****

**UNIVERSITY OF VICTORIA**

**Department of Electrical and Computer Engineering**

**ELEC 534 Digital Signal Processing III**

**Project Report**

Title: Supervised Multiclass Image Classification Using Convolutional Neural Network and Multilayer Perceptron

To: Dr. Wu-Sheng Lu

Name:

Amir Farzad V00934306

**Abstract**

Nowadays, using Machine Learning and Deep Learning algorithms are on the rise. These models can be used in so many different ways. One of the best use cases is image classification with convolutional neural networks. In this project firstly Multilayer perceptron neural network (MLP) and CNN networks are described as they are bounded together. Then these models for image classification are developed and trained with training data. Finally, the CNN and MLP networks are tested with testing data. In this project MNIST image dataset is used. Results have shown that CNN network is a better choice for image classification.

1. **Introduction**
   1. **Multilayer perceptron (MLP)**

The perceptron is a basic linear classifier for binary classification. For example, it predicts whether input is true or false. It classifies input by separate it into two or more categories with a simple equation (1). where w is weight vector, x is input feature vector and b is bias.



A multilayer perceptron (MLP) [1] is a neural network which contains more than one perceptron. Typically, MLP has input, output and hidden layers. The input layer is for getting the input data, output layer is for prediction with respect to the input data, and between them we can have multiple hidden layers. The networks which have been deployed in this project are supervised. MLPs used mostly for supervised classification tasks which means the output labels of dataset are available [2].

Feedforward networks like MLPs have two main stages. Within the forward pass, the data goes from the input layer through the hidden layers to the output layer, and the output layer labels is measured with respect to actual labels with an activation function. In the backward pass, exploitation backpropagation and also the chain rule of calculus, partial derivatives of the error operate with respect to the weights and biases area unit back-propagated through the MLP. That act of differentiation provides us a gradient and the parameters is also adjusted as they move the MLP one step nearer to the error minimum. This could be finished any gradient-based optimization formula like random gradient descent or Adam. The network will train in multiple epochs till the error will go no lower which is called convergence. A simple MLP network is shown in Figure 1.

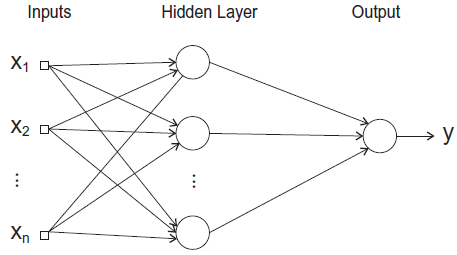


Figure1. A MLP network consists of one input layer, one hidden layer and one output layer.

* 1. **Convolutional Neural Network (CNN)**

Convolutional neural networks are deep artificial neural networks that are used mostly for classifying images [3]. CNN networks may determine faces, street signs, medical images and OCR [4].

Convolution is that the integral mensuration how much two functions overlap mutually passes over the opposite. Consider a convolution as the simplest way of blending two functions by multiplying them and rather than two-dimensional images, in convolutional networks images treated as 4 dimensions. It would be several filters over one image, each picking up a distinct data. After a convolutional layer, input is going through a nonlinear activation function like tangent hyperbolic which is able to squash input values between -1 and 1.

Feature detector or kernel is a window and it would be placed over the input image starting from position (0,0) in the image (upper left) then count the amount of cells during which the kernel matches the input image [5]. The amount of matching cells is then inserted within the position of (0,0) of the kernel. Then moving the kernel one cell to the right and do the same. If kernel moved one cell at time, that might be known as a stride of one element. After finishing the first row it will move to next row of image and do the same. A simple convolution of an input image with kernel size (3,3) is shown in Figure 2.

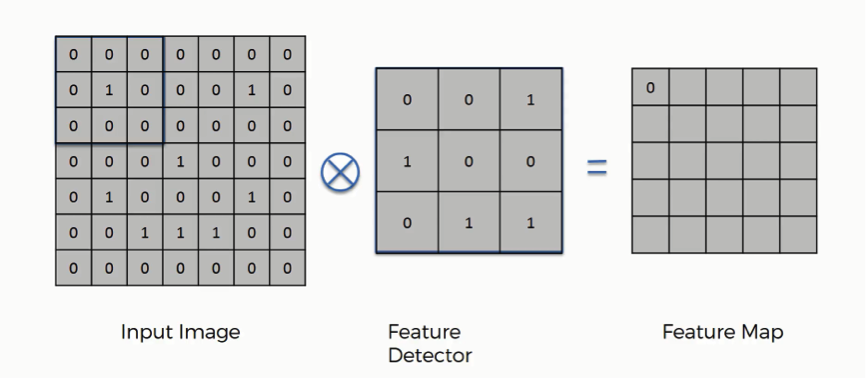


Figure2. Convolution of an input image with kernel size (3,3).

The next layer in convolutional network is max pooling [6]. The activation maps are fed into a max pooling layer, and the process would be like convolutions. In this step max pooling merely takes the biggest value from one kernel of an image and omits the rest of the values. Typically, for CNN we have one or more convolutional and max pooling layers. Then It fed into a MLP network for classification. A simple Max pooling with (2,2) filter (pool size) and stride of 2 is shown in Figure 3.

The feature map of Convolution network is shown in the equation (2). Where is the pixel index in the feature map, is the input patch centered at location , and k is used to index the channels of the feature map.



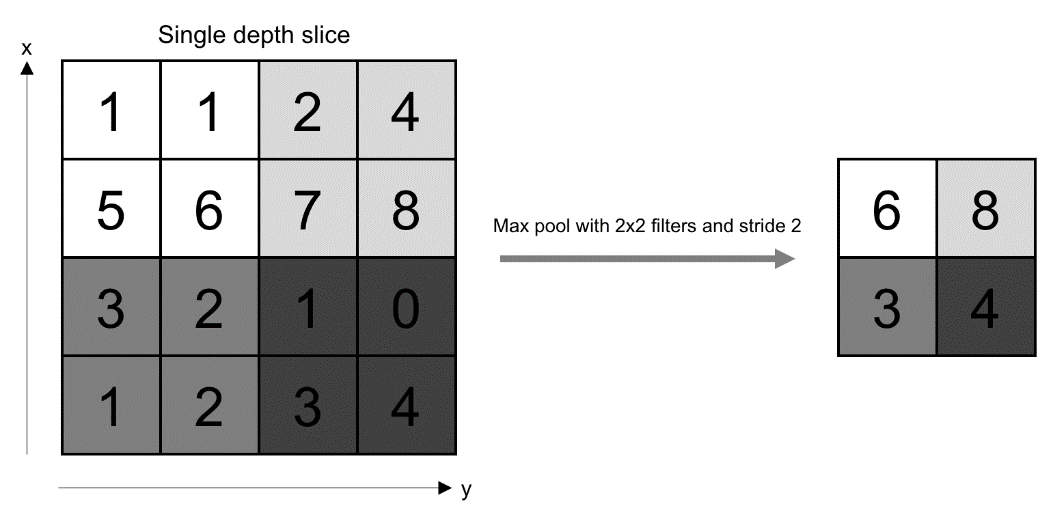


Figure3. Max pooling with (2,2) filter and stride of 2.

1. **Discussions**
   1. **MLP Code**

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout

import keras\_metrics

import matplotlib.pyplot as plt

import time

# split data

(train\_x, train\_y), (test\_x, test\_y) = mnist.load\_data()

#plot a simple digit number

img\_idx = 1000

print(*"Example of a digit: "*, train\_y[img\_idx]) # The label is 0

#plt.figure(1)

#plt.imshow(train\_x[img\_idx], cmap='Greys')

#plt.title(' Example of a digit ')

#plt.show()

#reshape data to 2D [X,28\*28]

train\_x = train\_x.reshape(train\_x.shape[0], 784)

test\_x = test\_x.reshape(test\_x.shape[0], 784)

input\_shape = (784,)

train\_x = train\_x.astype(*'float32'*)

test\_x = test\_x.astype(*'float32'*)

#Normalize data

train\_x = train\_x/255

test\_x = test\_x/255

print(*'train\_x shape:'*, train\_x.shape)

nclass = 10

# change class to binary class matrix

train\_y = keras.utils.to\_categorical(train\_y, nclass)

test\_y = keras.utils.to\_categorical(test\_y, nclass)

#timing

start = time.time()

#MLP Model

model = Sequential()

model.add(Dense(128, activation=*'relu'*, input\_shape=input\_shape))

model.add(Dropout(0.35))

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

model.add(Dropout(0.35))

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

model.add(Dropout(0.35))

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

model.add(Dropout(0.35))

model.add(Dense(nclass, activation=*'softmax'*))

model.summary()

model.compile(loss=*'categorical\_crossentropy'*, optimizer=*'Adam'*,

metrics=[*'accuracy'*, keras\_metrics.precision(), keras\_metrics.recall()])

hist=model.fit(train\_x, train\_y, batch\_size=64, epochs=20,

verbose=2, validation\_split=0.15)

accuray = model.evaluate(test\_x, test\_y, verbose=0)

print(*"Test Accuracy: "*, (accuray[1]\*100))

print(*'Test loss:'*, accuray[0])

end = time.time()

print(*"Time elapsed (seconds): "*, end - start)

#Plot accuracy for train and validation

plt.figure(2)

plt.plot(hist.history[*'acc'*])

plt.plot(hist.history[*'val\_acc'*], *'--'*)

plt.title(*'Accuracy of the model'*)

plt.ylabel(*'Accuracy'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Train'*, *'Validation'*], loc=*'lower right'*)

plt.show()

#Plot loss for train and validation

plt.figure(3)

plt.plot(hist.history[*'loss'*])

plt.plot(hist.history[*'val\_loss'*],*'--'*)

plt.title(*'Loss of the model'*)

plt.ylabel(*'loss'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Train'*, *'Validation'*], loc=*'upper right'*)

plt.show()

# plot Precision & Recall

plt.figure(4)

plt.plot(hist.history[*'precision'*])

plt.plot(hist.history[*'recall'*], *'--'*)

plt.title(*' Precision & Recall '*)

plt.ylabel(*'%'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Precision'*, *'Recall'*], loc=*'lower right'*)

plt.show()

img\_idx = 1005

plt.figure(5)

plt.imshow(test\_x[img\_idx].reshape(28, 28), cmap=*'Greys'*)

plt.title(*' Example of predict a digit '*)

plt.show()

pred = model.predict(test\_x[img\_idx].reshape(1, 784))

print(*"predicted output: "*,pred.argmax())

For MLP network, First, the MNIST [[1]](#footnote-1) dataset is loaded and split into test and train data. The MNIST dataset of handwritten digits has a training set of 60,000 examples (with 9000 examples for validation), and a test set of 10,000 examples. The image sizes are 28\*28 pixels. The network predicts 10 digits (0 to 9) or equivalently10 classes*.*

Then simply one of the digits of our dataset has been shown which is zero. Next train and test data are reshaped into 2D matrix which second dimension is 28\*28=784 as the MNIST dataset images has 28 rows and 28 columns. Then the dataset is normalized by dividing each element of images by 255. As it can be seen the train data now has the shape of (60000, 784) and test data has the shape of (10000, 784). There would be 10 classes for classification task as dataset is digits between 0 and 9 so the classes are changed to binary class matrix. Then MLP model is deployed. The MLP that here is developed has one input layer and several hidden layers with relu activation function and one output layer with softmax activation function (as this is a multiclass classification task). Each layer consists of more than one neuron and in this project it has 128 neurons. Between each layer it would be dropout layer to prevent overfitting. Then categorical cross entropy is applied as loss function and for optimizer the Adam optimizer is applied. Next training and validation data with mini-batch method is trained with different epochs and the accuracy, precision and recall of the trained model are reported.

Precision measures the exactness of a classifier and a higher precision means less false positives, whilst a lower precision means more false positives. A false positive is where you receive a positive, when you should receive a negative result (you falsely get positive result). Recall measures the completeness (sensitivity) of a classifier. Higher recall means less false negatives, whilst lower recall means more false negatives. A false negative is where you receive a negative, when you should receive a positive result (you falsely get negative result).

Then the model is tested on test data and the accuracy of the model is reported. The train, validation and test data are 51000, 9000 and 10000 samples, respectively. Next the accuracy and loss for train and validation is plotted, and precision and recall is plotted too. Finally, an example is predicted to see how much it works on real data. An example of a digit from MNIST dataset have shown in Figure 4.

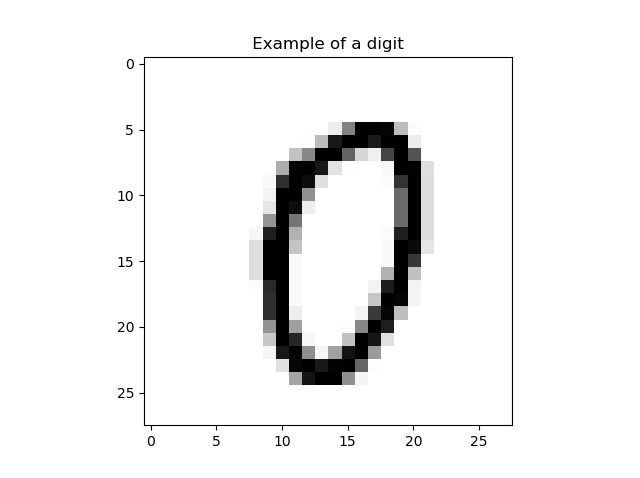


Figure4. An example of a digit from MNIST dataset which is 0.

* 1. **CNN Code**

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

import keras\_metrics

import matplotlib.pyplot as plt

import time

# split data

(train\_x, train\_y), (test\_x, test\_y) = mnist.load\_data()

#plot a simple digit number

img\_idx = 1000

print(*"Example of a digit: "*, train\_y[img\_idx])

#plt.figure(1)

#plt.imshow(train\_x[img\_idx], cmap='Greys')

#plt.title(' Example of a digit ')

#plt.show()

#reshape the data to 4D [X,28,28,1]

train\_x = train\_x.reshape(train\_x.shape[0], 28, 28, 1)

test\_x = test\_x.reshape(test\_x.shape[0], 28, 28, 1)

input\_shape = (28, 28, 1)

train\_x = train\_x.astype(*'float32'*)

test\_x = test\_x.astype(*'float32'*)

#Normalize data

train\_x = train\_x/255

test\_x = test\_x/255

print(*'train\_x shape:'*, train\_x.shape)

nclass = 10

# change class to binary class matrix

train\_y = keras.utils.to\_categorical(train\_y, nclass)

test\_y = keras.utils.to\_categorical(test\_y, nclass)

#timing

start = time.time()

#CNN Model

model = Sequential()

model.add(Conv2D(20, kernel\_size=(3, 3), activation=*'relu'*, input\_shape=input\_shape))

model.add(Dropout(0.35))

model.add(Conv2D(40, (2, 2), activation=*'relu'*))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.35))

model.add(Flatten())

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

model.add(Dropout(0.35))

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

model.add(Dropout(0.35))

model.add(Dense(nclass, activation=*'softmax'*))

model.summary()

model.compile(loss=*'categorical\_crossentropy'*, optimizer=*'Adam'*,

metrics=[*'accuracy'*, keras\_metrics.precision(), keras\_metrics.recall()])

hist = model.fit(train\_x, train\_y,batch\_size=128,epochs=10,

verbose=2, validation\_split=0.15)

accuray = model.evaluate(test\_x, test\_y, verbose=0)

print(*"Test Accuracy: "*, (accuray[1]\*100))

print(*'Test loss:'*, accuray[0])

end = time.time()

print(*"Time elapsed (seconds): "*, end - start)

#Plot accuracy for train and validation

plt.figure(2)

plt.plot(hist.history[*'acc'*])

plt.plot(hist.history[*'val\_acc'*], *'--'*)

plt.title(*'Accuracy of the model'*)

plt.ylabel(*'Accuracy'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Train'*, *'Validation'*], loc=*'lower right'*)

plt.show()

#Plot loss for train and validation

plt.figure(3)

plt.plot(hist.history[*'loss'*])

plt.plot(hist.history[*'val\_loss'*],*'--'*)

plt.title(*'Loss of the model'*)

plt.ylabel(*'loss'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Train'*, *'Validation'*], loc=*'upper right'*)

plt.show()

# plot Precision & Recall

plt.figure(4)

plt.plot(hist.history[*'precision'*])

plt.plot(hist.history[*'recall'*], *'--'*)

plt.title(*' Precision & Recall '*)

plt.ylabel(*'%'*)

plt.xlabel(*'epoch'*)

plt.legend([*'Precision'*, *'Recall'*], loc=*'lower right'*)

plt.show()

img\_idx = 1005

plt.figure(5)

plt.imshow(test\_x[img\_idx].reshape(28, 28), cmap=*'Greys'*)

plt.title(*' Example of predict a digit '*)

plt.show()

pred = model.predict(test\_x[img\_idx].reshape(1, 28, 28, 1))

print(*"predicted output: "*,pred.argmax())

For CNN network first the MNIST dataset is loaded and split into test and train data. Then one of the digits of the dataset have been shown which is zero. Next the train and test data split into 4D matrix (X, 28, 28, 1) as the MNIST dataset images have 28 rows and 28 columns. Then the dataset is normalized by dividing each number by 255. It is evident that the train data now has the shape of (60000, 28, 28, 1) and test data has the shape of (10000, 28, 28, 1). It has been 10 classes for classification task as dataset has digits between 0 and 9 so classes are changed to binary class matrix. Then the CNN model is deployed. The CNN which has been deployed has one input layer and several hidden layers with Relu activation function and one output layer with softmax activation function (as this is a multiclass classification task). Each layer consists of more than one neuron. In the input layer a convolution layer with 20 neurons with the kernel size of (3,3) for convolution is developed. Then another convolutional layer with 40 neurons and (2,2) kernel size has been developed. Next there is a need for Max pooling layer which consists of (2,2) pool size. Then the model has been flattened as it needs to added to a MLP model for classification. Then there would be different hidden layers and finally there would be output layer with size of 10 as there is a classifying task for digits into 10 categories. Between each layer there is a dropout layer to prevent overfitting. Then categorical cross entropy is used for loss function and for optimizer there would be Adam optimizer. Next training and validation data are trained with mini-batch method with different epochs and accuracy, precision and recall of our trained model is reported. Then the model is tested on test data and the accuracy of the model is reported. The train, validation and test data are 51000, 9000 and 10000 samples, respectively. Next the accuracy and loss are plotted for train and validation, and precision and recall is plotted as well. Finally, an example is predicted to see how much it works on real data.

1. **Results**
   1. **MLP Results**

The best test result for MLP model was 97.72% with the time of 98.76 seconds with one hidden layer 128 neurons plus two hidden layers each consist of 64 neurons (see the Appendix section). 20 epochs are used with batch size of 64. As it can be seen accuracy and validation, precision and recall (Figures 6 and 10) are high enough to show that the model is converged and trained very well (around 97% in Figures 5 and 9). The loss shows around 9% in Figures 7 and 11 which is good for converging. Finally, it can be seen that predicted output is exactly 9 which was expected (Figure 8).

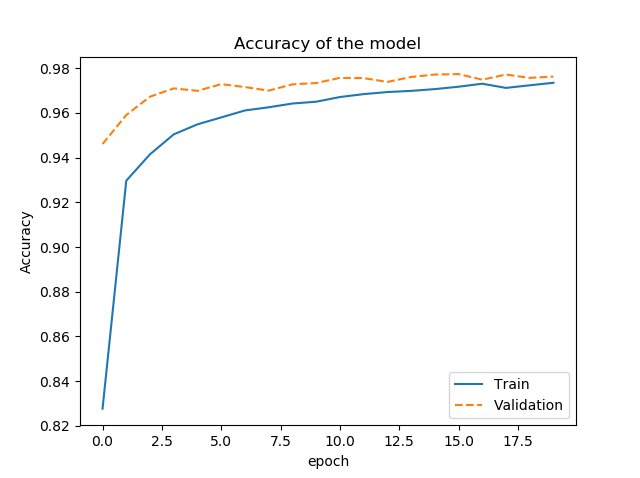


Figure5. Train and validation accuracy for MLP with 4 hidden layers each 128 neurons.

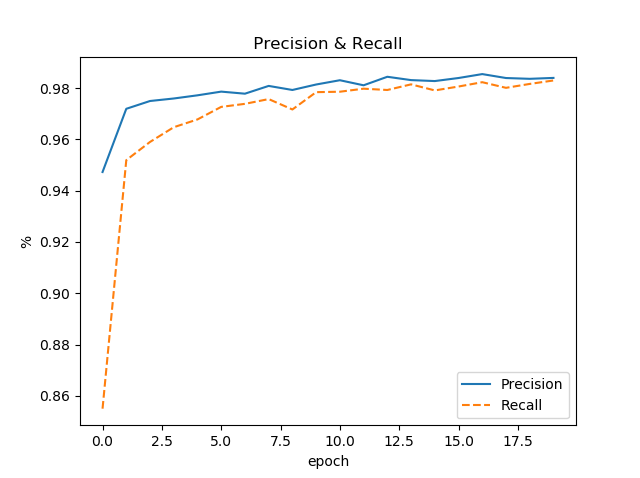


Figure6. Precision and recall for MLP with 4 hidden layers each 128 neurons.

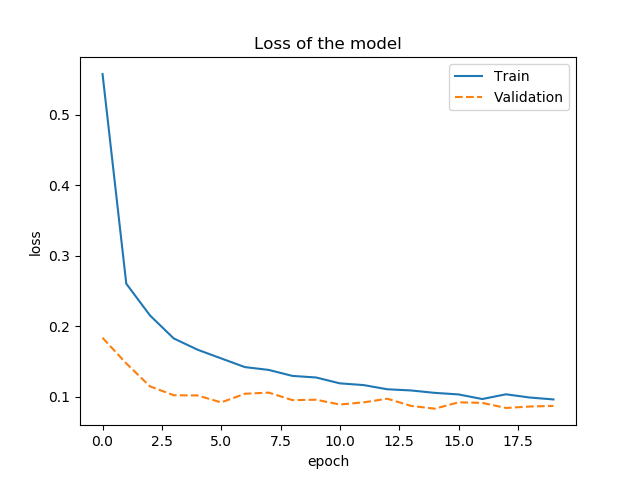


Figure7. Train and validation loss for MLP with 4 hidden layers each 128 neurons.

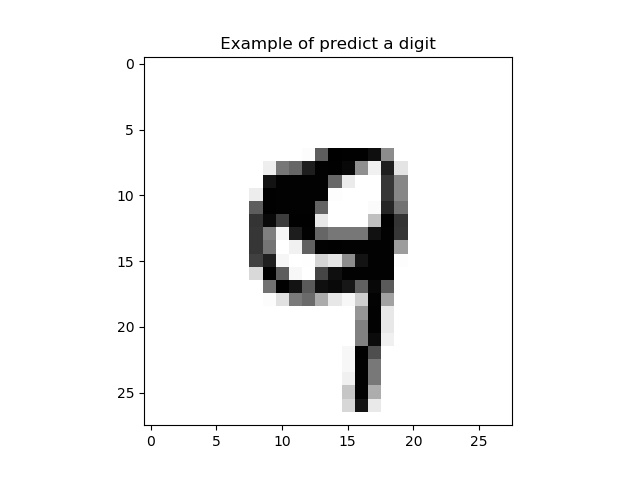


Figure8. Example of predicted image which is 9.

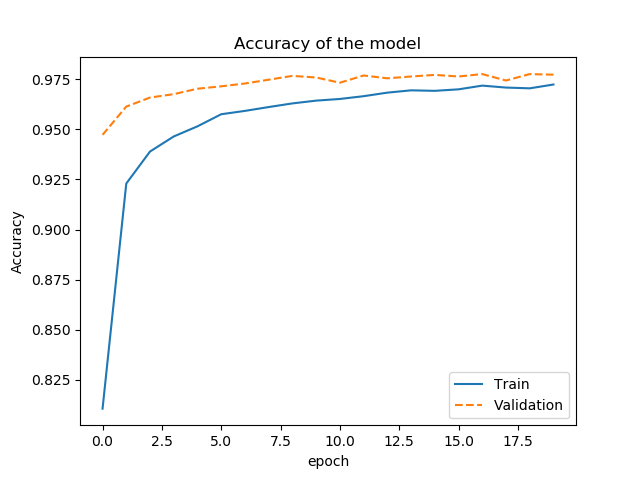


Figure9. Train and validation accuracy for MLP with 3 hidden layers 128+64+64 neurons.

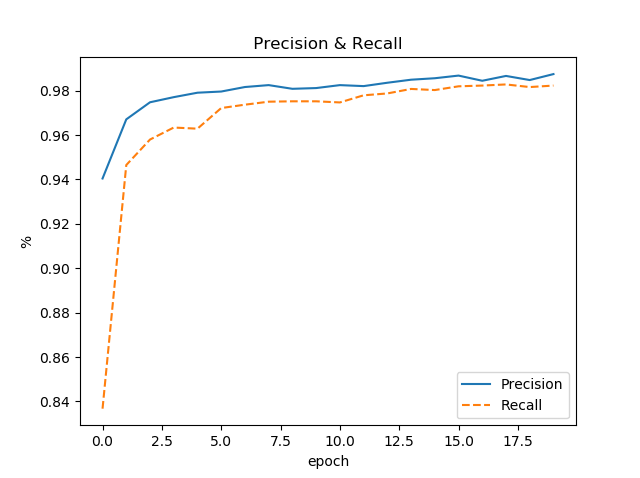


Figure10. Precision and recall for MLP with 3 hidden layers 128+64+64 neurons.

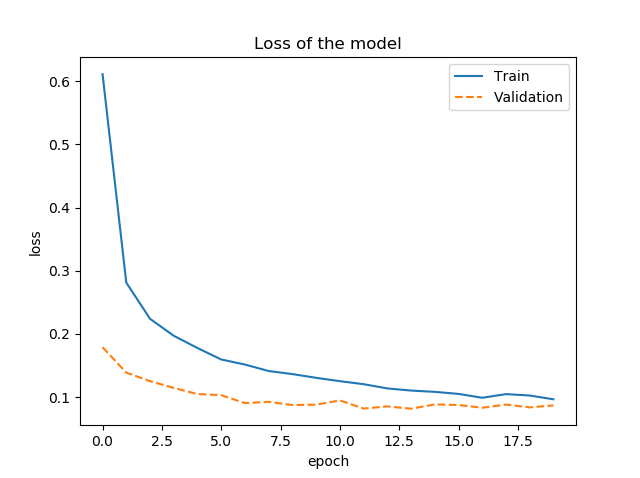


Figure11. Train and validation loss for MLP with 3 hidden layers 128+64+64 neurons.

* 1. **CNN Results**

The best test result for CNN model was 99% with the time of 1338.21 seconds with two convolutional layer (20 and 40 neurons) a Max pooling layer and two hidden layers each consist of 128 neurons (see the Appendix section) and 10 epochs with batch size of 128 are used. As it can be seen accuracy and validation, precision and recall (Figures 13 and 16) are high enough to show that the model is converged and trained very well (around 98% in Figures 12 and 15). The loss shows around 5% in Figures 14 and 17 which is good for converging. Finally, it can be seen that the predicted output is exactly 9 which was expected.

A compare of all results of MLP and CNN networks is shown in Table 1. This is evident that CNN network with 99% of test accuracy has the best results. However, the time for convergence for CNN networks are so much (1338 seconds for CNN in compare with 98 seconds for MLP) as the complexity of these networks are much higher.

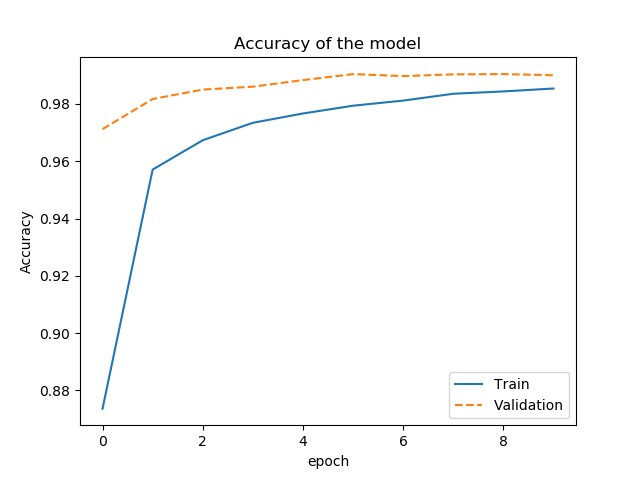


Figure12. Train and validation accuracy for CNN with 2 convolutional layers (20 and 40) and 2 hidden layers each 128 neurons.

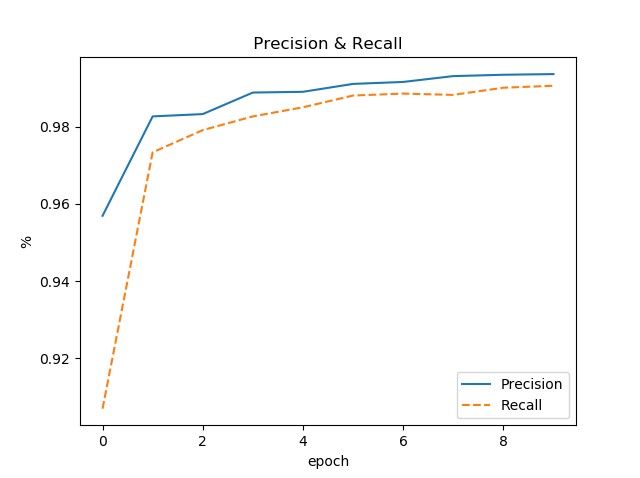


Figure13. Precision and recall for CNN with 2 convolutional layers (20 and 40) and 2 hidden layers each 128 neurons.

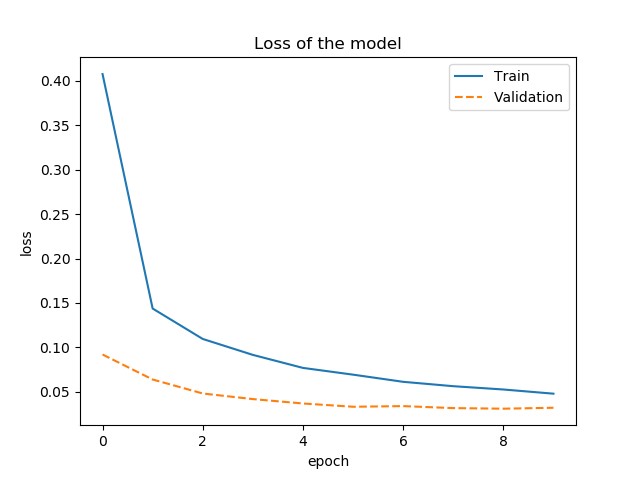


Figure14. Train and validation loss for CNN with 2 convolutional layers (20 and 40) and 2 hidden layers 128 neurons.

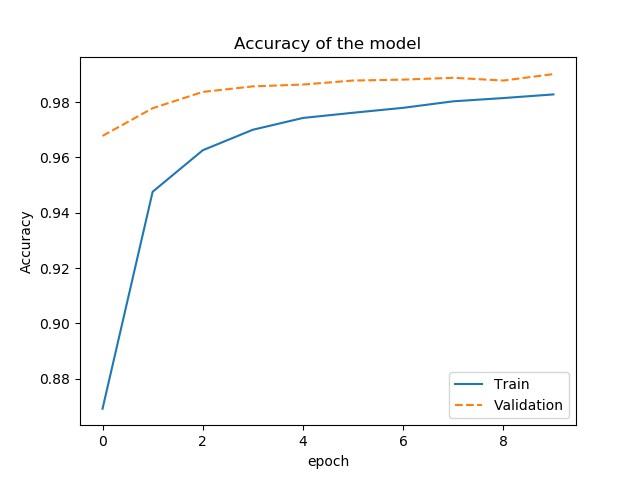


Figure15. Train and validation accuracy for CNN with 2 convolutional layers (20 and 20) and 1 hidden layer with 128 neurons.

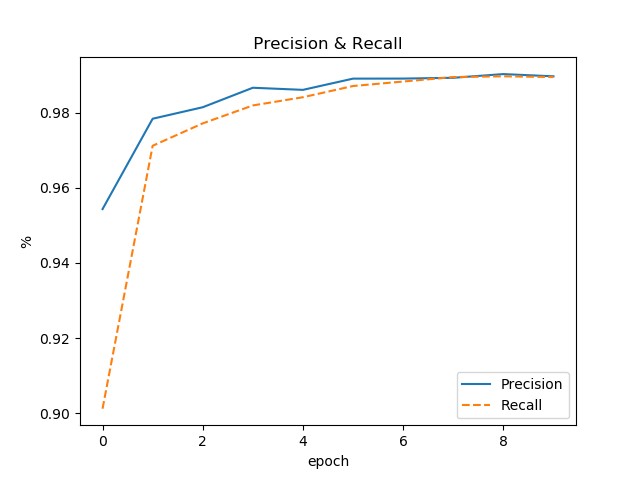


Figure16. Precision and recall for CNN with 2 convolutional layers (20 and 20) and 1 hidden layer with 128 neurons.

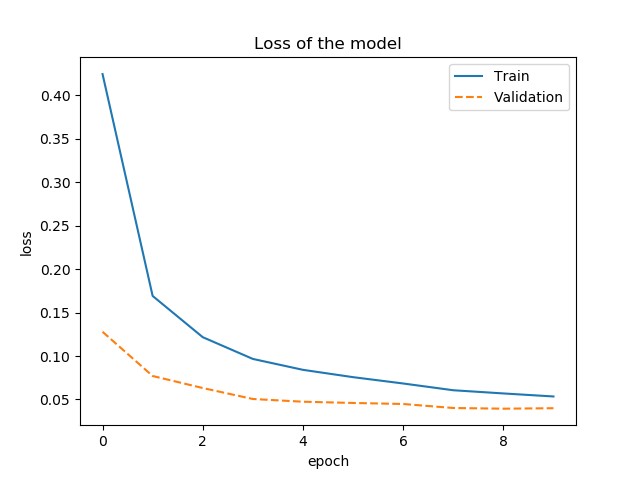


Figure17. Train and validation loss for CNN with 2 convolutional layers (20 and 20) and 1 hidden layer with 128 neurons.

Table1. Compare results of MLPs and CNNs with different hidden layers.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Network | Training loss | Training Accuracy | Validation loss | Validation Accuracy | Precision | Recall | Test Accuracy | Test loss | Time (s) |
| MLP  4\*128 | 0.096 | 0.9735 | 0.0868 | 0.9763 | 0.9839 | 0.9829 | 97.63 | 0.086 | 98.76 |
| MLP  128+(2\*64) | 0.0965 | 0.9723 | 0.0867 | 0.9772 | 0.9874 | 0.9823 | 97.72 | 0.086 | 71.00 |
| CNN  20+40+(2\*128) | 0.0480 | 0.9854 | 0.0322 | 0.990 | 0.9936 | 0.9905 | **99.0** | 0.0321 | 1338.21 |
| CNN  20+20+128 | 0.0534 | 0.9828 | 0.0399 | 0.9901 | 0.9897 | 0.9895 | 98.96 | 0.0316 | 777.34 |

1. **Conclusions**

In this project two important Deep Learning algorithms namely Multilayer Perceptron and Convolutional Neural Network have been introduced. The MNIST dataset have been used for multiclass image classification task for the MLP and CNN models. Different models with different hidden layers have been developed. Results have shown that CNN had the best results with 99% accuracy for the test data. However, it is evident that CNN network is more complex than MLP and need more time to convergence.

1. **References**

[1] “[1704.08068] Understanding the Feedforward Artificial Neural Network Model from the Perspective of Network Flow.” [Online]. Available: https://arxiv.org/abs/1704.08068. [Accessed: 08-Mar-2019].

[2] “On complexity analysis of supervised MLP-learning for algorithmic comparisons - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/document/939044. [Accessed: 08-Mar-2019].

[3] “Simple convolutional neural network on image classification - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/document/8078730. [Accessed: 08-Mar-2019].

[4] “Convolutional neural networks: an overview and application in radiology | SpringerLink.” [Online]. Available: https://link.springer.com/article/10.1007/s13244-018-0639-9. [Accessed: 08-Mar-2019].

[5] W.-S. Lu, *Applications of Digital Processing Techniques*. 2019.

[6] “Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction.” [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5998766/. [Accessed: 08-Mar-2019].

1. **Appendix**

**MLP Output**

x\_train shape: (60000, 784)

60000 train samples

10000 test samples

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_1 (Dense) (None, 128) 100480

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 128) 16512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 128) 16512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 128) 16512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_4 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 10) 1290

=================================================================

Total params: 151,306

Trainable params: 151,306

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 51000 samples, validate on 9000 samples

Epoch 1/20

- 6s - loss: 0.5579 - acc: 0.8277 - precision: 0.9473 - recall: 0.8550 - val\_loss: 0.1835 - val\_acc: 0.9461 - val\_precision: 0.9583 - val\_recall: 0.9837

Epoch 2/20

- 5s - loss: 0.2603 - acc: 0.9297 - precision: 0.9719 - recall: 0.9519 - val\_loss: 0.1472 - val\_acc: 0.9591 - val\_precision: 0.9906 - val\_recall: 0.9684

Epoch 3/20

- 6s - loss: 0.2153 - acc: 0.9415 - precision: 0.9749 - recall: 0.9590 - val\_loss: 0.1145 - val\_acc: 0.9673 - val\_precision: 0.9778 - val\_recall: 0.9888

Epoch 4/20

- 5s - loss: 0.1826 - acc: 0.9504 - precision: 0.9759 - recall: 0.9647 - val\_loss: 0.1020 - val\_acc: 0.9710 - val\_precision: 0.9729 - val\_recall: 0.9898

Epoch 5/20

- 5s - loss: 0.1667 - acc: 0.9549 - precision: 0.9772 - recall: 0.9678 - val\_loss: 0.1017 - val\_acc: 0.9699 - val\_precision: 0.9797 - val\_recall: 0.9857

Epoch 6/20

- 5s - loss: 0.1543 - acc: 0.9580 - precision: 0.9786 - recall: 0.9726 - val\_loss: 0.0918 - val\_acc: 0.9729 - val\_precision: 0.9798 - val\_recall: 0.9878

Epoch 7/20

- 5s - loss: 0.1419 - acc: 0.9611 - precision: 0.9778 - recall: 0.9738 - val\_loss: 0.1041 - val\_acc: 0.9716 - val\_precision: 0.9710 - val\_recall: 0.9908

Epoch 8/20

- 5s - loss: 0.1379 - acc: 0.9626 - precision: 0.9808 - recall: 0.9757 - val\_loss: 0.1057 - val\_acc: 0.9700 - val\_precision: 0.9768 - val\_recall: 0.9888

Epoch 9/20

- 5s - loss: 0.1294 - acc: 0.9642 - precision: 0.9792 - recall: 0.9716 - val\_loss: 0.0951 - val\_acc: 0.9728 - val\_precision: 0.9788 - val\_recall: 0.9878

Epoch 10/20

- 5s - loss: 0.1272 - acc: 0.9650 - precision: 0.9814 - recall: 0.9784 - val\_loss: 0.0956 - val\_acc: 0.9734 - val\_precision: 0.9868 - val\_recall: 0.9908

Epoch 11/20

- 5s - loss: 0.1189 - acc: 0.9671 - precision: 0.9830 - recall: 0.9786 - val\_loss: 0.0888 - val\_acc: 0.9757 - val\_precision: 0.9798 - val\_recall: 0.9908

Epoch 12/20

- 5s - loss: 0.1164 - acc: 0.9684 - precision: 0.9811 - recall: 0.9797 - val\_loss: 0.0919 - val\_acc: 0.9756 - val\_precision: 0.9857 - val\_recall: 0.9827

Epoch 13/20

- 5s - loss: 0.1105 - acc: 0.9694 - precision: 0.9844 - recall: 0.9792 - val\_loss: 0.0970 - val\_acc: 0.9739 - val\_precision: 0.9760 - val\_recall: 0.9939

Epoch 14/20

- 5s - loss: 0.1088 - acc: 0.9699 - precision: 0.9831 - recall: 0.9814 - val\_loss: 0.0868 - val\_acc: 0.9761 - val\_precision: 0.9827 - val\_recall: 0.9857

Epoch 15/20

- 5s - loss: 0.1054 - acc: 0.9707 - precision: 0.9827 - recall: 0.9791 - val\_loss: 0.0829 - val\_acc: 0.9772 - val\_precision: 0.9877 - val\_recall: 0.9867

Epoch 16/20

- 5s - loss: 0.1031 - acc: 0.9718 - precision: 0.9839 - recall: 0.9806 - val\_loss: 0.0919 - val\_acc: 0.9774 - val\_precision: 0.9818 - val\_recall: 0.9929

Epoch 17/20

- 5s - loss: 0.0966 - acc: 0.9731 - precision: 0.9854 - recall: 0.9823 - val\_loss: 0.0912 - val\_acc: 0.9749 - val\_precision: 0.9798 - val\_recall: 0.9918

Epoch 18/20

- 5s - loss: 0.1033 - acc: 0.9712 - precision: 0.9839 - recall: 0.9801 - val\_loss: 0.0839 - val\_acc: 0.9772 - val\_precision: 0.9867 - val\_recall: 0.9847

Epoch 19/20

- 5s - loss: 0.0987 - acc: 0.9724 - precision: 0.9836 - recall: 0.9816 - val\_loss: 0.0860 - val\_acc: 0.9757 - val\_precision: 0.9818 - val\_recall: 0.9918

Epoch 20/20

- 5s - loss: 0.0960 - acc: 0.9735 - precision: 0.9839 - recall: 0.9829 - val\_loss: 0.0868 - val\_acc: 0.9763 - val\_precision: 0.9739 - val\_recall: 0.9918

Test Accuracy: 97.63

Test loss: 0.08684454486782779

Time elapsed (seconds): 98.76264929771423

predicted output: 9

x\_train shape: (60000, 784)

60000 train samples

10000 test samples

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_1 (Dense) (None, 128) 100480

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 64) 8256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 64) 4160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 10) 650

=================================================================

Total params: 113,546

Trainable params: 113,546

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 51000 samples, validate on 9000 samples

Epoch 1/20

- 4s - loss: 0.6113 - acc: 0.8107 - precision: 0.9404 - recall: 0.8367 - val\_loss: 0.1789 - val\_acc: 0.9472 - val\_precision: 0.9736 - val\_recall: 0.9796

Epoch 2/20

- 3s - loss: 0.2812 - acc: 0.9229 - precision: 0.9671 - recall: 0.9465 - val\_loss: 0.1387 - val\_acc: 0.9613 - val\_precision: 0.9767 - val\_recall: 0.9847

Epoch 3/20

- 3s - loss: 0.2239 - acc: 0.9389 - precision: 0.9747 - recall: 0.9580 - val\_loss: 0.1252 - val\_acc: 0.9658 - val\_precision: 0.9719 - val\_recall: 0.9888

Epoch 4/20

- 3s - loss: 0.1971 - acc: 0.9464 - precision: 0.9771 - recall: 0.9634 - val\_loss: 0.1145 - val\_acc: 0.9675 - val\_precision: 0.9758 - val\_recall: 0.9857

Epoch 5/20

- 3s - loss: 0.1776 - acc: 0.9515 - precision: 0.9791 - recall: 0.9629 - val\_loss: 0.1048 - val\_acc: 0.9702 - val\_precision: 0.9797 - val\_recall: 0.9847

Epoch 6/20

- 3s - loss: 0.1595 - acc: 0.9575 - precision: 0.9796 - recall: 0.9721 - val\_loss: 0.1030 - val\_acc: 0.9714 - val\_precision: 0.9847 - val\_recall: 0.9837

Epoch 7/20

- 3s - loss: 0.1515 - acc: 0.9591 - precision: 0.9816 - recall: 0.9737 - val\_loss: 0.0904 - val\_acc: 0.9728 - val\_precision: 0.9768 - val\_recall: 0.9898

Epoch 8/20

- 3s - loss: 0.1412 - acc: 0.9611 - precision: 0.9825 - recall: 0.9750 - val\_loss: 0.0925 - val\_acc: 0.9747 - val\_precision: 0.9671 - val\_recall: 0.9908

Epoch 9/20

- 3s - loss: 0.1364 - acc: 0.9629 - precision: 0.9808 - recall: 0.9752 - val\_loss: 0.0874 - val\_acc: 0.9766 - val\_precision: 0.9807 - val\_recall: 0.9878

Epoch 10/20

- 3s - loss: 0.1305 - acc: 0.9643 - precision: 0.9811 - recall: 0.9752 - val\_loss: 0.0880 - val\_acc: 0.9758 - val\_precision: 0.9710 - val\_recall: 0.9918

Epoch 11/20

- 3s - loss: 0.1252 - acc: 0.9651 - precision: 0.9825 - recall: 0.9747 - val\_loss: 0.0947 - val\_acc: 0.9732 - val\_precision: 0.9710 - val\_recall: 0.9908

Epoch 12/20

- 3s - loss: 0.1204 - acc: 0.9665 - precision: 0.9820 - recall: 0.9779 - val\_loss: 0.0818 - val\_acc: 0.9768 - val\_precision: 0.9818 - val\_recall: 0.9888

Epoch 13/20

- 3s - loss: 0.1137 - acc: 0.9682 - precision: 0.9835 - recall: 0.9787 - val\_loss: 0.0853 - val\_acc: 0.9754 - val\_precision: 0.9817 - val\_recall: 0.9878

Epoch 14/20

- 3s - loss: 0.1103 - acc: 0.9694 - precision: 0.9849 - recall: 0.9808 - val\_loss: 0.0816 - val\_acc: 0.9763 - val\_precision: 0.9856 - val\_recall: 0.9806

Epoch 15/20

- 3s - loss: 0.1082 - acc: 0.9691 - precision: 0.9856 - recall: 0.9802 - val\_loss: 0.0883 - val\_acc: 0.9771 - val\_precision: 0.9808 - val\_recall: 0.9898

Epoch 16/20

- 3s - loss: 0.1050 - acc: 0.9699 - precision: 0.9868 - recall: 0.9819 - val\_loss: 0.0874 - val\_acc: 0.9763 - val\_precision: 0.9858 - val\_recall: 0.9888

Epoch 17/20

- 4s - loss: 0.0989 - acc: 0.9718 - precision: 0.9844 - recall: 0.9823 - val\_loss: 0.0831 - val\_acc: 0.9775 - val\_precision: 0.9818 - val\_recall: 0.9888

Epoch 18/20

- 3s - loss: 0.1047 - acc: 0.9708 - precision: 0.9866 - recall: 0.9828 - val\_loss: 0.0882 - val\_acc: 0.9743 - val\_precision: 0.9897 - val\_recall: 0.9796

Epoch 19/20

- 3s - loss: 0.1025 - acc: 0.9704 - precision: 0.9848 - recall: 0.9816 - val\_loss: 0.0837 - val\_acc: 0.9775 - val\_precision: 0.9788 - val\_recall: 0.9908

Epoch 20/20

- 3s - loss: 0.0965 - acc: 0.9723 - precision: 0.9874 - recall: 0.9823 - val\_loss: 0.0867 - val\_acc: 0.9772 - val\_precision: 0.9837 - val\_recall: 0.9867

Test Accuracy: 97.72

Test loss: 0.08670758328863012

Time elapsed (seconds): 71.00206112861633

**CNN Output**

x\_train shape: (60000, 28, 28, 1)

60000 train samples

10000 test samples

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 20) 200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 26, 26, 20) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 25, 25, 40) 3240

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 40) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 12, 12, 40) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 5760) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 737408

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 128) 16512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_4 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 10) 1290

=================================================================

Total params: 758,650

Trainable params: 758,650

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 51000 samples, validate on 9000 samples

Epoch 1/10

- 101s - loss: 0.4073 - acc: 0.8736 - precision: 0.9569 - recall: 0.9070 - val\_loss: 0.0921 - val\_acc: 0.9712 - val\_precision: 0.9845 - val\_recall: 0.9745

Epoch 2/10

- 112s - loss: 0.1438 - acc: 0.9571 - precision: 0.9826 - recall: 0.9733 - val\_loss: 0.0640 - val\_acc: 0.9817 - val\_precision: 0.9808 - val\_recall: 0.9929

Epoch 3/10

- 112s - loss: 0.1095 - acc: 0.9673 - precision: 0.9832 - recall: 0.9791 - val\_loss: 0.0482 - val\_acc: 0.9850 - val\_precision: 0.9839 - val\_recall: 0.9949

Epoch 4/10

- 149s - loss: 0.0916 - acc: 0.9734 - precision: 0.9888 - recall: 0.9826 - val\_loss: 0.0419 - val\_acc: 0.9860 - val\_precision: 0.9908 - val\_recall: 0.9908

Epoch 5/10

- 168s - loss: 0.0770 - acc: 0.9767 - precision: 0.9890 - recall: 0.9850 - val\_loss: 0.0370 - val\_acc: 0.9883 - val\_precision: 0.9859 - val\_recall: 0.9959

Epoch 6/10

- 117s - loss: 0.0694 - acc: 0.9794 - precision: 0.9910 - recall: 0.9880 - val\_loss: 0.0333 - val\_acc: 0.9904 - val\_precision: 0.9949 - val\_recall: 0.9908

Epoch 7/10

- 114s - loss: 0.0614 - acc: 0.9812 - precision: 0.9915 - recall: 0.9885 - val\_loss: 0.0341 - val\_acc: 0.9897 - val\_precision: 0.9908 - val\_recall: 0.9918

Epoch 8/10

- 125s - loss: 0.0564 - acc: 0.9835 - precision: 0.9930 - recall: 0.9882 - val\_loss: 0.0318 - val\_acc: 0.9903 - val\_precision: 0.9909 - val\_recall: 0.9959

Epoch 9/10

- 211s - loss: 0.0527 - acc: 0.9844 - precision: 0.9934 - recall: 0.9900 - val\_loss: 0.0311 - val\_acc: 0.9904 - val\_precision: 0.9849 - val\_recall: 0.9969

Epoch 10/10

- 121s - loss: 0.0480 - acc: 0.9854 - precision: 0.9936 - recall: 0.9905 - val\_loss: 0.0322 - val\_acc: 0.9900 - val\_precision: 0.9819 - val\_recall: 0.9969

Test Accuracy: 99.0

Test loss: 0.032198574410984296

Time elapsed (seconds): 1338.2145411968231

predicted output: 9

x\_train shape: (60000, 28, 28, 1)

60000 train samples

10000 test samples

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 20) 200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 26, 26, 20) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 25, 25, 20) 1620

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 20) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 12, 12, 20) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 2880) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 368768

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 10) 1290

=================================================================

Total params: 371,878

Trainable params: 371,878

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 51000 samples, validate on 9000 samples

Epoch 1/10

- 66s - loss: 0.4246 - acc: 0.8692 - precision: 0.9543 - recall: 0.9012 - val\_loss: 0.1279 - val\_acc: 0.9678 - val\_precision: 0.9931 - val\_recall: 0.9719

Epoch 2/10

- 118s - loss: 0.1692 - acc: 0.9476 - precision: 0.9784 - recall: 0.9712 - val\_loss: 0.0770 - val\_acc: 0.9778 - val\_precision: 0.9866 - val\_recall: 0.9899

Epoch 3/10

- 86s - loss: 0.1217 - acc: 0.9626 - precision: 0.9814 - recall: 0.9771 - val\_loss: 0.0632 - val\_acc: 0.9837 - val\_precision: 0.9921 - val\_recall: 0.9921

Epoch 4/10

- 72s - loss: 0.0967 - acc: 0.9700 - precision: 0.9866 - recall: 0.9819 - val\_loss: 0.0505 - val\_acc: 0.9857 - val\_precision: 0.9955 - val\_recall: 0.9910

Epoch 5/10

- 72s - loss: 0.0841 - acc: 0.9743 - precision: 0.9861 - recall: 0.9841 - val\_loss: 0.0474 - val\_acc: 0.9863 - val\_precision: 0.9910 - val\_recall: 0.9933

Epoch 6/10

- 71s - loss: 0.0757 - acc: 0.9761 - precision: 0.9890 - recall: 0.9871 - val\_loss: 0.0460 - val\_acc: 0.9878 - val\_precision: 0.9989 - val\_recall: 0.9899

Epoch 7/10

- 71s - loss: 0.0684 - acc: 0.9779 - precision: 0.9891 - recall: 0.9883 - val\_loss: 0.0448 - val\_acc: 0.9881 - val\_precision: 0.9955 - val\_recall: 0.9921

Epoch 8/10

- 72s - loss: 0.0606 - acc: 0.9803 - precision: 0.9893 - recall: 0.9895 - val\_loss: 0.0402 - val\_acc: 0.9888 - val\_precision: 0.9966 - val\_recall: 0.9944

Epoch 9/10

- 72s - loss: 0.0569 - acc: 0.9815 - precision: 0.9903 - recall: 0.9897 - val\_loss: 0.0394 - val\_acc: 0.9878 - val\_precision: 0.9944 - val\_recall: 0.9933

Epoch 10/10

- 72s - loss: 0.0534 - acc: 0.9828 - precision: 0.9897 - recall: 0.9895 - val\_loss: 0.0399 - val\_acc: 0.9901 - val\_precision: 0.9966 - val\_recall: 0.9944

Test Accuracy: 98.96000000000001

Test loss: 0.031684402347984725

Time elapsed (seconds): 777.3478367328644

predicted output: 9

1. <http://yann.lecun.com/exdb/mnist/> [↑](#footnote-ref-1)