**Author**

**Mohammad Javad Ranjbar**

mohammadjavadranjbarkalahroodi@gmail.com

Abstract. With the availability of large databases and recent developments in deep learning methods, the performance of artificial intelligence systems in many of the advanced tasks has reached the human level or even beyond. Significant examples of this progress can be found in areas such as image classification, emotion analysis, speech recognition, etc. In this report, I will explain the steps of training simple handwritten digit recognition systems with different properties (such as different optimizers, learning rates, etc.) and then I will explain what caused the difference in the results.

## 1. Introduction:

With the developments of the society and the increase in the need for automation, handwritten digits recognition task has been a popular and important area of research in pattern recognition field, In this project, I trained several of deep neural networks and deep convolutional neural networks introduced by [3], improved by [4], refined and simplified by [5, 6, 7] to analyze what are the key components of the training a network and how to train the best network for the MNIST dataset. In order to discover the best approach amongst the usual methods, I compared the two most frequent of them: Multi-Layer Perceptron and Convolutional Neural Networks but some differences make CNN better than MLP in the pattern recognition task (However, they both have good accuracy on the MNIST dataset). For example, MLPs are not translation-invariant which means they react differently to an image and its shifted version.

## 2. MNIST dataset:

The MNIST dataset is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. MNIST dataset contains 60,000 training images and 10,000 testing images. These images are 28x28 pixels and in black and white format.

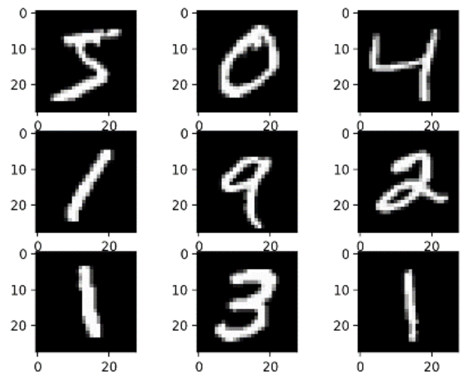
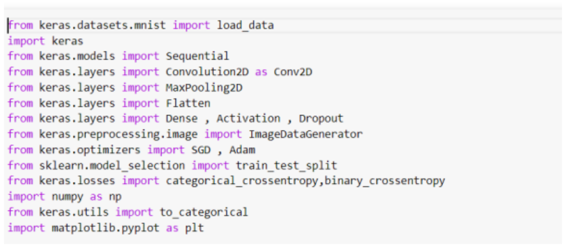


Figure 1 Sample images from MNIST

## 3. **Code**:

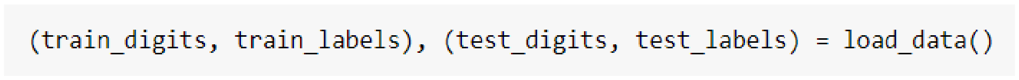
First, we import the required Modules



```
from keras.datasets.mnist import load_data
import keras
from keras.models import Sequential
from keras.layers import Convolution2D as Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense , Activation , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD , Adam
from sklearn.model_selection import train_test_split
from keras.losses import categorical_crossentropy,binary_crossentropy
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

Figure 2 Modules

Then we load the training and testing data

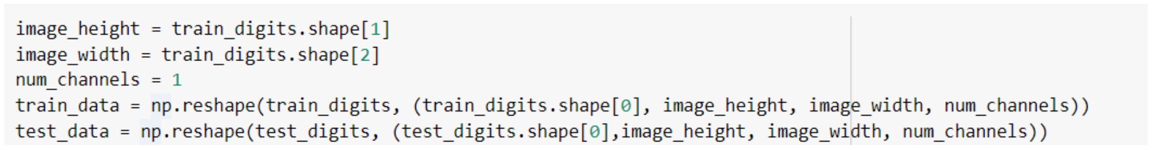


```
(train_digits, train_labels), (test_digits, test_labels) = load_data()
```

Figure 3 Splitting Data

Now we prepare the data before giving it to the network.

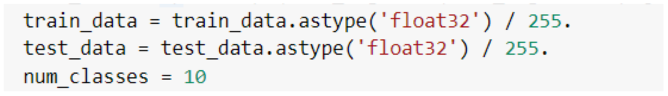
As mentioned earlier, the images are in 28x28 pixels and black and white (one channel).



```
image_height = train_digits.shape[1]
image_width = train_digits.shape[2]
num_channels = 1
train_data = np.reshape(train_digits, (train_digits.shape[0], image_height, image_width, num_channels))
test_data = np.reshape(test_digits, (test_digits.shape[0],image_height, image_width, num_channels))
```

Figure 4 Preprocessing

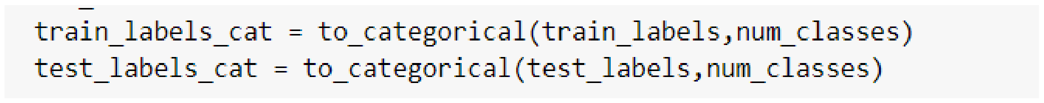
We normalize the value of pixels in each image to make calculations easier and to prevent overflow of the network, we have ten digits from 0 to 9, which is ten classes.



```
train_data = train_data.astype('float32') / 255.
test_data = test_data.astype('float32') / 255.
num_classes = 10
```

Figure 5 Normalizing

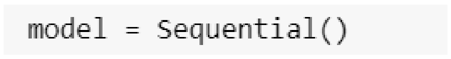
Since the output of the network is a 1x10 Matrix with values between zeros and ones, which are the membership rates in each class, we have to convert the training labels into a 1x10 Matrix too, in which the value of the cell related to the class is one and the rest of the matrix cells are assigned zero.



```
train_labels_cat = to_categorical(train_labels,num_classes)
test_labels_cat = to_categorical(test_labels,num_classes)
```

Figure 6 Labels

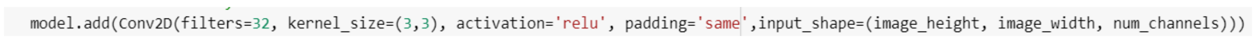
Now we create the model, the model is sequential, meaning that all the layers are placed one after another.



```
model = Sequential()
```

Figure 7 creating a model

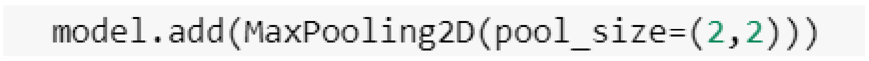
In this part, we add different layers. I choose the number of layers arbitrary (It's not optimized but for the purpose of this project, it works)



```
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same',input_shape=(image_height, image_width, num_channels)))
```

Figure 8 Convolution layers

We add the convolutional layers, which includes 32 filters of the 3x3 matrix with ReLU function as the activation function, we used the same padding so we retain the matrix size.

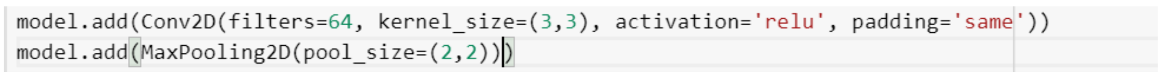


```
model.add(MaxPooling2D(pool_size=(2,2)))
```

Figure 9 Pooling layer

We use the pooling layer to reduce the calculation. Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer.

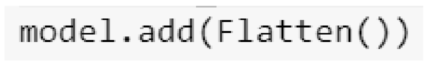
We apply more convolution layers and Pooling.



```
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

Figure 10 Conv2D and Pooling Layers

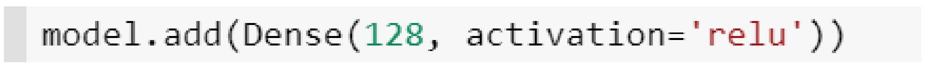
The input matrix of the neural network must be in the form of a vector, so we must convert the multidimensional matrix to a one-dimensional matrix.



```
model.add(Flatten())
```

Figure 11 Flatten Layer

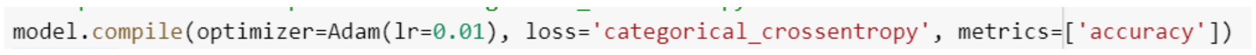
Now, we add the first layer of the neural network with 128 neurons and the ReLU function as the activation function.



```
model.add(Dense(128, activation='relu'))
```

Figure 12 Dense Layer

And the last layer has 10 neurons, which is the number of classes. The activation function of this layer is Softmax, which makes the outputs between zero and one so that the sum of the neurons becomes one, and the class is the number of the largest cell of the output vector.



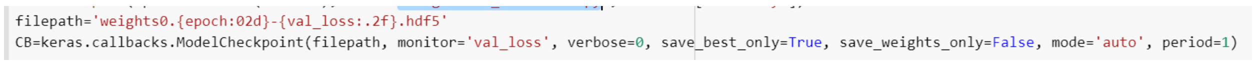
```
model.compile(optimizer=Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 13 Compiling

Now we compile the model and determine the learning rate and the optimizer (In this project we used Adam and SGD).

**Saving the Best Model**:

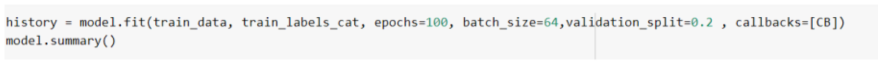
I Used call back to save the best weights.



```
filepath='weights0.{epoch:02d}-{val_loss:.2f}.hdf5'
CB=keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=False, mode='auto', period=1)
```

Figure 14 Callback

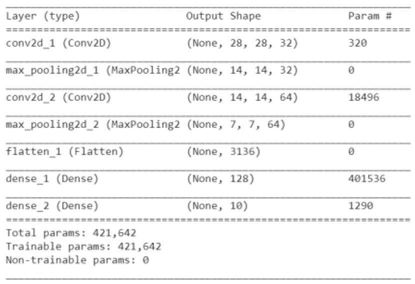
The filepath variable is used to define the name and method of storing weights (based on the val_loss) .We define a callback in such a way that reducing the Validation loss is important to us. It will save the models with the least Validation loss.
Now we train the model. I used 20% of training data for validation and we trained the model for 100 epochs.



```
history = model.fit(train_data, train_labels_cat, epochs=100, batch_size=64,validation_split=0.2 , callbacks=[CB])
model.summary()
```

Figure 15 Training

The summary shows the number of the model’s parameters.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536    
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
_________________________________________________________________
```

Figure 16 Summary

## 4. Optimizers:

We want to minimize the cost function by changing weights or biases. Therefore, we calculate the errors and change the weights and biases based on the error value. We use different optimizing methods such as SGD or ADAM on which I will elaborate more.

**SGD**:

To minimize the cost function we want to move in the opposite direction of error and as you can see in the below equation we calculate error and change the weights in respect of error.

$$\Delta \overline{W}_j = -\eta \nabla \overline{E} = \eta (d_j - y_j) f'(\overline{x}\overline{w}_j) \overline{x}^T$$

Figure 17 SGD

**Adam**:

Adam is similar to SGD but it has some extra parameters for example, in SGD, the learning rate is constant during learning and does not change. In Adam, it acts as if the learning rate was large at first to be able to see the entire error space. Then it reduces this rate according to the error to reach the minimum error.

Categorical cross-entropy is one of the loss functions that show how different the network output is from the desired output. In this function, this difference is shown by the fact that y^ represents the output of the network and y represents the desired output, which is either zero or one.

$$L(y,\hat{y}) =- \sum_{j=0}^{M} \sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

Figure 18 Loss Function

## 5. **Difference between ReLU and Sigmoid**:

Sigmoid is a function that converts input to a range of zero to one

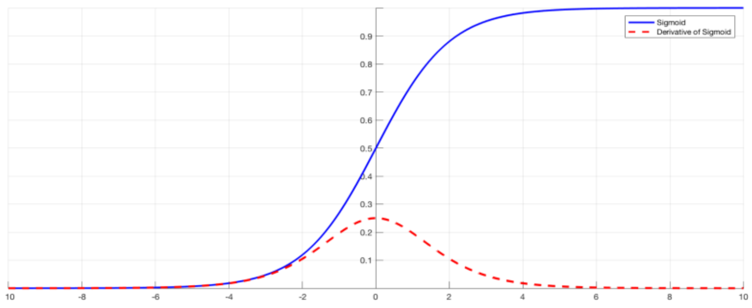


Figure 19 Sigmoid

To update the parameters, the backpropagation method requires a function derivative, which is multiplied by a chain law, and if these functions and derivatives are too small, they approach zero and prevent the propagation of errors, as the result, the parameters cannot be updated. We call this problem vanishing gradient, we can avoid this problem by using the ReLU function. In practice, networks with ReLU tend to show better convergence performance than sigmoid. But there are some disadvantages to ReLU too. For example, there is no mechanism to constrain the output of the neuron or the Dying ReLU problem (This can be handled, to some extent, by using Leaky-Relu instead).

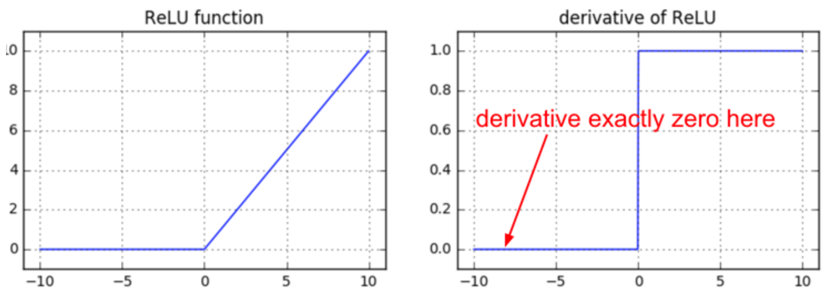


Figure 20 ReLU

## 6. **Learning Rate**:

Choosing the right learning rate can be difficult. If the learning rate is too small, it can cause the convergence to be very slow, and on the other hand, if it is too large, it can interfere with the convergence and cause the loss function to fluctuate or even diverge around the minimum value. Also, a very small learning rate can cause underfitting or only finding the local minimum. For example, In the case that the learning rate is small and we used the sigmoid as the activation function, we might have the same small derivative problem and as a result of the small learning rate the network cannot be fitted to the data at all (vanishing gradient problem) and the accuracy is very low. Also, if we use Adam and a big learning rate the loss function might fluctuate and the network might not be able to find the global minimum.

## 7. **Results**:

I have trained different models with different properties and the full results (Including plots of loss and accuracy on validation data) are available on my GitHub.

Here are two examples, as suspected, the sigmoid function not only led to the vanishing gradient but also the model suffers from a high loss. Whereas, the ReLU function worked fine.

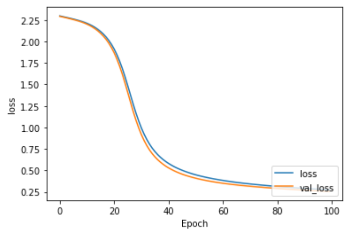


Figure 21 CNN/SGD/Relu_lr_0.0001

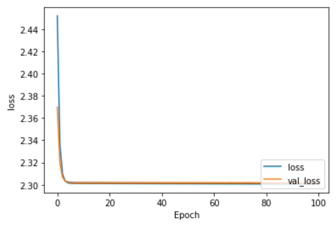


Figure 22 CNN/SGD/Sig_lr_0.0001

## 8. Conclusion:

In this project, first I implemented two frequent methods of pattern recognition (CNN and MLP) with different optimizers, learning rates and activation functions second I compared the results with the experiment of training these networks on the MNIST dataset. The experiments led to the conclusion that considering the good performance of MLP on this dataset CNN acquires better accuracy. Also, as we can see from the error and accuracy plots when we use SGD, it takes longer for the network to acquire high accuracy, However, Adam is better and faster. In the matter of activation function, ReLU mostly does a better job than sigmoid and in some cases, ReLU did not face some problems like vanishing gradient. In addition, a high value of learning rate can cause the network to not converge and a low learning rate can cause to only find a local minimum. In conclusion, choosing any of these properties is a delicate job and needs close attention and also, there are some more properties to be considered in future such as how many layers and neurons are enough?

## 9. References:

[1]. Y. LeCun, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist.

[2]. D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pages 3642–3649. IEEE, 2012

[3]. K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4):193–202, 1980.

[4]. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998.

[5]. S. Behnke. Hierarchical Neural Networks for Image Interpretation, volume 2766 of Lecture Notes in Computer Science. Springer, 2003.

[6]. P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In Seventh International Conference on Document Analysis and Recognition, pages 958–963, 2003.

[7]. D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In International Joint Conference on Artificial Intelligence, pages 1237–1242, 2011.

[8]. Dan Cireșan, Ueli Meier, Juergen Schmidhuber,Multi-column Deep Neural Networks for Image Classification arXiv:1202.2745