solution06.1

November 10, 2020

Exercise Sheet 6.1 **Convolutional neural networks**

[10]:
```
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import optimizers
import matplotlib.pyplot as plt
import keras


class LogHistory(keras.callbacks.Callback):

    def on_train_begin(self, logs={}):
        self.acc = []
        self.loss = []

    def on_batch_end(self, batch, logs={}):
        self.acc.append(logs.get('acc'))
        self.loss.append(logs.get('loss'))


def main():
    # parameters
    batch_size = 100
    epochs = 17
    learning_rate = 0.5

    # loading the data from local directory
    f = np.load('mnist.npz')
    x_train, y_train = f['x_train'], f['y_train']
    x_valid, y_valid = f['x_test'], f['y_test']
    f.close()

    # number of training and test (validation) samples
    n_train = x_train.shape[0]
    n_valid = x_valid.shape[0]
```

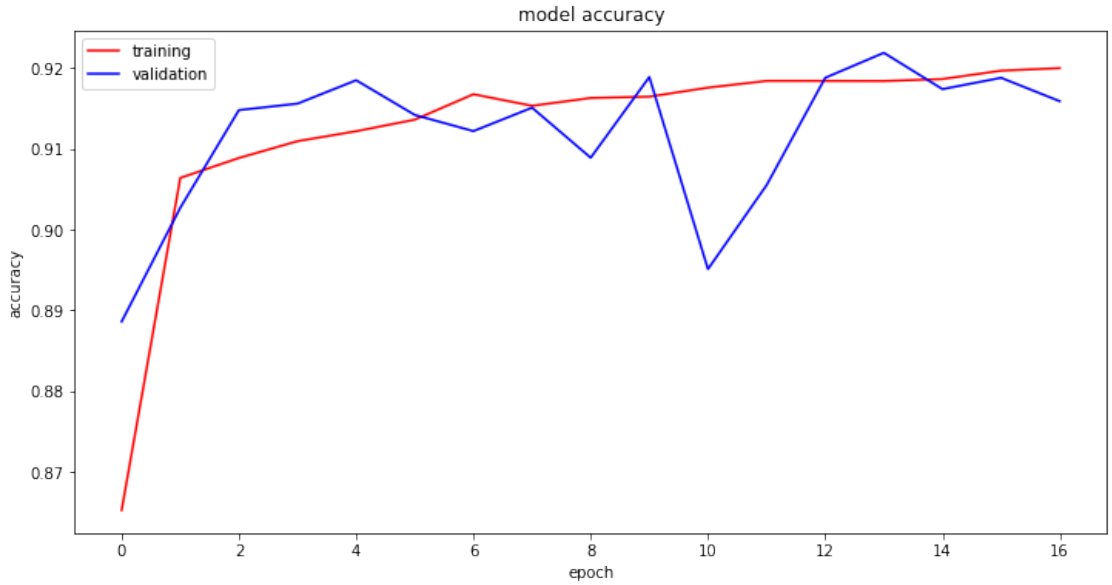
```
    # reshaping the input from [60000 28 28] to [60000 784]
    x_train = x_train.reshape(x_train.shape[0], 28 * 28)
    x_valid = x_valid.reshape(x_valid.shape[0], 28 * 28)

    # scaling the input to [0,1]
    x_train_scaled = x_train/255
    x_valid_scaled = x_valid/255

    # one-hot-encoding the labels
    y_train = np_utils.to_categorical(y_train, 10)
    y_valid = np_utils.to_categorical(y_valid, 10)

    # Call back function
    log_history = LogHistory()

    # defining the model
    model = Sequential()

    model.add(layers.Dense(units=10,
                           activation='linear',
                           use_bias=True,
                           kernel_initializer='zeros',
                           bias_initializer='zeros',
                           input_shape=(28 * 28,)))

    model.add(layers.Dense(units=10,
                           activation='softmax'))

    # compiling
    sgd = optimizers.SGD(lr=learning_rate)

    model.compile(loss='categorical_crossentropy',
                  optimizer=sgd,
                  metrics=['accuracy'])

    # training
    training_log = model.fit(x_train_scaled, y_train,
                             batch_size=batch_size,
                             epochs=epochs,
                             validation_data=(x_valid_scaled, y_valid),
                             callbacks=[log_history])

    # final evaluation
    score, acc = model.evaluate(x_valid_scaled, y_valid, batch_size=n_valid)
    print('final model accuracy over the validation set is %.1f%%' % (100 *␣
↪acc))
```

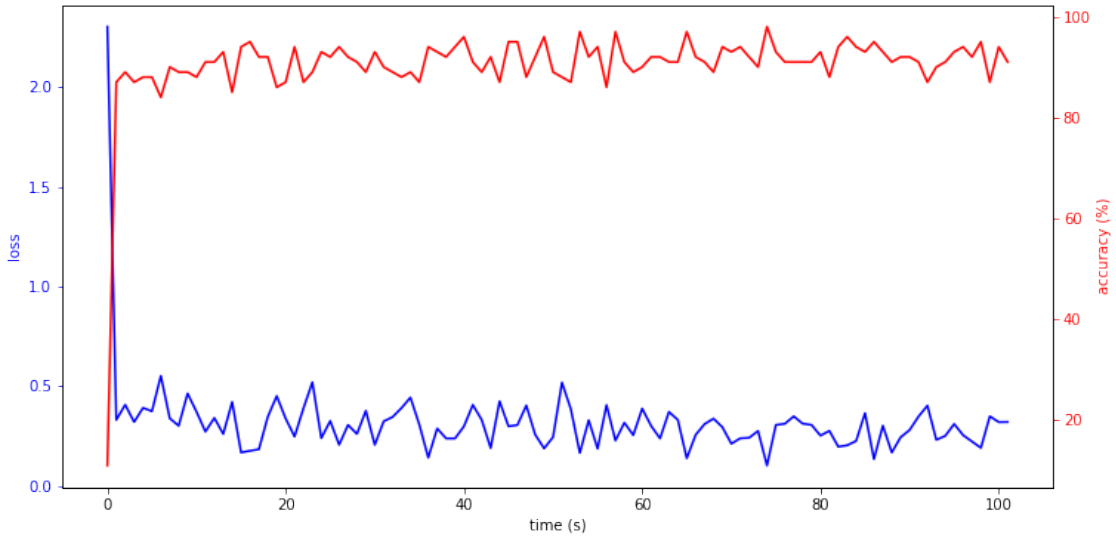
```
    # plotting history for accuracy
    plt.figure(figsize=(12,6))
    plt.plot(training_log.history['acc'], color='r', label='training')
    plt.plot(training_log.history['val_acc'], color='b', label='validation')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.show()

    # plotting history for accuracy 100th iteration
    np_loss = np.array(log_history.loss)[0::100]
    np_acc = np.array(log_history.acc)[0::100]

    fig, ax1 = plt.subplots(figsize=(12,6))
    ax1.plot(np.arange(len(np_loss)), np_loss, color='b', label='loss')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel('loss', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(np.arange(len(np_acc)),100*np_acc, color='r', label='accuracy')
    ax2.set_ylabel('accuracy (%)', color='r')
    ax2.tick_params('y', colors='r')
    plt.show()
```

[11]:
```
main()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/17
60000/60000 [==============================] - 1s 15us/step - loss: 0.4655 -
acc: 0.8652 - val_loss: 0.3794 - val_acc: 0.8886
Epoch 2/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.3290 -
acc: 0.9064 - val_loss: 0.3225 - val_acc: 0.9027
Epoch 3/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.3193 -
acc: 0.9089 - val_loss: 0.2957 - val_acc: 0.9148
Epoch 4/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.3128 -
acc: 0.9110 - val_loss: 0.3015 - val_acc: 0.9156
Epoch 5/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.3097 -
acc: 0.9122 - val_loss: 0.2811 - val_acc: 0.9185
Epoch 6/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.3043 -
acc: 0.9136 - val_loss: 0.3070 - val_acc: 0.9142
Epoch 7/17
```

```
60000/60000 [==============================] - 1s 13us/step - loss: 0.2988 -
acc: 0.9168 - val_loss: 0.3103 - val_acc: 0.9122
Epoch 8/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2968 -
acc: 0.9153 - val_loss: 0.3067 - val_acc: 0.9151
Epoch 9/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2960 -
acc: 0.9163 - val_loss: 0.3230 - val_acc: 0.9089
Epoch 10/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2936 -
acc: 0.9165 - val_loss: 0.2936 - val_acc: 0.9189
Epoch 11/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2929 -
acc: 0.9176 - val_loss: 0.3588 - val_acc: 0.8951
Epoch 12/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2913 -
acc: 0.9184 - val_loss: 0.3301 - val_acc: 0.9055
Epoch 13/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2883 -
acc: 0.9184 - val_loss: 0.2957 - val_acc: 0.9188
Epoch 14/17
60000/60000 [==============================] - 1s 14us/step - loss: 0.2892 -
acc: 0.9184 - val_loss: 0.2839 - val_acc: 0.9219
Epoch 15/17
60000/60000 [==============================] - 1s 13us/step - loss: 0.2877 -
acc: 0.9187 - val_loss: 0.3026 - val_acc: 0.9174
Epoch 16/17
60000/60000 [==============================] - 1s 14us/step - loss: 0.2865 -
acc: 0.9197 - val_loss: 0.2962 - val_acc: 0.9188
Epoch 17/17
60000/60000 [==============================] - 1s 14us/step - loss: 0.2838 -
acc: 0.9200 - val_loss: 0.3114 - val_acc: 0.9159
10000/10000 [==============================] - 0s 6us/step
final model accuracy over the validation set is 91.6%
```





[13]:
```
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
import matplotlib.pyplot as plt
import keras

class LogHistory(keras.callbacks.Callback):
```

```
    def on_train_begin(self, logs={}):
        self.acc = []
        self.loss = []

    def on_batch_end(self, batch, logs={}):
        self.acc.append(logs.get('acc'))
        self.loss.append(logs.get('loss'))

def main():
    # parameters
    batch_size = 100
    epochs = 34
    learning_rate = 0.001
    Beta1 = 0.9
    Beta2 = 0.999
    epsilon = 1e-8

    # loading the data from local directory
    f = np.load('mnist.npz')
    x_train, y_train = f['x_train'], f['y_train']
    x_valid, y_valid = f['x_test'], f['y_test']
    f.close()

    # number of training and test (validation) samples
    n_train = x_train.shape[0]
    n_valid = x_valid.shape[0]

    # reshaping the input from [60000 28 28] to [60000 784]
    x_train = x_train.reshape(x_train.shape[0], 28*28)
    x_valid = x_valid.reshape(x_valid.shape[0], 28*28)

    # scaling the input to [0,1]
    x_train = x_train/255
    x_valid = x_valid/255

    # one-hot-encoding the labels
    y_train = np_utils.to_categorical(y_train, 10)
    y_valid = np_utils.to_categorical(y_valid, 10)

    # Call back function
    log_history = LogHistory()

    # initialization
    const_init = keras.initializers.Constant(value=0.1)
    trunc_init = keras.initializers.TruncatedNormal(mean=0.0, stddev=0.01)

    # defining the model
```

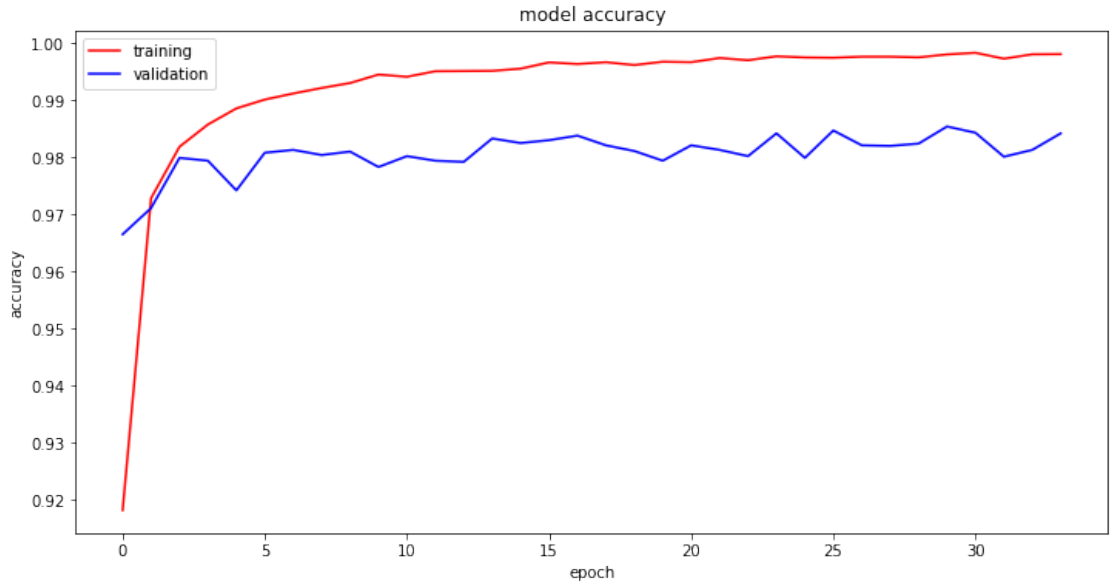
```
model = Sequential()
model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init,
                       input_shape=(28*28,)))

model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init))

model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init))

model.add(layers.Dense(units=10,
                       activation='softmax'))

# compiling
adam = keras.optimizers.Adam(lr=learning_rate,
                             beta_1=Beta1,
                             beta_2=Beta2,
                             epsilon=epsilon)

model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])

# training
training_log = model.fit(x_train, y_train,
                         batch_size=batch_size,
                         epochs=epochs,
                         validation_data=(x_valid, y_valid),
                         callbacks=[log_history])

# final evaluation
score, acc = model.evaluate(x_valid, y_valid, batch_size=n_valid)
print('final model accuracy over the validation set is %.1f%%' % (100*acc))

# plotting history for accuracy
plt.figure(figsize=(12,6))
plt.plot(training_log.history['acc'], color='r', label='training')
```

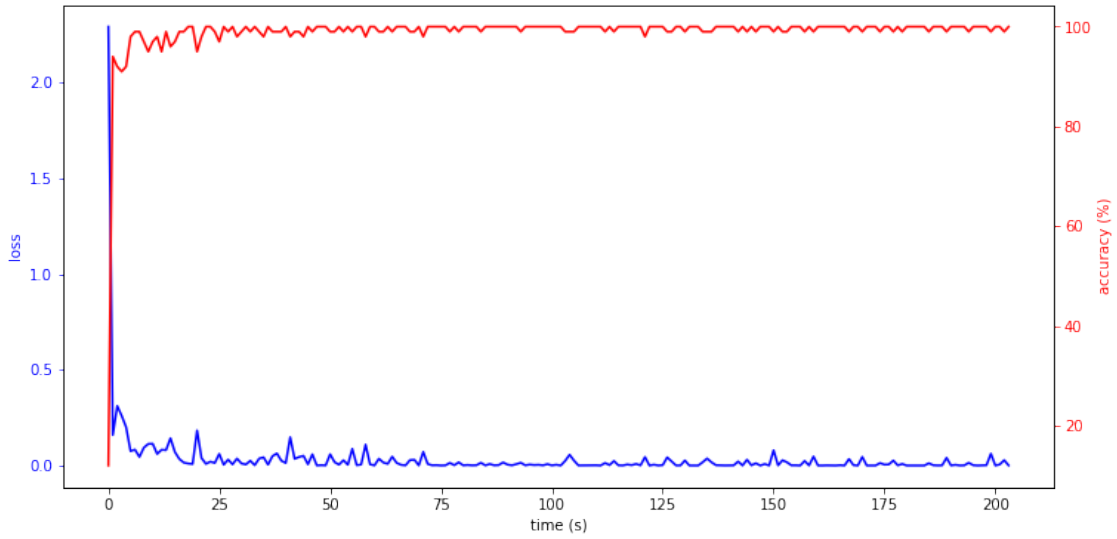
```
    plt.plot(training_log.history['val_acc'], color='b', label='validation')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.show()

    # plotting history for accuracy 100th iteration
    np_loss = np.array(log_history.loss)[0::100]
    np_acc = np.array(log_history.acc)[0::100]

    fig, ax1 = plt.subplots(figsize=(12,6))
    ax1.plot(np.arange(len(np_loss)), np_loss, color='b', label='loss')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel('loss', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(np.arange(len(np_acc)), 100 * np_acc, color='r', label='accuracy')
    ax2.set_ylabel('accuracy (%)', color='r')
    ax2.tick_params('y', colors='r')
    plt.show()

main()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.2673 -
acc: 0.9182 - val_loss: 0.1059 - val_acc: 0.9665
Epoch 2/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0893 -
acc: 0.9728 - val_loss: 0.0924 - val_acc: 0.9711
Epoch 3/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0600 -
acc: 0.9819 - val_loss: 0.0686 - val_acc: 0.9799
Epoch 4/34
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0457 -
acc: 0.9858 - val_loss: 0.0691 - val_acc: 0.9794
Epoch 5/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0371 -
acc: 0.9886 - val_loss: 0.0890 - val_acc: 0.9742
Epoch 6/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0304 -
acc: 0.9901 - val_loss: 0.0748 - val_acc: 0.9808
Epoch 7/34
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0283 -
acc: 0.9912 - val_loss: 0.0725 - val_acc: 0.9813
Epoch 8/34
```

```
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0261 -
acc: 0.9922 - val_loss: 0.0764 - val_acc: 0.9804
Epoch 9/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0213 -
acc: 0.9930 - val_loss: 0.0783 - val_acc: 0.9810
Epoch 10/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0184 -
acc: 0.9945 - val_loss: 0.0948 - val_acc: 0.9783
Epoch 11/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0196 -
acc: 0.9941 - val_loss: 0.1026 - val_acc: 0.9802
Epoch 12/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0161 -
acc: 0.9951 - val_loss: 0.0977 - val_acc: 0.9794
Epoch 13/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0163 -
acc: 0.9951 - val_loss: 0.0915 - val_acc: 0.9792
Epoch 14/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0161 -
acc: 0.9952 - val_loss: 0.0822 - val_acc: 0.9833
Epoch 15/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0159 -
acc: 0.9955 - val_loss: 0.0911 - val_acc: 0.9825
Epoch 16/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0122 -
acc: 0.9966 - val_loss: 0.0822 - val_acc: 0.9830
Epoch 17/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0131 -
acc: 0.9964 - val_loss: 0.0942 - val_acc: 0.9838
Epoch 18/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0116 -
acc: 0.9967 - val_loss: 0.0943 - val_acc: 0.9821
Epoch 19/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0139 -
acc: 0.9962 - val_loss: 0.0881 - val_acc: 0.9811
Epoch 20/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0123 -
acc: 0.9968 - val_loss: 0.1083 - val_acc: 0.9794
Epoch 21/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0117 -
acc: 0.9967 - val_loss: 0.0992 - val_acc: 0.9821
Epoch 22/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0108 -
acc: 0.9974 - val_loss: 0.0984 - val_acc: 0.9813
Epoch 23/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0097 -
acc: 0.9970 - val_loss: 0.1119 - val_acc: 0.9802
Epoch 24/34
```

```
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0090 -
acc: 0.9977 - val_loss: 0.0878 - val_acc: 0.9842
Epoch 25/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0096 -
acc: 0.9975 - val_loss: 0.1071 - val_acc: 0.9799
Epoch 26/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0098 -
acc: 0.9975 - val_loss: 0.0863 - val_acc: 0.9847
Epoch 27/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0082 -
acc: 0.9976 - val_loss: 0.0981 - val_acc: 0.9821
Epoch 28/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0091 -
acc: 0.9976 - val_loss: 0.0872 - val_acc: 0.9820
Epoch 29/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0091 -
acc: 0.9975 - val_loss: 0.1026 - val_acc: 0.9824
Epoch 30/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0079 -
acc: 0.9980 - val_loss: 0.0882 - val_acc: 0.9854
Epoch 31/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0070 -
acc: 0.9983 - val_loss: 0.0925 - val_acc: 0.9843
Epoch 32/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0104 -
acc: 0.9973 - val_loss: 0.1106 - val_acc: 0.9801
Epoch 33/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0079 -
acc: 0.9981 - val_loss: 0.0968 - val_acc: 0.9813
Epoch 34/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0071 -
acc: 0.9981 - val_loss: 0.0977 - val_acc: 0.9842
10000/10000 [==============================] - 1s 96us/step
final model accuracy over the validation set is 98.4%
```





[15]:
```
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
import matplotlib.pyplot as plt
import keras


class LogHistory(keras.callbacks.Callback):
```

```
    def on_train_begin(self, logs={}):
        self.acc = []
        self.loss = []

    def on_batch_end(self, batch, logs={}):
        self.acc.append(logs.get('acc'))
        self.loss.append(logs.get('loss'))


def main():
    # parameters
    batch_size = 100
    epochs = 34
    learning_rate = 0.001
    Beta1 = 0.9
    Beta2 = 0.999
    epsilon = 1e-8

    # loading the data from local directory
    f = np.load('mnist.npz')
    x_train, y_train = f['x_train'], f['y_train']
    x_valid, y_valid = f['x_test'], f['y_test']
    f.close()

    # number of training and test (validation) samples
    n_train = x_train.shape[0]
    n_valid = x_valid.shape[0]

    # reshaping the input from [60000 28 28] to [60000 784]
    x_train = x_train.reshape(x_train.shape[0], 28*28)
    x_valid = x_valid.reshape(x_valid.shape[0], 28*28)

    # scaling the input to [0,1]
    x_train = x_train/255
    x_valid = x_valid/255

    # one-hot-encoding the labels
    y_train = np_utils.to_categorical(y_train, 10)
    y_valid = np_utils.to_categorical(y_valid, 10)

    # initialization
    const_init = keras.initializers.Constant(value=0.1)
    trunc_init = keras.initializers.TruncatedNormal(mean=0.0, stddev=0.01)

    # Call back function
    log_history = LogHistory()
```

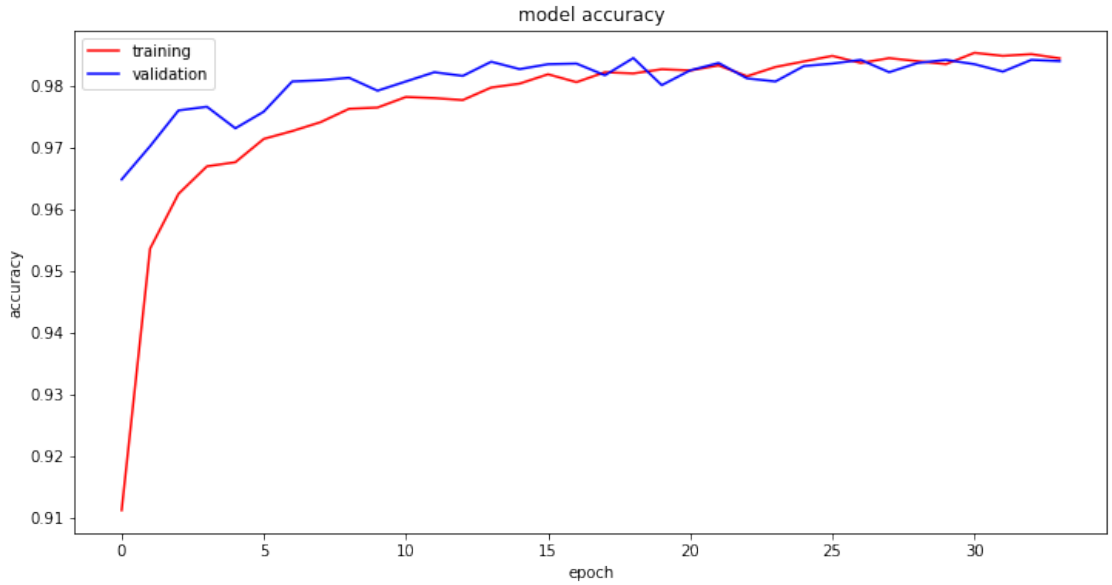
```
# defining the model
model = Sequential()
model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init,
                       input_shape=(28*28,)))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(units=1500,
                       activation='relu',
                       use_bias=True,
                       kernel_initializer=trunc_init,
                       bias_initializer=const_init))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(units=10,
                       activation='softmax'))

# compiling
adam = keras.optimizers.Adam(lr=learning_rate,
                             beta_1=Beta1,
                             beta_2=Beta2,
                             epsilon=epsilon)

model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])

# training
training_log = model.fit(x_train, y_train,
                         batch_size=batch_size,
                         epochs=epochs,
                         validation_data=(x_valid, y_valid),
                         callbacks=[log_history])

# final evaluation
score, acc = model.evaluate(x_valid, y_valid, batch_size=n_valid)
```

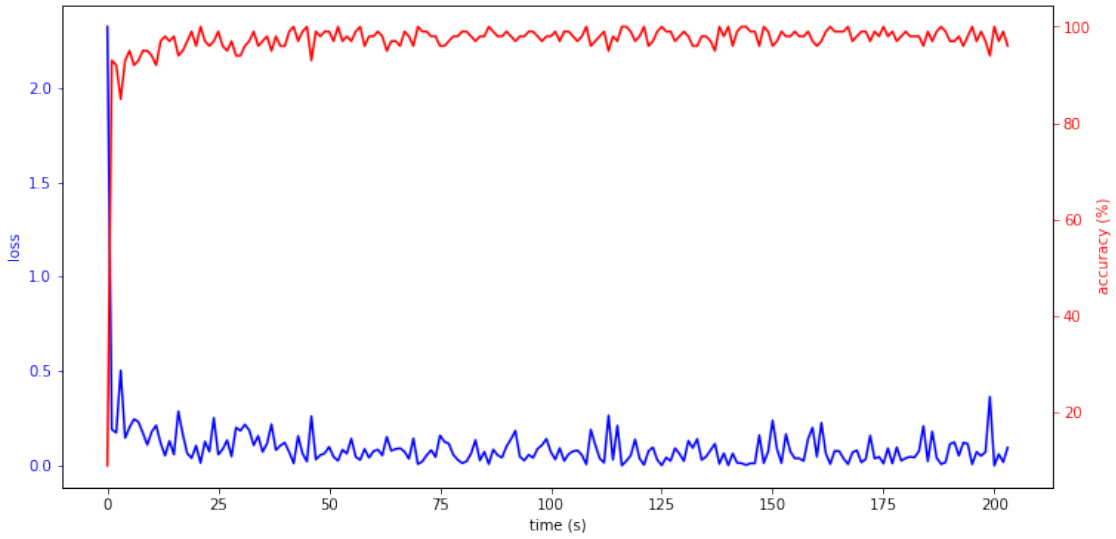
```
    print('final model accuracy over the validation set is %.1f%%' % (100*acc))

    # plotting history for accuracy
    plt.figure(figsize=(12,6))
    plt.plot(training_log.history['acc'], color='r', label='training')
    plt.plot(training_log.history['val_acc'], color='b', label='validation')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.show()

    # plotting history for accuracy 100th iteration
    np_loss = np.array(log_history.loss)[0::100]
    np_acc = np.array(log_history.acc)[0::100]

    fig, ax1 = plt.subplots(figsize=(12,6))
    ax1.plot(np.arange(len(np_loss)), np_loss, color='b', label='loss')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel('loss', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(np.arange(len(np_acc)), 100 * np_acc, color='r', label='accuracy')
    ax2.set_ylabel('accuracy (%)', color='r')
    ax2.tick_params('y', colors='r')
    plt.show()

main()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.2871 -
acc: 0.9112 - val_loss: 0.1134 - val_acc: 0.9648
Epoch 2/34
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1542 -
acc: 0.9536 - val_loss: 0.0998 - val_acc: 0.9702
Epoch 3/34
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1312 -
acc: 0.9625 - val_loss: 0.0869 - val_acc: 0.9760
Epoch 4/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.1144 -
acc: 0.9669 - val_loss: 0.0806 - val_acc: 0.9766
Epoch 5/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.1099 -
acc: 0.9676 - val_loss: 0.0944 - val_acc: 0.9731
Epoch 6/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0991 -
```

```
acc: 0.9714 - val_loss: 0.0793 - val_acc: 0.9758
Epoch 7/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0933 -
acc: 0.9727 - val_loss: 0.0699 - val_acc: 0.9807
Epoch 8/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0909 -
acc: 0.9741 - val_loss: 0.0714 - val_acc: 0.9809
Epoch 9/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0836 -
acc: 0.9763 - val_loss: 0.0686 - val_acc: 0.9813
Epoch 10/34
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0828 -
acc: 0.9765 - val_loss: 0.0773 - val_acc: 0.9792
Epoch 11/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0790 -
acc: 0.9782 - val_loss: 0.0842 - val_acc: 0.9807
Epoch 12/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0790 -
acc: 0.9780 - val_loss: 0.0726 - val_acc: 0.9822
Epoch 13/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0803 -
acc: 0.9777 - val_loss: 0.0744 - val_acc: 0.9816
Epoch 14/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0726 -
acc: 0.9797 - val_loss: 0.0668 - val_acc: 0.9839
Epoch 15/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0731 -
acc: 0.9804 - val_loss: 0.0732 - val_acc: 0.9827
Epoch 16/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0660 -
acc: 0.9819 - val_loss: 0.0798 - val_acc: 0.9835
Epoch 17/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0734 -
acc: 0.9806 - val_loss: 0.0767 - val_acc: 0.9836
Epoch 18/34
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0698 -
acc: 0.9822 - val_loss: 0.0940 - val_acc: 0.9817
Epoch 19/34
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0661 -
acc: 0.9820 - val_loss: 0.0755 - val_acc: 0.9845
Epoch 20/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0688 -
acc: 0.9827 - val_loss: 0.0871 - val_acc: 0.9801
Epoch 21/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0689 -
acc: 0.9825 - val_loss: 0.0787 - val_acc: 0.9825
Epoch 22/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0641 -
```

```
acc: 0.9833 - val_loss: 0.0838 - val_acc: 0.9837
Epoch 23/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0746 -
acc: 0.9816 - val_loss: 0.0991 - val_acc: 0.9812
Epoch 24/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0640 -
acc: 0.9831 - val_loss: 0.0861 - val_acc: 0.9807
Epoch 25/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0620 -
acc: 0.9840 - val_loss: 0.0784 - val_acc: 0.9832
Epoch 26/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0636 -
acc: 0.9849 - val_loss: 0.0755 - val_acc: 0.9836
Epoch 27/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0657 -
acc: 0.9837 - val_loss: 0.0748 - val_acc: 0.9842
Epoch 28/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0689 -
acc: 0.9845 - val_loss: 0.0816 - val_acc: 0.9822
Epoch 29/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0675 -
acc: 0.9840 - val_loss: 0.0784 - val_acc: 0.9837
Epoch 30/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0702 -
acc: 0.9835 - val_loss: 0.0770 - val_acc: 0.9842
Epoch 31/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0614 -
acc: 0.9854 - val_loss: 0.0828 - val_acc: 0.9835
Epoch 32/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0641 -
acc: 0.9849 - val_loss: 0.0863 - val_acc: 0.9823
Epoch 33/34
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0640 -
acc: 0.9851 - val_loss: 0.0788 - val_acc: 0.9842
Epoch 34/34
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0661 -
acc: 0.9845 - val_loss: 0.0852 - val_acc: 0.9840
10000/10000 [==============================] - 1s 94us/step
final model accuracy over the validation set is 98.4%
```





[16]:
```
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import initializers
from keras import optimizers
import matplotlib.pyplot as plt
from keras.callbacks import Callback
```

```
class LogHistory(Callback):

    def on_train_begin(self, logs={}):
        self.acc = []
        self.loss = []

    def on_batch_end(self, batch, logs={}):
        self.acc.append(logs.get('acc'))
        self.loss.append(logs.get('loss'))


def main():
    # parameters
    batch_size = 100
    epochs = 34
    learning_rate = 0.001
    Beta1 = 0.9
    Beta2 = 0.999
    epsilon = 1e-8

    # loading the data from local directory
    f = np.load('mnist.npz')
    x_train, y_train = f['x_train'], f['y_train']
    x_valid, y_valid = f['x_test'], f['y_test']
    f.close()

    # number of training and test (validation) samples
    n_train = x_train.shape[0]
    n_valid = x_valid.shape[0]

    # reshaping the input from [60000 28 28] to [60000 28 28 1]
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    x_valid = x_valid.reshape(x_valid.shape[0], 28, 28, 1)

    # scaling the input to [0,1]
    x_train = x_train/255
    x_valid = x_valid/255

    # one-hot-encoding the labels
    y_train = np_utils.to_categorical(y_train, 10)
    y_valid = np_utils.to_categorical(y_valid, 10)

    # Call back function
    log_history = LogHistory()

    # initialization
```

```
const_init = initializers.Constant(value=0.1)
trunc_init = initializers.TruncatedNormal(mean=0.0, stddev=0.01)

# defining the model
model = Sequential()
model.add(layers.Conv2D(filters=32,
                        kernel_size=(5, 5),
                        strides=(1, 1),
                        padding='same',
                        use_bias=True,
                        kernel_initializer=trunc_init,
                        bias_initializer=const_init,
                        activation='relu',
                        input_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D(pool_size=(2,2),
                              strides=(2,2),
                              padding='same'))

model.add(layers.Conv2D(filters=64,
                        kernel_size=(5, 5),
                        strides=(1, 1),
                        padding='same',
                        use_bias=True,
                        kernel_initializer=trunc_init,
                        bias_initializer=const_init,
                        activation='relu'))

model.add(layers.MaxPooling2D(pool_size=(2,2),
                              strides=(2,2),
                              padding='same'))

model.add(layers.Flatten())

model.add(layers.Dense(units=10,
                       activation='softmax'))

# compiling
adam = optimizers.Adam(lr=learning_rate,
                       beta_1=Beta1,
                       beta_2=Beta2,
                       epsilon=epsilon)

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

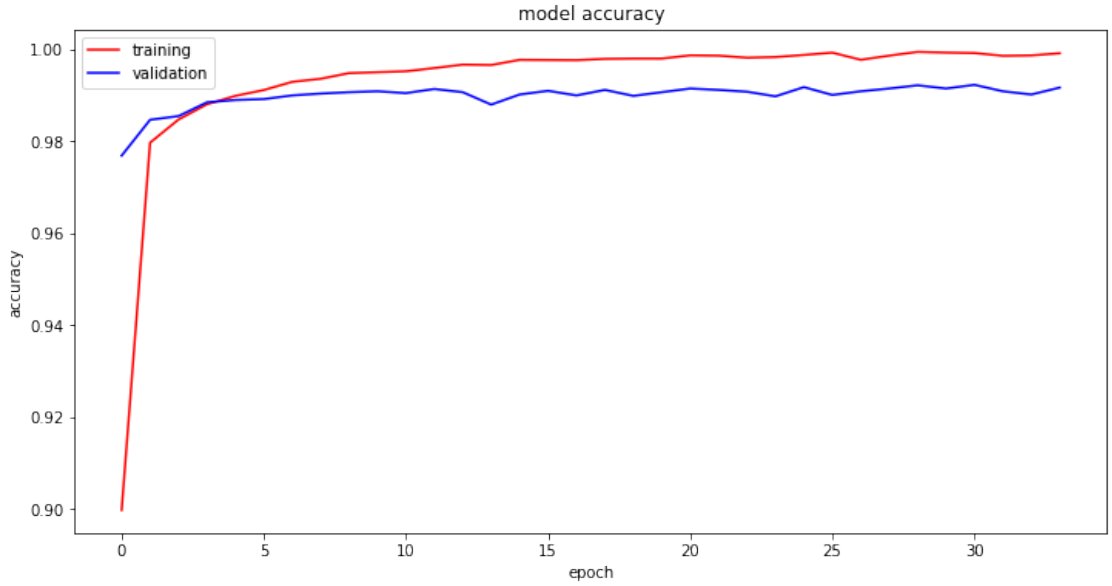
```
    # training
    training_log = model.fit(x_train, y_train,
                             batch_size=batch_size,
                             epochs=epochs,
                             validation_data=(x_valid, y_valid),
                             callbacks=[log_history])

    # final evaluation
    score, acc = model.evaluate(x_valid, y_valid, batch_size=n_valid)
    print('final model accuracy over the validation set is %.1f%%' % (100*acc))


    # plotting history for accuracy
    plt.figure(figsize=(12,6))
    plt.plot(training_log.history['acc'], color='r', label='training')
    plt.plot(training_log.history['val_acc'], color='b', label='validation')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.show()

    # plotting history for accuracy 100th iteration
    np_loss = np.array(log_history.loss)[0::100]
    np_acc = np.array(log_history.acc)[0::100]

    fig, ax1 = plt.subplots(figsize=(12,6))
    ax1.plot(np.arange(len(np_loss)), np_loss, color='b', label='loss')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel('loss', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(np.arange(len(np_acc)), 100 * np_acc, color='r', label='accuracy')
    ax2.set_ylabel('accuracy (%)', color='r')
    ax2.tick_params('y', colors='r')
    plt.show()

main()
```

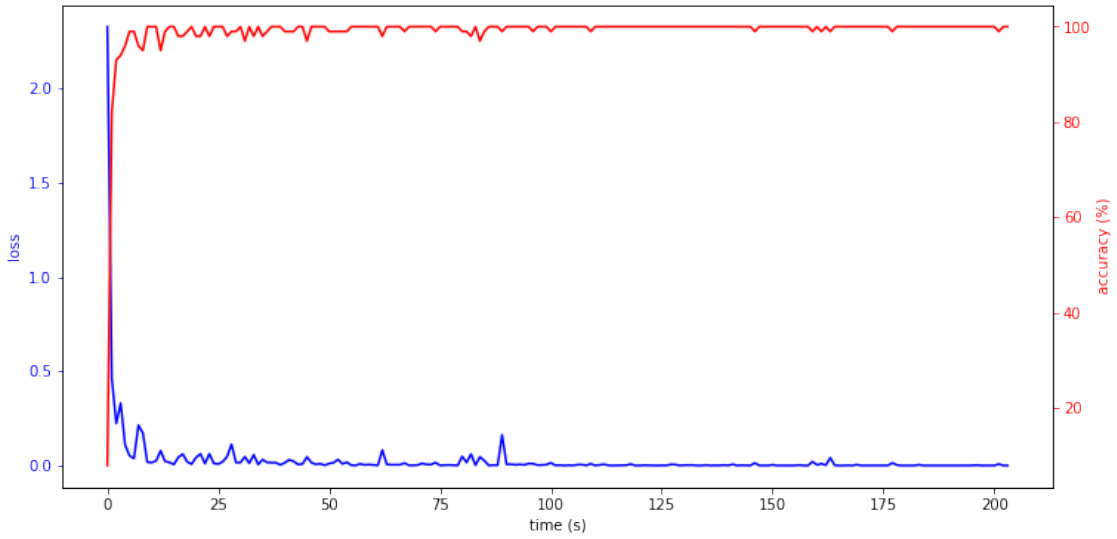
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.3246 -
acc: 0.8998 - val_loss: 0.0746 - val_acc: 0.9769
Epoch 2/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0685 -
acc: 0.9797 - val_loss: 0.0452 - val_acc: 0.9847
Epoch 3/34
```

```
60000/60000 [==============================] - 82s 1ms/step - loss: 0.0501 -
acc: 0.9848 - val_loss: 0.0442 - val_acc: 0.9855
Epoch 4/34
60000/60000 [==============================] - 82s 1ms/step - loss: 0.0390 -
acc: 0.9881 - val_loss: 0.0333 - val_acc: 0.9885
Epoch 5/34
60000/60000 [==============================] - 82s 1ms/step - loss: 0.0317 -
acc: 0.9899 - val_loss: 0.0381 - val_acc: 0.9890
Epoch 6/34
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0272 -
acc: 0.9912 - val_loss: 0.0331 - val_acc: 0.9892
Epoch 7/34
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0229 -
acc: 0.9930 - val_loss: 0.0291 - val_acc: 0.9900
Epoch 8/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0203 -
acc: 0.9936 - val_loss: 0.0304 - val_acc: 0.9904
Epoch 9/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0172 -
acc: 0.9949 - val_loss: 0.0298 - val_acc: 0.9907
Epoch 10/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0151 -
acc: 0.9951 - val_loss: 0.0286 - val_acc: 0.9909
Epoch 11/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0140 -
acc: 0.9953 - val_loss: 0.0280 - val_acc: 0.9905
Epoch 12/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0120 -
acc: 0.9960 - val_loss: 0.0287 - val_acc: 0.9914
Epoch 13/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0102 -
acc: 0.9967 - val_loss: 0.0314 - val_acc: 0.9907
Epoch 14/34
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0108 -
acc: 0.9966 - val_loss: 0.0403 - val_acc: 0.9880
Epoch 15/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0072 -
acc: 0.9977 - val_loss: 0.0323 - val_acc: 0.9902
Epoch 16/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0070 -
acc: 0.9977 - val_loss: 0.0351 - val_acc: 0.9910
Epoch 17/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0068 -
acc: 0.9977 - val_loss: 0.0401 - val_acc: 0.9900
Epoch 18/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0061 -
acc: 0.9980 - val_loss: 0.0333 - val_acc: 0.9912
Epoch 19/34
```

```
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0057 -
acc: 0.9980 - val_loss: 0.0427 - val_acc: 0.9899
Epoch 20/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0056 -
acc: 0.9980 - val_loss: 0.0351 - val_acc: 0.9907
Epoch 21/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0044 -
acc: 0.9987 - val_loss: 0.0392 - val_acc: 0.9915
Epoch 22/34
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0042 -
acc: 0.9987 - val_loss: 0.0396 - val_acc: 0.9912
Epoch 23/34
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0053 -
acc: 0.9982 - val_loss: 0.0429 - val_acc: 0.9908
Epoch 24/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0050 -
acc: 0.9984 - val_loss: 0.0472 - val_acc: 0.9898
Epoch 25/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0035 -
acc: 0.9988 - val_loss: 0.0392 - val_acc: 0.9918
Epoch 26/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0022 -
acc: 0.9993 - val_loss: 0.0441 - val_acc: 0.9901
Epoch 27/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0064 -
acc: 0.9978 - val_loss: 0.0435 - val_acc: 0.9909
Epoch 28/34
60000/60000 [==============================] - 85s 1ms/step - loss: 0.0042 -
acc: 0.9987 - val_loss: 0.0384 - val_acc: 0.9915
Epoch 29/34
60000/60000 [==============================] - 85s 1ms/step - loss: 0.0020 -
acc: 0.9995 - val_loss: 0.0372 - val_acc: 0.9922
Epoch 30/34
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0024 -
acc: 0.9993 - val_loss: 0.0433 - val_acc: 0.9915
Epoch 31/34
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0025 -
acc: 0.9992 - val_loss: 0.0367 - val_acc: 0.9923
Epoch 32/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0044 -
acc: 0.9986 - val_loss: 0.0434 - val_acc: 0.9909
Epoch 33/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0038 -
acc: 0.9987 - val_loss: 0.0474 - val_acc: 0.9902
Epoch 34/34
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0023 -
acc: 0.9992 - val_loss: 0.0453 - val_acc: 0.9917
10000/10000 [==============================] - 7s 723us/step
```

```
final model accuracy over the validation set is 99.2%
```





The linear model, seems to be overfitting because the accuracy over the training data is increasing but it is reducing over the validation data. Also, the final accuracy is about 92% for test data and 90% for validation data. For the MLP without dropout (simple), overfitting is not as pronounced as the linear model but the learning rate reduces (saturates) after 10 batch training. The final accuracy over the training data is about 99.8% and 98% over the validation data. Regarding the MLP with dropout, the model does not overfit and the accuracy for both training and validation data leans towards 98% and would keep growing for more itteration. Finally, the convolutional model has marginal overfitting but the accuracy over both training data and validation data us

very high (99.9% for training and 99.2% for validation)

Dropout has better results against overfitting but convolutional network is has higher accuracy.

the number of weights and biases in convolutional network is much much less than the MLP method (even with dropout). By the approximation, the number of parameters to be optimized in the MLP is the sqrt of those in convolutional one.