $[0, 1)$, the terms with exponents $t+1$ become smaller with each time step, the denominators become closer to one, and this modification has a diminishing effect.

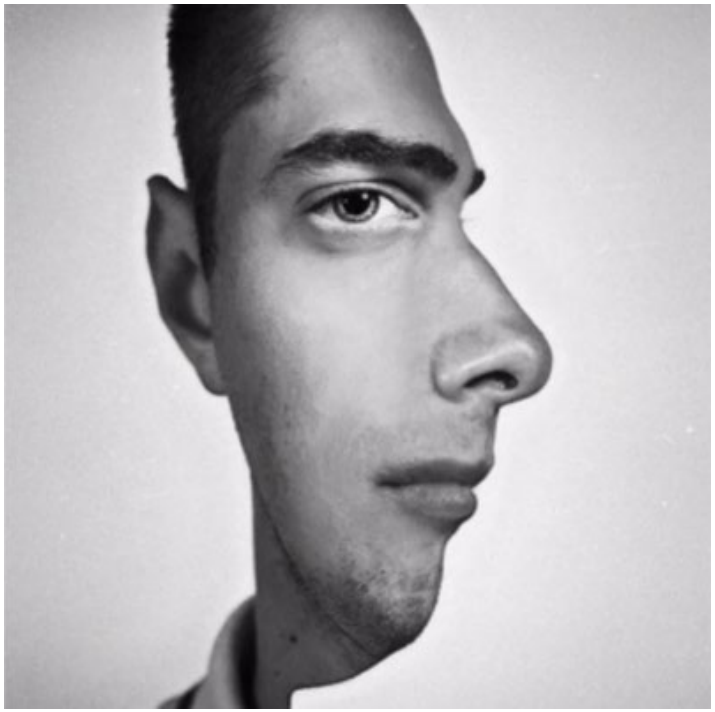
Finally, we update the parameters as before, but with the modified terms:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}} + \epsilon}. \tag{6.17}$$

$$\begin{aligned}\mathbf{m}_{t+1} &\leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\sum_{i \in \mathcal{B}_t}\frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}} \\ \mathbf{v}_{t+1} &\leftarrow \gamma \cdot \mathbf{v}_t + (1-\gamma)\sum_{i \in \mathcal{B}_t}\left(\frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}\right)^2, \end{aligned} \tag{6.18}$$

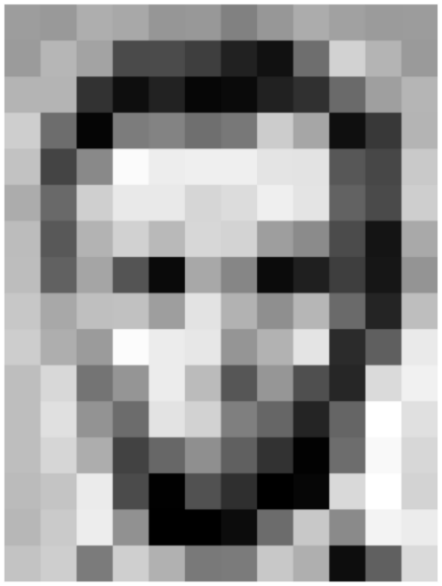
Prince, S. J. (January 28, 2024). Understanding Deep Learning.

What do You See?

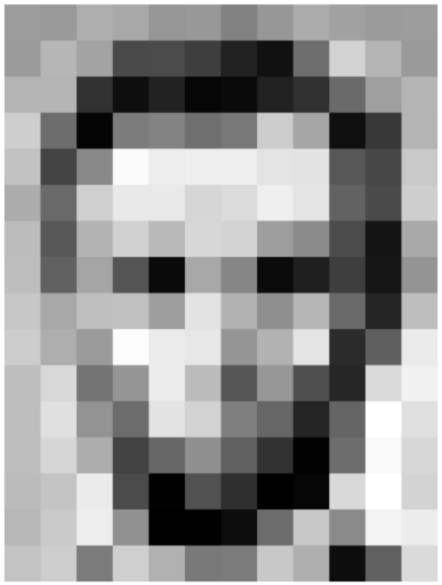


What do You See?

What do Computers See?

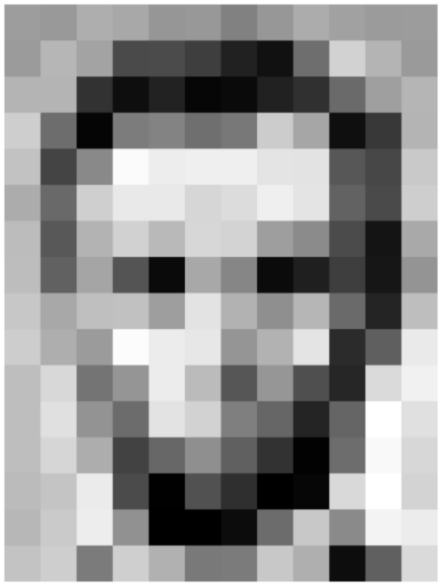


Images are Numbers



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	106	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	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
196	206	123	207	177	121	123	200	175	13	96	218

Images are Numbers



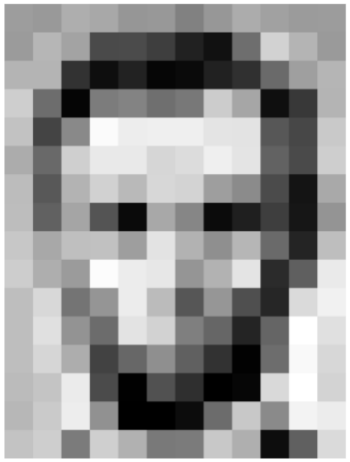
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	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	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

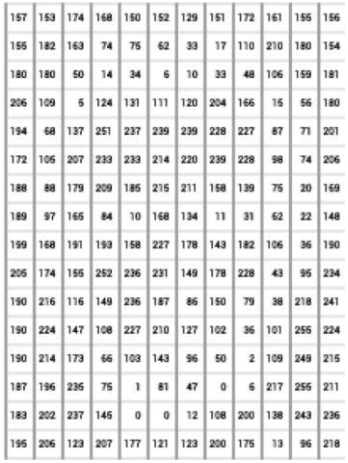
What the computer sees

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

An image is just a matrix of numbers [0,255]
ex. 1080x1080x3 for an RGB image

Tasks in Computer Vision





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

classification

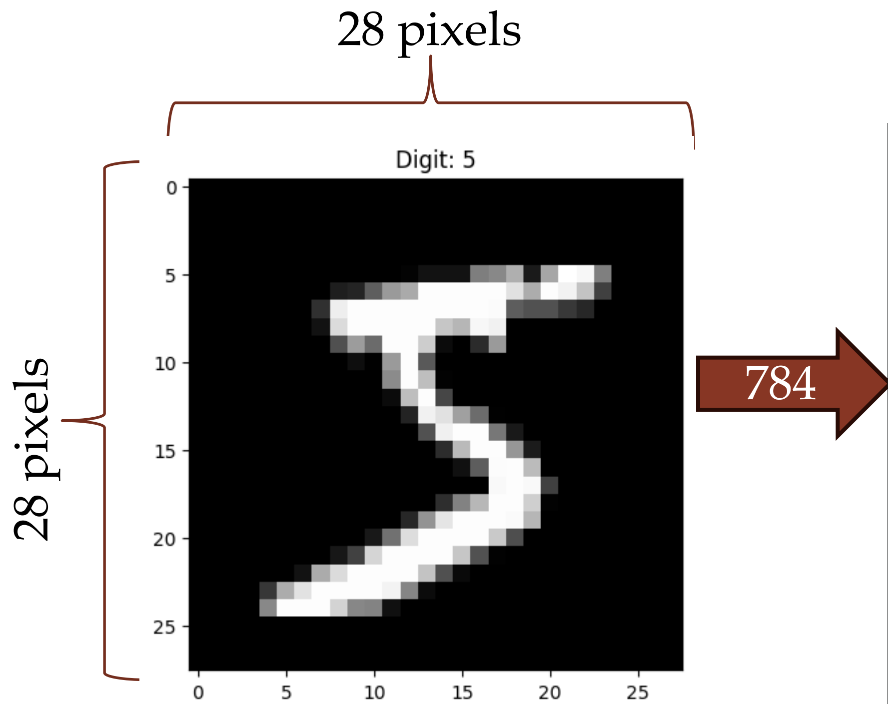
Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

Input Image

Pixel Representation

- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

Images as “seen” by ANN




```
Pixel values (0-255) of the image:
[[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [   0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [   0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [   0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
http://introtodeeplearning.com/

High Level Feature Detection

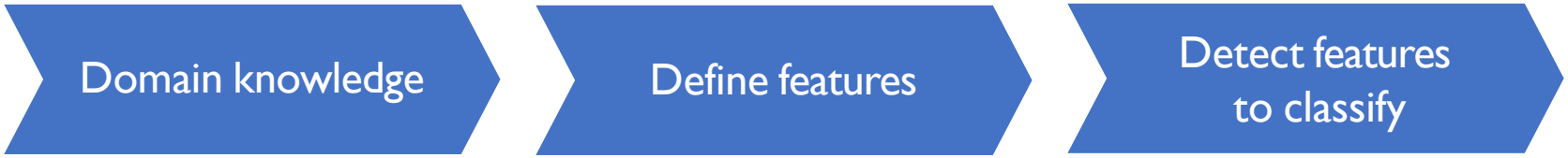
Let's identify key features in each image category

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

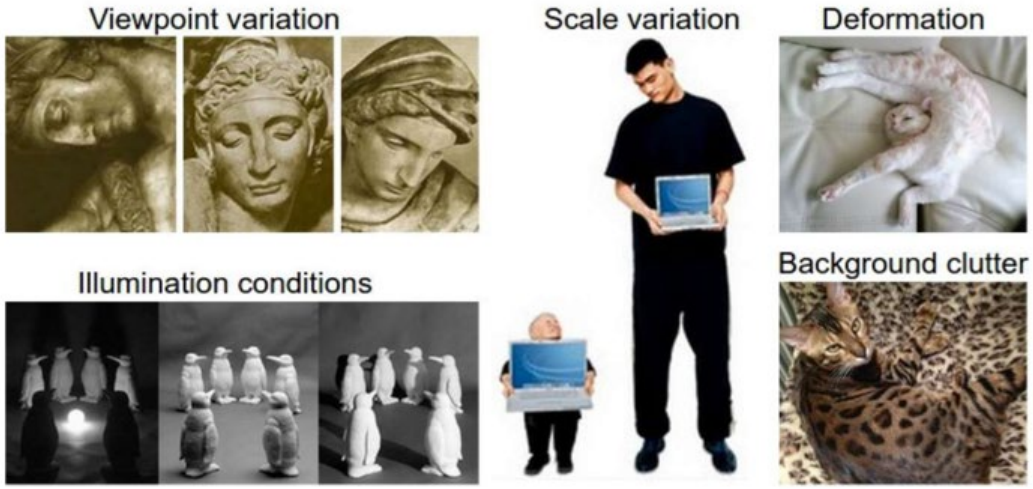
Manual Feature Extraction



Problems?

Manual Feature Extraction

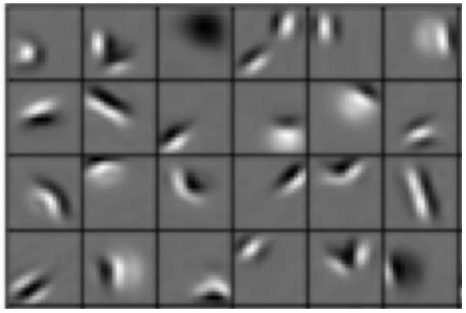
Domain knowledge → Define features → Detect features to classify



Learning Feature Representations

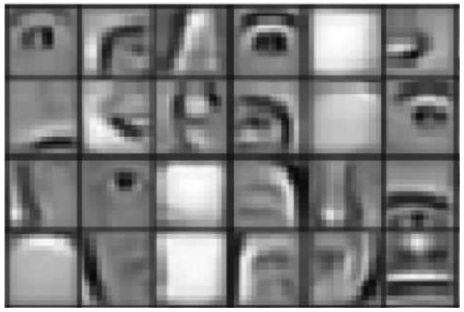
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

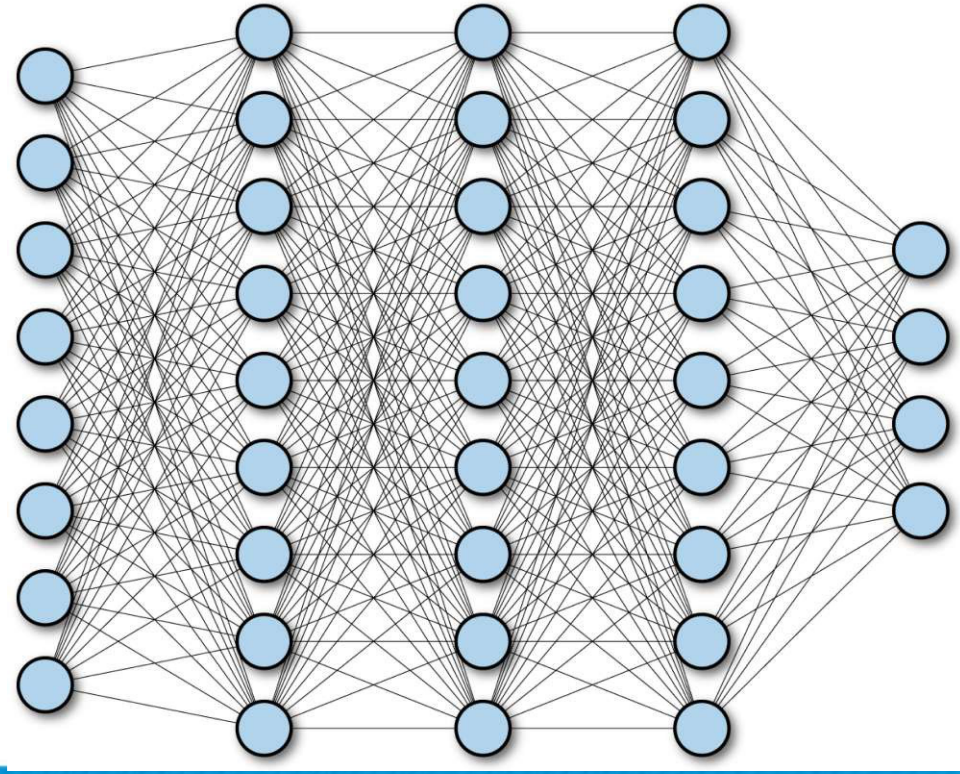
High level features

Facial structure

Convolutional Neural Networks (CNNs)

- **Definition**: CNNs are a category of deep neural networks specialized for analyzing visual imagery.
- **Local Connectivity**: Unlike traditional neural networks, CNNs connect each neuron to only a local region of the input data, capturing spatial relationships and local patterns.
- **Applications**: Beyond image classification, CNNs are used for object detection, image segmentation, facial recognition, and even in non-image tasks like audio processing or time series analysis.

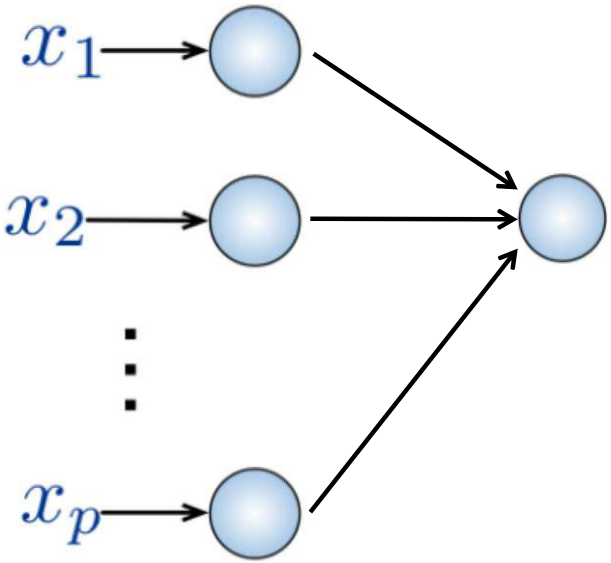
Fully Connected Neural Network



Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



Fully Connected:

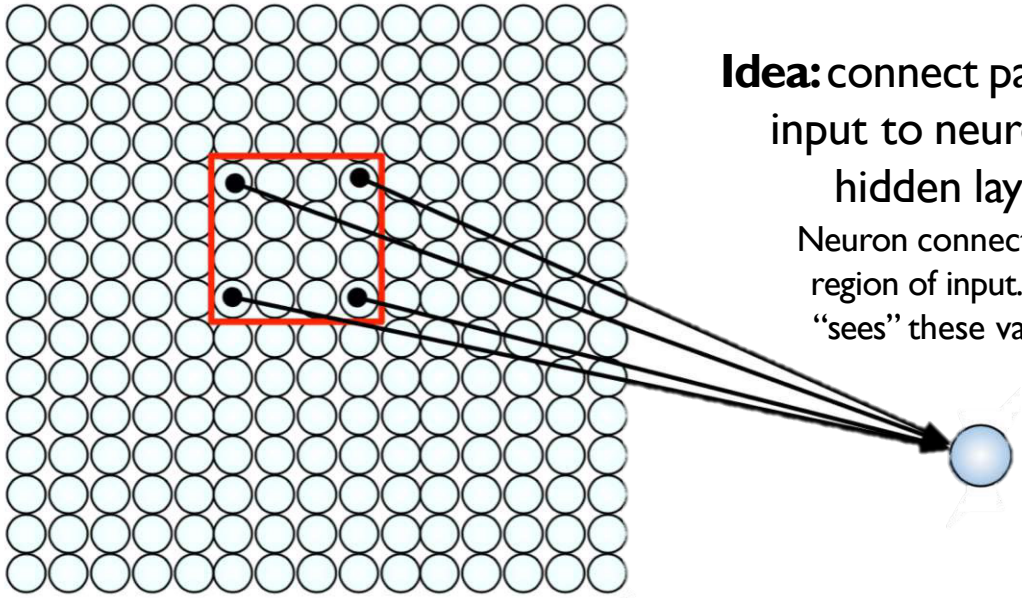
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Welcome to Convolution

- **Purpose**: Extracts local features and detects spatial patterns from input data, making it essential for image processing tasks.
- **Operation**: Filters (or kernels) slide over the input data to produce feature maps. These filters are matrices of learnable parameters.
- **Shared Weights**: A single filter is applied across the entire input, enabling the network to detect a particular feature irrespective of its location.
- **Feature Detection**: Initial filters might detect edges, textures, or colors. As we delve deeper into the network, the features become more complex and abstract.

Using Spatial Structure

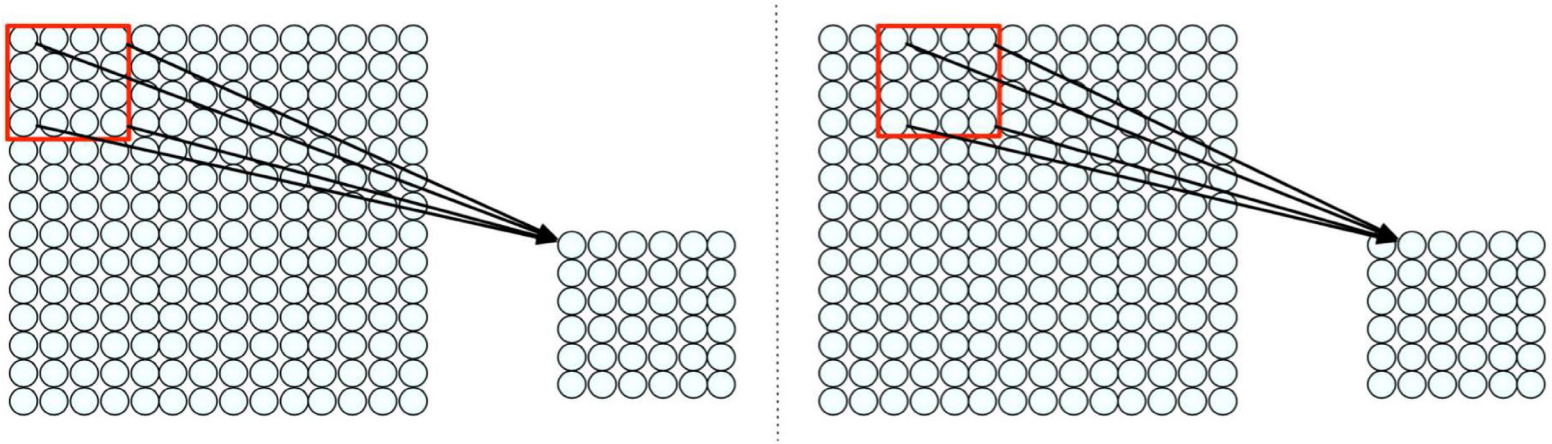
Input: 2D image. Array of pixel values



Idea: connect patches of input to neurons in hidden layer.

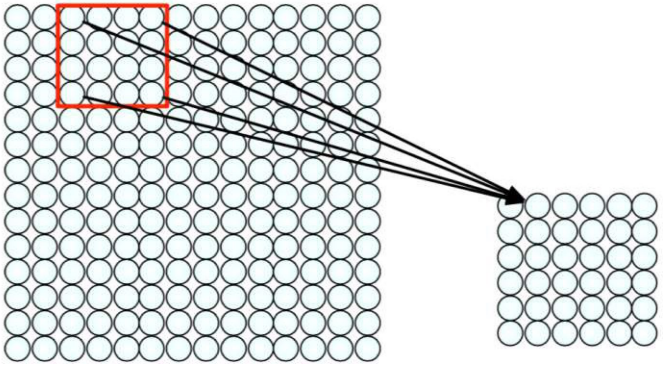
Neuron connected to region of input. Only "sees" these values.

Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by n pixels for next patch

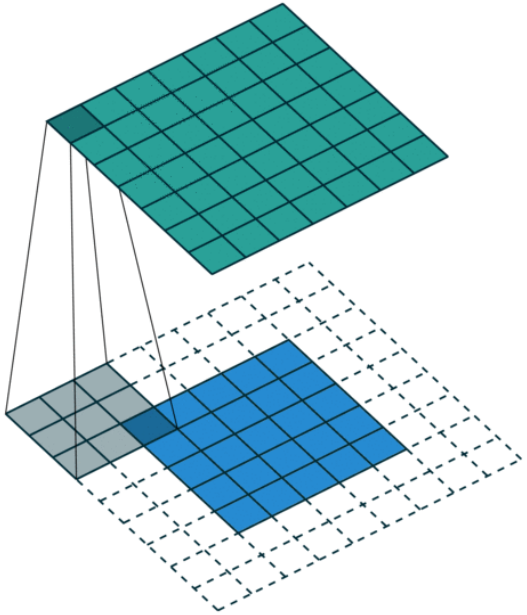
This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

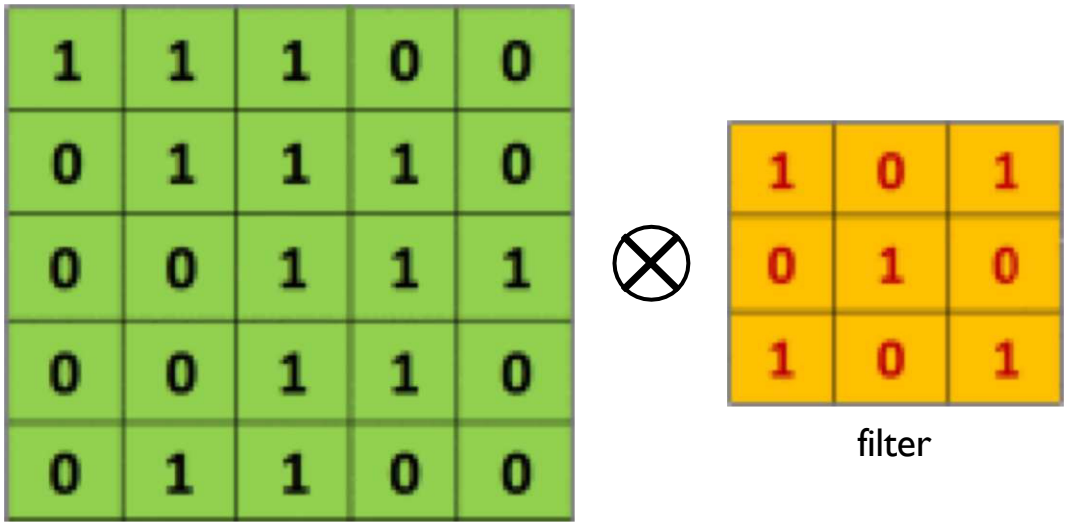
3) Spatially share parameters of each filter

Convolution animation



The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



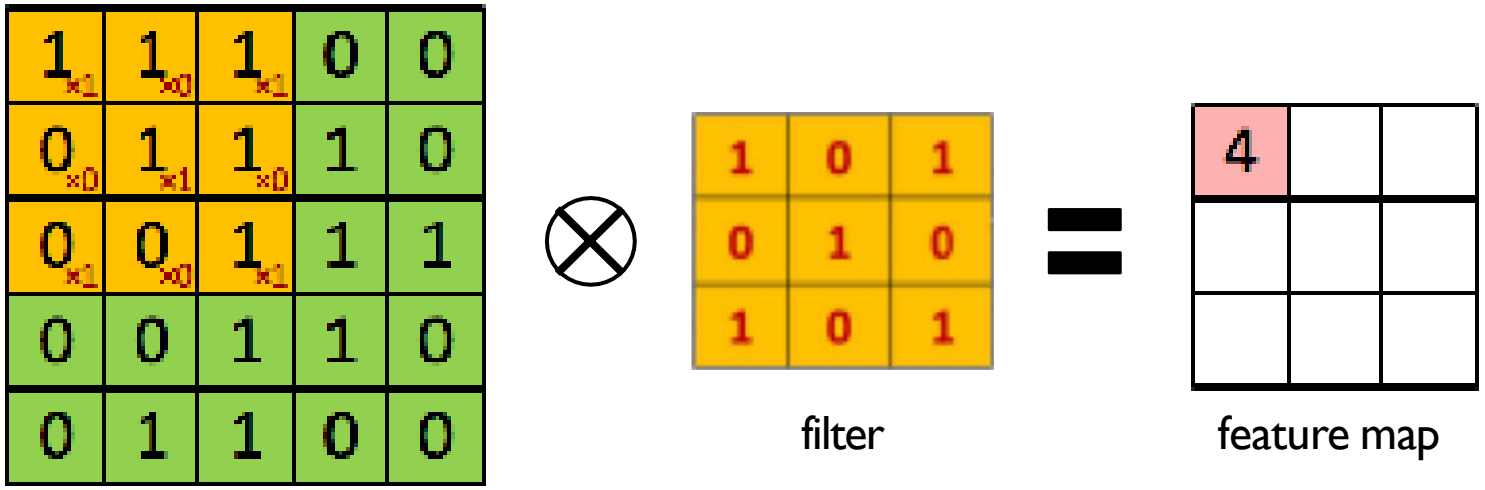
image

filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

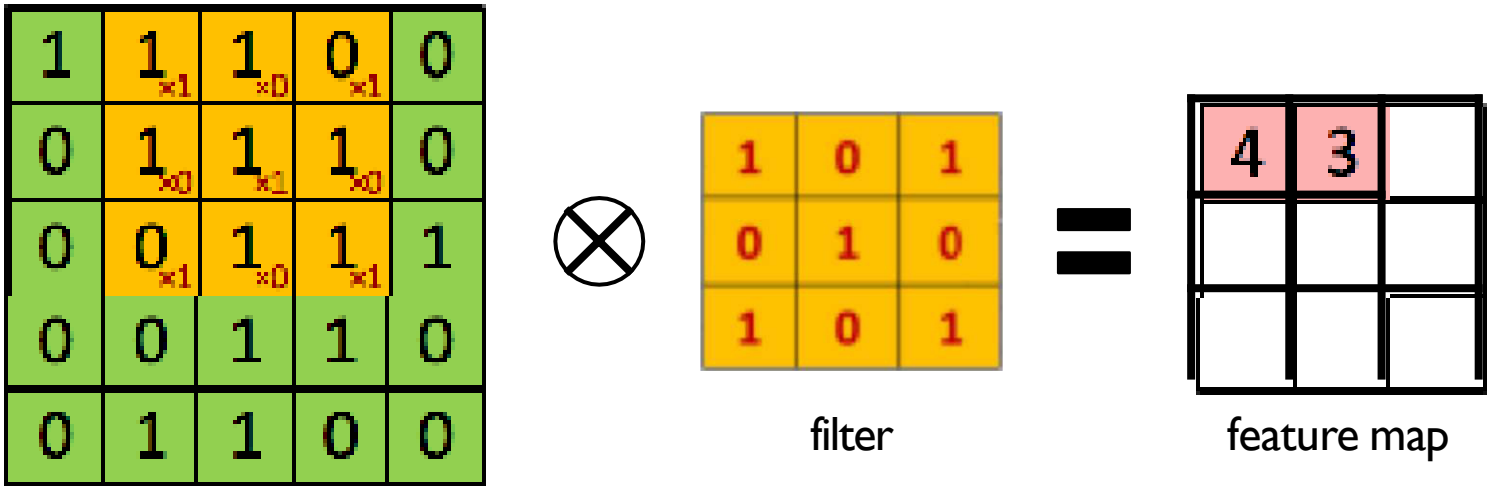
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



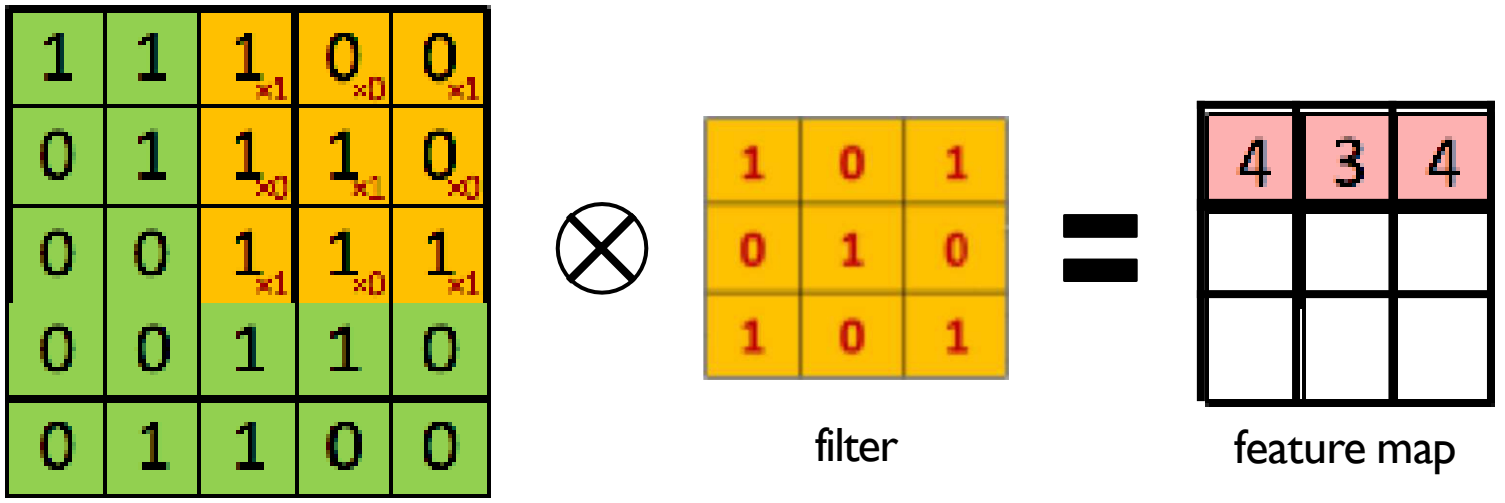
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



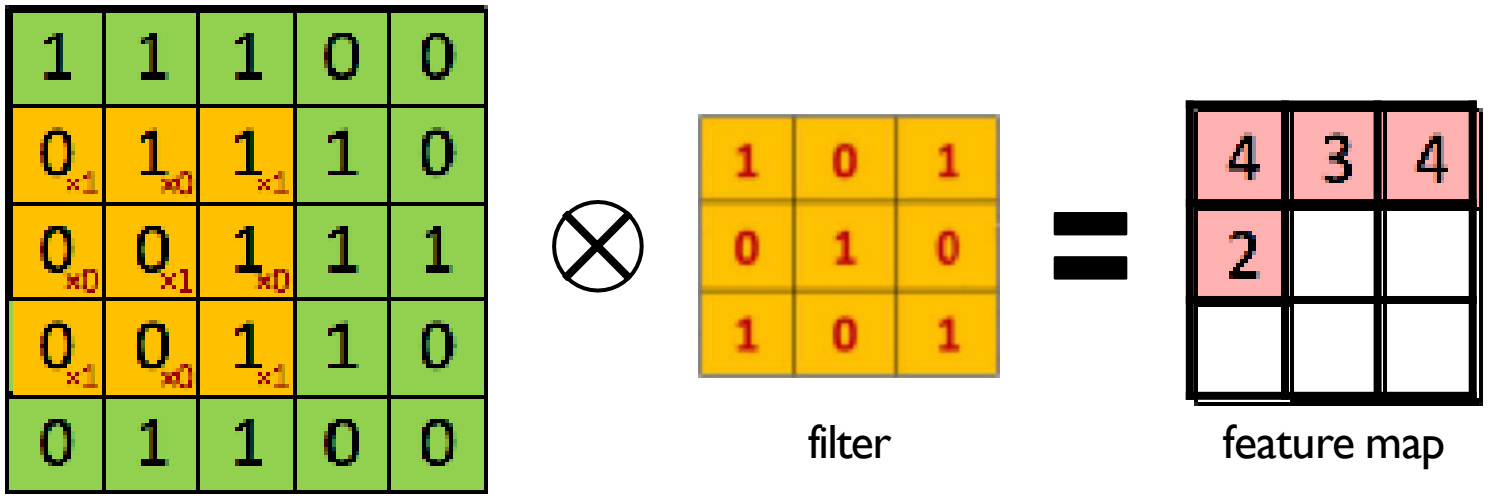
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



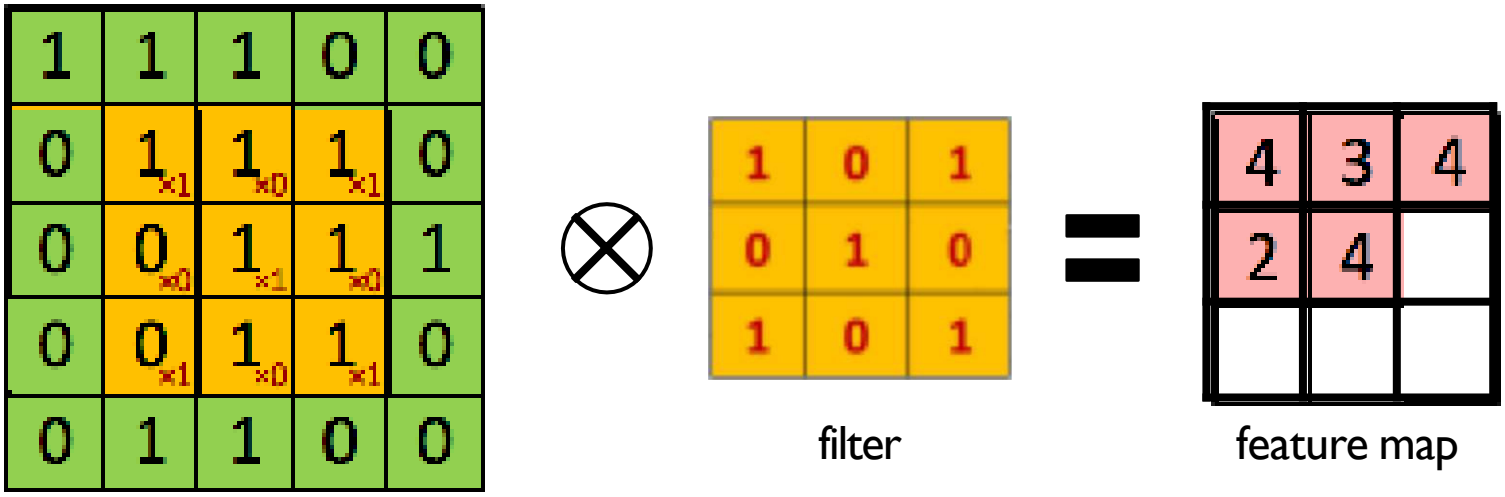
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



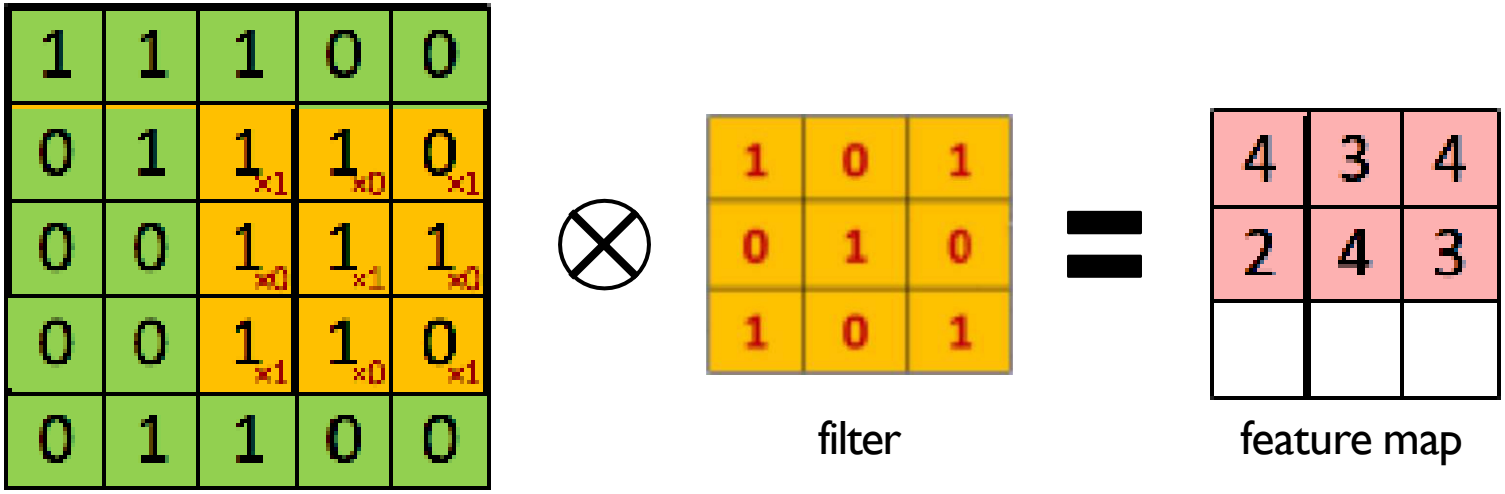
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



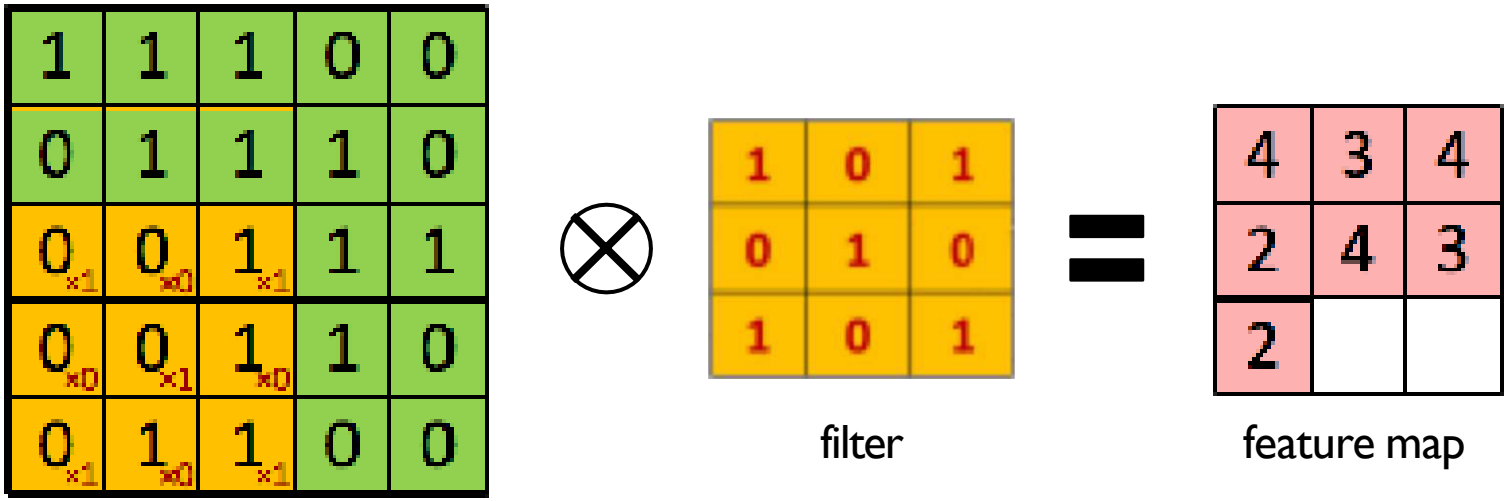
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



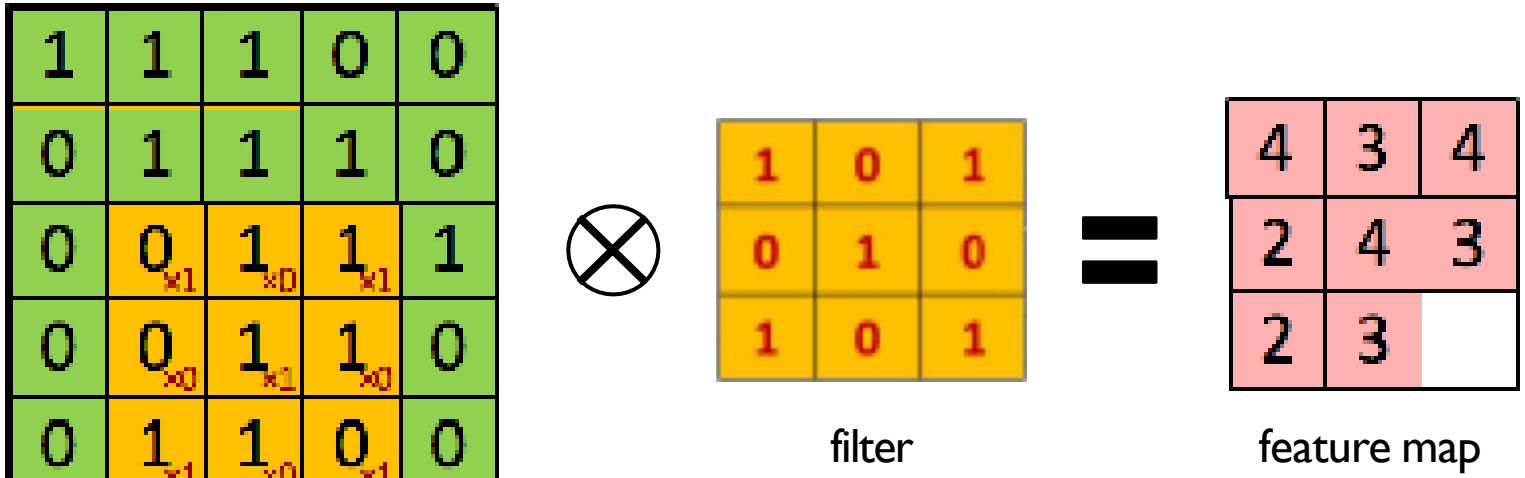
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



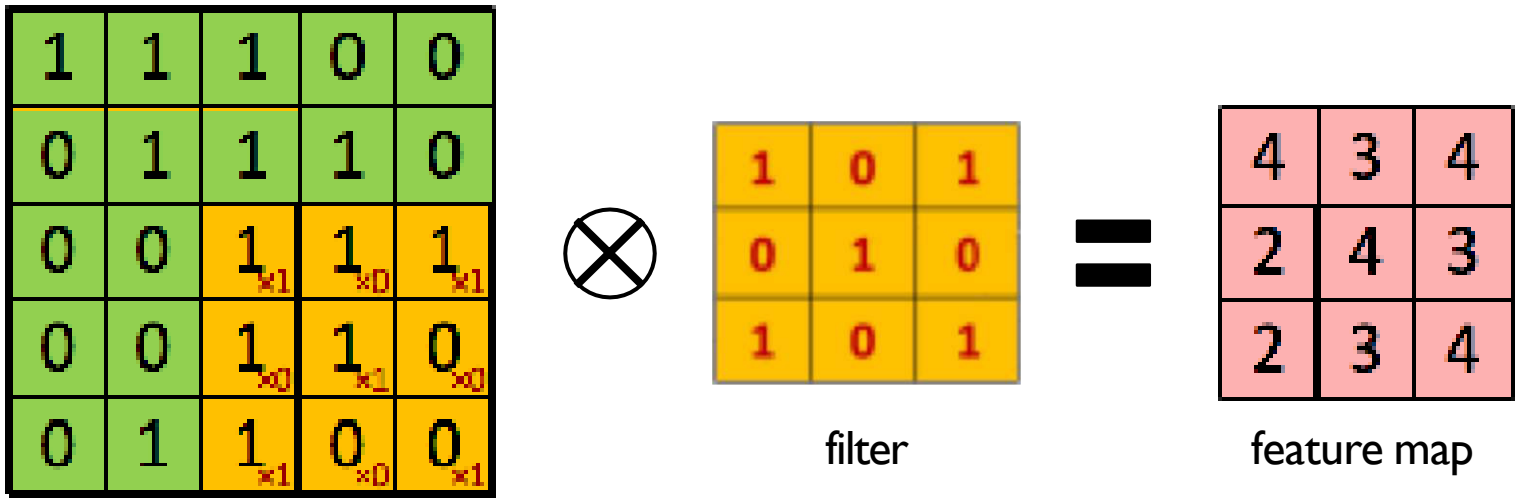
The Convolution Operation

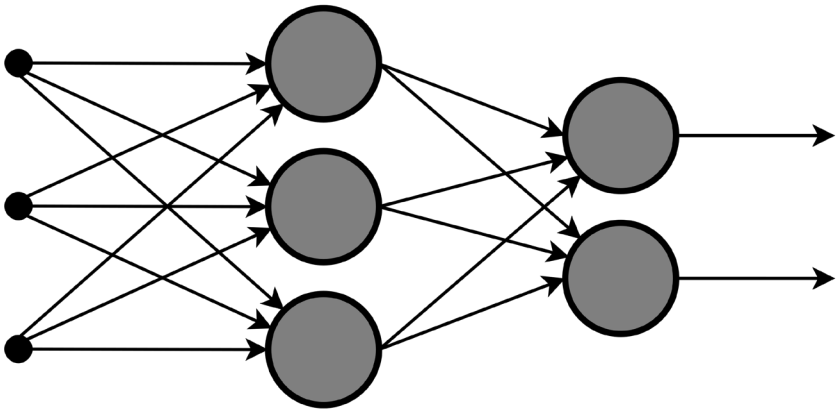
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:





Convolutional Neural ~~Networks~~

edge detection

Vertical edge detection

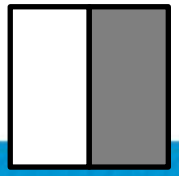
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

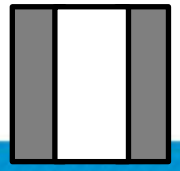
1	0	-1
1	0	-1
1	0	-1

=

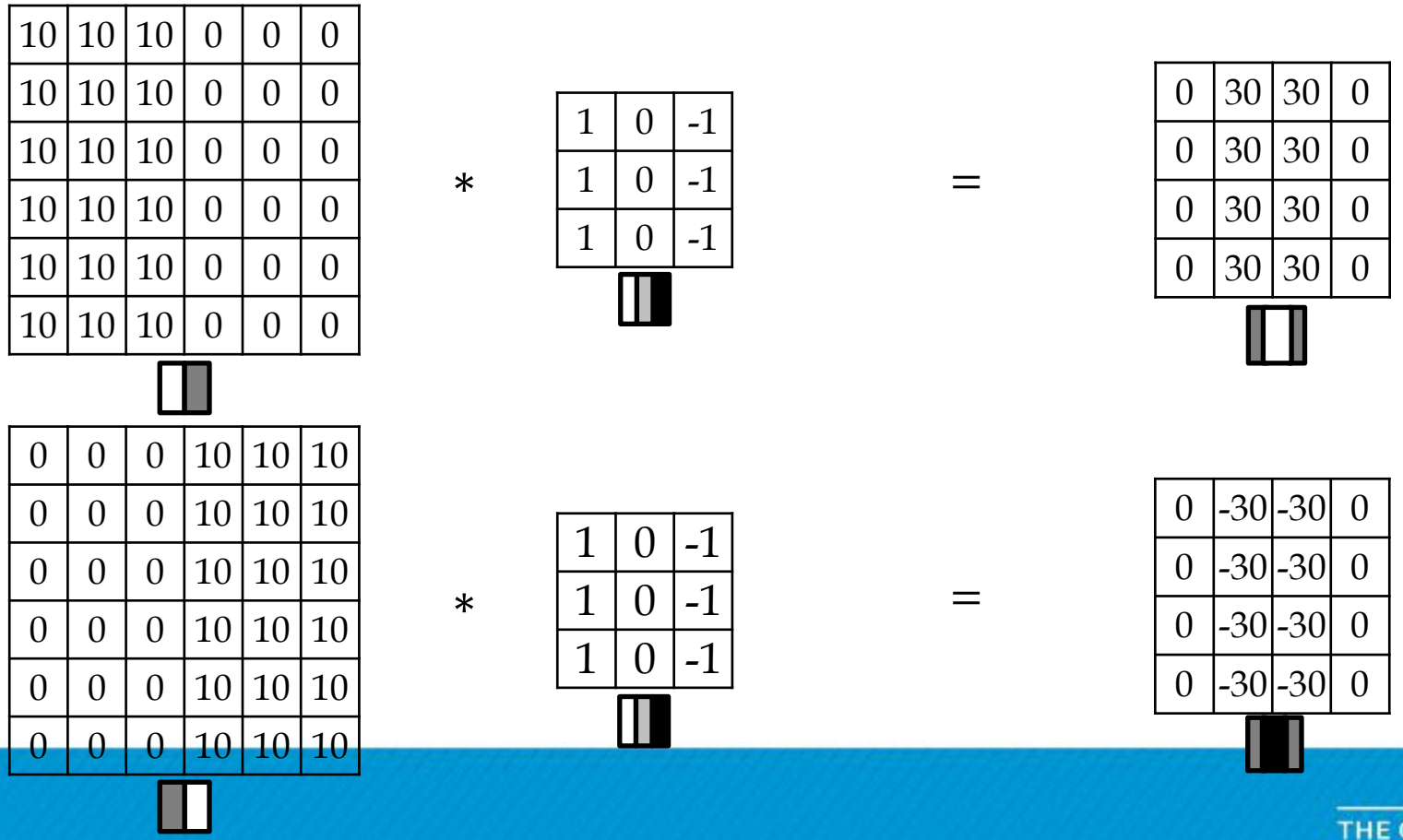
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



*



Vertical edge detection examples



Vertical and horizontal edge detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	0	-1
2	0	-2
1	0	-1

“Sobel filter”

3	0	-3
10	0	-10
3	0	-3

“Scharr filter”

1	1	1
0	0	0
-1	-1	-1

Horizontal

Learning to detect edges

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

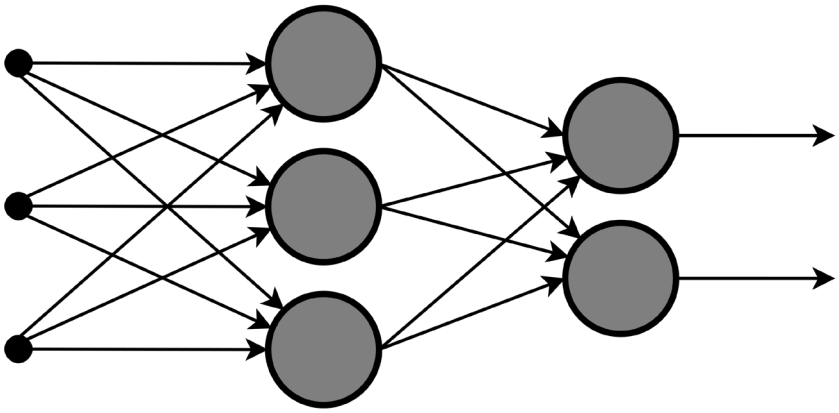
*

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Programs

- cnn_vertical_edge_detection_with_annotation.ipyb: This program demonstrates how filters are used to detect vertical edges of an image.

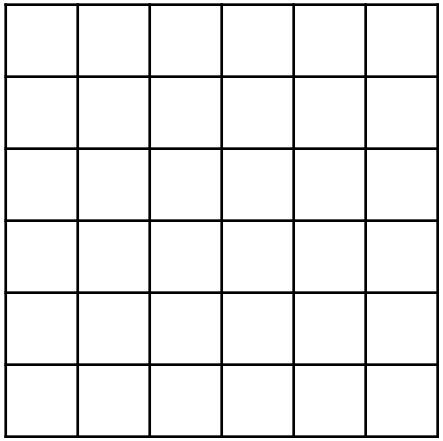
- mnist_conv_layer_visualization.ipyb: This program visualizes the feature maps of a Convolutional Neural Network (CNN) after each convolutional and pooling layer when processing a digit image from the MNIST dataset.
- It demonstrates that the features extracted by a convolutional network do not necessarily conform to our interpretation of a feature.



Convolutional Neural Networks

Padding

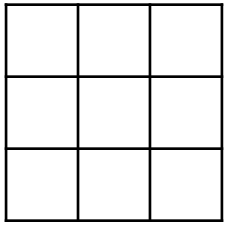
Problems of convolution operation



6×6

$n \times n$

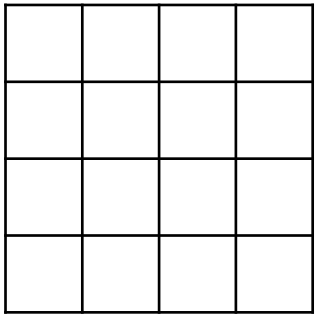
$*$



3×3

$f \times f$

$=$



4×4

$(n - f + 1) \times (n - f + 1)$

Problems:

Shrinky output

Throw away info from edge

Padding

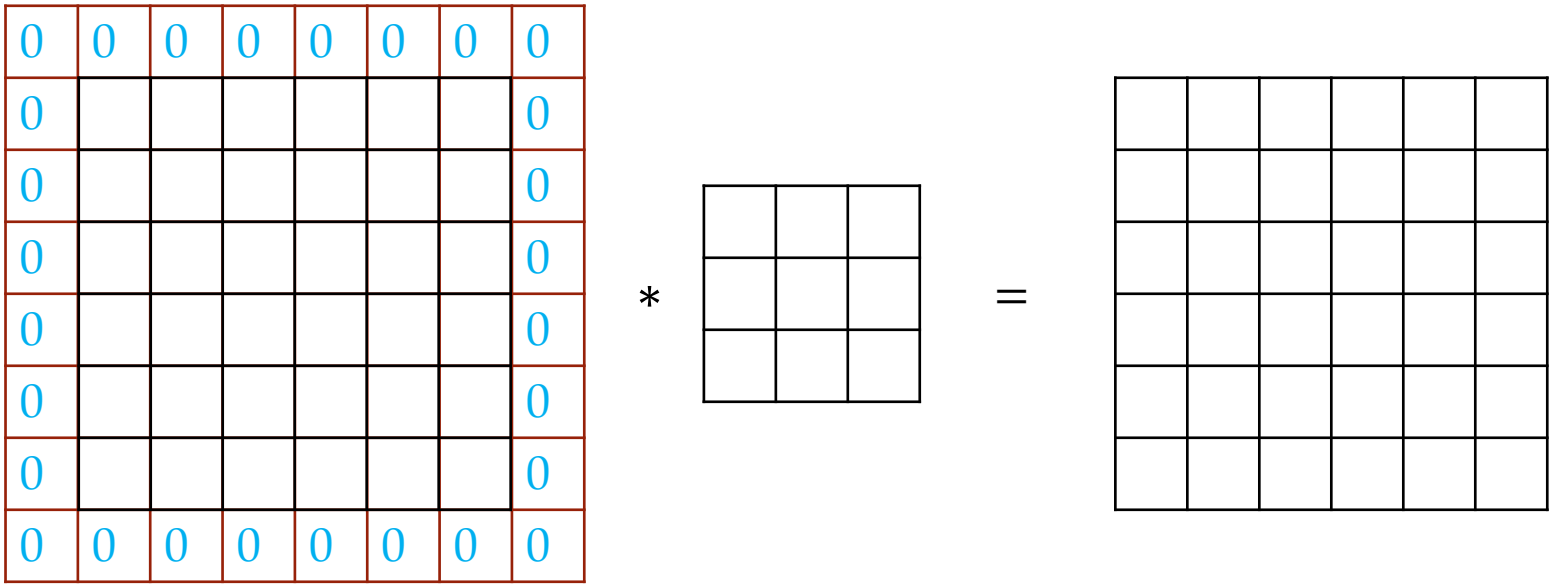
Padding refers to the process of adding layers of zeros (or other values) outside the original image boundaries before applying the convolution operation.

Padding is used to control the spatial dimensions of the output feature map.

With padding, we can apply convolutions to an image and keep the output dimensions either the same or adjust them according to our needs.

Padding ensures that the pixels on the edge of the image are used effectively, allowing the model to learn from the entire image, not just the central part.

Padding



$n = 6$

$p = padding = 1$

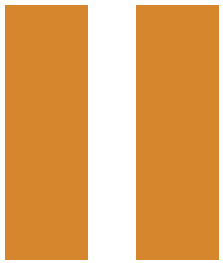
$(n + 2p) \times (n + 2p)$

3×3

$f \times f$

6×6

$(n + 2p - f + 1) \times (n + 2p - f + 1)$

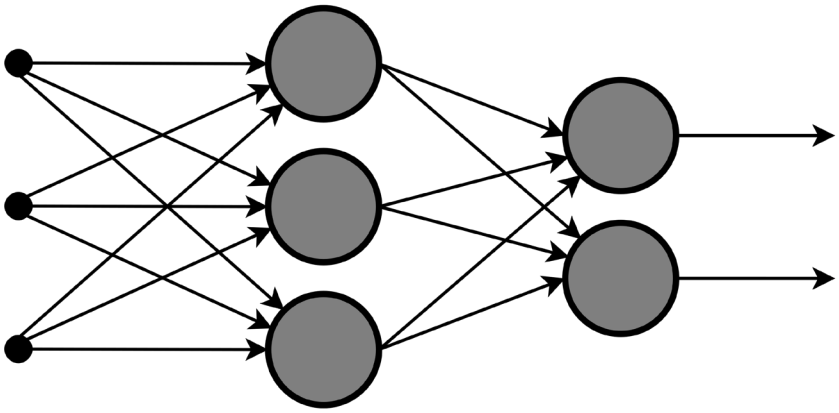


BREAK

Please come back @

2:30 PM EST

1:30 PM CST



Convolutional Neural Networks

Strided convolutions

Strides

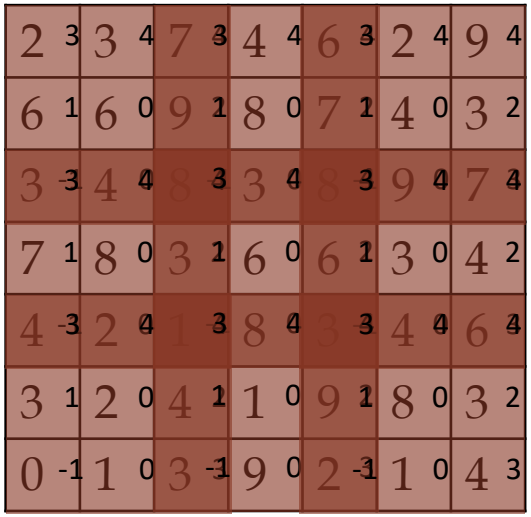
Stride Definition: The stride is the number of pixels we skip when we move the filter across the input image. A stride of 1 means the filter moves one pixel at a time.

Dimensionality Reduction: Using a stride larger than 1 will reduce the spatial dimensions of the resulting feature map. This can be beneficial when working with large images, as it prevents the creation of excessively large feature maps which could make the network too computationally expensive.

Computational Efficiency: Larger strides reduce the number of operations required to compute the output feature maps. This makes the convolution operation faster.

Field of View: With larger strides, the filter covers a larger area of the input image. This can help the network to capture more global information about the input image. However, a larger field of view can sometimes mean that finer details are lost.

Strided convolution



$*$

3	4	4
1	0	2
-1	0	3

$=$

9	100	83
6	9	12
4	12	74

$s = stride = 2$

$n = 7$

$p = padding = 0$

$s = stride = 2$

$(n + 2p) \times (n + 2p)$

3×3

$f \times f$

3×3

$$(\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor) \times (\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor)$$

Floor operation: round down

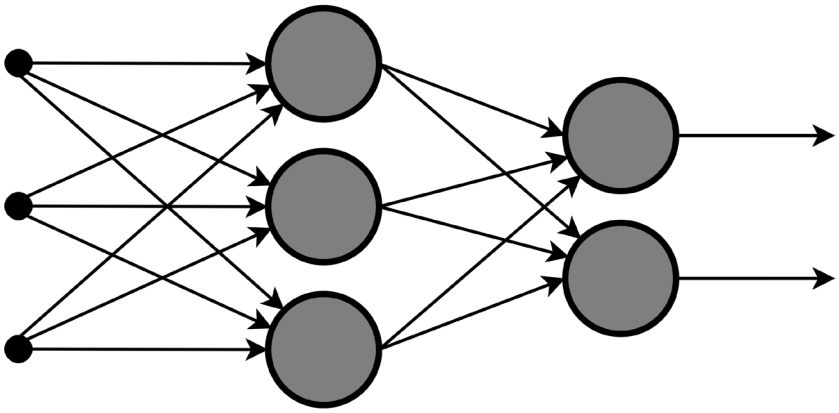
Summary of convolutions

$n \times n$ image $\quad f \times f$ filter

padding p stride s

Output matrix size:

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor$$

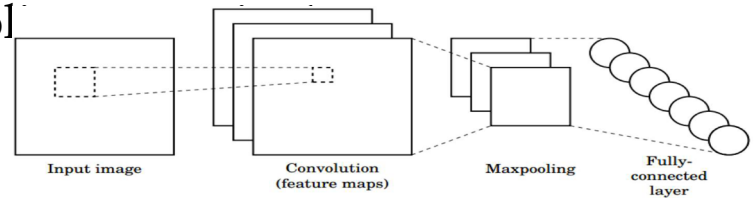


Convolutional Neural Networks

Pooling layers

CNN Processing Steps: Early Layers

1. **Richness of Image Data:** Images can contain myriad features, from edges to complex patterns, representing a significant volume of information.
2. **Challenge:** Capturing every subtle detail can lead to:
    1. **Overwhelming Parameters:** Large number of neurons and weights, causing computational overhead.
    2. **Risk of Overfitting:** The model might memorize noise and anomalies, leading to reduced generalization.
3. **Solution:** Pool



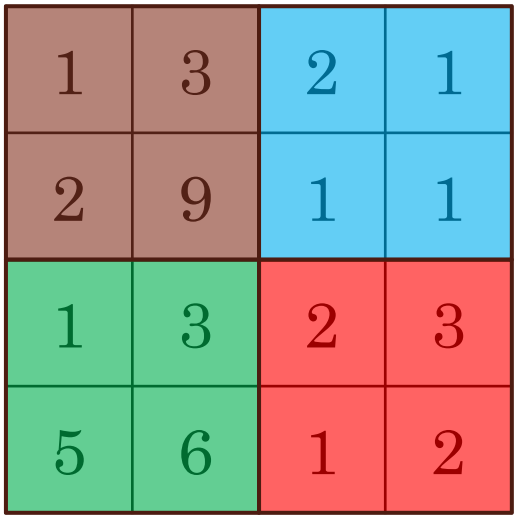
CNN Processing Steps: Early Layers

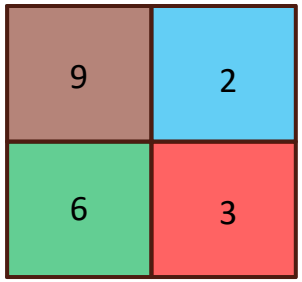
1. **What is Pooling?**
    1. A down-sampling operation reducing the spatial dimensions of feature maps.
    2. Focuses on prominent features while discarding redundant or less critical details.

2. **Types of Pooling:**
    1. **Max Pooling:** Extracts the most prominent feature (max value) in a local region.
    2. **Average Pooling:** Averages out the features, providing a smoothed out perspective.

3. **Benefits:**
    1. **Efficiency:** Fewer parameters, faster processing.
    2. **Spatial Invariance:** Model remains robust to minor shifts and distortions.
    3. **Generalization:** Reduced risk of overfitting, capturing only essential patterns.

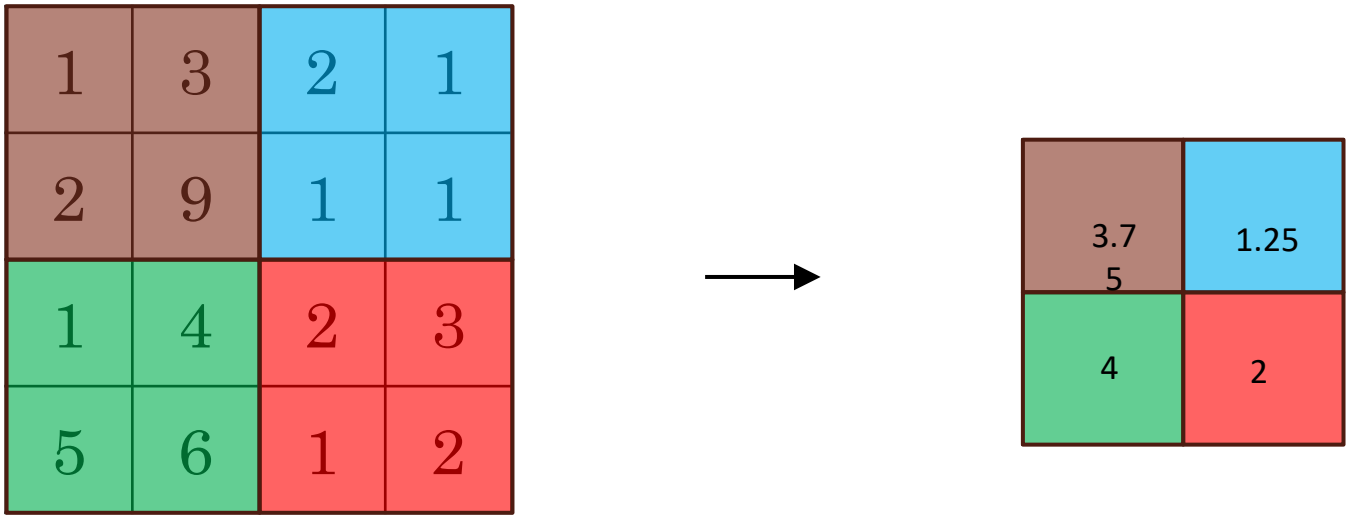
Pooling layer: max pooling





Hyperparameters:
f : filter size = 2
s : stride = 2

Pooling layer: average pooling



Summary of pooling

Hyperparameters:

f : filter size

s : stride

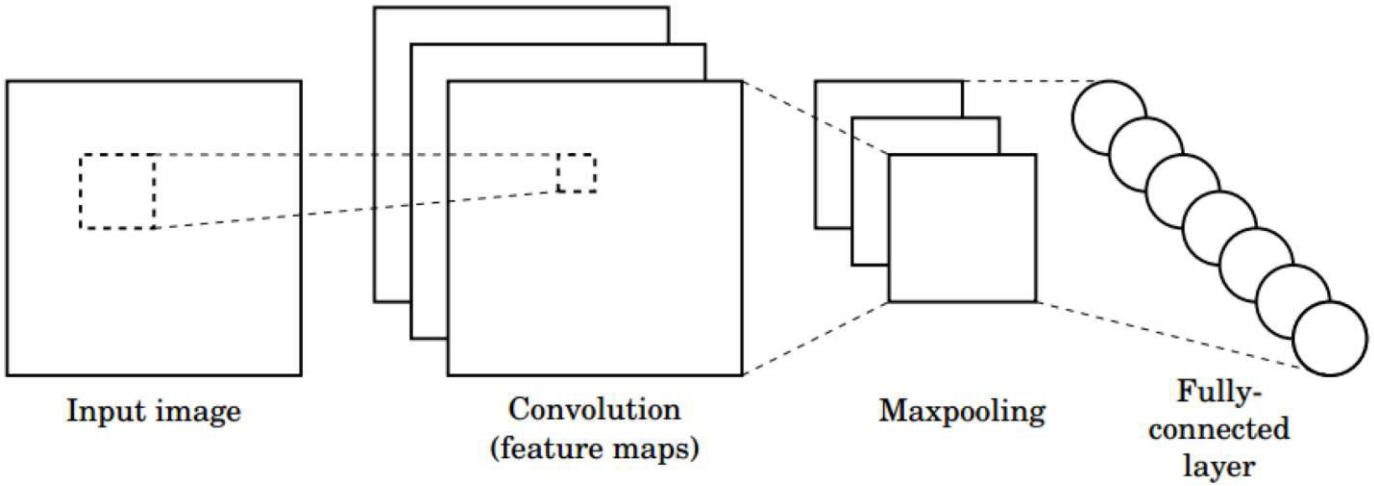
Max or average pooling

No param to learn!

$$Input: n_H \times n_W \times n_c$$

$$Output: \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_c$$

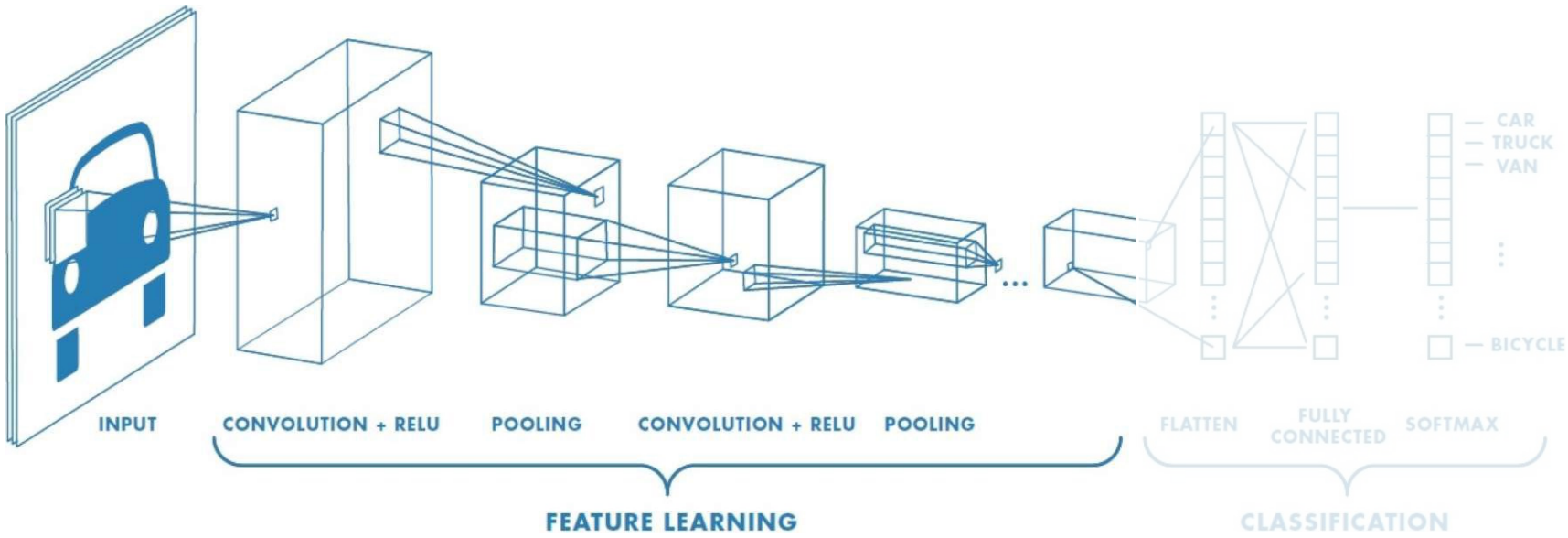
CNNs for Classification



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

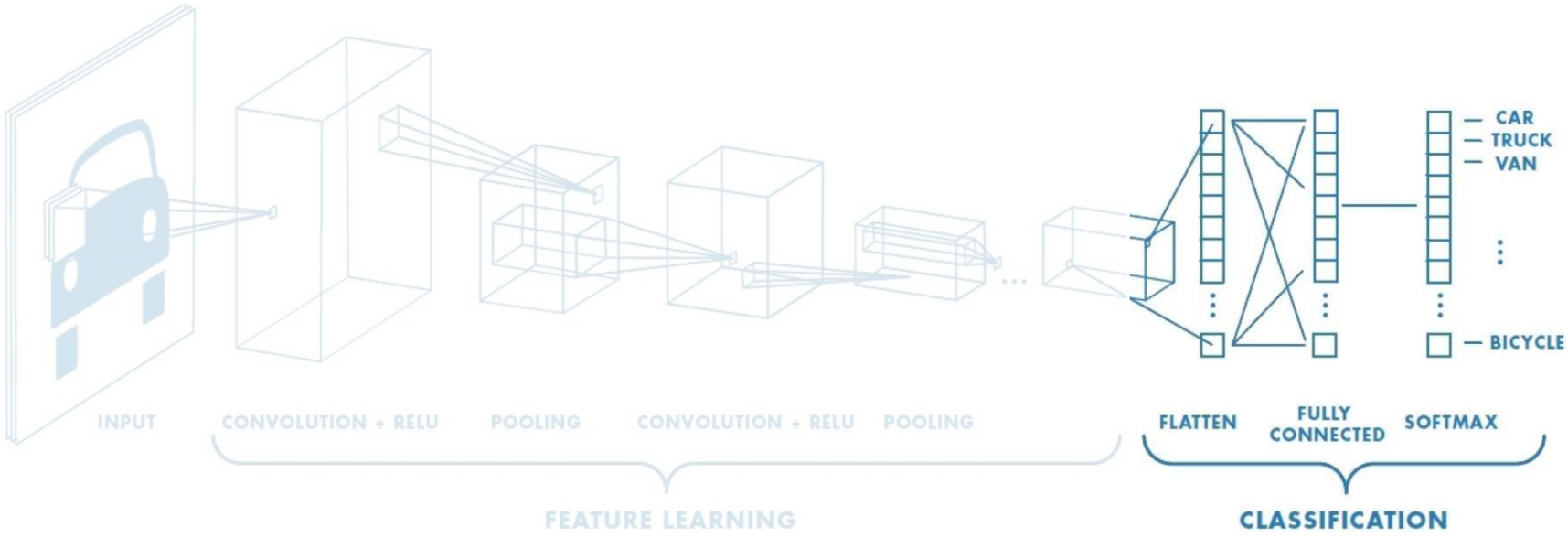
Train model with image data.
Learn weights of filters in convolutional layers.

CNNs for Classification: Feature Learning



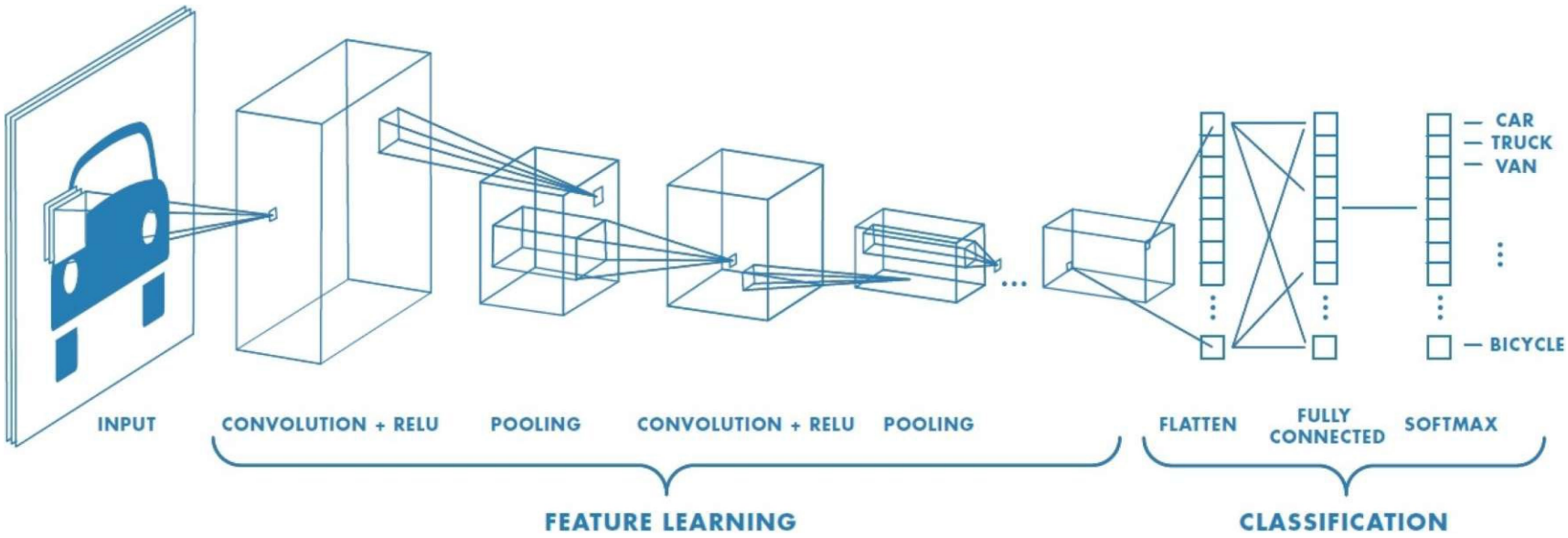
1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability(using Softmax)** of image belonging to a particular class

CNNs:Training with Backpropagation



Learn weights for convolutional filters and fully connected layers
Backpropagation: cross-entropy loss

MNIST Program

MNIST

- The MNIST("Modified National Institute of Standards and Technology“) dataset is a large database of handwritten digits commonly used for training various image processing systems and for training and testing in the field of machine learning.
- The original training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students.
- The dataset contains 70,000 images of handwritten digits: 60,000 for training and 10,000 for testing.
- Each image is 28x28 pixels, representing a grayscale image of a digit (from 0 to 9). Each pixel has a value ranging from 0 (black) to 255 (white).
- Each image is labeled with the digit it represents.
- MNIST has been widely adopted in the machine learning community, and many methods have been tried on it. It's considered as the "Hello World" of machine learning in the image classification context.

MNIST CNN Program Overview
cnnv_mnist_with_annotation.ipynb

- Data Preparation: The program begins by importing necessary libraries and loading the MNIST dataset. The images are normalized to fall between 0 and 1, aiding in the model's training efficiency. Labels are converted into a one-hot encoded format, facilitating the classification task.

- Model Architecture: A Sequential CNN model is constructed. The model comprises convolutional layers (which extract features from the images), pooling layers (that reduce the spatial dimensions, retaining important information), and fully connected layers (to interpret these features and produce a classification result).

- Training: Model is compiled and trained using the training subset of the MNIST dataset.

- Evaluation: After training, the model's performance is evaluated on a separate testing subset, giving insights into its accuracy and potential real-world applicability.

- Demonstration of the use of netron.app to visualize the network.

FMNIST

- Introduction: Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

- Purpose: It was created as a more challenging replacement for the traditional MNIST dataset of handwritten digits. It serves the same purpose: as a benchmark for machine learning algorithms, but with a more demanding problem domain.

- Classes: The dataset includes 10 types of clothing and accessories, specifically: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. This variety allows for interesting machine learning tasks such as classification and image recognition.

FMNIST Programs

- fmnist_classification_with_annotation.ipynb: Uses CNN to classify images
- fmnist_classification_ann.ipynb : Uses ANN to classify images
- fmnist_classification_with_image_prediction.ipynb: Uses CNN to predict a fashion image
- fmnist_with_data_augmentation.ipynb: Demonstrates the use of data augmentation

References

In addition to the references in each slide, the following are leveraged throughout this lecture:

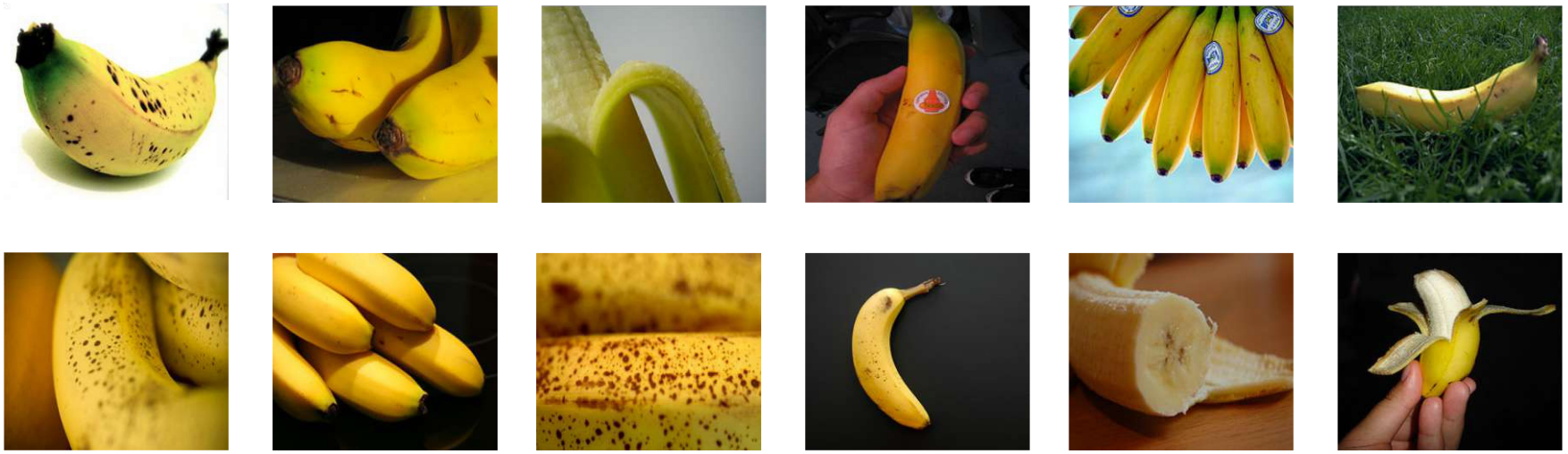
DeepLearning.AI(https://www.deeplearning.ai/)

CNNs for Classification: ImageNet

ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

"Elongated crescent-shaped yellow fruit with soft sweet flesh"



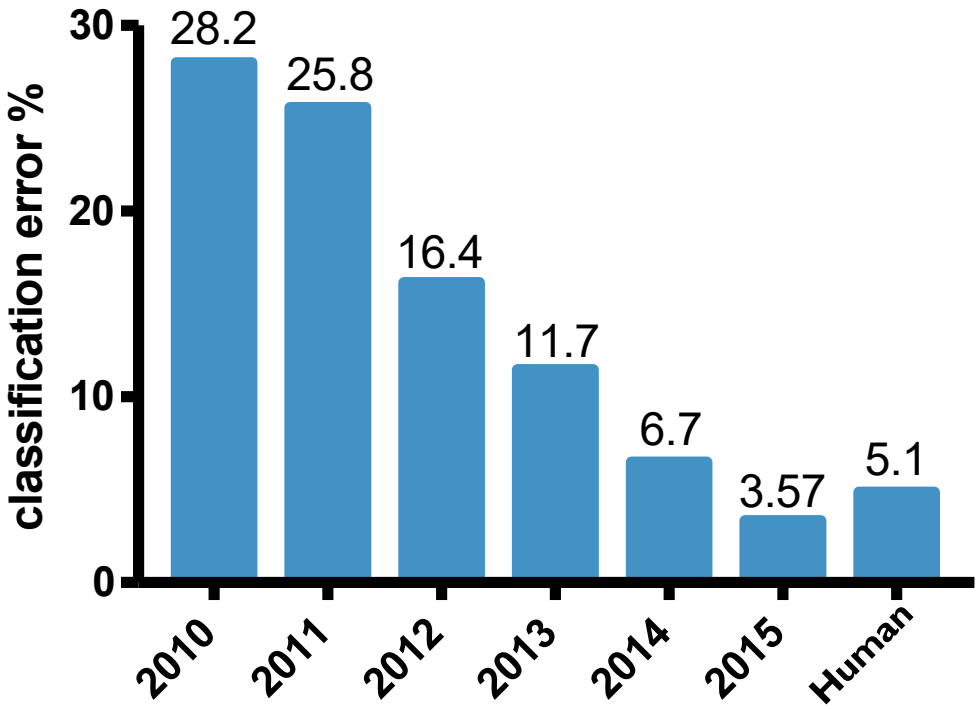
1409 pictures of bananas.

ImageNet Challenge

Classification task: produce a list of object categories present in image. 1000 categories.

Other tasks include:
single-object localization, object detection from video/image, scene classification, scene parsing

ImageNet Challenge



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

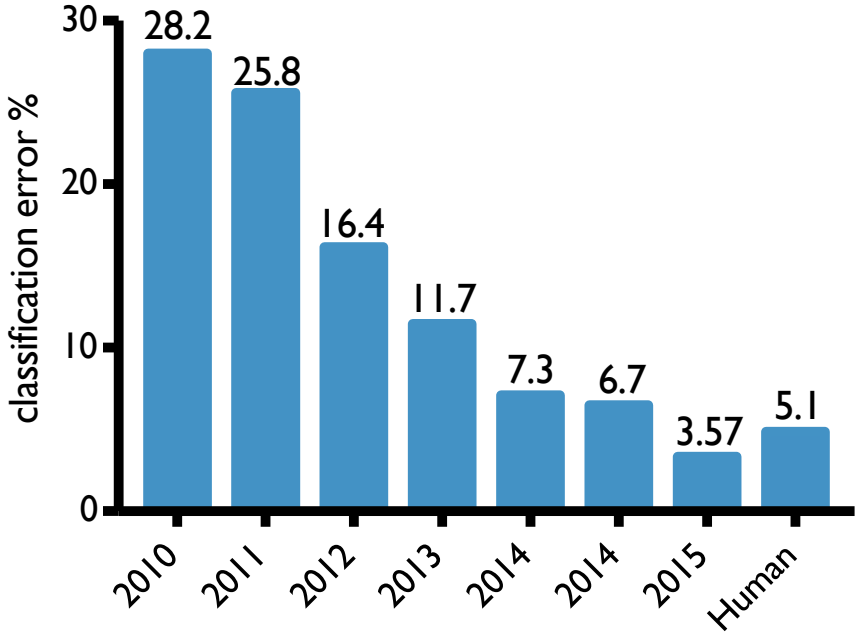
2013: ZFNet
- 8 layers, more filters

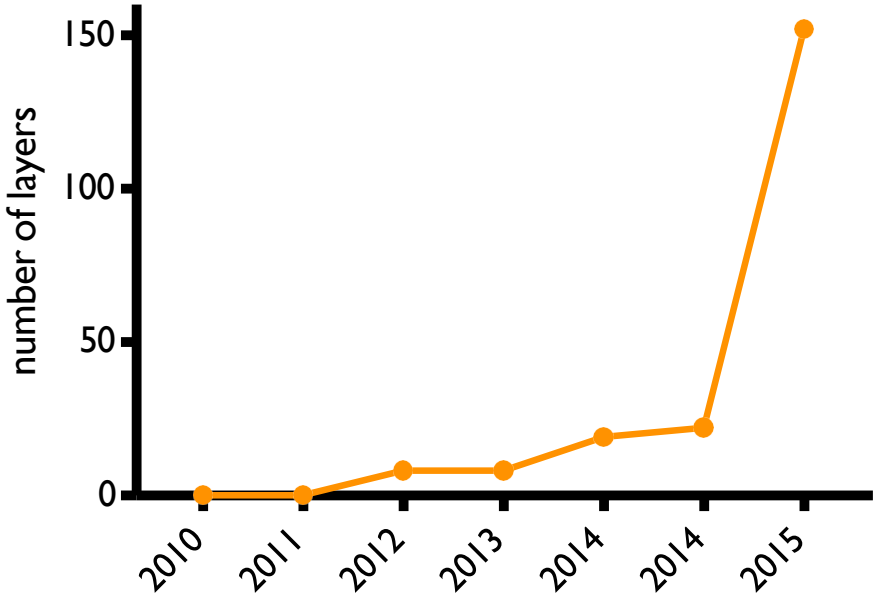
2014:VGG
- 19 layers

2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

ImageNet Challenge





Number of Filters in Convolutional Layers

- The number of filters determines how many distinct features or patterns each convolutional layer can learn.
- Smaller numbers (e.g., 32) are often used in the initial layers to capture basic features like edges and textures.
- Larger numbers (e.g., 64) in deeper layers can capture more complex and abstract features.
- These numbers can be tuned during experimentation to optimize model performance.

Number of Neurons in Dense Layers

- The number of neurons in dense layers can vary depending on the complexity of the task.
- A common practice is to gradually decrease the number of neurons in deeper layers to reduce the model's capacity and prevent overfitting.
- The choice of 128 neurons is reasonable for a moderately complex task like MNIST digit classification.
- In practice, the selection of architecture and hyperparameters often involves experimentation and tuning to find the best configuration for a specific task. It's common to try different architectures and hyperparameter values to achieve the desired performance on the dataset.

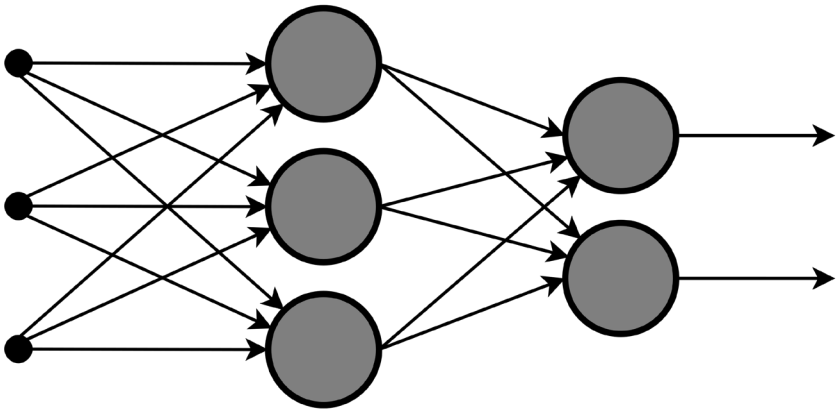
Why We Need ReLU in Convolutional Layers

Non-linearity: Stacking multiple convolutions without ReLU would be equivalent to a single convolution. ReLU breaks this linearity, allowing layers to genuinely build upon each other.

Computational Simplicity: ReLU is computationally light, making it efficient for large-scale images and feature maps.

Avoiding Vanishing Gradients: ReLU helps in preventing the vanishing gradient problem, ensuring effective learning across deep networks.

Empirical Success: Networks with ReLU tend to train faster and achieve better performance in practice.



Convolutional Neural ~~Networks~~

Data Augmentation

Introduction to Data Augmentation Techniques

1. Data augmentation is a strategy to increase the diversity of training data without actually collecting new data.

2. It involves making slight modifications to existing images, which can help models learn more generalized features.

3. Common techniques include flipping, rotation, zooming, translation, shearing and color adjustment.

4. The goal is to create a robust model that performs well on new, unseen data, by exposing it to varied versions of training data.

1. Zooming alters the scale of the image, either magnifying or shrinking parts of it.

2. It helps the model learn to identify objects regardless of their size in the image.

3. Random zooming in and out mimics the effect of objects being nearer or farther away.

4. This technique is particularly useful for datasets where object scale varies significantly.

5. Zooming can improve a model's accuracy by training it on images that simulate different distances.

1. Keras TensorFlow simplifies the application of zooming on images.

2. The ImageDataGenerator class allows for the addition of random zoom augmentation.

3. The zoom_range parameter specifies the zoom range for scaling. With zoom_range=0.2, the images will be randomly zoomed in or out by a factor between 1 - 0.2 (i.e., 0.8 or 80%) and 1 + 0.2 (i.e., 1.2 or 120%).

4. Each image in the training set gets randomly zoomed during the model training.

5. Code snippet:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(zoom_range=0.2)
```

Understanding Shearing in Image Augmentation

1. Shearing is a geometric transformation that slants the shape of an image.

2. It distorts the image along the X or Y axis, creating a skewed effect.

3. This technique is used to simulate the effect of viewing objects from different angles.

4. In training models, shearing can help in recognizing objects under different perspectives.

5. It's an effective way to introduce variability and reduce the model's sensitivity to the orientation of objects.

Implementing Shearing Using Keras

1. Keras TensorFlow offers easy-to-use tools for image augmentation, including shearing.

2. The ImageDataGenerator class in Keras is utilized to apply on-the-fly transformations.

3. Setting the shear_range parameter in ImageDataGenerator applies the shearing effect.

4. The shearing angle is randomly picked from a range, introducing randomness in the training data. For example, shear_range=0.2 means the image will be sheared (or skewed) with a random magnitude between -20% to +20%.

Guidelines for Choosing Shearing Parameters

1. **Start with Small Angles**: Usually, small shear angles (around 5% to 20%) are sufficient. Larger angles might distort the image too much, making it difficult for the model to learn useful features.

2. **Consider the Dataset**: The optimal shear range depends on your specific dataset. For datasets where objects are likely to be seen from varied angles, a slightly higher shear range can be beneficial.

3. **Balance with Other Augmentations**: If you're using multiple augmentation techniques, ensure that the combined transformations don't overly distort the images.

4. **Test and Iterate**: Start with a lower value and gradually increase it while monitoring model performance. Look for improvements in validation accuracy or reductions in overfitting.

5. A range of 0.1 to 0.3 (10% to 30%) is often a good starting point for experimentation.

Guidelines for Choosing Zooming Parameters

1. Range Selection: Similar to shearing, a zoom range between 5% to 20% is often effective.

2. Dataset Specificity: For datasets with objects of varying sizes or images taken from different distances, a higher zoom range can be helpful.

3. Avoid Extreme Zooms: Extreme zooms might result in loss of important features.

4. Monitoring Effects: As with shearing, observe the impact of zooming on model performance and adjust accordingly.

5. A zoom range of 0.1 to 0.2 (10% to 20%) is commonly used.

6. Experimentation is KeyIt's often necessary to experiment with different values to find the optimal settings.

CNN Processing Steps

Below is a sequence of operations commonly found in a Convolutional Neural Network (CNN) pipeline, from the start of processing an input image to connecting to a fully connected layer:

1. **Input**: Start with the input image or feature map.

2. **Padding**: (Optional) Add padding to the input to control the spatial dimensions after convolution. This can help in preserving the spatial dimensions of the original input.

3. **Convolution**: Apply filters to the input (or padded input) to create feature maps. This operation detects patterns like edges, textures, and more, depending on the depth in the network.

4. **ReLU**: Introduce non-linearity with the ReLU activation function.

5. **Pooling**: Reduce the spatial dimensions of the feature maps to decrease computation, enhance translation invariance, and condense learned features. Max pooling is a common method used.

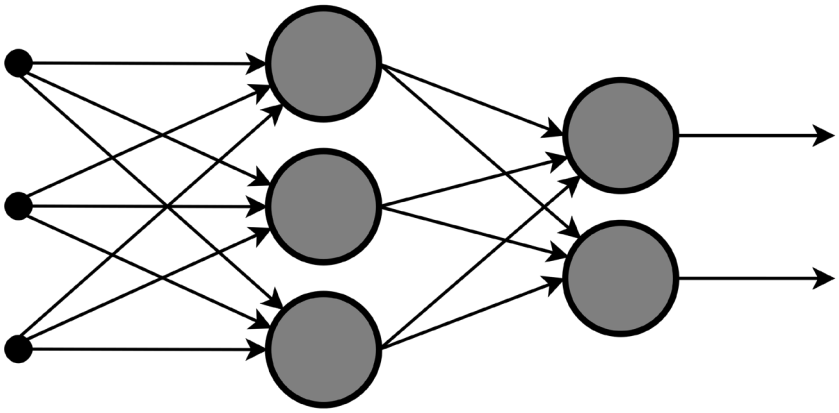
CNN Processing Steps

6. Flattening: Once all the convolutional and pooling layers are done, flatten the final feature map into a one-dimensional vector. This step is necessary before connecting to a fully connected layer.

7. Fully Connected Layer: Use the flattened vector as input for fully connected layers, which perform high-level reasoning from the extracted features.

8. Softmax or Activation: For tasks like classification, the final layer might employ a softmax function to convert network outputs to probabilities. For regression tasks, another activation function might be used.

Back-up



Convolutional Neural ~~Networks~~

Strides and Pooling

Pooling Layers vs Strided Convolutions

Pooling Layers:

- Advantages: Quick dimension reduction, non-parametric.
- Disadvantages: Potential spatial information loss.

Strided Convolutions:

- Advantages: Reduces dimensions while learning, captures complex patterns.
- Disadvantages: More trainable parameters, computationally intensive.

General Guidelines

- Starting Out: Begin with a mix of convolutional layers and pooling for quick dimension reduction.
- Optimization: Consider replacing pooling with strided convolutions for potential performance gains.
- Deep Networks: Prefer strided convolutions; pooling can reduce dimensions too aggressively.
- Recent Trends: Modern state-of-the-art models tend to favor strided convolutions and deeper architectures over extensive pooling.

Max Pooling Intuition

Dominant Feature Presence:

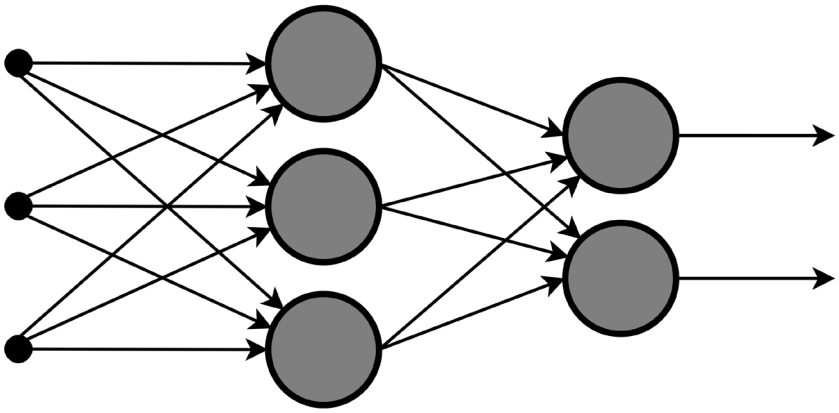
High value indicates the presence of a specific feature in the input region.

Dimensionality Reduction:

- Reduces spatial size of feature maps.
- Reduces computational overhead in subsequent layers.

Robustness:

- Makes the network resilient to minor local changes in the input.
- Focuses on the dominant feature, minimizing impact of noise.



Convolutional Neural ~~Networks~~

Dropout

Application of Dropout in Convolutional Layers

1. **How it Works:**
    1. Similar to its use in fully connected layers, dropout in convolutional layers involves randomly setting a fraction of the input units (in this case, the feature maps) to zero during training. This process is applied at each training step.

2. **Effect:**
    1. Dropout in convolutional layers promotes the learning of more robust features that are not overly reliant on specific paths or combinations of features. It can help in preventing overfitting by reducing the interdependence of feature maps.

Common Practice and Considerations

1. **Less Common in Convolutional Layers:**
    1. Dropout is more commonly used in fully connected layers due to their propensity to overfit more than convolutional layers. The shared weights and local connectivity in convolutional layers inherently provide some level of regularization.

2. **Careful Application:**
    1. When used in convolutional layers, dropout is typically applied with a lower dropout rate compared to fully connected layers. This is because convolutional layers are more sensitive to the dropout operation, and aggressive dropout can lead to loss of important feature information.

Intrinsic Regularization in Convolutional Layers

Convolutional Neural Networks (CNNs) have certain architectural features that inherently contribute to regularization, reducing the risk of overfitting compared to fully connected layers. These features include:

1. **Shared Weights:**
    1. In a convolutional layer, the same filter (set of weights) is applied across different parts of the input image or feature map. This means each filter learns to recognize a feature (like an edge or a pattern) irrespective of its position in the input, leading to a significant reduction in the total number of parameters compared to a fully connected layer with a similar number of units.
    2. By sharing weights, the model is forced to learn more generalized features that are useful across the entire input space, rather than memorizing specific patterns at specific locations.

Dropout in Convolutional Layers

1. **Dropping Entire Feature Maps:**
    1. The more common approach is to drop entire feature maps. In this scenario, during each forward pass, some feature maps (produced by applying filters) are entirely ignored at random. This means that all the activations of the selected feature map are set to zero.
    2. This method ensures that the network does not rely too heavily on any particular feature map, promoting the development of more robust features.

2. **Dropping Components of Feature Maps:**
    1. Alternatively, dropout can be applied to individual components (pixels) of the feature maps. In this case, random pixels in each feature map are set to zero, rather than the entire map.
    2. This variant is less common as it might disrupt the spatial features learned by the filters, which are crucial in tasks like image recognition.

Important Components of CNN

Convolution: The primary purpose of the convolution operation is feature extraction. Convolutional filters are designed to identify patterns such as edges, textures, and other features that are important for understanding the content of the image. While convolution with a stride greater than one or without padding can reduce dimensionality, its fundamental role is to detect features from the input.

Padding: The purpose of padding is indeed to preserve spatial information, allowing the convolution operation to produce feature maps that retain the full spatial extent of the original input. This is especially important for maintaining information at the borders of the image.

Important Components of CNN

Max Pooling: Max pooling is used to reduce the spatial dimensions of the feature maps, which helps to make the representation smaller and more manageable. It also introduces spatial invariance, allowing the network to recognize features regardless of their position in the visual field.

Strides: Strides determine the step size at which the convolution filter moves across the image. Using strides larger than one reduces the spatial dimensions of the output feature map, effectively downsampling the image. This can serve as a method for reducing dimensionality while performing the convolution operation.

In essence, convolution extracts features, padding preserves spatial dimensions (especially at the borders), max pooling provides spatial invariance and reduces dimensions, and strides can be used to control the reduction in dimensionality during the convolution operation. Each component serves its purpose in the architecture to process the image effectively and efficiently, contributing to the CNN's ability to learn and make predictions.

Understanding Softmax

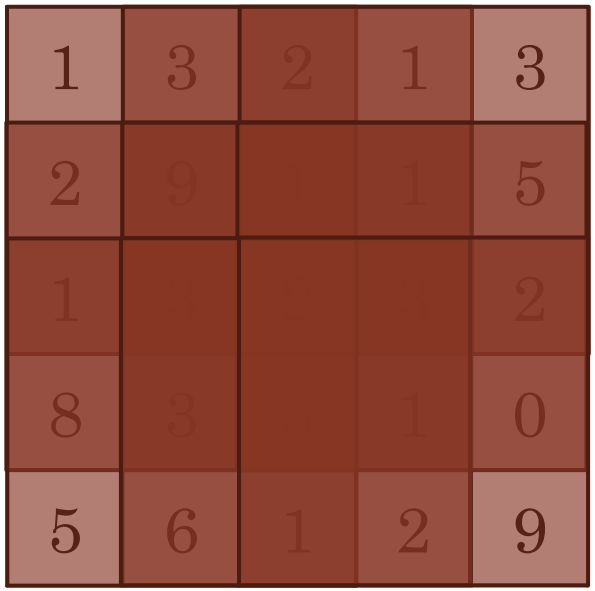
- Definition: The softmax function is a mathematical function used in machine learning and deep learning for multi-class classification problems.
- Purpose: It converts a vector of raw scores (logits) into a probability distribution over multiple classes.
- Output: The output of the softmax function is a set of probabilities, where each probability represents the likelihood of a particular class being the correct one.

How Softmax Works

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum e^{z_j}}$$

- Where:
- z_i is the raw score (logit) for class "i."
- e represents the exponential function.
- The denominator is the sum of exponentiated logits over all classes.
- Result: The output of the softmax function is a probability distribution. The class with the highest probability is the model's predicted class.

Pooling layer: max pooling



9	9	5
9	9	5
8	6	9

Hyperparameters:
f : filter size = 3
s : stride = 1

Types of max-pooling layers (1D, 2D, and 3D)

Max-Pooling Type	Data Dimension	Typical Use Cases
MaxPooling1D	1D (Sequences)	Natural Language Processing (NLP) tasks, time series data, audio analysis, and other 1D sequential data.
MaxPooling2D	2D (Images)	Computer vision tasks, image classification, object detection, image segmentation, and feature extraction from 2D images.
MaxPooling3D	3D (Volumes)	Medical image analysis (e.g., MRI or CT scans), video analysis, 3D object recognition, and 3D data representation.

Each type of max-pooling layer is designed to operate on data with a specific dimensionality. MaxPooling1D is suitable for sequences, MaxPooling2D for images, and MaxPooling3D for volumetric data.
These layers are commonly used in various machine learning and deep learning tasks to reduce the spatial dimensions of the data while preserving important features by selecting the maximum value in each local region.

Spatial Invariance Intuition

Spatial invariance, in the context of CNNs and particularly with pooling, refers to the model's ability to recognize features or patterns regardless of their position or minor variations in the input space.

Example: Recognizing a Cat

Imagine you're training a CNN to recognize cats in images. Cats can appear in various positions within a picture: centered, towards the left, right, top, or bottom. They can be sitting, standing, or even upside-down.

With Spatial Invariance (thanks to pooling): Pooling layers help the model look at the 'bigger picture' by summarizing features. This means that even if the cat's position changes slightly (or even if some of its features are slightly distorted), the model can still detect the crucial aspects, like the shape of its eyes, the curve of its tail, etc., and successfully recognize it as a cat.

Keras Params

loss functions:

'binary_crossentropy': Used for binary classification problems.

'categorical_crossentropy': Used for multi-class classification problems where labels are one-hot encoded.

'sparse_categorical_crossentropy': Used for multi-class classification problems where labels are integers.

'mse' or 'mean_squared_error': Used for regression problems.

'mae' or 'mean_absolute_error': Used for regression problem

metrics:

'accuracy': Used for classification problems.

'mae': Mean Absolute Error, used for regression problems.

'mse': Mean Squared Error, used for regression problems.

'AUC': Area Under the ROC Curve.

'Precision', 'Recall': Used for evaluating precision and recall.

Back-up Slides & References

Hands On…..

References

In addition to the references in each slide the below are leveraged throughout this course.

Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly.
Wolff, S. G. (2018). Less is more: optimizing classification performance through feature selection in a very-high-resolution remote sensing object-based urban application. GIScience & Remote Sensing. doi:10.1080/15481603.2017.1408892

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.