CNN_MNIST

August 12, 2019

In [6]:
```
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
from keras.layers import regularizers
from keras.layers import BatchNormalization
from prettytable import PrettyTable
```

In [2]:
```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

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')
```

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

In [3]:
```
        # 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)
```

In [4]:
```
        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.
W0811 17:59:04.471589  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.487215  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.487215  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.643455  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.643455  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.659076  7896 deprecation.py:506] From C:\Users\hp\Anaconda3\lib\site-packages\ke
```

```
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0811 17:59:04.752821  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.768444  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 17:59:04.877831  7896 deprecation.py:323] From C:\Users\hp\Anaconda3\lib\site-packages\te
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 [==============================] - 113s 2ms/step - loss: 0.2674 - acc: 0.9171 - val
Epoch 2/12
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0871 - acc: 0.9742 - val
Epoch 3/12
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0658 - acc: 0.9798 - val
Epoch 4/12
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0521 - acc: 0.9847 - val
Epoch 5/12
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0454 - acc: 0.9863 - val
Epoch 6/12
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0418 - acc: 0.9874 - val
Epoch 7/12
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0366 - acc: 0.9887 - val
Epoch 8/12
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0325 - acc: 0.9899 - val
Epoch 9/12
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0302 - acc: 0.9903 - val
Epoch 10/12
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0273 - acc: 0.9917 - val
Epoch 11/12
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0279 - acc: 0.9915 - val
Epoch 12/12
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0250 - acc: 0.9920 - val
Test loss: 0.028258230628092677
Test accuracy: 0.9924
```

1 Assignment

We have to make about 3 distinct architectures of convolutional netwroks.

1.0.1 Architecture 1:

```
In [7]: model_1 = Sequential()
        model_1.add(Conv2D(32, kernel_size=(3, 3),
```

```
                 activation='relu',
                 input_shape=input_shape))
model_1.add(Conv2D(32, (3, 3), activation='relu'))
model_1.add(MaxPooling2D(pool_size=(3, 3)))
model_1.add(BatchNormalization())
model_1.add(Dropout(0.5))
model_1.add(Flatten())
model_1.add(Dense(64, 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])
```

```
W0811 18:35:14.560059  7896 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 73s 1ms/step - loss: 0.1829 - acc: 0.9419 - val
Epoch 2/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0683 - acc: 0.9788 - val
Epoch 3/12
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0523 - acc: 0.9841 - val
Epoch 4/12
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0447 - acc: 0.9857 - val
Epoch 5/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0367 - acc: 0.9883 - val
Epoch 6/12
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0325 - acc: 0.9894 - val
Epoch 7/12
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0302 - acc: 0.9902 - val
Epoch 8/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0279 - acc: 0.9908 - val
Epoch 9/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0244 - acc: 0.9922 - val
Epoch 10/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0221 - acc: 0.9928 - val
```

```
Epoch 11/12
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0198 - acc: 0.9940 - val
Epoch 12/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0205 - acc: 0.9933 - val
Test loss: 0.028312752436022673
Test accuracy: 0.9919
```

1.0.2 Arhcitecture 2

```
In [9]: model_2 = Sequential()
        model_2.add(Conv2D(32, kernel_size=(5, 5),
                           activation='relu',
                           input_shape=input_shape))
        model_2.add(Conv2D(64, (3, 3), activation='relu'))
        model_2.add(BatchNormalization())
        model_1.add(Dropout(0.1))
        model_2.add(Conv2D(64, (2, 2), activation='relu'))
        model_2.add(MaxPooling2D(pool_size=(3, 3)))
        model_2.add(BatchNormalization())
        model_2.add(Flatten())
        model_2.add(Dense(64, activation='relu'))
        model_2.add(BatchNormalization())
        model_2.add(Dropout(0.25))
        model_2.add(Dense(num_classes, activation='softmax'))

        model_2.compile(loss=keras.losses.categorical_crossentropy,
                        optimizer=keras.optimizers.Adam(),
                        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 1/12
60000/60000 [==============================] - 303s 5ms/step - loss: 0.1254 - acc: 0.9651 - val
Epoch 2/12
60000/60000 [==============================] - 303s 5ms/step - loss: 0.0431 - acc: 0.9879 - val
Epoch 3/12
60000/60000 [==============================] - 302s 5ms/step - loss: 0.0320 - acc: 0.9904 - val
Epoch 4/12
60000/60000 [==============================] - 300s 5ms/step - loss: 0.0230 - acc: 0.9928 - val
```

```
Epoch 5/12
60000/60000 [==============================] - 305s 5ms/step - loss: 0.0198 - acc: 0.9941 - val
Epoch 6/12
60000/60000 [==============================] - 328s 5ms/step - loss: 0.0168 - acc: 0.9945 - val
Epoch 7/12
60000/60000 [==============================] - 332s 6ms/step - loss: 0.0143 - acc: 0.9953 - val
Epoch 8/12
60000/60000 [==============================] - 317s 5ms/step - loss: 0.0134 - acc: 0.9957 - val
Epoch 9/12
60000/60000 [==============================] - 325s 5ms/step - loss: 0.0105 - acc: 0.9966 - val
Epoch 10/12
60000/60000 [==============================] - 311s 5ms/step - loss: 0.0097 - acc: 0.9969 - val
Epoch 11/12
60000/60000 [==============================] - 311s 5ms/step - loss: 0.0101 - acc: 0.9966 - val
Epoch 12/12
60000/60000 [==============================] - 315s 5ms/step - loss: 0.0077 - acc: 0.9977 - val
Test loss: 0.03223905730191964
Test accuracy: 0.9913
```

1.0.3 Architecture 3:

```
In [10]: model_3 = Sequential()
         model_3.add(Conv2D(32, kernel_size=(2, 2),
                            activation='sigmoid',
                            input_shape=input_shape))
         model_3.add(Conv2D(64, (3, 3), activation='sigmoid'))
         model_3.add(BatchNormalization())
         model_3.add(MaxPooling2D(pool_size=(3, 3)))
         model_3.add(Dropout(0.5))
         model_3.add(Flatten())
         model_3.add(Dense(128, activation='sigmoid'))
         model_3.add(BatchNormalization())
         model_3.add(Dropout(0.2))
         model_3.add(Dense(num_classes, activation='softmax'))

         model_3.compile(loss=keras.losses.categorical_crossentropy,
                         optimizer=keras.optimizers.Adagrad(),
                         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 1/12
60000/60000 [==============================] - 316s 5ms/step - loss: 0.1750 - acc: 0.9469 - val
Epoch 2/12
60000/60000 [==============================] - 319s 5ms/step - loss: 0.0886 - acc: 0.9730 - val
Epoch 3/12
60000/60000 [==============================] - 317s 5ms/step - loss: 0.0738 - acc: 0.9772 - val
Epoch 4/12
60000/60000 [==============================] - 327s 5ms/step - loss: 0.0647 - acc: 0.9802 - val
Epoch 5/12
60000/60000 [==============================] - 326s 5ms/step - loss: 0.0578 - acc: 0.9822 - val
Epoch 6/12
60000/60000 [==============================] - 324s 5ms/step - loss: 0.0552 - acc: 0.9826 - val
Epoch 7/12
60000/60000 [==============================] - 324s 5ms/step - loss: 0.0515 - acc: 0.9846 - val
Epoch 8/12
60000/60000 [==============================] - 323s 5ms/step - loss: 0.0474 - acc: 0.9854 - val
Epoch 9/12
60000/60000 [==============================] - 323s 5ms/step - loss: 0.0452 - acc: 0.9857 - val
Epoch 10/12
60000/60000 [==============================] - 319s 5ms/step - loss: 0.0427 - acc: 0.9868 - val
Epoch 11/12
60000/60000 [==============================] - 318s 5ms/step - loss: 0.0407 - acc: 0.9878 - val
Epoch 12/12
60000/60000 [==============================] - 332s 6ms/step - loss: 0.0381 - acc: 0.9885 - val
Test loss: 0.03459447893754113
Test accuracy: 0.9881
```

1.0.4 Architecture 4:

```
In [5]: # Activation = selu & softmax , without Batch normalization and dropout, Initializer =
        # bias_initializer, Optimizer = Adagrad
        epochs = 10
        batch_size = 256

        model_4 = Sequential()
        model_4.add(Conv2D(32, kernel_size=(5, 5),
                           activation='tanh',
                           input_shape=input_shape))
        model_4.add(Dense(45,
                          kernel_initializer='random_uniform'))
        model_4.add(Conv2D(64, (4, 4), activation='softmax'))
        model_4.add(BatchNormalization())
        model_4.add(Dropout(0.25))
        model_4.add(MaxPooling2D(pool_size=(3, 3)))
        model_4.add(Flatten())
        model_4.add(Dense(128, activation='sigmoid'))
```

```
        model_4.add(BatchNormalization())
        model_4.add(Dropout(0.5))
        model_4.add(Dense(num_classes, activation='softmax'))

        model_4.compile(loss=keras.losses.categorical_crossentropy,
                        optimizer=keras.optimizers.Adamax(),
                        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])
```

```
W0811 22:37:34.260540  6576 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 22:37:34.260540  6576 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 22:37:34.338661  6576 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 22:37:34.416781  6576 deprecation.py:506] From C:\Users\hp\Anaconda3\lib\site-packages\ke
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0811 22:37:34.432404  6576 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 22:37:34.573019  6576 deprecation_wrapper.py:119] From C:\Users\hp\Anaconda3\lib\site-pac

W0811 22:37:34.682407  6576 deprecation.py:323] From C:\Users\hp\Anaconda3\lib\site-packages\te
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 [==============================] - 354s 6ms/step - loss: 0.1780 - acc: 0.9479 - val
Epoch 2/10
60000/60000 [==============================] - 378s 6ms/step - loss: 0.0719 - acc: 0.9778 - val
Epoch 3/10
60000/60000 [==============================] - 360s 6ms/step - loss: 0.0588 - acc: 0.9825 - val
Epoch 4/10
60000/60000 [==============================] - 368s 6ms/step - loss: 0.0494 - acc: 0.9842 - val
Epoch 5/10
60000/60000 [==============================] - 366s 6ms/step - loss: 0.0438 - acc: 0.9871 - val
Epoch 6/10
60000/60000 [==============================] - 362s 6ms/step - loss: 0.0389 - acc: 0.9884 - val
```

```
Epoch 7/10
60000/60000 [==============================] - 397s 7ms/step - loss: 0.0340 - acc: 0.9893 - val
Epoch 8/10
60000/60000 [==============================] - 404s 7ms/step - loss: 0.0320 - acc: 0.9901 - val
Epoch 9/10
60000/60000 [==============================] - 387s 6ms/step - loss: 0.0275 - acc: 0.9914 - val
Epoch 10/10
60000/60000 [==============================] - 381s 6ms/step - loss: 0.0259 - acc: 0.9918 - val
Test loss: 0.036352840721810935
Test accuracy: 0.9884
```

1.1 Conclusions

```
In [11]: number= [1,2,3,4]
         model = ["Architecture 1","Architecture 2","Architecture 3","Architecture 4"]
         opt   = ['Adadelta', 'Adam', 'Adagrad', 'Adamax']
         act   = ['relu', 'relu', 'sigmoid', 'tanh,sigmoid']
         loss  = [0.028312752436022673,0.03223905730191964,0.03459447893754113,score_4[0]]
         acc   = [0.9919,0.9913,0.9881,score_4[1]]

         #Initialize Prettytable
         pt = PrettyTable()
         pt.add_column("Sr.No.", number)
         pt.add_column("Model", model)
         pt.add_column("Optimizer", opt)
         pt.add_column('Activation ', act)
         pt.add_column("Test Loss", loss)
         pt.add_column("Test Accuracy", acc)
         print(pt)
```

```
+--------+----------------+-----------+--------------+----------------------+---------------+
| Sr.No. |     Model      | Optimizer | Activation   |      Test Loss       | Test Accuracy |
+--------+----------------+-----------+--------------+----------------------+---------------+
|   1    | Architecture 1 |  Adadelta |     relu     | 0.028312752436022673 |     0.9919    |
|   2    | Architecture 2 |    Adam   |     relu     | 0.03223905730191964  |     0.9913    |
|   3    | Architecture 3 |  Adagrad  |   sigmoid    | 0.03459447893754113  |     0.9881    |
|   4    | Architecture 4 |   Adamax  | tanh,sigmoid | 0.036352840721810935 |     0.9884    |
+--------+----------------+-----------+--------------+----------------------+---------------+
```

Architecture 1 is getting highest accuracy of 99.19% with minimum loss, in which I have used Adadelta optimizer and relu activation.

I have tried 4 CNN architectures:

1. with differnt number of layers

2. different kernel values: (2,2), (3,3), (5,5)

3. different activations: relu, tanh, sigmoid

4. different optimizers: adam, adadelta, adagrad, adamax

5. adding batchnormalisation layers

6. adding some dropout layers, maxpool layers, flattening layers and dense layers