Convolutional Neural Networks

In this notebook, we train an Multi-Layer Perceptrons (MLP). to classify images from the MNIST database.

1. Load MNIST Database

In [1]:

```
from keras.datasets import mnist

# use Keras to import pre-shuffled MNIST database
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print("The MNIST database has a training set of %d examples." % len(X_train))
print("The MNIST database has a test set of %d examples." % len(X_test))
```

```
Using TensorFlow backend.
The MNIST database has a training set of 60000 examples.
The MNIST database has a test set of 10000 examples.
```

2. Visualize the First Six Training Images

In [3]:

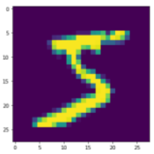
```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.cm as cm
import numpy as np
```

In [5]:

```
fig = plt.figure(figsize=(5,5))
plt.imshow(X_train[0])
X_train[0][5]
```

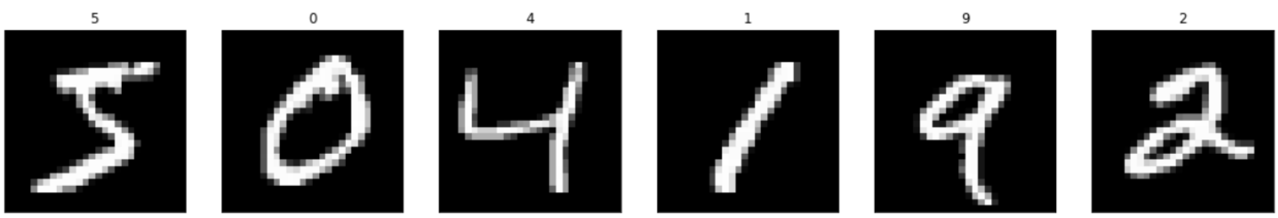
Out[5]:

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
        18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
         0,   0], dtype=uint8)
```



In [6]:

```
# plot first six training images
fig = plt.figure(figsize=(20,20))
for i in range(6):
    ax = fig.add_subplot(1, 6, i+1, xticks=[], yticks=[])
    ax.imshow(X_train[i], cmap='gray')
    ax.set_title(str(y_train[i]))
```

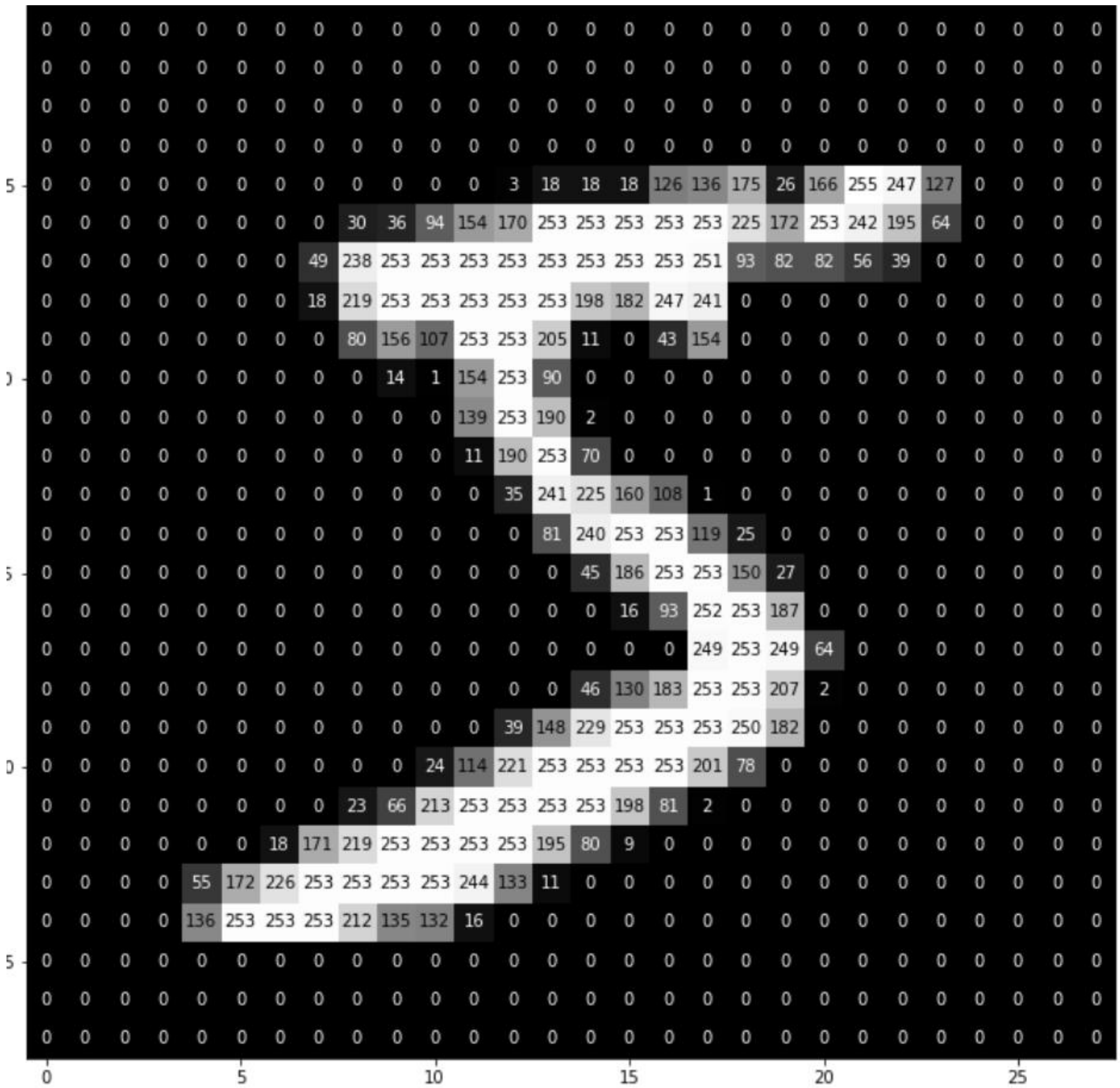


3. View an Image in More Detail

In [7]:

```
def visualize_input(img, ax):
    ax.imshow(img, cmap='gray')
    width, height = img.shape
    thresh = img.max()/2.5
    for x in range(width):
        for y in range(height):
            ax.annotate(str(round(img[x][y],2)), xy=(y,x),
                        horizontalalignment='center',
                        verticalalignment='center',
                        color='white' if img[x][y]<thresh else 'black')

fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(111)
visualize_input(X_train[0], ax)
```



4. Rescale the Images by Dividing Every Pixel in Every Image by 255

In [8]:

```
# rescale [0,255] --> [0,1]
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
```

In [9]:

```
X_train.shape#[1:]
```

Out[9]:

```
(60000, 28, 28)
```

In [10]:

```
X_train[0][5]
```

Out[10]:

```
array([ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.01176471,  0.07058824,  0.07058824,
        0.07058824,  0.49411765,  0.53333336,  0.68627453,  0.10196079,
        0.65098041,  1.        ,  0.96862745,  0.49803922,  0.        ,
        0.        ,  0.        ,  0.        ], dtype=float32)
```

5. Encode Categorical Integer Labels Using a One-Hot Scheme

In [11]:

```
from keras.utils import np_utils

# print first ten (integer-valued) training labels
print('Integer-valued labels:')
print(y_train[:10])

# one-hot encode the labels
y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)

# print first ten (one-hot) training labels
print('One-hot labels:')
print(y_train[:10])
```

```
Integer-valued labels:
[5 0 4 1 9 2 1 3 1 4]
One-hot labels:
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]]
```

6. Define the Model Architecture

In [12]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
```

In [13]:

```
# define the model
model = Sequential()
model.add(Flatten(input_shape=X_train.shape[1:]))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# summarize the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130      
=================================================================
Total params: 669,706.0
Trainable params: 669,706.0
Non-trainable params: 0.0
_________________________________________________________________
```

7. Compile the Model

In [14]:

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

8. Calculate the Classification Accuracy on the Test Set (Before Training)

In [16]:

```
# evaluate test accuracy
score = model.evaluate(X_test, y_test, verbose=0)
accuracy = 100*score[1]

# print test accuracy
print('Test accuracy: %.4f%%' % accuracy)
```

```
Test accuracy: 7.9200%
```

9. Train the Model

In [ ]:

```
from keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='mnist.model.best.hdf5',
                               verbose=1, save_best_only=True)
hist = model.fit(X_train, y_train, batch_size=128, epochs=10,
          validation_split=0.2, callbacks=[checkpointer],
          verbose=1, shuffle=True)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
47872/48000 [============================>.] - ETA: 0s - loss: 0.2721 - acc: 0.9161
…….
Epoch 4/10
 4992/48000 [==>...........................] - ETA: 8s - loss: 0.0554 - acc: 0.9828
```

10. Load the Model with the Best Classification Accuracy on the Validation Set

In [18]:

```
# load the weights that yielded the best validation accuracy
model.load_weights('mnist.model.best.hdf5')
```

11. Calculate the Classification Accuracy on the Test Set

In [19]:

```
# evaluate test accuracy
score = model.evaluate(X_test, y_test, verbose=0)
accuracy = 100*score[1]

# print test accuracy
print('Test accuracy: %.4f%%' % accuracy)
```

```
Test accuracy: 97.7200%
```