

Computer Engineering Department

# BIM 470 – Neural Networks

**MNIST Digit Recognition with CNN**

Prepared By

Ecem ŞEN

Necip AŞIK

Date: 07.01.2021

**What is the CNN?**

Artificial Neural Networks (ANNs) are computational processing systems of which are heavily inspired by way biological nervous systems (such as the human brain) operate. ANNs are mainly comprised of a high number of interconnected computational nodes (referred to as neurons), of which work entwine in a distributed fashion to collectively learn from the input in order to optimise its final output.

The two key learning paradigms in image processing tasks are supervised and unsupervised learning. Supervised learning is learning through pre-labelled inputs, which act as targets. For each training example there will be a set of input values (vectors) and one or more associated designated output values. The goal of this form of training is to reduce the models overall classification error, through correct calculation of the output value of training example by training.

Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimise through learning. Each neuron will still receive an input and perform a operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply

The only notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model.

One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. Common machine learning benchmarking datasets such as the MNIST database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just 28 × 28. With this dataset a single neuron in the first hidden layer will contain 784 weights (28×28×1 where 1 bare in mind that MNIST is normalised to just black and white values), which is manageable for most forms of ANN.

If you consider a more substantial coloured image input of 64 × 64, the number of weights on just a single neuron of the first layer increases substantially to 12, 288. Also take into account that to deal with this scale of input, the network will also need to be a lot larger than one used to classify colour-normalised MNIST digits, then you will understand the drawbacks of using such models.

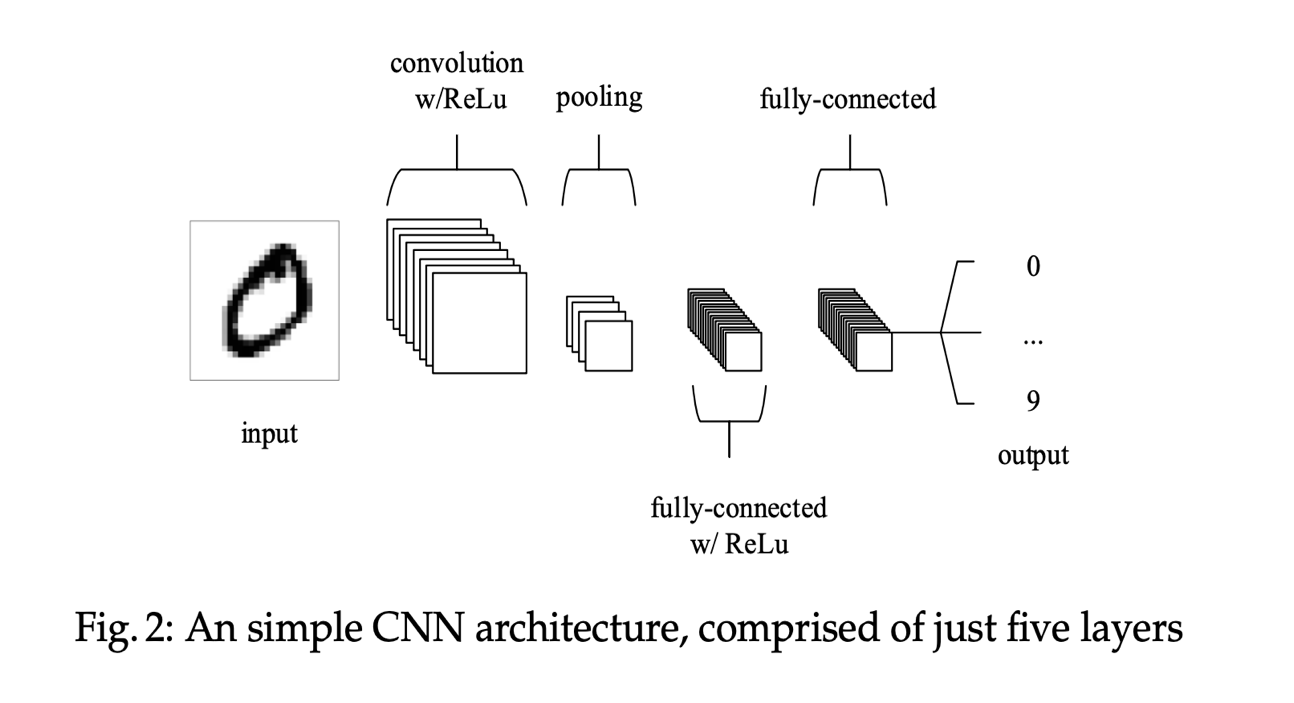
**CNN architecture**

CNNs primarily focus on the basis that the input will be comprised of images. This focuses the architecture to be set up in way to best suit the need for dealing with the specific type of data

One of the key differences is that the neurons that the layers within the CNN are comprised of neurons organised into three dimensions, the spatial dimensionality of the input (height and the width) and the depth. The depth does not refer to the total number of layers within the ANN, but the third dimension of a activation volume. Unlike standard ANNS, the neurons within any given layer will only connect to a small region of the layer preceding it.

In practice this would mean that for the example given earlier, the input ’volume’ will have a dimensionality of 64 × 64 × 3 (height, width and depth), leading to a final output layer comprised of a dimensionality of 1 × 1 × n (where n represents the possible number of classes) as we would have condensed the full input dimensionality into a smaller volume of class scores filed across the depth dimension.

CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure 2.



The basic functionality of the example CNN above can be broken down into four key areas.

1. As found in other forms of ANN, the input layer will hold the pixel values of the image.
2. The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to ReLu) aims to apply Introduction to Convolutional Neural Networks 5 an ’elementwise’ activation function such as sigmoid to the output of the activation produced by the previous layer.

1. The pooling layer will then simply perform downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.
2. The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance[1].

**What is the MNIST Database ?**

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field [2].

**CODE**

**Libraries:**

1. # CNN for the MNIST Dataset
2. **from** keras.datasets **import** mnist
3. **from** keras.models **import** Sequential
4. **from** keras.layers **import** Dense
5. **from** keras.layers **import** Dropout
6. **from** keras.layers **import** Flatten
7. **from** keras.layers.convolutional **import** Conv2D
8. **from** keras.layers.convolutional **import** MaxPooling2D
9. **from** keras.utils **import** np\_utils

**Load Data:**

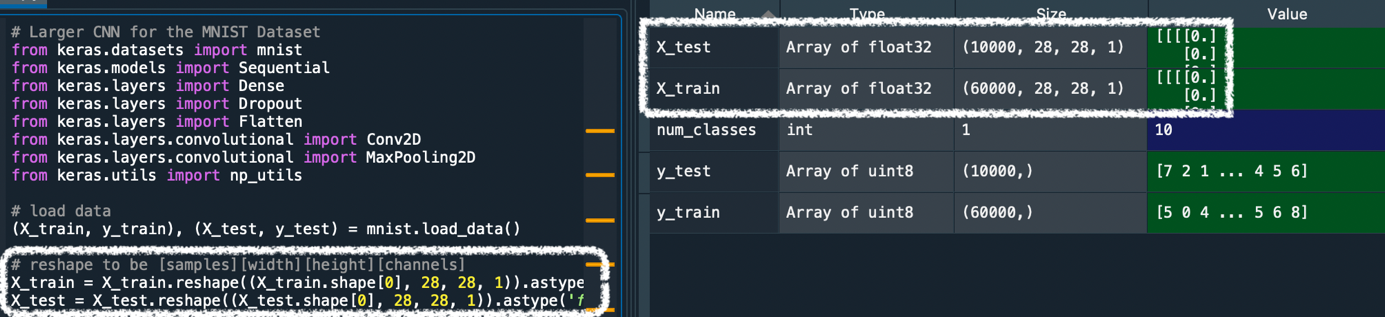
1. (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

metin, ekran görüntüsü, skorbord içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Reshape Data:**

1. X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1)).astype('float32')
2. X\_test = X\_test.reshape((X\_test.shape[0], 28, 28, 1)).astype('float32')



**Normalize inputs:**

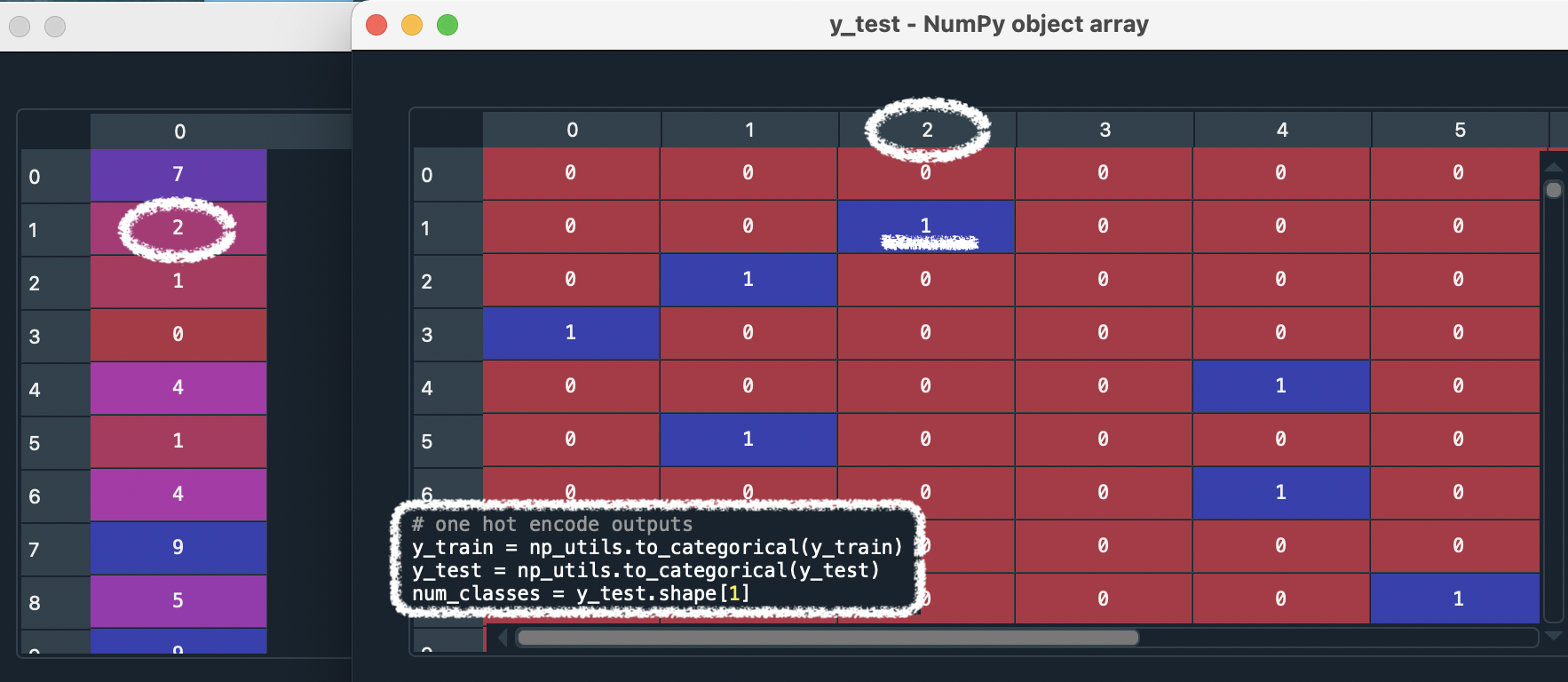
1. X\_train = X\_train / 255
2. X\_test = X\_test / 255

metin, skorbord, ekran, kargo konteyneri içeren bir resim

Açıklama otomatik olarak oluşturuldu

**One-hot encode outputs:**

1. y\_train = np\_utils.to\_categorical(y\_train)
2. y\_test = np\_utils.to\_categorical(y\_test)
3. num\_classes = y\_test.shape[1]



**Define the model:**

1. **def** model():
2. model = Sequential()
3. model.add(Conv2D(30, (5, 5), input\_shape=(28, 28, 1), activation='relu'))
4. model.add(MaxPooling2D())
5. model.add(Conv2D(15, (3, 3), activation='relu'))
6. model.add(MaxPooling2D())
7. model.add(Dropout(0.2))
8. model.add(Flatten())
9. model.add(Dense(128, activation='relu'))
10. model.add(Dense(50, activation='relu'))
11. model.add(Dense(num\_classes, activation='softmax'))

**Compile the model:**

1. model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])
2. **return** model

**Build the model:**

1. model = model()

**Fit the model:**

1. model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200)

**Final evaluation the model:**

1. scores = model.evaluate(X\_test, y\_test, verbose=0)
2. **print**("CNN Error: %.2f%%" % (100-scores[1]\*100))

REFERENCES

**[1]** O’Shea, K.: Introduction to Convolutional Neural Networks Aberystwyth University, (2015).

# [2] THE MNIST DATABASE of handwritten digits [Yann LeCun](http://yann.lecun.com/), Courant Institute, NYU, [Corinna Cortes](http://homepage.mac.com/corinnacortes/), Google Labs, New York, [Christopher J.C. Burges](http://research.microsoft.com/en-us/people/cburges/), Microsoft Research, Redmond