

Convolution Neural Networks Tutorial (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2026

June 18, 2026

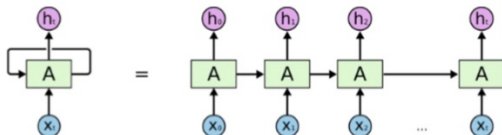
Overview

- 1 Convolution Neural Networks (CNN)
 - Convolution Neural Networks (CNNs) and Image Analysis
- 2 Computational Formalisms
 - Convolution Operations
- 3 Examples
 - Example 1: Convolution Neural Network from Scratch
 - Example 2: Recognition of Handwritten Digits
 - Example 3: Recognition of Handwritten Sketches
- 4 Appendix A: Object Detection with YOLO

Quick Review

Machine Learning Capabilities (1997-2014)

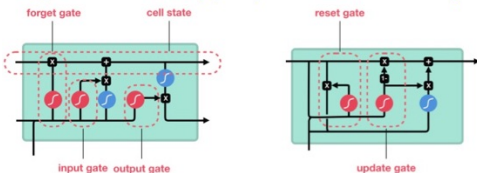
Recurrent Neural Networks (RNN): Learn sequences in data streams (text, speech)



Hidden state "h" serves two purposes:

- Make an output prediction.
- Represent features in the previous steps

Long Short-Term Memory (1997) Gated Recurrent Units (2014)



Key Features of LSTM:

- Standard RNN suffers from vanishing gradients for modeling of long-term dependencies.
- LSTM gives cells the ability to **remember values for long periods of time**.
- Gates regulate the flow of information in / out of the cell, and what should be remembered or discarded.



sigmoid



tanh



pointwise
multiplication



pointwise
addition



vector
concatenation

Applications:

- Time series prediction.
- Time-series anomaly detection.

Machine Learning Capabilities (1997-2014)

Learning Streams of Text

- Download complete works of Shakespeare (5.4 million characters)
- Train machine to remember text.
- Write new Shakespeare!

Sample after 1000 Iterations

OTHERS. Allow up, whe, Will a moutice,
Reason that boy skents allies; comn jusgius,
And encarembower heth lasty!
TASTOR. When no; devires at though me beligo jody?
NANCHELOT. Trom juther and bur itnot spokk,
That as take have wendisho

Sample after 2000 Iterations

VIBOTARD. Walk this boy and door as am 'stone!
NTRURLEN. Being entainare af Haion.
Els that by much that, I mion's now, who make foil the kidll
CLOTE. Why which Ham'd?
QUICKLY. And stand quast of I; this Fi

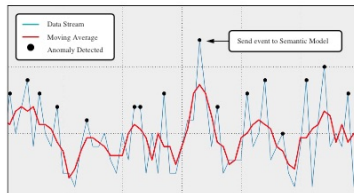
Sample after 2500 Iterations

FORD. Nay, You're, excount: and now did yet.
PAROLLES. Take DUTBY
This who is begin Cnoban a bows; but yet which that have be,
Oll, thou stan, and me not ready withered gids
And he in the pleassues or pardon us.
Her. I pray you, how can, and tu

Sample after 3000 Iterations

Thume, 'tigoning, dear?
My lam, an hour, chork'd more in my grain,
I am thing forent innocon, maden! I thank I do speak you?
ARRESSTURO. But his bounes, giving to know, foward to
the distvenail.' The to you well know yes, my lowi.
SECOND CMMENOR. He needs, for the reforeds are;

Time Series Anomaly Detection



Time Series Prediction



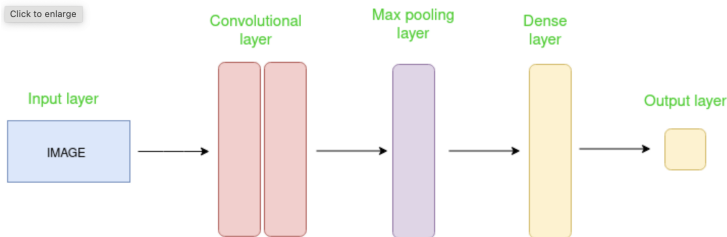
◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Convolution Neural Networks (CNNs)

Convolution Neural Network

Convolution neural networks are multi-layer neural networks, designed to analyze and interpret images comprised of complex patterns, shapes and textures.

Simple CNN Architecture:



Convolution Neural Networks (CNNs)

Simple CNN Architecture Layers:

- **Input Layer:** Receive raw image data, usually as tensors with dimensions width $\times$ height $\times$ channels (e.g., $224 \times 224 \times 3$).
- **Convolution Layers:** Small learnable matrices (e.g., 5×5) slide over the image input, automatically detecting features such as edges or textures or shapes.
- **Max Pooling Layer:** Downsampling operation to reduce feature map dimensions (width/height) by selecting the maximum value from sub-regions (e.g., 2-by-2 window).
- **Dense Layers:** After feature extraction, these layers make the final prediction based on the detected patterns.
- **Softmax Output:** Converts the network output into probabilities, showing the likelihood of each class.

Convolution Neural Networks (CNNs)

Deep Learning Conceptual Diagram: Detect features in images, starting with edges and progressing to more complex patterns.

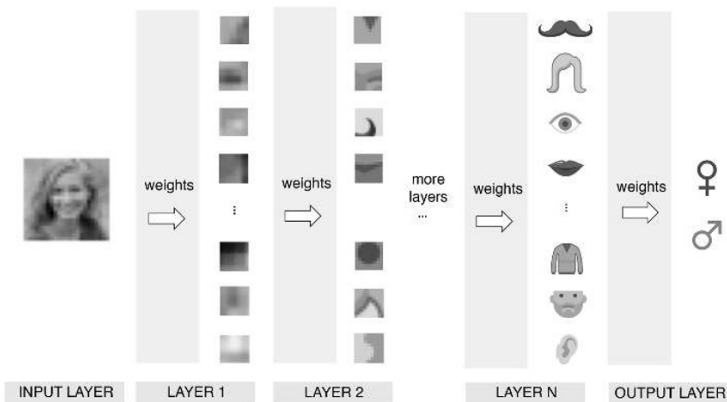


Figure 2-2. Deep learning conceptual diagram

Convolution Neural Networks (CNNs)

Convolution Architecture: Detect features in images, starting with edges and progressing to more complex patterns.

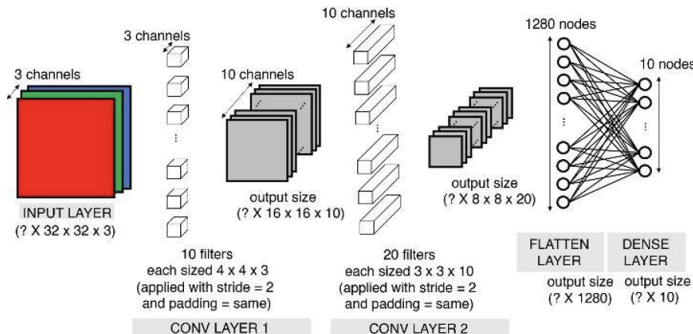


Figure 2-13. A diagram of a convolutional neural network

Computational Formalisms

Vectors, Matrices, Tensors, Color Images

Vectors, matrices, tensors.

vector



$$\mathbf{v} \in \mathbb{R}^{64}$$

matrix



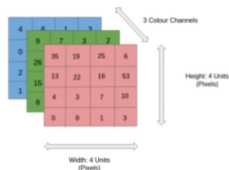
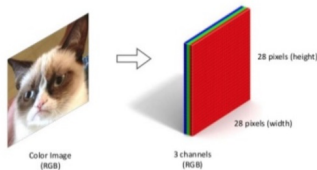
$$\mathbf{X} \in \mathbb{R}^{8 \times 8}$$

tensor



$$\mathbf{X} \in \mathbb{R}^{4 \times 4 \times 4}$$

Color images are three-dimensional tensors.



Convolutions

Mathematical: A convolution operation can be represented:

Here:

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻

Convolution Operations

1D Input Signals/Arrays:

- Applied to sequences or time series.
- Filter slides along the input signal and performs element-wise multiplication and accumulation at each position to produce output signal.

2D Input Signals/Arrays:

- Applied to greyscale/color images.
- Filter/kernel slides over the input array and performs element-wise multiplication and accumulation at each position.

3D Input Signals/Arrays:

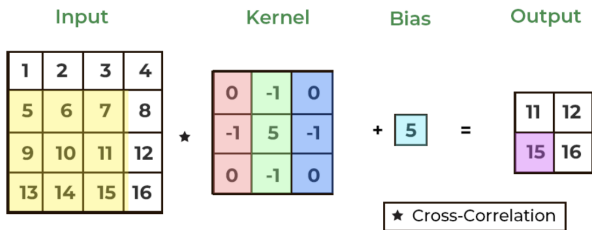
- Applied to videos and/or volumetric data.

2D Convolutions

2D Convolutions

A 2D convolution is a mathematical operation that applies a filter to an image, producing a filtered output (also called a feature map).

Simple Example: Sharpen filter ...



$$5 \times 0 + 9 \times -1 + 13 \times 0 + 6 \times -1 + 10 \times 5 + 14 \times -1 + 7 \times 0 + 11 \times -1 + 15 \times 0 + 5 = 15$$

Filter/Kernel Design:

2D Convolutions

Filter Applied to Grayscale Image:

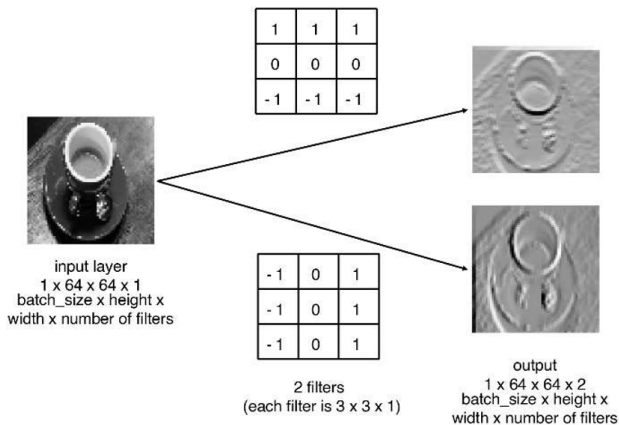


Figure 2-11. Two convolutional filters applied to a grayscale image

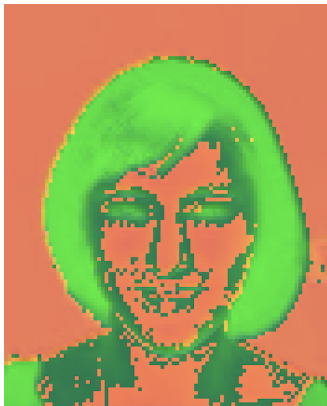
TensorFlow and Python Image Library (PIL)

```

30 Input = Input.unsqueeze(0)
31 print('Input Tensor :',Input.shape)
32
33 # Assemble convolution operation ...
34
35 print("--- Assemble and compute convolution operation ... ");
36
37 conv = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=2, bias=True)
38 output = conv(Input)
39
40 print(output.shape)
41
42 Out_img = output.squeeze(0)
43
44 print("--- Convert tensor to image ... ");
45
46 Out_img = T.ToPILImage()(Out_img)
47
48 print("--- Show filtered image ... ");
49
50 Out_img.show();
51
52 # call the main method ...
53
54 if __name__ == "__main__":
55     main()

```

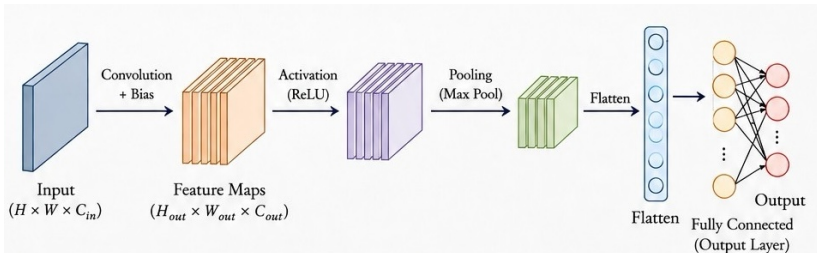
2D Convolutions



Examples

Convolution Neural Network from Scratch

Problem Statement:



- Convolution operation (actually cross-correlation):

$$(X * K)[i, j] = \sum_m \sum_n X[i + m, j + n] \cdot K[m, n] + b$$

Where:

- X : input feature map
- K : kernel (filter)
- b : bias term

- Activation function (ReLU):

$$\phi(z) = \max(0, z)$$

- Max Pooling operation:

$$Y[i, j] = \max_{0 \leq m, n < p, 0 \leq m < p} X[i \cdot p + m, j \cdot p + n]$$

Convolution Neural Network from Scratch

```
import numpy as np
from scipy.signal import correlate

def conv2d(input_matrix, kernel, bias=0):
    """Performs 2D cross-correlation."""
    h, w = input_matrix.shape
    kh, kw = kernel.shape
    oh, ow = h - kh + 1, w - kw + 1
    output = np.zeros((oh, ow))
    for i in range(oh):
        for j in range(ow):
            output[i, j] = np.sum(input_matrix[i:i+kh,
                                                j:j+kw] * kernel) + bias
    return output

def relu(x):
    """Applies ReLU activation."""
    return np.maximum(0, x)

def max_pool2d(x, pool_size=2):
    """Applies 2D max pooling."""
    h, w = x.shape
    ph, pw = pool_size, pool_size
    oh, ow = h // ph, w // pw
    output = np.zeros((oh, ow))
    for i in range(oh):
        for j in range(ow):
            output[i, j] = np.max(x[i*ph:(i+1)*ph, j*pw:(j+1)*pw])
    return output
```

```
# Example
# Input (5x5)
input_matrix = np.array([
    [1, 2, 0, 1, 2],
    [0, 1, 3, 1, 0],
    [2, 2, 1, 0, 1],
    [1, 0, 2, 1, 3],
    [2, 1, 0, 1, 2]
])

# Kernel (3x3)
kernel = np.array([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
])

bias = 0
```

```
conv_output = conv2d(input_matrix, kernel, bias)
relu_output = relu(conv_output)
pool_output = max_pool2d(relu_output, pool_size=2)

print("Cross-Correlation Output:\n", conv_output)
print("ReLU Output:\n", relu_output)
print("Max Pooling Output:\n", pool_output)

Cross-Correlation Output:\n [ 2. -1. -3.]
                             [ 3. -3. -1.]
                             [ 0. -1.  1.]

ReLU Output:\n [[2.  0.  0.]
                [3.  0.  0.]
                [0.  0.  1.]]

Max Pooling Output:\n [[3.  0.]
                      [0.  1.]]
```

Recognition of Handwritten Digits

Problem Statement

- Use CNNs to demonstrate [recognition of handwritten digits](#).

MNIST Handwritten Digit Dataset

- MNIST (Modified National Institute of Standards and Technology) is a database of handwritten digits provided by NIST.
- The database contains: 60,000 training images and 10,000 testing images.
- Each image is a scan of a handwritten image 0 through 9.
- Differences among images are due to variations in handwriting style.

Recognition of Handwritten Digits

Source Code: [python-code-ai.d/cnn/TestCNN-Digit-Recognition02.py](#)

Sample Digits



Recognition of Handwritten Digits

Model Summary

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d (MaxPooling2D)	(None, 12, 12, 8)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	3,216
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 37,610 (146.91 KB)

Trainable params: 37,610 (146.91 KB)

Non-trainable params: 0 (0.00 B)

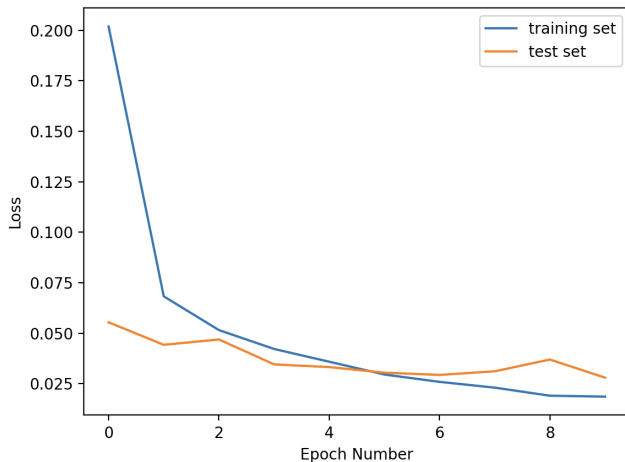
Recognition of Handwritten Digits

Training History:

Epoch 1/10	
1875/1875	4s 2ms/step - accuracy: 0.9421 - loss: 0.1915 - val_accuracy: 0.9831 - val_loss: 0.0563
Epoch 2/10	
1875/1875	4s 2ms/step - accuracy: 0.9809 - loss: 0.0632 - val_accuracy: 0.9878 - val_loss: 0.0375
Epoch 3/10	
1875/1875	4s 2ms/step - accuracy: 0.9847 - loss: 0.0473 - val_accuracy: 0.9853 - val_loss: 0.0432
Epoch 4/10	
1875/1875	4s 2ms/step - accuracy: 0.9883 - loss: 0.0379 - val_accuracy: 0.9883 - val_loss: 0.0332
Epoch 5/10	
1875/1875	4s 2ms/step - accuracy: 0.9902 - loss: 0.0312 - val_accuracy: 0.9899 - val_loss: 0.0292
Epoch 6/10	
1875/1875	4s 2ms/step - accuracy: 0.9918 - loss: 0.0265 - val_accuracy: 0.9893 - val_loss: 0.0331
Epoch 7/10	
1875/1875	4s 2ms/step - accuracy: 0.9922 - loss: 0.0241 - val_accuracy: 0.9909 - val_loss: 0.0289
Epoch 8/10	
1875/1875	4s 2ms/step - accuracy: 0.9928 - loss: 0.0208 - val_accuracy: 0.9900 - val_loss: 0.0343
Epoch 9/10	
1875/1875	4s 2ms/step - accuracy: 0.9943 - loss: 0.0176 - val_accuracy: 0.9916 - val_loss: 0.0292
Epoch 10/10	
1875/1875	4s 2ms/step - accuracy: 0.9942 - loss: 0.0171 - val_accuracy: 0.9916 - val_loss: 0.0284

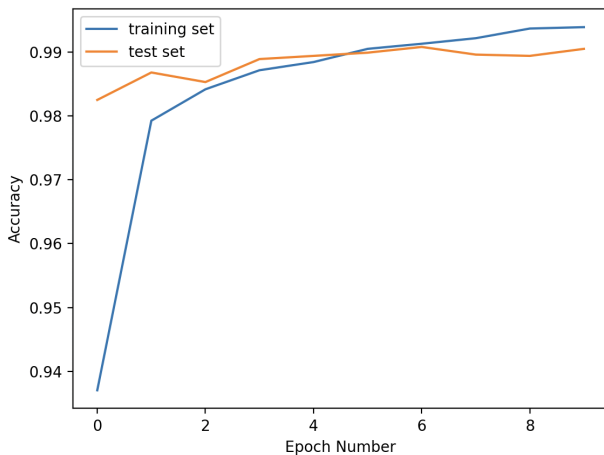
Recognition of Handwritten Digits

Training History:



Recognition of Handwritten Digits

Training History:



Recognition of Handwritten Digits

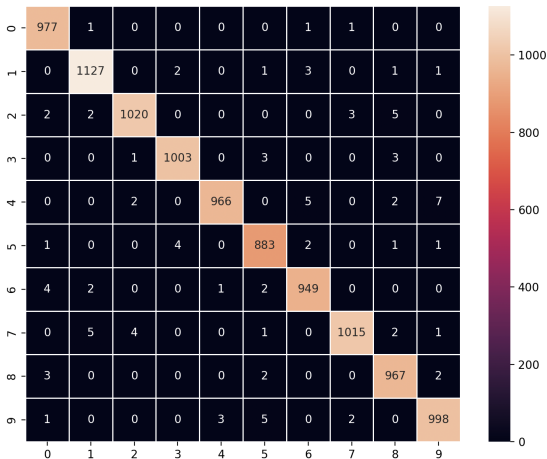
Digit Predictions:

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

Legend: Prediction correct → green; Prediction incorrect → red.

Recognition of Handwritten Digits

Confusion Matrix:



Recognition of Handwritten Sketches

QuickDraw Dataset

The **QuickDraw Dataset** is a collection of **50 million drawings** across **345 categories**, contributed by the players of the game Quick Draw.

Categories:

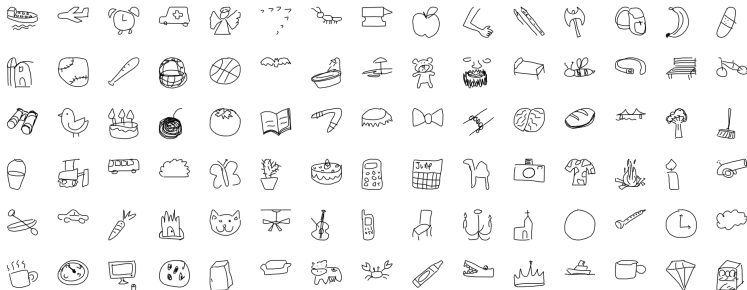
- | | | | |
|---------------------|------------------|-------------------|-----------------|
| 1. aircraft carrier | 16. barn | 31. binoculars | 46. bucket |
| 2. airplane | 17. baseball | 32. bird | 47. bulldozer |
| 3. alarm clock | 18. baseball bat | 33. birthday cake | 48. bus |
| 4. ambulance | 19. basket | 34. blackberry | 49. bush |
| 5. angel | 20. basketball | 35. blueberry | 50. butterfly |
| 6. animal migration | 21. bat | 36. book | 51. cactus |
| 7. ant | 22. bathtub | 37. boomerang | |
| 8. anvil | 23. beach | 38. bottlecap | |
| 9. apple | 24. bear | 39. bowtie | |
| 10. arm | 25. beard | 40. bracelet | 342. wristwatch |
| 11. asparagus | 26. bed | 41. brain | 343. yoga |
| 12. axe | 27. bee | 42. bread | 344. zebra |
| 13. backpack | 28. belt | 43. bridge | 345. zigzag |
| 14. banana | 29. bench | 44. broccoli | |
| 15. bandage | 30. bicycle | 45. broom | |

Recognition of Handwritten Sketches

What do 50 million drawings look like?

Over 15 million players have contributed millions of drawings playing [Quick, Draw!](#) These doodles are a unique data set that can help developers train new neural networks, help researchers see patterns in how people around the world draw, and help artists create things we haven't begun to think of. That's why [we're open-sourcing them](#), for anyone to play with.

Select a drawing



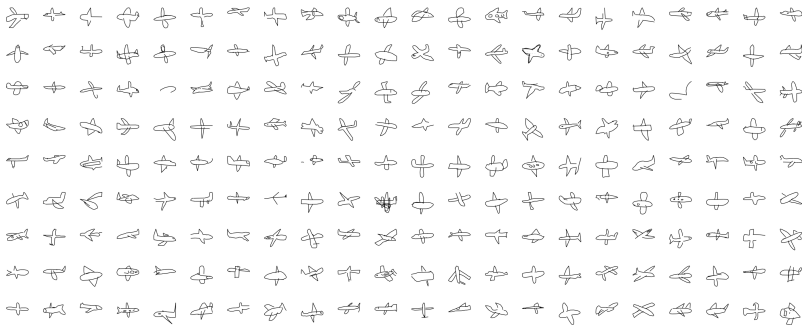
Recognition of Handwritten Sketches

Airplane Samples:

You are looking at 135,821 airplane drawings made by real people... on the internet.

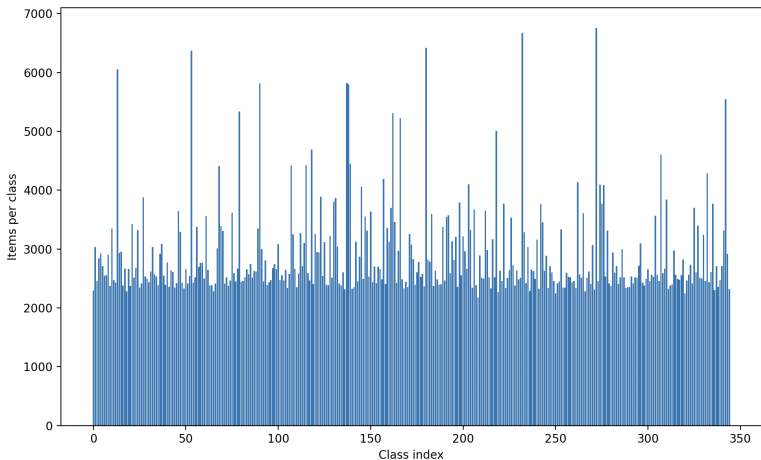
If you see something that shouldn't be here, simply select the drawing and click the flag icon.

It will help us make the collection better for everyone.



Recognition of Handwritten Sketches

Histogram of Drawing Categories:



Recognition of Handwritten Sketches

Dataset Format

Drawings are captured as **timestamped vectors**, tagged with **metadata** converning what the player was asked to draw and player location (country).

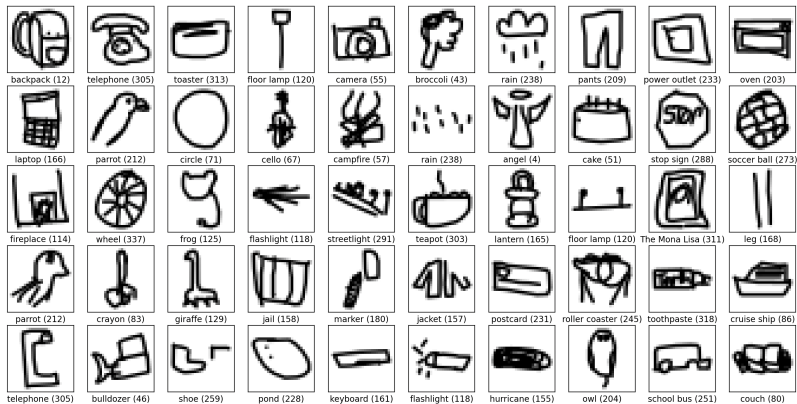
Drawing Array Format:

```
[  
  [ // first stroke ...  
    [ x0, x1, x2, x3, x4, .... ],  
    [ y0, y1, y2, y3, y4, .... ],  
    [ t0, t1, t2, t3, t4, .... ] ],  
  [ // second stroke ...  
    [ x0, x1, x2, x3, x4, .... ],  
    [ y0, y1, y2, y3, y4, .... ],  
    [ t0, t1, t2, t3, t4, .... ] ], ...  
]
```

Source Code: [python-code-ai.d/cnn/TestCNN-Sketch-Recognition01.py](#)

Recognition of Handwritten Sketches

Dataset Preview:



Recognition of Handwritten Sketches

Dataset Preview:

Image shape: (28, 28, 1)

```
[ [ 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 31 63 95 115
    86 50 2 0 0 0 0 0 0 0]
  [ 0 0 0 0 0 75 183 176 152 10 54 182 222 250 255 255 255 255
    255 255 231 151 61 0 0 0]
  [ 0 0 0 0 129 255 216 202 255 98 220 222 157 125 93 60 28 8
    38 82 162 239 254 108 0 0 0]
  [ 0 0 0 33 250 158 4 0 204 209 255 236 105 0 0 0 0
    0 0 0 6 164 254 88 0 0]
  [ 0 0 0 158 243 30 155 235 211 255 254 185 255 97 0 0 0
    0 0 0 3 193 229 10 0]
  [ 0 0 38 251 134 153 251 162 252 245 225 1 180 241 6 0 0
    0 0 0 0 69 255 62 0]
  [ 0 0 150 240 26 245 145 0 116 255 149 0 75 255 58 0 0
    0 0 0 0 29 255 94 0]
  [ 0 0 209 173 77 255 62 0 62 255 68 0 14 251 122 0 0
    0 0 0 0 2 249 128 0]
  [ 0 0 240 138 151 236 4 0 76 255 46 0 0 221 157 0 0
    0 0 0 0 0 218 160 0]
  [ 0 0 251 127 188 188 0 0 76 255 46 0 0 213 164 0 4 45
    0 0 0 0 0 186 193 0]
  [ 0 5 255 117 199 176 0 0 76 255 46 0 0 205 173 0 75 251
    9 0 44 154 2 0 158 226 0]
  [ 0 14 255 108 209 166 0 0 76 255 46 0 0 160 243 102 44 61
    34 35 98 199 132 170 251 252 6]
  [ 0 23 255 99 220 156 0 0 76 255 46 0 0 120 249 255 255 255
    255 255 255 255 247 214 202 255 24 0]
```

Dataset Preview (Normalized):

```
[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.      0.01176471 0.12156863 0.24705882 0.37254903 0.4509804
 0.3372549 0.19607843 0.00784314 0.      0.      0.
 0.      0.      0.      0.      ]
[0.      0.      0.      0.      0.      0.29411766
 0.7176471 0.6901961 0.59607846 0.03921569 0.21176471 0.7137255
 0.87058824 0.98039216 1.      1.      1.      1.
 1.      1.      0.90588236 0.5921569 0.23921569 0.
 0.      0.      0.      0.      ]
[0.      0.      0.      0.      0.5058824 1.
 0.84705883 0.7921569 1.      0.38431373 0.8627451 0.87058824
 0.6156863 0.49019608 0.3647059 0.23529412 0.10980392 0.03137255
 0.14901961 0.32156864 0.63529414 0.9372549 0.99607843 0.42352942
 0.      0.      0.      0.      ]
[0.      0.      0.      0.12941177 0.98039216 0.61960787
 0.01568628 0.      0.8      0.81960785 1.      0.9254902
 0.4117647 0.      0.      0.      0.      0.
 0.      0.      0.      0.02352941 0.6431373 0.99607843
 0.34509805 0.      0.      0.      ]
[0.      0.      0.      0.61960787 0.9529412 0.11764706
 0.60784316 0.92156863 0.827451 1.      0.99607843 0.7254902
 1.      0.38039216 0.      0.      0.      0.
 0.      0.      0.      0.      0.01176471 0.75686276
 0.8980392 0.03921569 0.      0.      ]
```

Recognition of Handwritten Sketches

Model Summary

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 512)	295,424
dense_1 (Dense)	(None, 345)	176,985

Total params: 500,985 (1.91 MB)

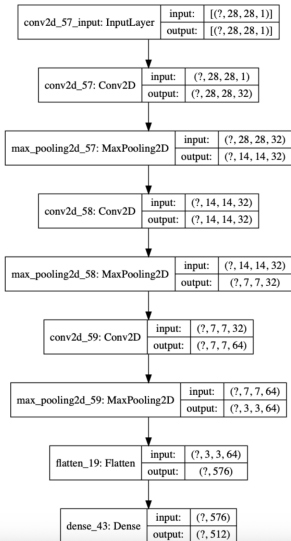
Trainable params: 500,985 (1.91 MB)

Non-trainable params: 0 (0.00 B)

Recognition of Handwritten Sketches

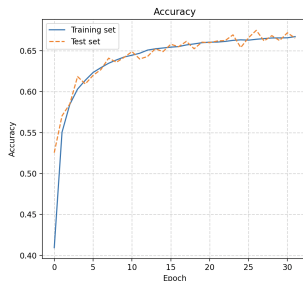
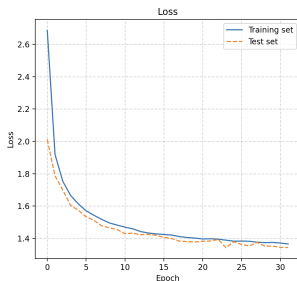
Graphviz of Model

[388]:



Recognition of Handwritten Sketches

Training History



Model Accuracy

Training Set Accuracy

Training loss: 1.36
Training accuracy: 0.66

Validation Set Accuracy

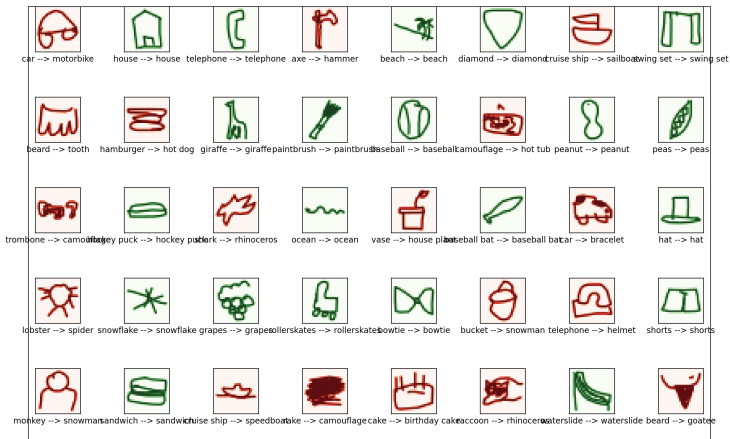
Validation loss: 1.43
Validation accuracy: 0.65

Test Set Accuracy

Test loss: 1.40
Test accuracy: 0.67

Recognition of Handwritten Sketches

Visualize Predictions: Training dataset ...



Legend: Prediction correct → green; Prediction incorrect → red.

Recognition of Handwritten Sketches

Visualize Predictions: Test dataset ...



Legend: Prediction correct → green; Prediction incorrect → red.

References

- Amidi A., and Amidi S., Cheatsheets on Machine Learning and Deep Learning, Various courses on ML at Stanford and MIT, 2018 – 2020.
- Foster D., Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play, O'Reilly, 2019.
- Hochreiter S., and Schmidhuber J., Long Short-Term Memory, Neural Computation, Vol. 9, No. 8, 1997, pp. 1735-1780.
- Nielsen A., Practical Time Series Analysis: Prediction with Statistics and Machine Learning, O'Reilly, 2020.

Appendix A: Object Detection with YOLO

Object Detection with YOLO

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻