CNN_MNIST

August 9, 2019

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.layers import Activation, BatchNormalization, regularizers
```

```
Using TensorFlow backend.
```

In [9]:
```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
        print('x_train shape:', x_train.shape)
        print(x_train.shape[0], 'train samples')
        print(x_test.shape[0], 'test samples')

        # convert class vectors to binary class matrices
        y_train = keras.utils.to_categorical(y_train, num_classes)
        y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [3]:
```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3, 3),
                         activation='relu',
                         input_shape=input_shape))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(num_classes, activation='softmax'))

        model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adadelta(),
                      metrics=['accuracy'])

        model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=1,
                  validation_data=(x_test, y_test))
        score = model.evaluate(x_test, y_test, verbose=0)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])
```

```
WARNING: Logging before flag parsing goes to stderr.
W0809 01:32:51.226855 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.320584 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.320584 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.383068 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p
```

```
W0809 01:32:51.383068 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.398689 12008 deprecation.py:506] From C:\Users\ACER\Anaconda3\lib\site-packages\
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0809 01:32:51.523696 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.539281 12008 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 01:32:51.711116 12008 deprecation.py:323] From C:\Users\ACER\Anaconda3\lib\site-packages\
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where


Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 104s 2ms/step - loss: 0.2648 - acc: 0.9184 - val
Epoch 2/12
60000/60000 [==============================] - 107s 2ms/step - loss: 0.0873 - acc: 0.9739 - val
Epoch 3/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0671 - acc: 0.9799 - val
Epoch 4/12
60000/60000 [==============================] - 105s 2ms/step - loss: 0.0552 - acc: 0.9830 - val
Epoch 5/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0464 - acc: 0.9856 - val
Epoch 6/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0428 - acc: 0.9872 - val
Epoch 7/12
60000/60000 [==============================] - 105s 2ms/step - loss: 0.0381 - acc: 0.9888 - val
Epoch 8/12
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0345 - acc: 0.9893 - val_
Epoch 9/12
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0299 - acc: 0.9902 - val_
Epoch 10/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0306 - acc: 0.9908 - val_
Epoch 11/12
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0277 - acc: 0.9915 - val_
Epoch 12/12
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0274 - acc: 0.9917 - val_
Test loss: 0.024807610545822537
Test accuracy: 0.9925
```

0.0.1 Example 1

```
In [10]: model_1 = Sequential()
         model_1.add(Conv2D(32, kernel_size=(2, 2),
```

```
                 activation='relu',
                 input_shape=input_shape))
model_1.add(Conv2D(32, (2, 2), activation='relu'))
model_1.add(MaxPooling2D(pool_size=(3, 3)))
model_1.add(BatchNormalization())
model_1.add(Dropout(0.25))
model_1.add(Flatten())
model_1.add(Dense(111, activation='relu'))
model_1.add(Dense(num_classes, activation='softmax'))

model_1.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model_1.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score_1 = model_1.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score_1[0])
print('Test accuracy:', score_1[1])
```

```
W0809 16:04:03.157577  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 37s 618us/step - loss: 0.1461 - acc: 0.9548 - va
Epoch 2/12
60000/60000 [==============================] - 40s 663us/step - loss: 0.0490 - acc: 0.9848 - va
Epoch 3/12
60000/60000 [==============================] - 39s 656us/step - loss: 0.0345 - acc: 0.9890 - va
Epoch 4/12
60000/60000 [==============================] - 40s 663us/step - loss: 0.0246 - acc: 0.9921 - va
Epoch 5/12
60000/60000 [==============================] - 40s 666us/step - loss: 0.0191 - acc: 0.9940 - va
Epoch 6/12
60000/60000 [==============================] - 39s 649us/step - loss: 0.0162 - acc: 0.9944 - va
Epoch 7/12
60000/60000 [==============================] - 40s 670us/step - loss: 0.0131 - acc: 0.9960 - va
Epoch 8/12
60000/60000 [==============================] - 41s 678us/step - loss: 0.0112 - acc: 0.9964 - va
Epoch 9/12
60000/60000 [==============================] - 39s 654us/step - loss: 0.0096 - acc: 0.9968 - va
Epoch 10/12
60000/60000 [==============================] - 41s 675us/step - loss: 0.0081 - acc: 0.9974 - va
```

```
Epoch 11/12
60000/60000 [==============================] - 40s 672us/step - loss: 0.0070 - acc: 0.9977 - va
Epoch 12/12
60000/60000 [==============================] - 39s 657us/step - loss: 0.0059 - acc: 0.9983 - va
Test loss: 0.03634111389952359
Test accuracy: 0.9902
```

0.0.2 Example 2

```
In [11]: epochs = 10

         model_2 = Sequential()
         model_2.add(Conv2D(32, kernel_size=(5, 5),
                            activation='relu',
                            input_shape=input_shape))
         model_2.add(Conv2D(64, (5, 5), activation='relu'))
         model_2.add(Conv2D(32, (5, 5), activation='relu'))
         model_2.add(MaxPooling2D(pool_size=(3, 3)))
         model_2.add(Flatten())
         model_2.add(Dense(128, activation='relu'))
         model_2.add(Dropout(0.5))
         model_2.add(BatchNormalization())
         model_2.add(Dropout(0.30))
         model_2.add(Dense(num_classes, activation='softmax'))

         model_2.compile(loss=keras.losses.categorical_crossentropy,
                         optimizer=keras.optimizers.Adadelta(),
                         metrics=['accuracy'])

         model_2.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(x_test, y_test))
         score_2 = model_2.evaluate(x_test, y_test, verbose=0)
         print('Test loss:', score_2[0])
         print('Test accuracy:', score_2[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 153s 3ms/step - loss: 0.2875 - acc: 0.9204 - val
Epoch 2/10
60000/60000 [==============================] - 185s 3ms/step - loss: 0.0909 - acc: 0.9750 - val
Epoch 3/10
60000/60000 [==============================] - 189s 3ms/step - loss: 0.0649 - acc: 0.9825 - val
Epoch 4/10
60000/60000 [==============================] - 188s 3ms/step - loss: 0.0532 - acc: 0.9850 - val
```

```
Epoch 5/10
60000/60000 [==============================] - 161s 3ms/step - loss: 0.0447 - acc: 0.9874 - val
Epoch 6/10
60000/60000 [==============================] - 143s 2ms/step - loss: 0.0383 - acc: 0.9892 - val
Epoch 7/10
60000/60000 [==============================] - 145s 2ms/step - loss: 0.0339 - acc: 0.9906 - val
Epoch 8/10
60000/60000 [==============================] - 143s 2ms/step - loss: 0.0323 - acc: 0.9906 - val
Epoch 9/10
60000/60000 [==============================] - 144s 2ms/step - loss: 0.0277 - acc: 0.9921 - val
Epoch 10/10
60000/60000 [==============================] - 145s 2ms/step - loss: 0.0257 - acc: 0.9930 - val
Test loss: 0.015346395371133803
Test accuracy: 0.9957
```

0.0.3 Example 3

```
In [12]: epochs = 11

         model_3 = Sequential()
         model_3.add(Conv2D(64, kernel_size=(6, 6),
                            activation='relu',
                            input_shape=input_shape))
         model_3.add(Conv2D(32, (6, 6), activation='relu'))
         model_3.add(MaxPooling2D(pool_size=(2, 2)))
         model_3.add(Dropout(0.25))
         model_3.add(Flatten())
         model_3.add(Dense(128, activation='relu'))
         model_3.add(Dropout(0.5))
         model_3.add(BatchNormalization())
         model_3.add(Dense(num_classes, activation='softmax'))

         model_3.compile(loss=keras.losses.categorical_crossentropy,
                         optimizer=keras.optimizers.Adam(),
                         metrics=['accuracy'])

         model_3.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(x_test, y_test))
         score_3 = model_3.evaluate(x_test, y_test, verbose=0)
         print('Test loss:', score_3[0])
         print('Test accuracy:', score_3[1])

Train on 60000 samples, validate on 10000 samples
Epoch 1/11
```

```
60000/60000 [==============================] - 133s 2ms/step - loss: 0.2035 - acc: 0.9440 - val
Epoch 2/11
60000/60000 [==============================] - 135s 2ms/step - loss: 0.0680 - acc: 0.9806 - val
Epoch 3/11
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0490 - acc: 0.9860 - val
Epoch 4/11
60000/60000 [==============================] - 128s 2ms/step - loss: 0.0405 - acc: 0.9882 - val
Epoch 5/11
60000/60000 [==============================] - 128s 2ms/step - loss: 0.0361 - acc: 0.9893 - val
Epoch 6/11
60000/60000 [==============================] - 127s 2ms/step - loss: 0.0291 - acc: 0.9915 - val
Epoch 7/11
60000/60000 [==============================] - 127s 2ms/step - loss: 0.0271 - acc: 0.9916 - val
Epoch 8/11
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0233 - acc: 0.9927 - val
Epoch 9/11
60000/60000 [==============================] - 127s 2ms/step - loss: 0.0213 - acc: 0.9938 - val
Epoch 10/11
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0210 - acc: 0.9937 - val
Epoch 11/11
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0179 - acc: 0.9944 - val
Test loss: 0.018701013443106058
Test accuracy: 0.9949
```

0.0.4 Example 4

In [3]:
```
# With activation = softmax, batch normalization and dropout layer

epochs = 10

model_4 = Sequential()
model_4.add(Conv2D(32, kernel_size=(4, 4),
                   activation='sigmoid',
                   input_shape=input_shape))
model_4.add(Conv2D(64, (4, 4), activation='sigmoid'))
model_4.add(MaxPooling2D(pool_size=(3, 3)))
model_4.add(Dropout(0.70))
model_4.add(Flatten())
model_4.add(Dense(128, activation='sigmoid'))
model_4.add(BatchNormalization(epsilon=0.001))
model_4.add(Dropout(0.30))
model_4.add(Dense(num_classes, activation='softmax'))

model_4.compile(loss=keras.losses.categorical_crossentropy,
                optimizer=keras.optimizers.Adadelta(),
                metrics=['accuracy'])
```

```
        model_4.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=1,
                  validation_data=(x_test, y_test))
        score_4 = model_4.evaluate(x_test, y_test, verbose=0)
        print('Test loss:', score_4[0])
        print('Test accuracy:', score_4[1])
```

```
WARNING: Logging before flag parsing goes to stderr.
W0809 14:12:37.660238  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:37.691510  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:37.707100  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:37.753999  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:37.769587  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:37.785243  2872 deprecation.py:506] From C:\Users\ACER\Anaconda3\lib\site-packages\
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0809 14:12:37.785243  2872 nn_ops.py:4224] Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x,
W0809 14:12:38.035182  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:38.050771  2872 deprecation_wrapper.py:119] From C:\Users\ACER\Anaconda3\lib\site-p

W0809 14:12:38.222638  2872 deprecation.py:323] From C:\Users\ACER\Anaconda3\lib\site-packages\
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where


Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 118s 2ms/step - loss: 1.8476 - acc: 0.3396 - val
Epoch 2/10
60000/60000 [==============================] - 118s 2ms/step - loss: 0.2793 - acc: 0.9129 - val
Epoch 3/10
60000/60000 [==============================] - 118s 2ms/step - loss: 0.1542 - acc: 0.9513 - val
Epoch 4/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.1222 - acc: 0.9622 - val
Epoch 5/10
60000/60000 [==============================] - 119s 2ms/step - loss: 0.1033 - acc: 0.9673 - val
Epoch 6/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.0941 - acc: 0.9703 - val
Epoch 7/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.0834 - acc: 0.9738 - val
```

```
Epoch 8/10
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0784 - acc: 0.9759 - val
Epoch 9/10
60000/60000 [==============================] - 120s 2ms/step - loss: 0.0756 - acc: 0.9768 - val
Epoch 10/10
60000/60000 [==============================] - 119s 2ms/step - loss: 0.0703 - acc: 0.9778 - val
Test loss: 0.03086712787185097
Test accuracy: 0.9897
```

0.0.5 Example 5

In [4]:
```
# With activation = tanh, without batch normalization, with dropout, with adam optimiz

        epochs = 10

        model_5 = Sequential()
        model_5.add(Conv2D(32, kernel_size=(3, 3),
                           activation='tanh',
                           input_shape=input_shape))
        model_5.add(Conv2D(32, (3, 3), activation='tanh'))
        model_5.add(MaxPooling2D(pool_size=(3, 3)))
        model_5.add(Dropout(0.44))
        model_5.add(Flatten())
        model_5.add(Dense(128, activation='sigmoid'))
        model_5.add(Dropout(0.30))
        model_5.add(Dense(num_classes, activation='softmax'))

        model_5.compile(loss=keras.losses.categorical_crossentropy,
                        optimizer=keras.optimizers.Adam(),
                        metrics=['accuracy'])

        model_5.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
        score_5 = model_5.evaluate(x_test, y_test, verbose=0)
        print('Test loss:', score_5[0])
        print('Test accuracy:', score_5[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.3902 - acc: 0.8877 - val_
Epoch 2/10
60000/60000 [==============================] - 64s 1ms/step - loss: 0.1223 - acc: 0.9660 - val_
Epoch 3/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0921 - acc: 0.9726 - val_
```

```
Epoch 4/10
60000/60000 [==============================] - 64s 1ms/step - loss: 0.0792 - acc: 0.9763 - val
Epoch 5/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0707 - acc: 0.9786 - val
Epoch 6/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0606 - acc: 0.9817 - val
Epoch 7/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0577 - acc: 0.9824 - val
Epoch 8/10
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0516 - acc: 0.9839 - val
Epoch 9/10
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0484 - acc: 0.9849 - val
Epoch 10/10
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0462 - acc: 0.9855 - val
Test loss: 0.03044679508442059
Test accuracy: 0.9902
```

0.0.6 Example 6

```
In [5]: # Activation = selu & softmax , without Batch normalization and dropout, Initializer =
        # bias_initializer, Optimizer = Adagrad

        epochs = 11

        model_6 = Sequential()
        model_6.add(Conv2D(64, kernel_size=(5, 5),
                           activation='selu',
                           input_shape=input_shape))
        model_6.add(Dense(64,
                          kernel_initializer='random_uniform',
                          bias_initializer='zeros'))
        model_6.add(Conv2D(32, (4, 4), activation='softmax'))
        model_6.add(MaxPooling2D(pool_size=(3, 3)))
        model_6.add(Flatten())
        model_6.add(Dense(128, activation='selu'))
        model_6.add(Dense(num_classes, activation='softmax'))

        model_6.compile(loss=keras.losses.categorical_crossentropy,
                        optimizer=keras.optimizers.Adagrad(),
                        metrics=['accuracy'])

        model_6.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
        score_6 = model_6.evaluate(x_test, y_test, verbose=0)
```

```
        print('Test loss:', score_6[0])
        print('Test accuracy:', score_6[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/11
60000/60000 [==============================] - 194s 3ms/step - loss: 0.2787 - acc: 0.9100 - val
Epoch 2/11
60000/60000 [==============================] - 192s 3ms/step - loss: 0.0592 - acc: 0.9824 - val
Epoch 3/11
60000/60000 [==============================] - 195s 3ms/step - loss: 0.0469 - acc: 0.9859 - val
Epoch 4/11
60000/60000 [==============================] - 191s 3ms/step - loss: 0.0396 - acc: 0.9885 - val
Epoch 5/11
60000/60000 [==============================] - 189s 3ms/step - loss: 0.0345 - acc: 0.9905 - val
Epoch 6/11
60000/60000 [==============================] - 189s 3ms/step - loss: 0.0310 - acc: 0.9913 - val
Epoch 7/11
60000/60000 [==============================] - 189s 3ms/step - loss: 0.0282 - acc: 0.9920 - val
Epoch 8/11
60000/60000 [==============================] - 191s 3ms/step - loss: 0.0259 - acc: 0.9928 - val
Epoch 9/11
60000/60000 [==============================] - 192s 3ms/step - loss: 0.0240 - acc: 0.9935 - val
Epoch 10/11
60000/60000 [==============================] - 208s 3ms/step - loss: 0.0223 - acc: 0.9942 - val
Epoch 11/11
60000/60000 [==============================] - 203s 3ms/step - loss: 0.0209 - acc: 0.9944 - val
Test loss: 0.03338987519220682
Test accuracy: 0.988
```

0.0.7 Example 7

```
In [7]: #Activation = softplus, Initializer = random_normal, with dropout and batch normalizat
        from keras import initializers

        epochs = 10

        model_7 = Sequential()
        model_7.add(Conv2D(32, kernel_size=(3, 3),
                           activation='softplus',
                           input_shape=input_shape))
        model_7.add(Conv2D(32, (4, 4), activation='softplus'))
        model_7.add(Dense(32,
                          kernel_initializer=initializers.random_normal(stddev=0.01)))
        model_7.add(MaxPooling2D(pool_size=(3, 3)))
        model_7.add(Dropout(0.50))
        model_7.add(Flatten())
        model_7.add(Dense(128, activation='softplus'))
```

```
model_7.add(BatchNormalization(epsilon=0.001))
model_7.add(Dropout(0.30))
model_7.add(Dense(num_classes, activation='softmax'))

model_7.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adamax(),
              metrics=['accuracy'])

model_7.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score_7 = model_7.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score_7[0])
print('Test accuracy:', score_7[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 114s 2ms/step - loss: 0.6562 - acc: 0.7877 - val
Epoch 2/10
60000/60000 [==============================] - 109s 2ms/step - loss: 0.2479 - acc: 0.9251 - val
Epoch 3/10
60000/60000 [==============================] - 108s 2ms/step - loss: 0.1668 - acc: 0.9492 - val
Epoch 4/10
60000/60000 [==============================] - 105s 2ms/step - loss: 0.1280 - acc: 0.9607 - val
Epoch 5/10
60000/60000 [==============================] - 107s 2ms/step - loss: 0.1124 - acc: 0.9657 - val
Epoch 6/10
60000/60000 [==============================] - 108s 2ms/step - loss: 0.1011 - acc: 0.9692 - val
Epoch 7/10
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0914 - acc: 0.9715 - val
Epoch 8/10
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0839 - acc: 0.9738 - val
Epoch 9/10
60000/60000 [==============================] - 107s 2ms/step - loss: 0.0793 - acc: 0.9754 - val
Epoch 10/10
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0748 - acc: 0.9765 - val
Test loss: 0.04983391810744069
Test accuracy: 0.9841
```

0.0.8 Prettytable

```
In [13]: from prettytable import PrettyTable

         number= [1,2,3,4,5,6,7]
         name= ["Model 1","Model 2","Model 3","Model 4","Model 5","Model 6","Model 7"]
```

```
loss= [score_1[0],score_2[0],score_3[0],score_4[0],score_5[0],score_6[0],score_7[0]]
acc= [score_1[1],score_2[1],score_3[1],score_4[1],score_5[1],score_6[1],score_7[1]]

#Initialize Prettytable
ptable = PrettyTable()
ptable.add_column("Index", number)
ptable.add_column("Model", name)
ptable.add_column("Test Loss", loss)
ptable.add_column("Test Accuracy", acc)
print(ptable)
```

```
+-------+---------+----------------------+---------------+
| Index |  Model  |      Test Loss       | Test Accuracy |
+-------+---------+----------------------+---------------+
|   1   | Model 1 | 0.03634111389952359  |     0.9902    |
|   2   | Model 2 | 0.015346395371133803 |     0.9957    |
|   3   | Model 3 | 0.018701013443106058 |     0.9949    |
|   4   | Model 4 | 0.03086712787185097  |     0.9897    |
|   5   | Model 5 | 0.03044679508442059  |     0.9902    |
|   6   | Model 6 | 0.03338987519220682  |     0.988     |
|   7   | Model 7 | 0.04983391810744069  |     0.9841    |
+-------+---------+----------------------+---------------+
```

0.0.9 Conclusions

1. For MNIST dataset, we build Convolutional Neural Networks using Conv2D.
2. We have used different kernel sizes like (3,3),(4,4),(5,5) etc.
3. Different layers are used in all the examples.
4. Some models with or without Batch normalization and Dropout layers to check the perforamnce.
5. Activation methods like softplus, relu, selu, sigmoid etc. are used
6. Optimizers like Adagrad, Adam, Adamax, Adadelta are used.
7. Test accuracy is almost same in all cases.

In [ ]: