MNIST Handwritten Digit Classification with Convolutional Neural Network

Introduction

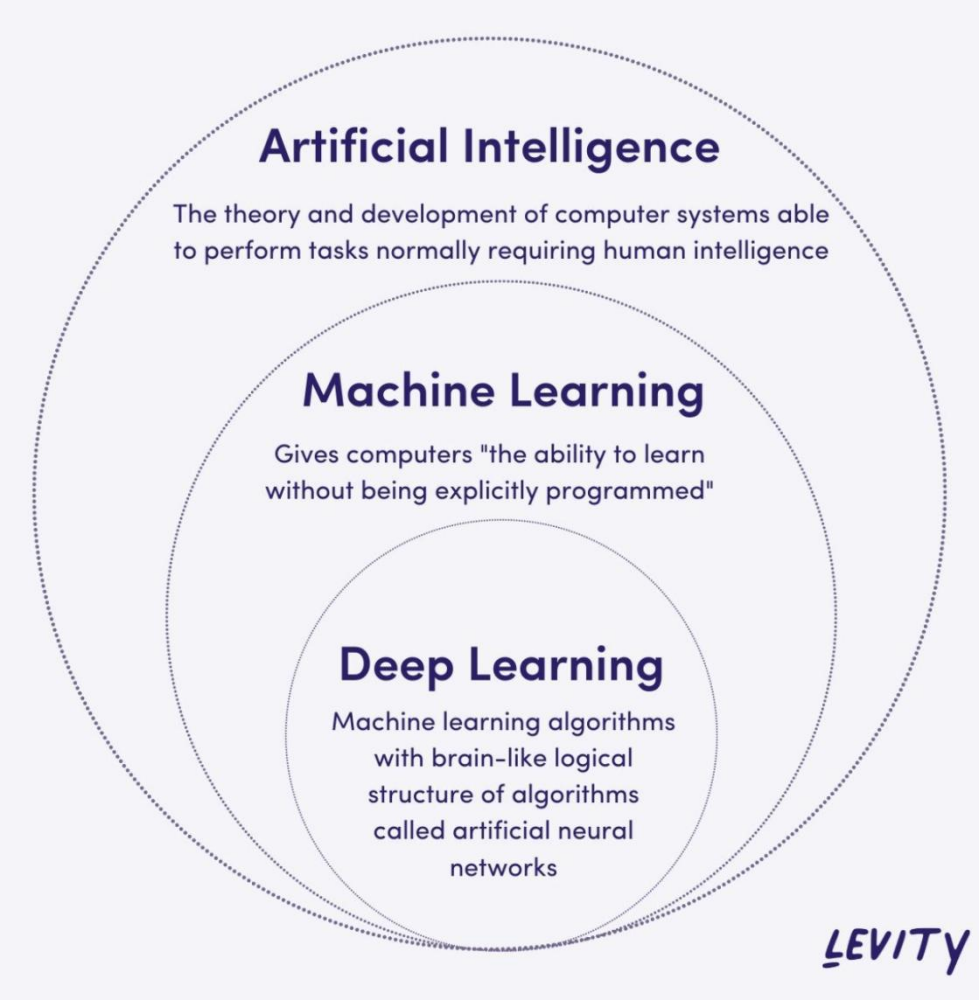


Figure 1. relationship between AI, Machine Learning and Deep Learning (Wolfevicz, 2022)

Artificial Intelligence, or AI, have always been discussed by the society. Its high efficiency of accomplishing different tasks and fast speed of learning draws attention from the world. Nowadays, AI can help improve the opportunity of accessing to education, healthcare, and fight against poverty and climate change. For instance, AI generating essays and pictures help people get inspired and make rough drafts, improving efficiency of creation to a very large extent, and it's still developing yet. Machine learning is a widely-used type of AI that employs algorithms trained on datasets to create models capable of accomplishing tasks typically done by humans.Supervised learning(SL), unsupervised learning(USL), and reinforcement learning(RL) are three primary categories for machine learning problems. SL learns from labeled data, which the expected results are given, to make prediction for future outcomes. For example, email spam detection is trained through the data labeled as either "spam" or "not spam". Each data paired with a correct output. The model will learn which should be collected

as "spam" or "not spam" based on the features of email. In contrast, USL, deals with unlabeled data set to recognize possible patterns or structures by itself. Take customer suggestion as an example, a company might use an unsupervised learning algorithm to find the affection of each customer based on users' features. RL, involves learning from mistakes and success by occasionally receiving feedback or rewards to reach the desired the outcomes. Autonomous driving uses reinforcement learning agent to controls a car in a simulated enviro nment. If the agent leads to a damage of the car, human will tell that it's a mistake, vice versa. The agent learns how to drive safely and efficiently based on the feedback received (Coursera, 2024). Machine learning algorithms can be as simple as an OLS regression (a technique for estimating coefficients of linear regression equations), or as complex as deep learning which analyzes data with a logical structure similar to how human would draw conclusions. Deep learning relies on a layered structure of algorithms known as artificial neural networks (ANN). It requires a huge amount of data to work effectively but minimal human interventions are needed (Wolfewicz, 2022). Being extremely good at finding patterns of complicated data, neural networks are often applied on fields such as speech recognition, natural language processing, computer vision, and recommendation engines (AWS, 2023). To understand how neural networks function, we conducted a research about handwritten digit recognition by a specific kind of neural networks called convolutional neural network (CNN). This paper will discuss the architecture of ANN and CNN, and the differences between them, as well as the learning processes.

Method

Neural Network

Neural network was inspired by the structure and function of biological neurons (Wikipedia, 2024). Normally, it consists of many layers including an input layer, an output layer, and multiple hidden layers in between. Each layer contains numerous neurons.

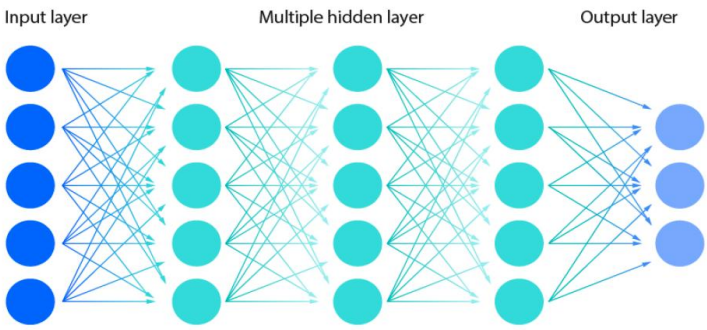


Figure 2. neural network layers (IBM, 2023)

Neural network is helpful in many industries such as medicine, business, pure sciences, data mining, telecommunications, and operations managements (NVIDIA Developer, 2018). It typically performs well on tasks such as speech recognition and image recognition. Regarding the image recognition, there is a specific application —— MNIST handwritten digit classification. The MNIST database, or Modified National Institute of Standards and Technology database, is a large database of handwritten digits that is commonly used for training and testing in the field of machine learning (Wikipedia, 2019). In this problem, the input will be a 28x28 pixel grayscale images of handwritten single digits between 0 and 9. The image will first be flattened, converting into a one-dimensional array. Pixel values will be normalized to the range [0,1], 0 representing black, and 1 representing white. Each neuron holds a specific number between 0 and 1, known as the activation. Through each layer, activations gain weights and biases to determine the activation in the next layer. Finally the results will be shown in the output layer.

During learning, the model goes through several key processes. First, all weights and biases are initialized randomly. Next, it computes the results with those weights and biases. This is the forward propagation. After that, with the labeled data set, it evaluates the difference between the predicted results and actual targets. Then it adjusts the weights and biases, starting from the output layer and move backward through the network. This is the backward propagation. The forward and backward propagation will be repeated many times until the network can correctly recognize the handwritten digits with a very high accuracy.

Convolutional Neural Network

Traditional artificial neural networks is limited to data set with lower complexity, whereas the convolutional neural network (CNN) can deal with more complicated images (O'Shea and Nash, 2015). The MNIST data set uses 28x28 pixel black and white images, which only 784 weights are needed for each neuron in the first layer, allowing ANN to perform well on this. However, colored images of objects with much more information are not manageable for ANN, so using CNN to perform this kind of tasks will be more suitable.

Convolutional neural network is similar to ANN but it's more specialized in handling spatial data more effectively and accurately. Since CNN is designed for images, its structure is made to be the best suit for its input. One key difference between the architecture of CNN and

ANN is that CNN's layers comprises neurons organized in three dimensions: height, width, and depth.

Besides from the input and the output layers, CNN consist of 3 important layers: convolutional layer (CONV), pooling layer (POOL), and fully connected layer (FC).

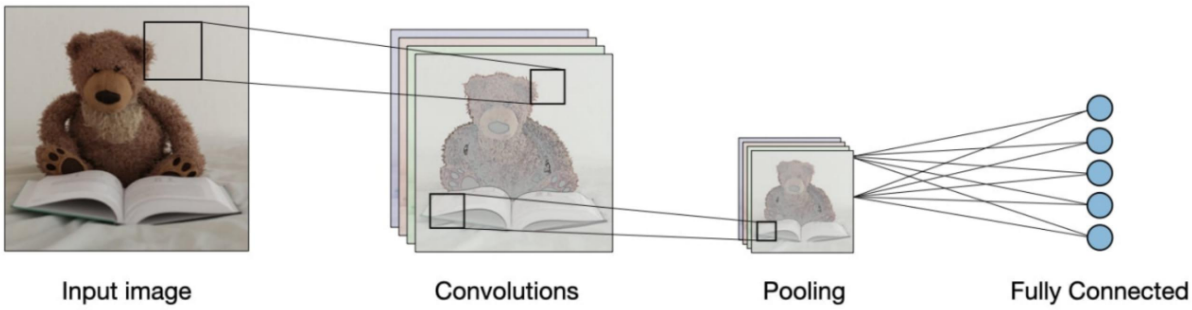
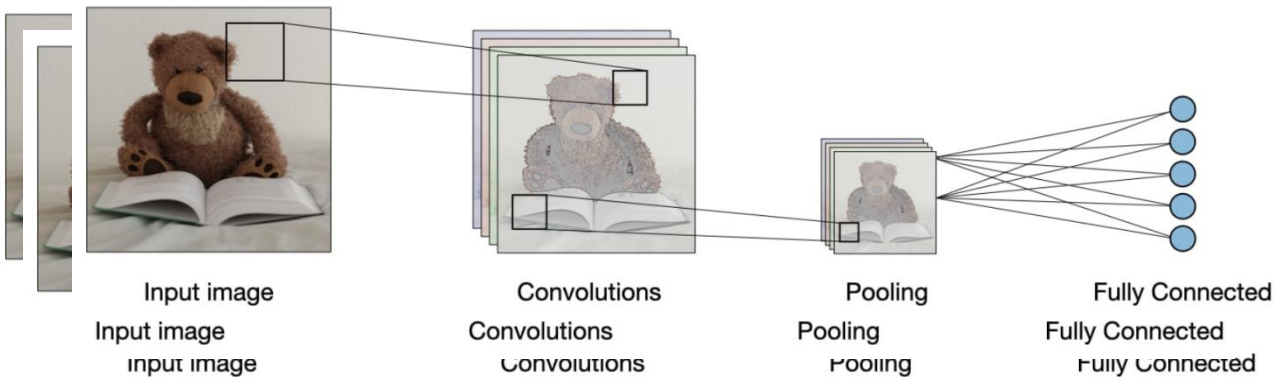


Figure 3. Layers of CNN (Amidi and Amidi, 2019)

Convolutional layer is designed to extract characteristics from input image through filter. The process of extracting is similar to a printer. First, provided that the input image is the handwritten numbers from 0 to 9 with size of 28*28*3, convolutional layer will divide the input image into several small cubes in same size. Each square is labeled with its RGB value. Convolutional layer has different filters called learned kernels. Each kernel could be regarded as a matrix with size of 3*3 with same depth with the input image. Kernel corresponds to a feature of the number from 0 to 9, represented by the numbers learned from



training. Kernel will overlap the first part of the input image on top left corner and start scanning from left to right and up to down with stride 1. After that, each kernel will calculate the dot product between kernel weight and the corresponding portion of input image. The new image printed out is called activation map. This new image highlights the feature of the original through trained weights and biases. In total, how many kernels depend how many activation maps are printed. Finally, all activation maps will be overlapped together.

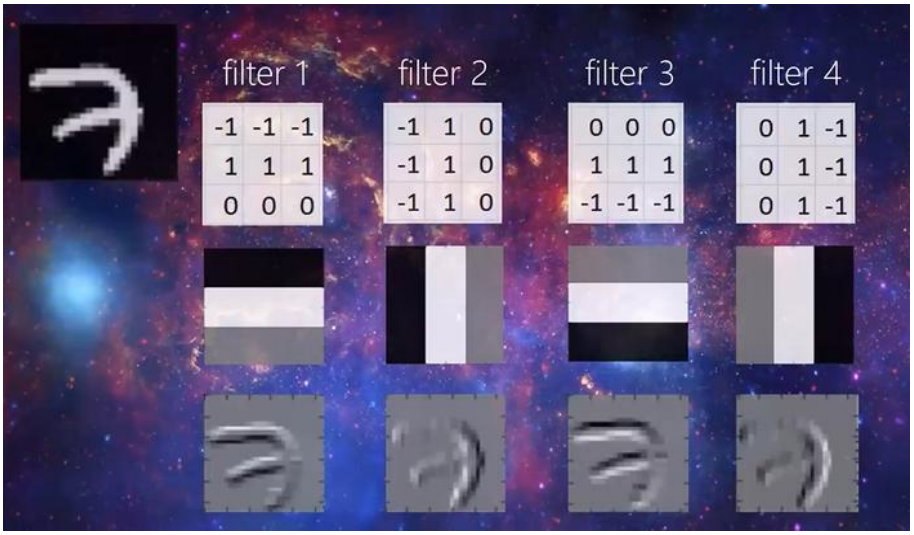


Figure 4. Convolutional Neural Networks (CNNs) explained (Deeplizard)

Pooling layer is used to reduce the size of image and computational cost, make the model finish training faster. In detail, max and average pooling, used most commonly, are applied to extract the maximum and average value. Thus, for each activation map, a filter is used again to scan through the map with a specific stride from the top left corner. After the operation of convolutional layer, each activation map of handwritten image is still in size of 28*28*3. The filter in pooling layer could be in size of 4*4 and stride 4. Therefore, the result output will be in dimension of 7*7*3. In max pooling, filter will extract a maximum number within the range of filter. For instance, one portion of map corresponding to filter has 8 as its maximum in 16 numbers. 8 will replace these numbers for the new map. Similarly, in average pooling, filter will calculate the average number within the range, adding all 16 number and dividing by 16.

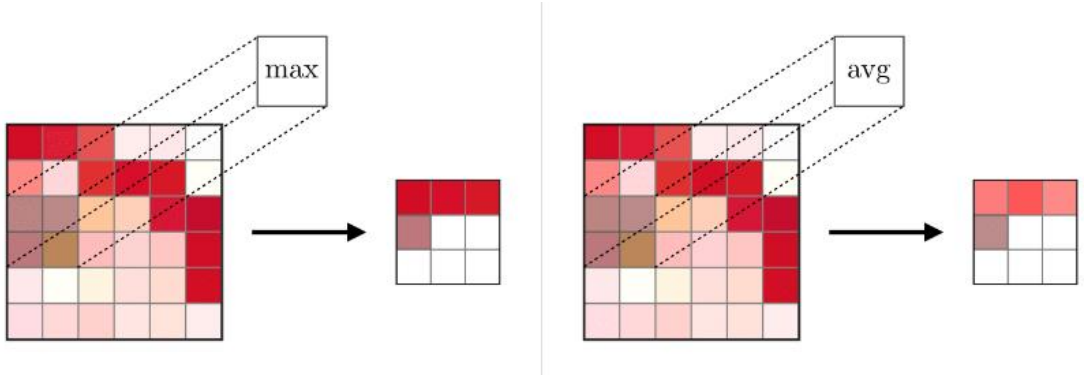


Figure 5. Pooling (Afshine Amidi and Shervine Amidi, 2019)

The operations of convolutional and pooling layers are repeated several times to get a totally new output with new dimension. For the next step, model will flatten the map by unfolding this three-dimension map as a straight line. In other words, map is sliced into pieces that can ignore the depth. Next, these pieces will be cut into tiny squares. Overall, the total quantities

of these sqaures are depth*length*width. Finally, these sqaures will be used as the input for fully connected layer.

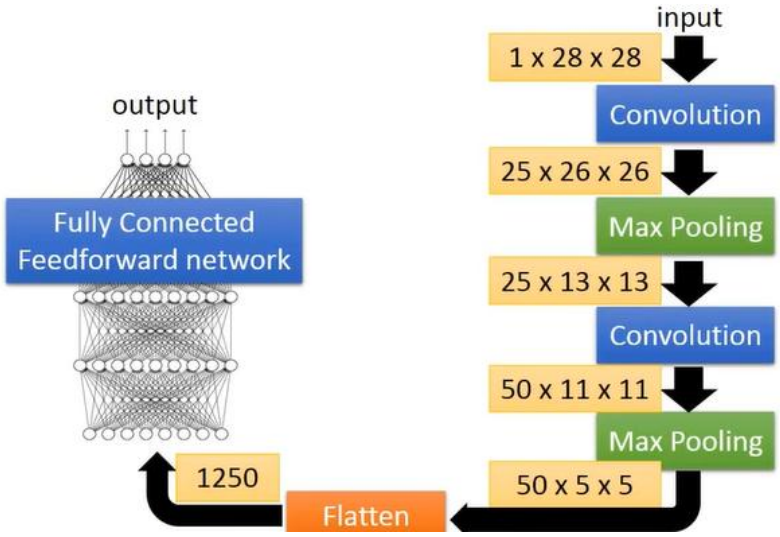


Figure 6. Convolutional Neural Network (Lee Hungyi, 2016)

Fully connected layer aims to make prediction and decision. These flatten inputs will be connected to all neurons in next layer. Neuron is the basic unit of processing that helps transmit. The neurons in previous layer will connect to the every neuron in next layer. Therefore, a interconnected network is formed. The final layer will classify the handwritten numbers into corresponding numbers from 0 to 9.

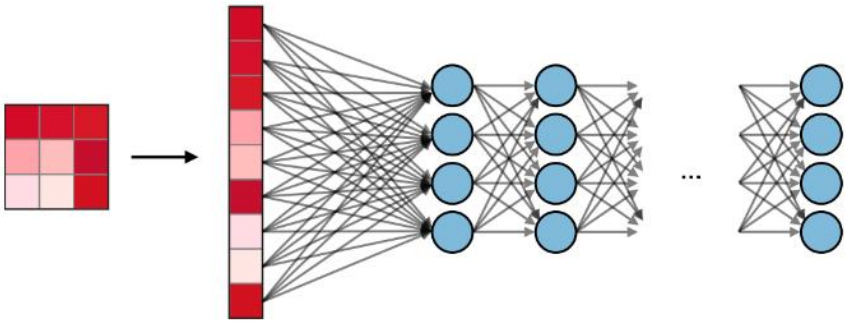


Figure 7. Full Connected (F. Amidi and S. Amidi)

Python Programme

In order to achieve classification of handwritten numbers from 0 to 1 through convolutional neural network, python plays an important role and helps.

For starters, libraries in python that are imported to achieve the function in efficient way. Tensorflow is used to help build up convolutional neural network. Matplotlib helps to draw the graph and numpy supports a large number of dimensional array and matrix operations.

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

After that, datasets, containing 60,000 images of handwritten numbers from 0 to 9 that has dimension of 28*28, are called to train the model. They are stored in x_train. Each image has correspond correct number it represents, which are stored in y_train.

```python
mnist=tf.keras.datasets.mnist
```

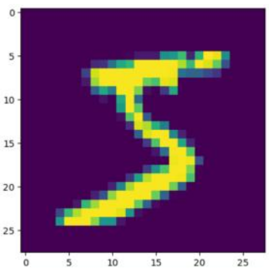
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

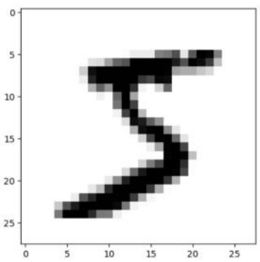
```python
x_train.shape
```

```
(60000, 28, 28)
```

Next, the images in datasets are converted to binary graphs which only have color black and white. As shown in the figure XII and XIII:

```python
plt.imshow(x_train[0])
plt.show()
plt.imshow(x_train[0], cmap = plt.cm.binary)
```

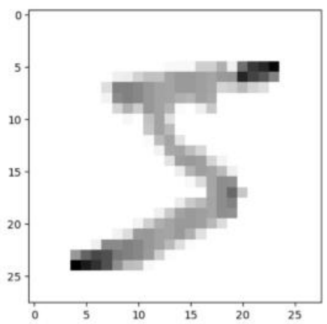




Then, the dots on the graph could be represented by numbers to show its color depth. Numbers that close to 0 mean it is more like white. Vice Versa, closing to 255 represents tendency of black. To help computer process these numbers efficiently, normalization is used to adjust numbers on each dot from 0 to 1.

```python
 x_train = tf.keras.utils.normalize (x_train, axis = 1)
x_text = tf.keras.utils.normalize(x_test, axis = 1)
plt.imshow(x_train[0], cmap = plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x2d4ed9d1b10>
```



```
   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]
```

```
 0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
 0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
 0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
 0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
 0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
 0.         0.         0.         0.        ]
```

Because Convolutional Neural Network can only handle with three-dimension image, the original images need to be added with another dimension called depth.

```
import numpy as np
IMG_SIZE=28

x_trainr=np.array(x_train).reshape(-1,IMG_SIZE, IMG_SIZE,1)
x_testr=np.array(x_text).reshape(-1,IMG_SIZE, IMG_SIZE,1)
print("Training Samples dimension",x_trainr.shape)
print("Training Samples dimension",x_testr.shape)
```

```
Training Samples dimension (60000, 28, 28, 1)
Training Samples dimension (10000, 28, 28, 1)
```

For the next step, training starts officially. Model uses 64 kernels with size of 3*3 to scan through the input image. Then, applying activation “ReLu” function to convert all negative products into 0. Max pooling with size of 2*2 is applied to simplify the input images. Repeating these steps for three times. Each image will be changed to size of 1*1 with 64 neurons. Then, flattening the image into a line with only one dimension. After that, the line will connect to dense layer (fully-connected layer) with 64 neurons. First dense layer will connect to second layer with 32 neurons. Second layer connects to final layer with only 10 neurons, representing corresponding 10 numbers. Activation function is used again for each connection.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
```

```
model=Sequential()

model.add(Conv2D(64,(3,3), input_shape = x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add (Flatten())
model.add (Dense(64))
model.add (Activation("relu"))

model.add (Dense(32))
model.add (Activation("relu"))

model.add(Dense(10))
model.add(Activation('softmax'))
```

Then, 70 percent of data are inputted into the model in order to increase the accuracy of model. The model will record the accuracy, loss, validation accuracy and loss for each epoch, containing 10,000 input images. Validation accuracy aims to supervise whether model is over-fitting and issues a sign for stopping the process. After enhancing the accuracy, the rest of 10,000 test samples is used to evaluate if the model reaches the accuracy it trains.

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
```

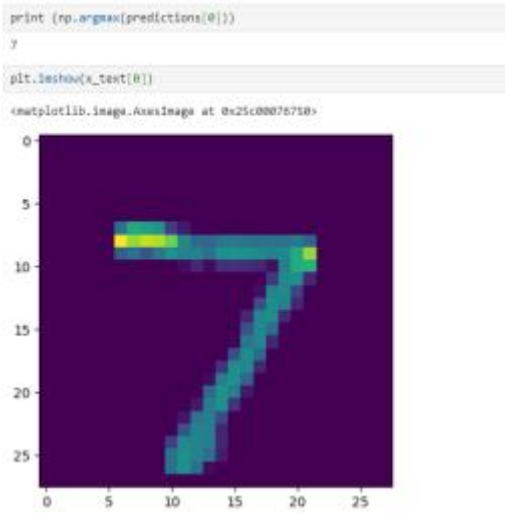
```
model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)
```

```
Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 32s 20ms/step - accuracy: 0.7963 - loss: 0.6321 - val_accuracy: 0.9649 - val_loss: 0
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 25s 19ms/step - accuracy: 0.9668 - loss: 0.1112 - val_accuracy: 0.9734 - val_loss: 0
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 24s 18ms/step - accuracy: 0.9777 - loss: 0.0713 - val_accuracy: 0.9746 - val_loss: 0
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 24s 18ms/step - accuracy: 0.9831 - loss: 0.0533 - val_accuracy: 0.9818 - val_loss: 0
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 24s 18ms/step - accuracy: 0.9883 - loss: 0.0390 - val_accuracy: 0.9711 - val_loss: 0
<keras.src.callbacks.history.History at 0x25c0004dc10>
```

```
test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9670 - loss: 0.1165
Test loss on 10,000 test samples 0.09380108118057251
Validation Accuracy on 10,000 test samples 0.9715999960899353
```

To examine the accuracy of prediction, comparing the output from prediction for first image and the correct corresponding answer. After all, whole training process is over. The model finishes learning and improving its precision.



Discussion

10 volunteers were invited to write a number between 0 to 9 for us to test the model.

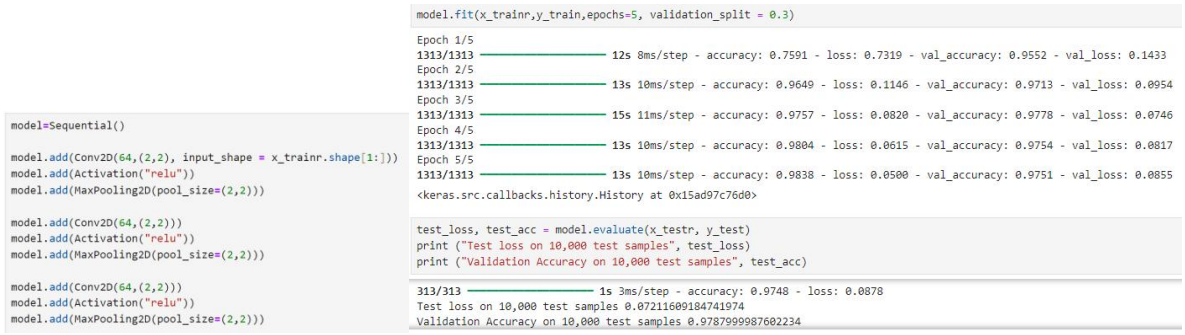
subject	success rate
S1	90%
S2	80%
S3	70%
S4	80%
S5	80%
S6	80%
S7	90%
S8	50%
S9	60%
S10	90%
mean ± standard deviation	77% ±13.375%

The mean accuracy of the classification results of the 10 subjects is 77%, with a standard deviation of 13.375%. The highest accuracy is 90%, indicating that the model correctly classifies 9 of the 10 digits written by the subject. The lowest accuracy is 50%, indicating that

only half of the 10 digits written by the subject are classified correctly. 7 written by 9 subjects are recognized as 1 and the prediction shows that the model is confident of the result. That is probably because that handwritten 7 are similar to 1. Cultural difference may also contribute to the low accuracy. The database mainly contains handwritten digit from North American people, but our subjects are all Chinese.

To investigate the effect of different parameters on the accuracy, we adjusted parameters to observe the changes.

First, we changed kernel size from 3*3 to 2*2 and 4*4, there is not significant changes. The accuracy still remains on the range of 97 to 98.

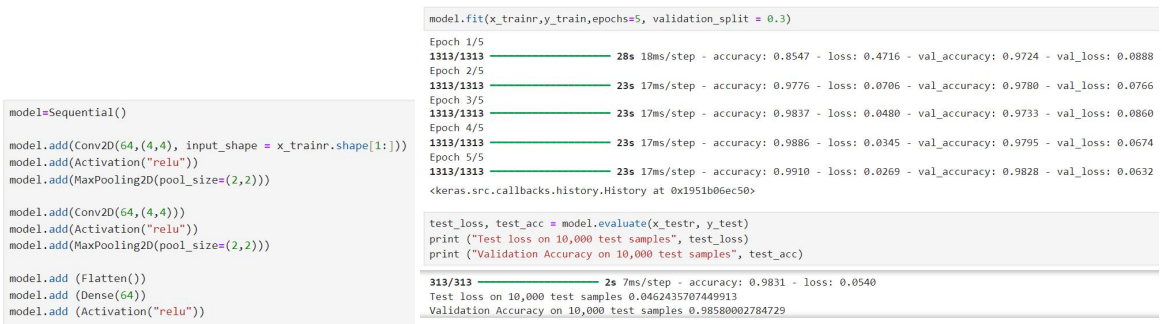


```
model=Sequential()

model.add(Conv2D(64,(2,2), input_shape = x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(2,2)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(2,2)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)

Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.7591 - loss: 0.7319 - val_accuracy: 0.9552 - val_loss: 0.1433
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.9649 - loss: 0.1146 - val_accuracy: 0.9713 - val_loss: 0.0954
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - accuracy: 0.9757 - loss: 0.0820 - val_accuracy: 0.9778 - val_loss: 0.0746
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.9804 - loss: 0.0615 - val_accuracy: 0.9754 - val_loss: 0.0817
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - accuracy: 0.9838 - loss: 0.0500 - val_accuracy: 0.9751 - val_loss: 0.0855
<keras.src.callbacks.history.History at 0x15ad97c76d0>
```

```
test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9748 - loss: 0.0878
Test loss on 10,000 test samples 0.07211609184741974
Validation Accuracy on 10,000 test samples 0.9787999987602234
```



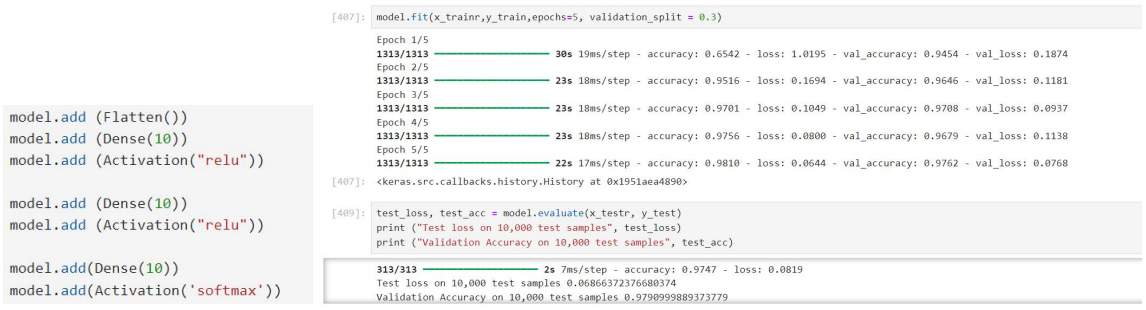
```
model=Sequential()

model.add(Conv2D(64,(4,4), input_shape = x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(4,4)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add (Flatten())
model.add (Dense(64))
model.add (Activation("relu"))
```

```
model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)

Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.8547 - loss: 0.4716 - val_accuracy: 0.9724 - val_loss: 0.0888
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9776 - loss: 0.0706 - val_accuracy: 0.9780 - val_loss: 0.0766
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9837 - loss: 0.0480 - val_accuracy: 0.9733 - val_loss: 0.0860
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9886 - loss: 0.0345 - val_accuracy: 0.9795 - val_loss: 0.0674
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9910 - loss: 0.0269 - val_accuracy: 0.9828 - val_loss: 0.0632
<keras.src.callbacks.history.History at 0x1951b06ec50>
```

```
test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9831 - loss: 0.0540
Test loss on 10,000 test samples 0.0462435707449913
Validation Accuracy on 10,000 test samples 0.98580002784729
```

Next, we reduce the number of neurons in each dense layer from 64 and 32 to 10. Still, the accuracy is not influenced.



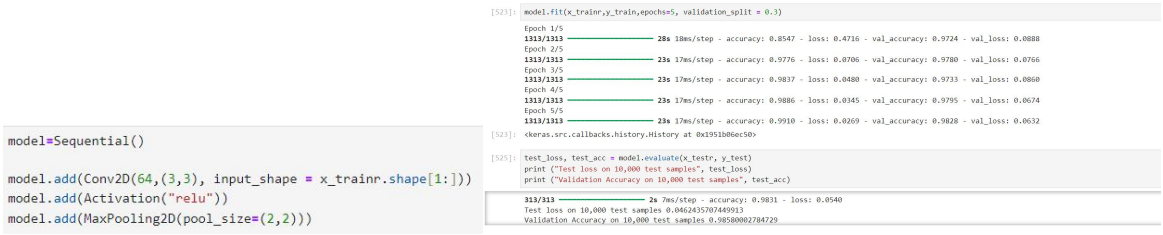
```
model.add (Flatten())
model.add (Dense(10))
model.add (Activation("relu"))

model.add (Dense(10))
model.add (Activation("relu"))

model.add(Dense(10))
model.add(Activation('softmax'))
```

```
[407]: model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)

Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.6542 - loss: 1.0195 - val_accuracy: 0.9454 - val_loss: 0.1874
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 18ms/step - accuracy: 0.9516 - loss: 0.1694 - val_accuracy: 0.9646 - val_loss: 0.1181
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 18ms/step - accuracy: 0.9701 - loss: 0.1049 - val_accuracy: 0.9708 - val_loss: 0.0937
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 18ms/step - accuracy: 0.9756 - loss: 0.0800 - val_accuracy: 0.9679 - val_loss: 0.1138
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 22s 17ms/step - accuracy: 0.9810 - loss: 0.0644 - val_accuracy: 0.9762 - val_loss: 0.0768
[407]: <keras.src.callbacks.history.History at 0x1951aea4890>
```

```
[409]: test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9747 - loss: 0.0819
Test loss on 10,000 test samples 0.06866372376680374
Validation Accuracy on 10,000 test samples 0.9790999889373779
```

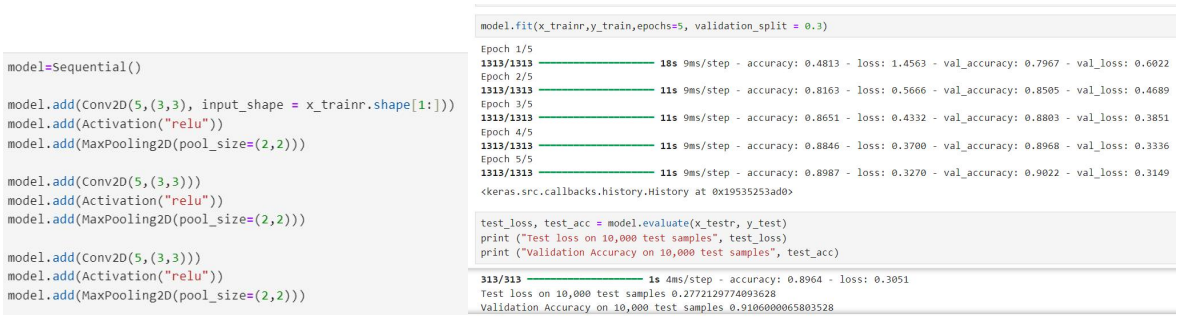
Moreover, we try to use only one convolutional layer instead of three. the change is also negligible.



```
model=Sequential()

model.add(Conv2D(64,(3,3), input_shape = x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
[523]: model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)

Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.8547 - loss: 0.4716 - val_accuracy: 0.9724 - val_loss: 0.0888
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9776 - loss: 0.0706 - val_accuracy: 0.9780 - val_loss: 0.0766
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9837 - loss: 0.0480 - val_accuracy: 0.9733 - val_loss: 0.0860
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9886 - loss: 0.0345 - val_accuracy: 0.9795 - val_loss: 0.0674
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - accuracy: 0.9910 - loss: 0.0269 - val_accuracy: 0.9828 - val_loss: 0.0632
[523]: <keras.src.callbacks.history.History at 0x1951b06ec50>

[525]: test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9831 - loss: 0.0540
Test loss on 10,000 test samples 0.04624357074499113
Validation Accuracy on 10,000 test samples 0.9858000278472900
```

Finally, after changing the number of kernels from 64 to 5, we discover that accuracy reduces from about 98 to 91. In case of uncertainty, we run the model several times to observe the accuracy when number of kernels is 5. The result indicates that the reduction of number of kernels will lead to the decrease of accuracy.



```
model=Sequential()

model.add(Conv2D(5,(3,3), input_shape = x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(5,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(5,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.fit(x_trainr,y_train,epochs=5, validation_split = 0.3)

Epoch 1/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - accuracy: 0.4813 - loss: 1.4563 - val_accuracy: 0.7967 - val_loss: 0.6022
Epoch 2/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step - accuracy: 0.8163 - loss: 0.5666 - val_accuracy: 0.8505 - val_loss: 0.4689
Epoch 3/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step - accuracy: 0.8651 - loss: 0.4332 - val_accuracy: 0.8803 - val_loss: 0.3851
Epoch 4/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step - accuracy: 0.8846 - loss: 0.3700 - val_accuracy: 0.8968 - val_loss: 0.3336
Epoch 5/5
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step - accuracy: 0.8987 - loss: 0.3270 - val_accuracy: 0.9022 - val_loss: 0.3149
<keras.src.callbacks.history.History at 0x19535253ad0>

test_loss, test_acc = model.evaluate(x_testr, y_test)
print ("Test loss on 10,000 test samples", test_loss)
print ("Validation Accuracy on 10,000 test samples", test_acc)

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8964 - loss: 0.3051
Test loss on 10,000 test samples 0.2772129774093628
Validation Accuracy on 10,000 test samples 0.9106000065803528
```

Conclusion

Convolutional Neural Network, containing several convolutional, pooling and fully-connected layers, can be applied achieve the classification of handwritten numbers. By using Jupyter Notebook, we construct and train our model with data sets. Finally, we collect handwritten numbers from the people around us and input into the model to study the result. Even thought the accuracy of the result is not as perfect as we thought, we learn the existent problem that motivates us to spend more time to research and improve it in the future.

Future Work

To further improve our model, we will explore and investigate more about the CNN architecture and hyper-parameters to reach higher test accuracy. We may also collect our own data set to train a model that fits better f or students and teacher in our school so that we can help teachers to record test scores more efficiently. We can also apply our model on the processing of bank check, postal code identification and so on.

Acknowledgement

Thanks for our Adviser Wang Siwen who has always supported us throughout the whole research. Without his help, we would not be able to establish a deep understanding of our topic and research method to finish our project.We also appreciate the videos contributed by 3Blue1Blue and DeepLearning_by_PhDScholar on YouTube, which assisted us to understand and build the model. We would also like to give credits to our AI friend ChatGPT who always answers our questions and helps us with coding problems.

Reference

AWS (2023). What is a neural network? AI and ML guide - AWS. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/what-is/neural-network/.

Coursera Staff. (2024). Machine Learning vs. Neural Networks: What's the Difference? [online] Available at: https://www.coursera.org/articles/machine-learning-vs-neural-networks [Accessed 12 June. 2024].

Geeksforgeeks. (2024). What is a neural network? [online] Availavle at: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/ [Accessed 12 June. 2024].

NVIDIA Developer. (2018). Artificial Neural Network. [online] Available at: https://developer.nvidia.com/discover/artificial-neural-network [Accessed 12 June. 2024].

IBM (2023). What Are Neural Networks? [online] Available at: https://www.ibm.com/topics/neural-networks [Accessed 11 June. 2024].

O'shea, K. and Nash, R. (2015). An Introduction to Convolutional Neural Networks. [online] Available at: https://arxiv.org/pdf/1511.08458 [Accessed 12 June. 2024].

Wikipedia (2019). MNIST database. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/MNIST_database [Accessed 12 June. 2024].

Wikipedia. (2024). Neural network (machine learning). [online] Available at: https://en.wikipedia.org/wiki/Neural_network_(machine_learning) [Accessed 11 June. 2024].

Wolfewicz, A. (2022). Deep learning vs. machine learning – What's the difference? [online] levity.ai. Available at: https://levity.ai/blog/difference-machine-learning-deep-learning.

Afshine Amidi and Shervine Amidi. (2019). Convolutional Neural Networks cheatsheet. [online] Available at: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks