Deep Neural Network Application in Digit Recognition

Suohuijia Wang

17200170

Abstract

This paper introduces a famous neural network called CNN and applies it to digit recognition. MNIST database is used in this model. It is divided into three parts, training set, validation set and test set. SGD is used as the optimizer. In the meantime, call_back function records the loss and accuracy of three sets after every 100 batches. Three hyperparameters, learning rate, l2 and top dense hidden units are tuned to get the better accuracy of our model. We analyze the performance based on these three hyperparameters by plotting accuracy and loss figures. Then we present a comparison between CNN, MLP and SVM on digit recognition. In MLP, we tune the same hyperparameters, learning rate, l2, and top dense hidden units. In SVM, we use the existing algorithm SVC in sklearn, and use grid search to find the 'best' hyperparameters. Based on our experiment, we get the conclusion that in the similar situation, CNN does perform better than other two algorithms, with the highest accuracy, 98.63%.

1 Introduction

Digit recognition is very popular in the business world since it can make our lives more efficient. The main application areas of handwritten digital recognition are as follows: (1) Applied to large-scale data statistics. For instance, censuses, industry annual inspection, financial statements entry which need to enter a large amount of data. (2) Applied in finance and taxation. With the rapid economic development in our country, there will be a great deal of daily financial, taxation, checks and other needs to be dealt with. (3) Automatic sorting mail system.

Currently, neural network is the mainstream approach to solve digit recognition. The basic idea is to take a large amount of handwritten digits, which we call training set, and then the neural network uses the training set to learn rules of recognizing handwritten digits automatically. In addition, if we increase the size of the training set, our network can learn more details so as to improve the accuracy. However, there are different kinds of neural networks now, with the development of techniques, neural networks are gradually becoming more advanced.

In this paper, we mainly focus on Convolutional Neural Network, also named CNN which is widely used in digit recognition in the business world. In addition, we compare CNN with other two algorithms, MLP and SVM to observe their performance. As for the hyperparameters, we choose three hyperparameters to tune, learning rate, l2, and top dense hidden units. We use the

MNIST database and divide it into three parts: training data, validation data, and test data. We use training data to train our models, and then, use validation data to assess our hyperparameters according to analyze the loss and accuracy of every 100 batches. After the two steps, we have a set of 'good' hyperparameters, so we can use test data to test our models. Section 2 provides the background for those non-technical readers. Section 3 introduces the experiments, formulates the model for CNN, and presents results. Section 4 discusses the comparison between these three models. Conclusions are finally drawn in section 5.

2 Background

2.1 Neural networks

Neural network, also named Artificial neural network, is an information processing paradigm. It is inspired by the biological nervous systems, such as the brain and process information. Neural network is composed of a large amount of highly interconnected processing elements, which are named as neurons, working in unison to solve specific problems. ANN can be used in pattern recognition or data classification.

In a neural network, there are some basic components, perceptrons, sigmoid neurons, input neurons, output layer, output neurons, hidden layer and also we should have weights and biases to help our network learn better.

2.2 Convolutional Neural Network

2.2.1 Development of CNN

The originator of the CNN (Convolutional Neural Network), LeCun[1], introduced the LeNet-5 model in 1998, marking the official birth of neural networks. In 2006, Geoff Hinton[2] pointed out that the neural network with multi-hidden layers had the superior feature-learning ability and its training complexity could be reduced by layer-by-layer initialization. In 2012, Alexnet[3] got two firsts in the ImageNet contest, and the accuracy rate exceeded the second nearly 10%. Taigman et al.[4] published an article in 2014 CVPR. At the same time, DeepFace has become a landmark research result of CNN in the field of face recognition. In 2015, LeCun et al.[5] systematically summarizing the development of deep learning in past and present lives.

Today, with the rapid development of GPU and big data technology, CNN has moved from academic to engineering area, and has been widely applied in practical engineering fields such as image classification, object detection, image segmentation, image tagging and image generation.

2.2.2 CNN

Convolutional Neural Network (CNN) is a feed-forward neural network, and its artificial neurons can respond to a part of the surrounding cells within the coverage area, so it performs well in the large-scale image processing.

A convolutional neural network consists of one or more convolutional layers, rectified linear units layer and the top fully connected layer (corresponding to a classical neural network). And it also contains associated weights, pooling layers, and loss layer. This structure allows the convolutional neural network to use the two-dimensional structure of the input data.

2.2.3 Backpropagation

Backpropagation, short for "backward propagation of errors," is an algorithm which uses gradient descent to conduct the supervised learning of artificial neural networks. It is considered as a very efficient approach to train ANN. Given an artificial neural network and an error function, this approach can calculate the gradient of the error function with respect to the neural network's weights and bias, then in the process of backpropagation, these hyperparameters can be adjusted according to the error. The whole process is iterated until the error converges. Backpropagation is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks.

2.2.4 SGD

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function which is written as a sum of differentiable functions. SGD is popular in optimisation problems since its learning speed is fast and to some extent, it can avoid getting stuck into a local optimum. In other words, SGD tries to find minima or maxima by iteration.

2.2.5 L2 Regularization

L2 regularization is also named as weight decay. The main idea is to add an extra term to the cost function, which is called the regularization term. Intuitively, the effect of regularization is to make the network prefer to learn small weights, all other things being equal. Large weights will only be allowed if they considerably improve the first part of the cost function. The relative importance of the two elements of the compromise depends on the value of λ: when λ is small we prefer to minimize the original cost function, but when λ is large, we prefer small weights.

3 Experiments

In our experiment, we test our model on the MNIST dataset. We focus on three hyperparameters, learning rate, l2 and top dense hidden units. We use SGD as our optimizer. In the meantime, call_back function is used to record the test loss and test accuracy after every 100 batches. Then we can plot figures based on different sets of hyperparameters and analyze their performance.

In addition, we compare our model with other two algorithms, MLP and SVM, to observe their performance on digit recognition.

3.1 Problem description

3.1.1 Problem

In this paper, we apply CNN in digit recognition and then compare it with MLP and SVM to analyze their performance by tuning their hyperparameters. Our goal is to find the optimal hyperparameters which can get the highest validation and test accuracy in our model.

3.1.2 Data source

Our database is MNIST database. It is a large database of handwritten digits that is commonly used for training various image processing systems. We divide it into three parts, training data is used to train our model to get weights and biases, validation data is used to test whether our

hyperparameters are good enough, finally, test data is used to test our model and classify new dataset.

Figure 1 shows the three sets used in CNN, we have 50,000 data in the training set, and 10,000 data in the validation set as well as the test set. (28, 28) is the input image dimension, 10 means we have 10 digits to classify, which are our targets.

x_train	(50000, 28, 28, 1)	y_train	(50000, 10)
x_val	(10000, 28, 28, 1)	y_val	(10000, 10)
x_test	(10000, 28, 28, 1)	y_test	(10000, 10)

Figure 1. Three sets used in CNN

3.1.3 Planning Objective

Every time we train our data based on different hyperparameters, we record the test accuracy and loss. Test accuracy means the possibility that we can predict correctly when we use our model to predict a digit image. Our goal is to find the optimal set of hyperparameters which can get the highest accuracy in the validation and test data set.

3.2 Experimental settings

3.2.1 Model description

We use keras to create our model. Keras is a high-level neural networks API, which supports convolutional neural networks. Following is the code of our model definition:

```
conv_1_filters = 32
conv_2_filters = 64
l2_regularizer = l2(l=l2_lambda)
model = Sequential()
model.add(Conv2D(filters=conv_1_filters,
                 kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(filters=conv_2_filters,
                 kernel_size=(3, 3),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=top_dense_hidden_units,
                kernel_regularizer=l2_regularizer,
                activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=categorical_crossentropy,
              optimizer=SGD(lr=learning_rate),
              metrics=['accuracy'])
```

Our CNN has five layers, first convolutional layer with 32 filters, second convolutional layer with 64 filters, next is the max polling layer with size 2*2, then a flatten layer, and last is a top dense layer. ‘relu’ is the rectified linear unit, which is used to output the hidden neurons, see Figure 2. Kernel size is 3*3.

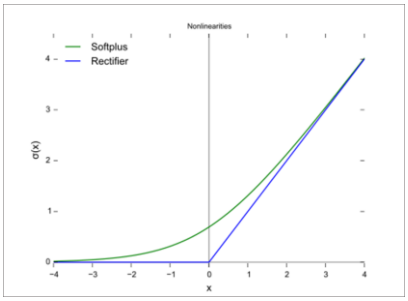


Figure 2. Relu activation function[1]

We use categorical cross-entropy as our loss function, l2 to improve our model which can avoid being overfitting, SGD as our optimizer. As for SGD, we define the batch size as 32, the epoch as 2, and for every 100 batches, we record the accuracy and loss of our three sets. After the definition, we can use MNIST dataset to train our model.

3.2.2 Hyperparameters

Tuning hyperparameters is a huge and important work in CNN. In this paper, we choose three hyperparameters, learning rate, l2 and top dense hidden units. For every hyperparameter, we create a list with several proper values and then we try every permutation in our model to observe their performance, see Figure 3.

Learning rate (lr)	[0.1, 0.01, 0.005, 0.001]
L2 lambda (l2)	[1e-4, 1e-5, 1e-6]
Dense hidden units (Dhu)	[128, 256, 512]

Figure 3. Possible values for each hyperparameter

3.3 Experiment results

Our objective considered in this paper is to maximize the accuracy and minimize the loss of our test data. After running our program, we get lots of results. For the sake of simplicity, we use control variates method to do the analysis. Therefore, we plot figures to compare the performance of accuracy and loss between training set, validation set and test set based on different hyperparameters and find the optimal hyperparameters with the highest accuracy.

3.3.1 Analysis of learning rate with fixed l2 and dense hidden units

Learning rate is a very important hyperparameter in optimization models. Therefore, we first observe the performance of the three datasets based on different learning rates. We set l2 as 0.00001 and dense hidden units as 256, with four different learning rates, see Figure 4.

Learning rate	0.001	0.005	0.01	0.1
L2 lambda	0.00001	0.00001	0.00001	0.00001
Dense hidden units	256	256	256	256

Figure 4. Different lrs with fixed l2 and Dhu

[1] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Figure 5 shows the performance of three sets with different learning rates. We can see that the difference in their performance is obvious. For each set, the yellow line (lr=0.1) has the best performance, with very fast learning speed, it jumps to 0.95 accuracy at the very beginning, and then converges to the highest accuracy. However, the blue line (lr=0.001) rises significantly after 200 batches. When we train our model, we can see the accuracy fluctuates in the whole process, but the yellow line always lies at the top of other lines. Then we use the validation set to find the best hyperparameters, still learning rate with 0.1 performs the best. Finally, we use the test set to do prediction, we can see the yellow line (lr=0.1) reaches around 0.98 accuracy, and the blue line (lr=0.001) reaches around 0.91 accuracy. The difference is 0.07.

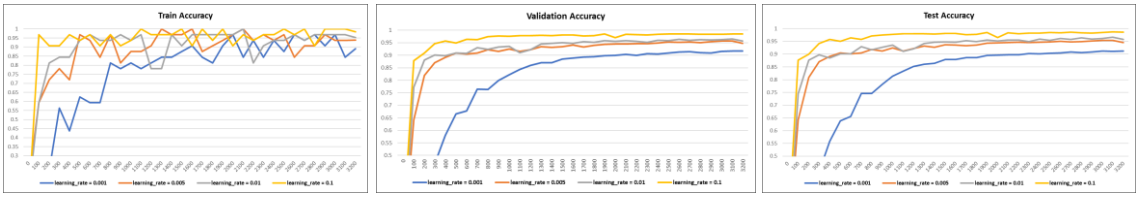


Figure 5. Comparison of accuracy between three sets based on different lrs

Figure 6 demonstrates the performance of loss in three sets. We can see the blue line (lr=0.001) reduces very slow, and the performance of other three lines is similar, obviously, yellow line (lr=0.1) has the best performance, reaches the smallest loss, around 0.05.

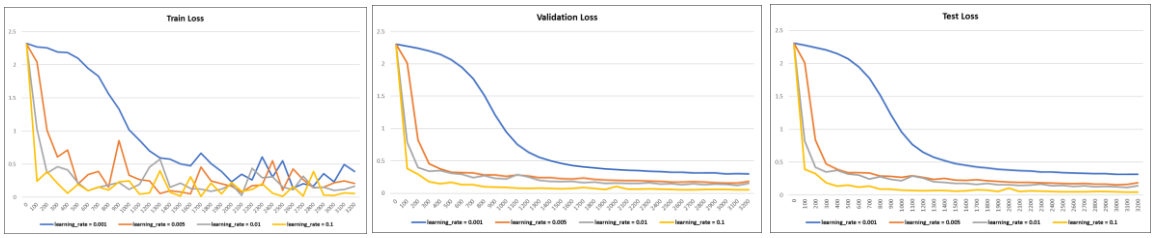


Figure 6. Comparison of loss between three sets based on different lrs

3.3.2 Analysis of dense hidden units with fixed learning rate and l2

From the previous analysis, we choose a proper learning rate, 0.01 as our fixed learning rate, we don't use the best learning rate 0.1 since its convergence is so fast that we may not observe the performance clearly. In this case, we set l2 as 0.00001, and observe the performance by tuning different dense hidden units. See Figure 7.

Learning rate	0.01	0.01	0.01
L2 lambda	0.00001	0.00001	0.00001
Dense hidden units	128	256	512

Figure 7. Different Dhus with fixed lr and l2

Figure 8 shows the performance of different dense hidden units. We can see that their performance is similar. In the training set, the accuracy can reach to around 0.95. However, our training model fluctuates randomly when it reaches the minima, we can see there are some values which are close to 1. In the validation set and test, the accuracy is about 0.96. Overall, the orange line (Dhu=256) performs a little better in the validation and test set.

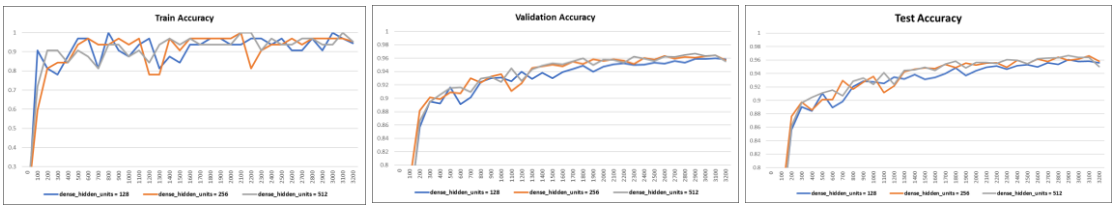


Figure 8. Comparison of accuracy between three sets based on different Dhus

Figure 9 shows the performance of loss in three sets. Still, their trends are similar, and the whole performance is worse than the previous experiment (analysis of learning rate), the optimal is 0.05 while in this experiment, 0.12.

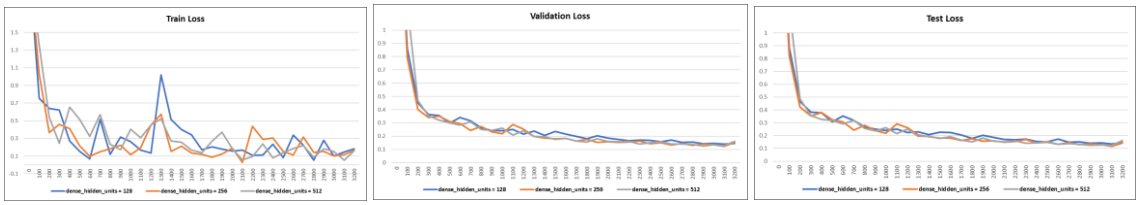


Figure 9. Comparison of loss between three sets based on different Dhus

3.3.3 Analysis of l2 with fixed learning rate and dense hidden units

After the analysis of learning rate and dense hidden units, now we analyze the third hyperparameter, l2. As for the other hyperparameters, we set 0.01 and 256 respectively, see Figure 10.

Learning rate	0.01	0.01	0.01
L2 lambda	0.000001	0.00001	0.0001
Dense hidden units	256	256	256

Figure 10. Different l2 with fixed lr and Dhus

Figure 11 and Figure 12 illustrate the performance of accuracy and loss based on different l2, we can see that their performance is similar as what we did before (analysis of dense hidden units), so we omit the analysis. Overall, the orange line (l2=0.00001) performs a little better, 0.96 and 0.12.

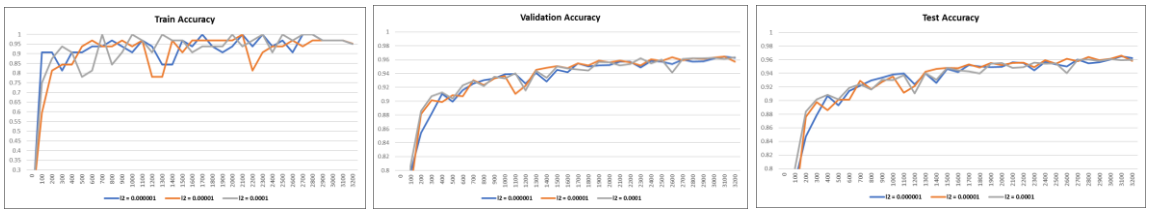


Figure 11. Comparison of accuracy between three sets based on different l2s

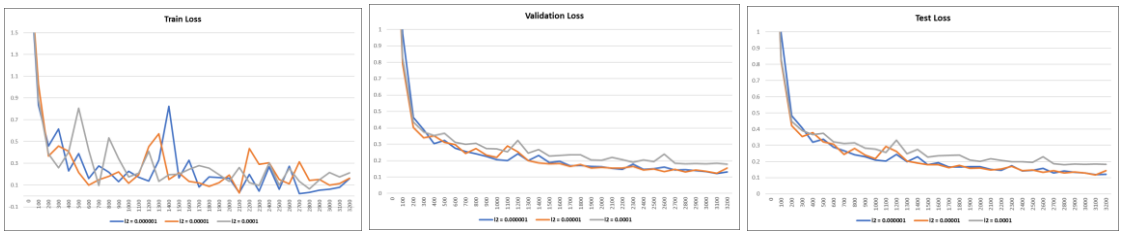


Figure 12. Comparison of loss between three sets based on different l2s

4 Discussion

4.1 Comparison of CNN and MLP

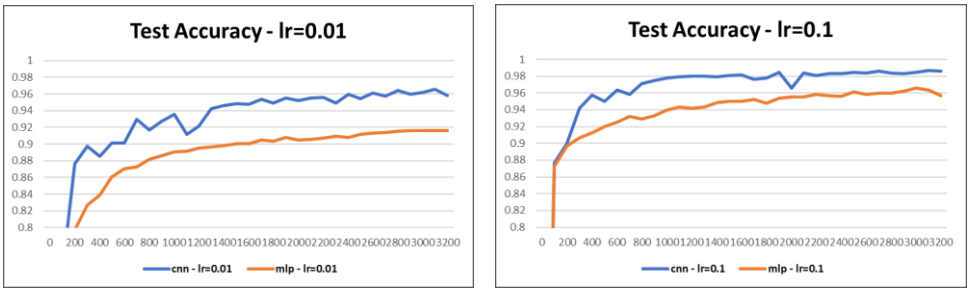
From our experiment results, we discover that learning rate plays a very important role in the performance of accuracy and loss. Therefore, we compare the performance of CNN with MLP[2] on the test set with different learning rates. We set l2 and dense hidden units as 0.00001 and 256 respectively, and then we choose two learning rates, 0.1 and 0.01 to compare the performance of these two algorithms. See Figure 13.

learning rate = 0.01 / 0.1		
	CNN	MLP
l2 lambda	0.00001	0.00001
Dense hidden units	256	256

Figure 13. Fixed l2 and Dhu with lr 0.01 and 0.1 on CNN and MLP

Figure 14 shows the performance of test accuracy, we can see that these two models' performance can be improved when learning rate rises from 0.01 to 0.1, from around 0.96 to 0.982 and 0.92 to 0.96 respectively. And the difference of the highest accuracy is shrunk. However, for both learning rates, CNN performs better than MLP.

When learning rate is 0.01, their test accuracy starts to rise significantly at around 200, then CNN (blue line) increases more markedly and above MLP (orange line) in the whole process. When learning rate is 0.1, the optimal accuracy of CNN is around 0.982, however, MLP is only about 0.958. The difference is 0.024.



[2] MLP has one hidden layer

Figure 14. Test Accuracy on CNN and MLP based on different lrs

The performance of test loss demonstrates the analogous result, see Figure 15, CNN reduces faster than MLP in both figures. In addition, when learning rate is 0.1, test loss of both algorithms reduces faster than 0.01, however, the difference between them is smaller. As for lr=0.01, the difference is 0.16, for lr=0.1, 0.09.

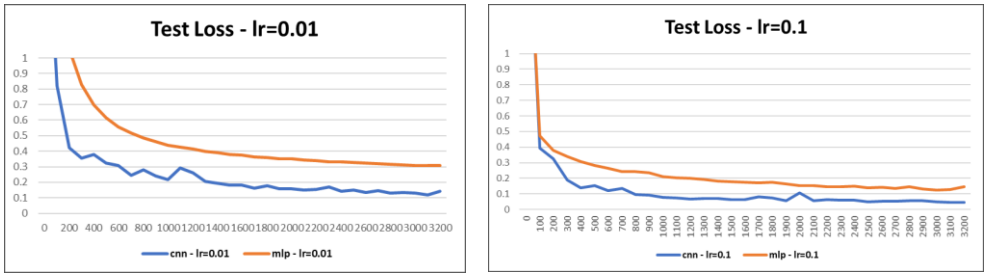


Figure 15. Test loss on CNN and MLP based on different lrs

4.2 Comparison of CNN, MLP and SVM

After our analysis in section 4.1, we get the highest accuracy of CNN and MLP. Then we compare them with SVM. As for SVM, we use SVC which is in sklearn.svm to define our model, and use GridSearch to tune hyperparameters automatically. Figure 16 shows the three algorithms' highest accuracy. According to these results, we can see that CNN's performance is the best, with highest accuracy 0.9863.

	CNN	SVM	MLP
learning rate	0.1	/	0.1
l2_lambda	0.00001	/	0.00001
Dense hidden units	256	/	256
Accuracy	0.9863	0.9769	0.9571

Figure 16. Highest accuracy in three models

5 Conclusions

5.1 Conclusion

In conclusion, we achieve CNN on digit recognition, and compare the performance with MLP and SVM. In CNN, we have three hyperparameters, learning rate, l2, and dense hidden units. Based on our analysis, the optimal learning rate is 0.1, as the learning rate gets smaller, the accuracy gets lower, and the speed of convergence gets slower. The performance of l2 and dense hidden units is similar. So learning rate plays a very important role in our model. As a result, the optimal values of these three hyperparameters in our experiment are 0.1, 0.00001, 256 respectively, with the highest accuracy 0.9863. In SVM, we use GridSearch to tune our hyperparameters automatically, and the best accuracy is 0.9769. In MLP, we tune the same hyperparameters, and the best values are also 0.1, 0.00001, and 256, with the highest accuracy 0.9571.

As a result, CNN has the best performance among these three algorithms based on the similar situation.

5.2 Future work

CNN, MLP, and SVM are all popular algorithms on digit recognition, in this paper, we confirm that in the similar situation, CNN performs the best, and SVM performs better than MLP. However, there are still some improvements for us to do on digit recognition.

5.2.1 Combining these classifiers

Every algorithm has its own strength, if we can combine them and exploit their respective advantages, maybe we can get better results.

Actually, there are some people who do further research on this field. Niu[6] and his partners achieved a hybrid model of integrating the synergy of CNN and SVM. In their paper, CNN works as a trainable feature extractor and SVM performs as a recognizer. In the end, they get a pretty good result: a recognition rate of 99.81% without rejection.

5.2.2 Tuning hyperparameters automatically

In this paper, we tune the three hyperparameters manually, sometimes it is not a perfect method. We use GridSearch in our SVM to tune hyperparameters automatically, in fact, based on the property of GridSearch, we spend a lot of time running this algorithm. As a result, we can consider a better method, Bayesian Optimization, as our tuning method.

6 References

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

[2] Hinton, G.E. and Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. *science*, *313*(5786), pp.504-507.

[3] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[4] Taigman, Y., Yang, M., Ranzato, M.A. and Wolf, L., 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1701-1708).

[5] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, *521*(7553), p.436.

[6] Niu, X.X. and Suen, C.Y., 2012. A novel hybrid CNN–SVM classifier for recognizing handwritten digits. *Pattern Recognition*, *45*(4), pp.1318-1325.

Code references:

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

https://github.com/keras-team/keras/blob/master/examples/mnist_mlp.py

http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py