

Keras on MNIST dataset

Objective : To apply multiple hidden layer [3,5,7,9...] MLP architecture on MNIST dataset

```
1 import warnings
2 from sklearn.exceptions import DataConversionWarning
3 warnings.filterwarnings(action='ignore', category=DataConversionWarning)
4
5 # For plotting purposes
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 from sklearn.preprocessing import MinMaxScaler
9 from keras.utils import to_categorical
10 from keras.models import Sequential
11 from keras.initializers import he_normal
12 from keras.layers import BatchNormalization, Dense, Dropout
13
14 # Import MNIST Dataset
15 from keras.datasets import mnist
```

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

```
1 print("x_train shape: ", x_train.shape)
2 print("x_test shape: ", x_test.shape)
3 print("Number of training examples :", x_train.shape[0], "and each image is of shape (%d, %d)"%(x_train
4 print("Number of testing examples :", x_test.shape[0], "and each image is of shape (%d, %d)"%(x_test.shi
```

x_train shape: (60000, 28, 28)

x_test shape: (10000, 28, 28)

Number of training examples : 60000 and each image is of shape (28, 28)

Number of testing examples : 10000 and each image is of shape (28, 28)

```
1  #converting from 3d to 1*784
2  x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]*x_train.shape[2])
3  x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[2])
4
5  # after converting the input images from 3d to 2d vectors
6  print("x_train shape: ", x_train.shape)
7  print("x_test shape: ", x_test.shape)
8  print("Number of training examples :", x_train.shape[0], "and each image is of shape (%d)"%(x_train.shap
9  print("Number of training examples :", x_test.shape[0], "and each image is of shape (%d)"%(x_test.shape
```

x_train shape: (60000, 784)

x_test shape: (10000, 784)

Number of training examples : 60000 and each image is of shape (784)

Number of training examples : 10000 and each image is of shape (784)

```
1 #Normalising data
2 minMaxScaler = MinMaxScaler()
3
4 x_train = minMaxScaler.fit_transform(x_train)
5 x_test = minMaxScaler.transform(x_test)
6
7 # x_train data point after normlizing.
8 print(x_train[0])
```

[illegible]

```
1 temp = y_train[0]
2 y_train = keras.utils.to_categorical(y_train)
3 y_test = keras.utils.to_categorical(y_test)
4
5 print("After converting the output {0} into a vector : {1}".format(temp,y_train[0]))
```

After converting the output 5 into a vector : [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

```
1 # Function for plotting train and cross validation loss
2 def plot_train_cv_loss(trained_model, epochs, colors=['b']):
3     fig, ax = plt.subplots(1,1)
4     ax.set_xlabel('epoch')
5     ax.set_ylabel('Categorical Crossentropy Loss')
6     x_axis_values = list(range(1,epochs+1))
7
8     validation_loss = trained_model.history['val_loss']
9     train_loss = trained_model.history['loss']
10
11     ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
12     ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
13     plt.legend()
14     plt.grid()
15     fig.canvas.draw()
```

```
1 # Defining batch size and epochs
2 # Batch size
3 batch_size = 128
4
5 # Number of time whole data is trained
6 epochs =50
7
8 # Input Layer dimension
9 input_dimension = x_train.shape[1]
10
11 # Output Layer dimension
12 output_dimension = y_train.shape[1]
```

Try 3 Hidden layer architecture

```
1  # Instantiate sequential model
2  model = Sequential()
3
4  # Add 1st hidden Layer : dense Layer
5  dense_layer1 = Dense(750,
6                        activation="relu",
7                        input_shape=(input_dimension,),
8                        kernel_initializer= he_normal(seed=None))
9  model.add(dense_layer1)
10
11 # Add batch normalization
12 model.add(BatchNormalization())
13
14 # Add dropout
15 model.add(Dropout(0.5))
16
17 # Add 2nd hidden Layer : dense Layer
18 dense_layer2 = Dense(500,
19                      activation="relu",
20                      kernel_initializer= he_normal(seed=None))
21 model.add(dense_layer2)
22
23 # Add batch normalization
24 model.add(BatchNormalization())
25
26 # Add dropout
27 model.add(Dropout(0.5))
28
29
30 # Add 3rd hidden Layer : dense Layer
31 dense_layer3 = Dense(250,
32                      activation="relu",
33                      kernel_initializer= he_normal(seed=None))
34 model.add(dense_layer3)
```

```
35
36 # Add batch normalization
37 model.add(BatchNormalization())
38
39 # Add dropout
40 model.add(Dropout(0.5))
41
42 # Add output layer : dense layer
43 dense_layer4 = Dense(output_dimension, activation='softmax')
44 model.add(dense_layer4)
45
46 # Summary of the model
47 print("Model Summary: \n")
48 model.summary()
49 print()
50 print()
51
52 # Compile the model
53 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
54
55 # Run the model
56 trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, validation_data=(x_test, y_test))

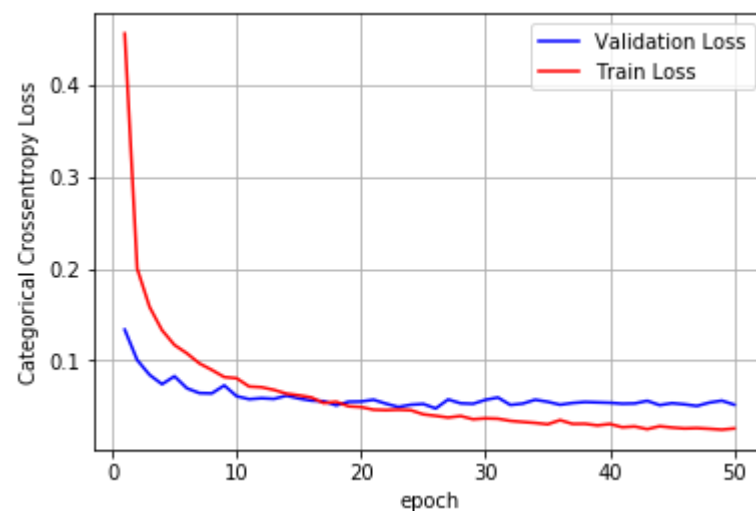
60000/60000 [=====] - 4s 61us/step - loss: 0.0399 - acc: 0.9876 - val_loss: 0.0480 - val_acc: 0.9862
Epoch 27/50
60000/60000 [=====] - 4s 73us/step - loss: 0.0381 - acc: 0.9876 - val_loss: 0.0578 - val_acc: 0.9844
Epoch 28/50
60000/60000 [=====] - 4s 60us/step - loss: 0.0398 - acc: 0.9874 - val_loss: 0.0536 - val_acc: 0.9847
Epoch 29/50
60000/60000 [=====] - 4s 61us/step - loss: 0.0362 - acc: 0.9882 - val_loss: 0.0530 - val_acc: 0.9859
Epoch 30/50
60000/60000 [=====] - 4s 61us/step - loss: 0.0372 - acc: 0.9879 - val_loss: 0.0574 - val_acc: 0.9843
Epoch 31/50
60000/60000 [=====] - 4s 61us/step - loss: 0.0368 - acc: 0.9877 - val_loss: 0.0599 - val_acc: 0.9843
```

```
833
Epoch 32/50
60000/60000 [=====] - 4s 62us/step - loss: 0.0345 - acc: 0.9889 - val_loss: 0.0518 - val_acc: 0.9
860
Epoch 33/50
60000/60000 [=====] - 4s 61us/step - loss: 0.0333 - acc: 0.9895 - val_loss: 0.0533 - val_acc: 0.9
```

```
1 # Accuracy and plotting train & validation loss
2 score = model.evaluate(x_test, y_test, verbose=0)
3 print('Test score:', score[0])
4 print('Test accuracy: {0:.2f}%'.format(score[1]*100))
5
6
7 # Plot train and cross validation error
8 plot_train_cv_loss(trained_model, epochs)
```

Test score: 0.05205892696186711

Test accuracy: 98.61%



Around 17 Epoch train error and validation error meets.

With 5 Hidden layer

```
1  # Instantiate sequential model
2  model = Sequential()
3
4  # Add 1st hidden Layer : dense Layer
5  dense_layer1 = Dense(1150,
6                        activation="relu",
7                        input_shape=(input_dimension,),
8                        kernel_initializer= he_normal(seed=None))
9  model.add(dense_layer1)
10
11 # Add batch normalization
12 model.add(BatchNormalization())
13
14 # Add dropout
15 model.add(Dropout(0.5))
16
17 # Add 2nd hidden Layer : dense Layer
18 dense_layer2 = Dense(900,
19                     activation="relu",
20                     kernel_initializer= he_normal(seed=None))
21 model.add(dense_layer2)
22
23 # Add batch normalization
24 model.add(BatchNormalization())
25
26 # Add dropout
27 model.add(Dropout(0.5))
28
29
30 # Add 3rd hidden Layer : dense Layer
31 dense_layer3 = Dense(750,
32                     activation="relu",
33                     kernel_initializer= he_normal(seed=None))
34 model.add(dense_layer3)
```



```
35
36 # Add batch normalization
37 model.add(BatchNormalization())
38
39 # Add dropout
40 model.add(Dropout(0.5))
41
42 # Add 4th hidden layer : dense Layer
43 dense_layer4 = Dense(500,
44                     activation="relu",
45                     kernel_initializer= he_normal(seed=None))
46 model.add(dense_layer4)
47
48 # Add batch normalization
49 model.add(BatchNormalization())
50
51 # Add dropout
52 model.add(Dropout(0.5))
53
54 # Add 5th hidden layer : dense Layer
55 dense_layer5 = Dense(250,
56                     activation="relu",
57                     kernel_initializer= he_normal(seed=None))
58 model.add(dense_layer5)
59
60 # Add batch normalization
61 model.add(BatchNormalization())
62
63 # Add dropout
64 model.add(Dropout(0.5))
65
66 # Add output layer : dense Layer
67 dense_layer6 = Dense(output_dimension, activation='softmax')
68 model.add(dense_layer6)
69
```

```
70 # Summary of the model
71 print("Model Summary: \n")
72 model.summary()
73 print()
74 print()
75
76 # Compile the model
77 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
78
79 # Run the model
80 trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, validation_data=(x_test, y_test))
```

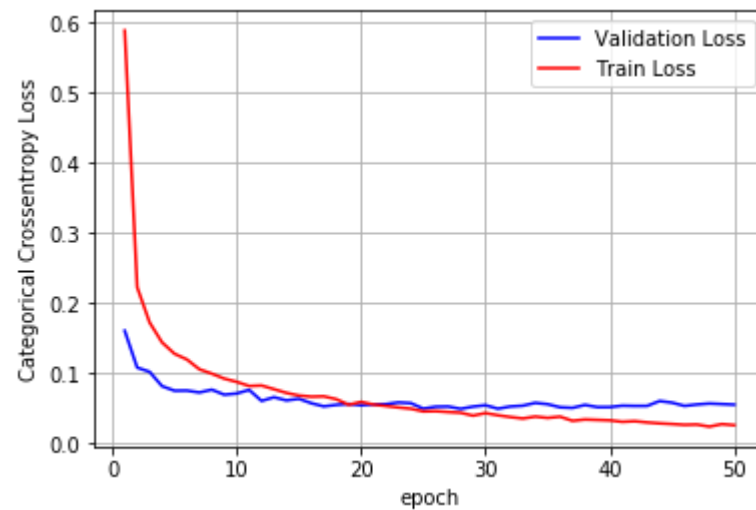
Model Summary:

Layer (type)	Output Shape	Param #
=====		
dense_8 (Dense)	(None, 1150)	902750
batch_normalization_6 (Batch Normalization)	(None, 1150)	4600
dropout_6 (Dropout)	(None, 1150)	0
dense_9 (Dense)	(None, 900)	1035900
batch_normalization_7 (Batch Normalization)	(None, 900)	3600
dropout_7 (Dropout)	(None, 900)	0
dense_10 (Dense)	(None, 750)	675750
batch_normalization_8 (Batch Normalization)	(None, 750)	3000

```
1 # Accuracy and plotting train & validation loss
2 score = model.evaluate(x_test, y_test, verbose=0)
3 print('Test score:', score[0])
4 print('Test accuracy: {:.2f}%'.format(score[1]*100))
5
6
7 # Plot train and cross validation error
8 plot_train_cv_loss(trained_model, epochs)
```

Test score: 0.05508785108429729

Test accuracy: 98.68%



Around 20 Epoch train error and validation error meets.

Try 7 Hidden Layer

```
1  # Instantiate sequential model
2  model = Sequential()
3
4  # Add 1st hidden Layer : dense Layer
5  dense_layer1 = Dense(1250,
6                        activation="relu",
7                        input_shape=(input_dimension,),
8                        kernel_initializer= he_normal(seed=None))
9  model.add(dense_layer1)
10
11 # Add batch normalization
12 model.add(BatchNormalization())
13
14 # Add dropout
15 model.add(Dropout(0.5))
16
17 # Add 2nd hidden Layer : dense Layer
18 dense_layer2 = Dense(1000,
19                      activation="relu",
20                      kernel_initializer= he_normal(seed=None))
21 model.add(dense_layer2)
22
23 # Add batch normalization
24 model.add(BatchNormalization())
25
26 # Add dropout
27 model.add(Dropout(0.5))
28
29
30 # Add 3rd hidden Layer : dense Layer
31 dense_layer3 = Dense(750,
32                      activation="relu",
33                      kernel_initializer= he_normal(seed=None))
34 model.add(dense_layer3)
```

```
35
36 # Add batch normalization
37 model.add(BatchNormalization())
38
39 # Add dropout
40 model.add(Dropout(0.5))
41
42 # Add 4th hidden layer : dense Layer
43 dense_layer4 = Dense(500,
44                     activation="relu",
45                     kernel_initializer= he_normal(seed=None))
46 model.add(dense_layer4)
47
48 # Add batch normalization
49 model.add(BatchNormalization())
50
51 # Add dropout
52 model.add(Dropout(0.5))
53
54 # Add 5th hidden layer : dense Layer
55 dense_layer5 = Dense(750,
56                     activation="relu",
57                     kernel_initializer= he_normal(seed=None))
58 model.add(dense_layer5)
59
60 # Add batch normalization
61 model.add(BatchNormalization())
62
63 # Add dropout
64 model.add(Dropout(0.5))
65
66 # Add 6th hidden layer : dense Layer
67 dense_layer6 = Dense(500,
68                     activation="relu",
69                     kernel_initializer= he_normal(seed=None))
```

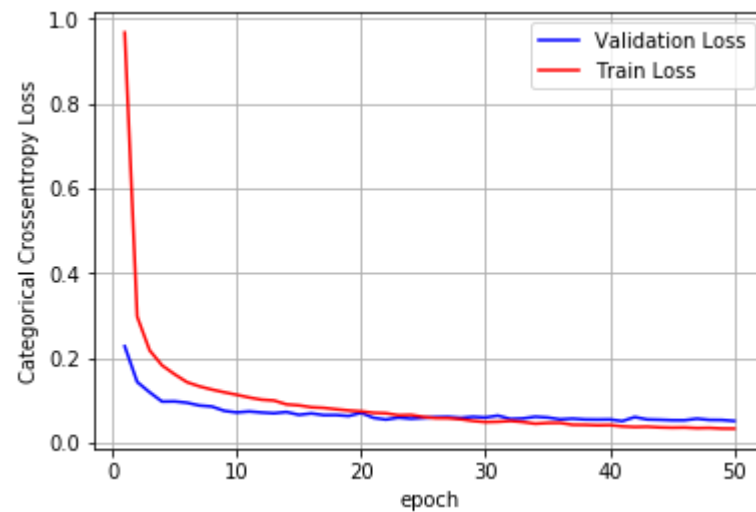
```
70 model.add(dense_layer6)
71
72 # Add batch normalization
73 model.add(BatchNormalization())
74
75 # Add dropout
76 model.add(Dropout(0.5))
77
78 # Add 7th hidden layer : dense Layer
79 dense_layer7 = Dense(250,
80                       activation="relu",
81                       kernel_initializer= he_normal(seed=None))
82 model.add(dense_layer7)
83
84 # Add batch normalization
85 model.add(BatchNormalization())
86
87 # Add dropout
88 model.add(Dropout(0.5))
89
90 # Add output layer : dense Layer
91 dense_layer8 = Dense(output_dimension, activation='softmax')
92 model.add(dense_layer8)
93
94 # Summary of the model
95 print("Model Summary: \n")
96 model.summary()
97 print()
98 print()
99
100 # Compile the model
101 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
102
103 # Run the model
104 trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, validation_data=(x_test, y_test))
```

```
60000/60000 [=====] - 7s 111us/step - loss: 0.0464 - acc: 0.9865 - val_loss: 0.0547 - val_acc: 0.9863
Epoch 37/50
60000/60000 [=====] - 7s 112us/step - loss: 0.0418 - acc: 0.9880 - val_loss: 0.0563 - val_acc: 0.9868
Epoch 38/50
60000/60000 [=====] - 7s 113us/step - loss: 0.0416 - acc: 0.9876 - val_loss: 0.0544 - val_acc: 0.9866
Epoch 39/50
60000/60000 [=====] - 7s 113us/step - loss: 0.0406 - acc: 0.9882 - val_loss: 0.0542 - val_acc: 0.9862
Epoch 40/50
60000/60000 [=====] - 7s 112us/step - loss: 0.0411 - acc: 0.9880 - val_loss: 0.0545 - val_acc: 0.9867
Epoch 41/50
```

```
1 # Accuracy and plotting train & validation loss
2 score = model.evaluate(x_test, y_test, verbose=0)
3 print('Test score:', score[0])
4 print('Test accuracy: {:.2f}%'.format(score[1]*100))
5
6
7 # Plot train and cross validation error
8 plot_train_cv_loss(trained_model, epochs)
```

Test score: 0.05090733877607854

Test accuracy: 98.70%



Around 22 Epoch train error and validation error meets.

Try 9 Hidden layer


```
1  # Instantiate sequential model
2  model = Sequential()
3
4  # Add 1st hidden Layer : dense Layer
5  dense_layer1 = Dense(2250,
6                        activation="relu",
7                        input_shape=(input_dimension,),
8                        kernel_initializer= he_normal(seed=None))
9  model.add(dense_layer1)
10
11 # Add batch normalization
12 model.add(BatchNormalization())
13
14 # Add dropout
15 model.add(Dropout(0.5))
16
17 # Add 2nd hidden Layer : dense Layer
18 dense_layer2 = Dense(2000,
19                      activation="relu",
20                      kernel_initializer= he_normal(seed=None))
21 model.add(dense_layer2)
22
23 # Add batch normalization
24 model.add(BatchNormalization())
25
26 # Add dropout
27 model.add(Dropout(0.5))
28
29
30 # Add 3rd hidden Layer : dense Layer
31 dense_layer3 = Dense(1750,
32                      activation="relu",
33                      kernel_initializer= he_normal(seed=None))
34 model.add(dense_layer3)
```

```
35
36 # Add batch normalization
37 model.add(BatchNormalization())
38
39 # Add dropout
40 model.add(Dropout(0.5))
41
42 # Add 4th hidden layer : dense Layer
43 dense_layer4 = Dense(1500,
44                     activation="relu",
45                     kernel_initializer= he_normal(seed=None))
46 model.add(dense_layer4)
47
48 # Add batch normalization
49 model.add(BatchNormalization())
50
51 # Add dropout
52 model.add(Dropout(0.5))
53
54 # Add 5th hidden layer : dense Layer
55 dense_layer5 = Dense(1250,
56                     activation="relu",
57                     kernel_initializer= he_normal(seed=None))
58 model.add(dense_layer5)
59
60 # Add batch normalization
61 model.add(BatchNormalization())
62
63 # Add dropout
64 model.add(Dropout(0.5))
65
66 # Add 6th hidden layer : dense Layer
67 dense_layer6 = Dense(1000,
68                     activation="relu",
69                     kernel_initializer= he_normal(seed=None))
```

```
70 model.add(dense_layer6)
71
72 # Add batch normalization
73 model.add(BatchNormalization())
74
75 # Add dropout
76 model.add(Dropout(0.5))
77
78 # Add 7th hidden layer : dense Layer
79 dense_layer7 = Dense(750,
80                     activation="relu",
81                     kernel_initializer= he_normal(seed=None))
82 model.add(dense_layer7)
83
84 # Add batch normalization
85 model.add(BatchNormalization())
86
87 # Add dropout
88 model.add(Dropout(0.5))
89
90 # Add 8th hidden layer : dense Layer
91 dense_layer8 = Dense(500,
92                     activation="relu",
93                     kernel_initializer= he_normal(seed=None))
94 model.add(dense_layer8)
95
96 # Add batch normalization
97 model.add(BatchNormalization())
98
99 # Add dropout
100 model.add(Dropout(0.5))
101
102 # Add 9th hidden layer : dense Layer
103 dense_layer9 = Dense(250,
104                     activation="relu",
```

```

105         kernel_initializer= he_normal(seed=None))
106 model.add(dense_layer9)
107
108 # Add batch normalization
109 model.add(BatchNormalization())
110
111 # Add dropout
112 model.add(Dropout(0.5))
113
114 # Add output Layer : dense Layer
115 dense_layer10 = Dense(output_dimension, activation='softmax')
116 model.add(dense_layer10)
117
118 # Summary of the model
119 print("Model Summary: \n")
120 model.summary()
121 print()
122 print()
123
124 # Compile the model
125 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
126
127 # Run the model
128 trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, valida

```

batch_normalization_20 (Batch Normalization)	(None, 1750)	7000
dropout_20 (Dropout)	(None, 1750)	0
dense_25 (Dense)	(None, 1500)	2626500
batch_normalization_21 (Batch Normalization)	(None, 1500)	6000
dropout_21 (Dropout)	(None, 1500)	0
dense_26 (Dense)	(None, 1250)	1876250
batch_normalization_22 (Batch Normalization)	(None, 1250)	5000

batch_normalization_22 (Batch Normalization)	(None, 1250)	5000
dropout_22 (Dropout)	(None, 1250)	0
dense_27 (Dense)	(None, 1000)	1251000
batch_normalization_23 (Batch Normalization)	(None, 1000)	4000
dropout_23 (Dropout)	(None, 1000)	0

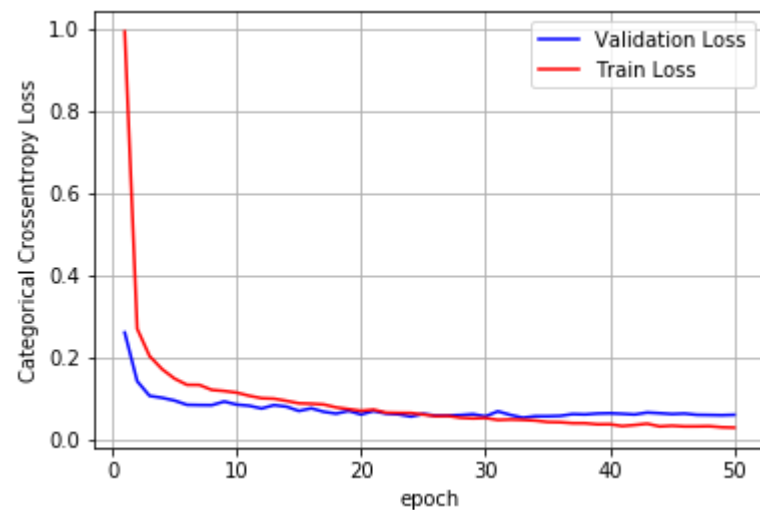
```

1 # Accuracy and plotting train & validation loss
2 score = model.evaluate(x_test, y_test, verbose=0)
3 print('Test score:', score[0])
4 print('Test accuracy: {:.2f}%'.format(score[1]*100))
5
6
7 # Plot train and cross validation error
8 plot_train_cv_loss(trained_model, epochs)

```

Test score: 0.06052361094756052

Test accuracy: 98.62%



Around 18 Epoch train error and validation error meets.

Summary

```
1 print("Activation function= Relu")
2 print("Epochs= 50")
3 print("Batch size= 128")
4 print("Dropout = 0.5")
5 print("Batch Normalization")
6
7 from prettytable import PrettyTable
8 x=PrettyTable()
9 x.field_names = ["Hidden layers","Test accuracy","Optimal_epochs"]
10 x.add_row(["3 Hidden layers","98.61%","17"])
11 x.add_row(["5 Hidden layers","98.68%","20"])
12 x.add_row(["7 Hidden layers","98.70%","22"])
13 x.add_row(["9 Hidden layers","98.62%","18"])
14
15 print(x)
```

Activation function= Relu

Epochs= 50

Batch size= 128

Dropout = 0.5

Batch Normalization

Hidden layers	Test accuracy	Optimal_epochs
3 Hidden layers	98.61%	17
5 Hidden layers	98.68%	20
7 Hidden layers	98.70%	22
9 Hidden layers	98.62%	18