12-FINAL

December 18, 2018

1 Assignment 12 - Neural Networks image recognition

Use both MLNN and the ConvNet to solve the following problem.

1. Add random noise (i.e. `np.random.normal`) to the images in training and testing. Make sure each image gets a different noise feature added to it. Inspect by printing out an image.
2. Compare the loss/accuracy (train, val) after N epochs for both MLNN and ConvNet with and without noise.
3. Vary the amount of noise (multiply `np.random.normal` by a factor) and keep track of the accuracy and loss (for training and validation) and plot these results.

2 Neural Networks - Image Recognition

```
In [1]: import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.optimizers import RMSprop
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from keras import backend
```

```
C:\Users\Erin\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
In [10]: import matplotlib.pyplot as plt
         %matplotlib inline
```

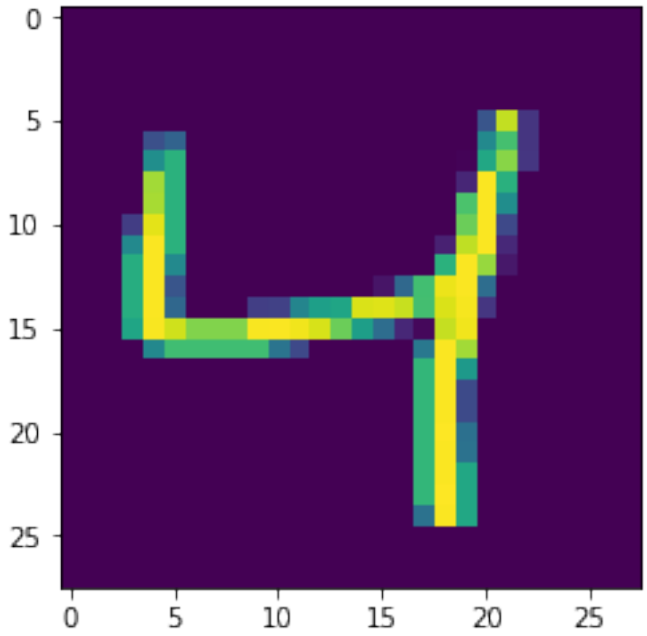
2.1 Multi Layer Neural Network

Trains a simple deep NN on the MNIST dataset. Gets to 98.40% test accuracy after 20 epochs (there is *a lot* of margin for parameter tuning).

```
In [11]: # the data, shuffled and split between train and test sets
         (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
In [18]: plt.imshow(x_train[2])
```

Out[18]: <matplotlib.image.AxesImage at 0x2174a9b35f8>



In [19]:
```
import numpy as np
np.max(x_train[2])
```

Out[19]: 255

In [16]:
```
y_train[2]
```

Out[16]: 4

In [24]:
```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

```
In [3]: x_train[0]

Out[3]: array([0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.01176471, 0.07058824, 0.07058824,
               0.07058824, 0.49411765, 0.53333336, 0.6862745 , 0.10196079,
               0.6509804 , 1.        , 0.96862745, 0.49803922, 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.11764706, 0.14117648, 0.36862746, 0.6039216 ,
               0.6666667 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
               0.99215686, 0.88235295, 0.6745098 , 0.99215686, 0.9490196 ,
               0.7647059 , 0.2509804 , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.19215687, 0.93333334,
               0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
               0.99215686, 0.99215686, 0.99215686, 0.9843137 , 0.3647059 ,
               0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.07058824, 0.85882354, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.7137255 ,
0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.3137255 , 0.6117647 , 0.41960785, 0.99215686, 0.99215686,
0.8039216 , 0.04313726, 0.        , 0.16862746, 0.6039216 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.05490196,
0.00392157, 0.6039216 , 0.99215686, 0.3529412 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.54509807,
0.99215686, 0.74509805, 0.00784314, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.04313726, 0.74509805, 0.99215686,
0.27450982, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.13725491, 0.94509804, 0.88235295, 0.627451  ,
0.42352942, 0.00392157, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.31764707, 0.9411765 , 0.99215686, 0.99215686, 0.46666667,
0.09803922, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.1764706 ,
0.7294118 , 0.99215686, 0.99215686, 0.5882353 , 0.10588235,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.0627451 , 0.3647059 ,
0.9882353 , 0.99215686, 0.73333335, 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.9764706 , 0.99215686,
0.9764706 , 0.2509804 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.18039216, 0.50980395,
0.7176471 , 0.99215686, 0.99215686, 0.8117647 , 0.00784314,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.15294118,
0.5803922 , 0.8980392 , 0.99215686, 0.99215686, 0.99215686,
0.98039216, 0.7137255 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.09411765, 0.44705883, 0.8666667 , 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.7882353 , 0.30588236, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.09019608, 0.25882354, 0.8352941 , 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.31764707,
0.00784314, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.07058824, 0.67058825, 0.85882354,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.7647059 ,
0.3137255 , 0.03529412, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.21568628, 0.6745098 ,
0.8862745 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.95686275, 0.52156866, 0.04313726, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.53333336, 0.99215686, 0.99215686, 0.99215686,
0.83137256, 0.5294118 , 0.5176471 , 0.0627451 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

```
In [22]: keras.utils.to_categorical(y_train, num_classes)
```

```
Out[22]: array([[[1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 ...,
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.]],

                [[0., 1., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 ...,
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.]],

                [[1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 ...,
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.]],

                ...,

                [[1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 ...,
```

```
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]],

      [[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]],

      [[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]]], dtype=float32)
```

In [23]:
```
y_train
```

Out[23]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

In [25]:
```
batch_size = 128
num_classes = 10
epochs = 20


# 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)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()
```

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

        history = 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])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.2433 - acc: 0.9250 - val
Epoch 2/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1024 - acc: 0.9690 - val_
Epoch 3/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0735 - acc: 0.9773 - val_
Epoch 4/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0597 - acc: 0.9823 - val_
Epoch 5/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0498 - acc: 0.9853 - val_
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0426 - acc: 0.9872 - val_
Epoch 7/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0366 - acc: 0.9888 - val_
Epoch 8/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0334 - acc: 0.9903 - val_
Epoch 9/20
```

```
60000/60000 [==============================] - 6s 94us/step - loss: 0.0301 - acc: 0.9913 - val
Epoch 10/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0304 - acc: 0.9913 - val
Epoch 11/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0263 - acc: 0.9925 - val
Epoch 12/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0257 - acc: 0.9928 - val
Epoch 13/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0240 - acc: 0.9930 - val
Epoch 14/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0218 - acc: 0.9939 - val
Epoch 15/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0220 - acc: 0.9941 - val
Epoch 16/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0189 - acc: 0.9947 - val
Epoch 17/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0197 - acc: 0.9948 - val
Epoch 18/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0183 - acc: 0.9954 - val
Epoch 19/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0170 - acc: 0.9954 - va
Epoch 20/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0180 - acc: 0.9958 - val
Test loss: 0.10086243585582774
Test accuracy: 0.9852
```

2.2 Conv Net

Trains a simple convnet on the MNIST dataset. Gets to 99.25% test accuracy after 12 epochs (there is still a lot of margin for parameter tuning).

```
In [26]: # input image dimensions
         img_rows, img_cols = 28, 28

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

         if backend.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 [27]: batch_size = 128
         num_classes = 10
         epochs = 12

         # 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)

         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])

Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 94s 2ms/step - loss: 0.2796 - acc: 0.9139 - val
```

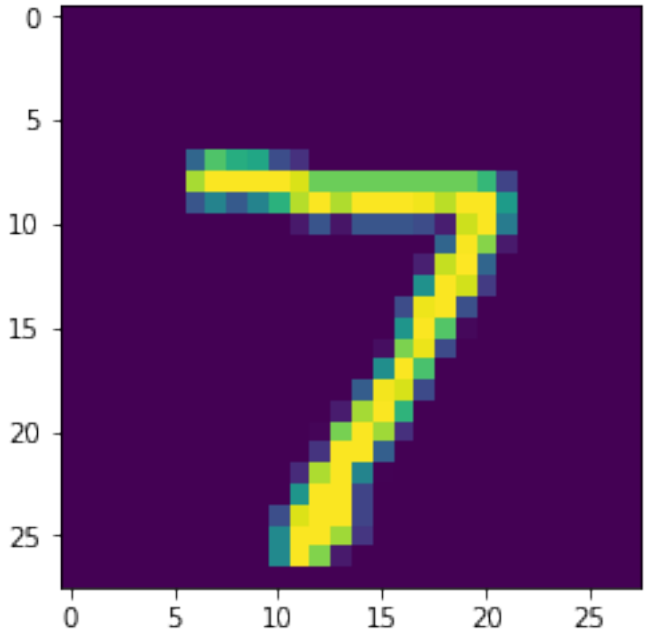
```
Epoch 2/12
60000/60000 [==============================] - 99s 2ms/step - loss: 0.0875 - acc: 0.9744 - val
Epoch 3/12
60000/60000 [==============================] - 98s 2ms/step - loss: 0.0677 - acc: 0.9798 - val
Epoch 4/12
60000/60000 [==============================] - 97s 2ms/step - loss: 0.0548 - acc: 0.9839 - val
Epoch 5/12
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0469 - acc: 0.9861 - va
Epoch 6/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0409 - acc: 0.9878 - va
Epoch 7/12
60000/60000 [==============================] - 90s 1ms/step - loss: 0.0387 - acc: 0.9882 - val
Epoch 8/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0348 - acc: 0.9893 - va
Epoch 9/12
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0330 - acc: 0.9898 - va
Epoch 10/12
60000/60000 [==============================] - 86s 1ms/step - loss: 0.0295 - acc: 0.9906 - val
Epoch 11/12
60000/60000 [==============================] - 86s 1ms/step - loss: 0.0299 - acc: 0.9906 - val
Epoch 12/12
60000/60000 [==============================] - 86s 1ms/step - loss: 0.0281 - acc: 0.9914 - val
Test loss: 0.026129956484261856
Test accuracy: 0.9928
```

```
In [28]: score = model.evaluate(x_test, y_test, verbose=0)
         print('Test loss:' , score[0])
         print('Test accuracy:' , score[1])
```

```
Test loss: 0.026129956484261856
Test accuracy: 0.9928
```

```
In [29]: plt.imshow(x_test[0].reshape(28,28))
```

```
Out[29]: <matplotlib.image.AxesImage at 0x2176ea9ee48>
```




```
In [30]: model.predict(x_test[0:1]), y_test[:1]

Out[30]: (array([[1.1235774e-12, 3.6403717e-11, 4.6634319e-10, 1.3857475e-09,
                  2.0686898e-11, 3.6422727e-12, 1.8520169e-16, 1.0000000e+00,
                  1.0492592e-11, 5.4979825e-09]], dtype=float32),
          array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]], dtype=float32))

In [31]: model.predict_classes(x_test[0:1]), y_test[:1]

Out[31]: (array([7], dtype=int64),
          array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]], dtype=float32))

In [40]: # CovNet with noise
         img_rows, img_cols = 28, 28

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

         if backend.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)
```

```
        np.random.normal(0, 0.1, 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
```