PROJECT

RECOGNISING HANDWRITTEN DIGITS FROM THE MNIST DATASET USING CONVOLUTIONAL NEURAL NETWORKS

RHEEYA UPPAAL
3rd Year, Computer Science

The workings and behaviour of Convolutional Neural Networks (CNNs) have been described in previous sections. In this project, a CNN was trained on the MNIST dataset to recognise handwritten digits.

The MNIST database (Mixed National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 20x20 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23 percent. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support vector machine to get an error rate of 0.8 percent.

The program uses a CNN with 2 hidden convolutional layers (32 hidden nodes each) , Dropout of 0.5, Rectified Linear Unit activation and Max Pooling. The system works with a 97.5% accuracy.

```
#CNN on MNIST dataset. The program classifies handwritten digits as '0','1' etc.

import matplotlib.pyplot as plt
from keras.datasets import mnist #dataset
from keras.utils import np_utils

from keras.models import Sequential #type of model
from keras.layers.core import Dense, Dropout, Activation, Flatten #layers
from keras.layers.convolutional import Convolution2D, MaxPooling2D #convolution layers
```

```
#%% Define values

batch_size=128  #Batch size to train
nb_classes=10   #Number of output classes (10 digits)
nb_epoch=12     #Number of epochs to train (i.e. number of trainings) -?

img_rows, img_cols=28, 28   #Dimensions of input images
nb_filters=32               #Number of convolutional filters to be used
nb_pool=2                   #Size of pooling area for max pooling
nb_conv=3                   #Size of convolution kernel
```

```
#%% Organise Data

#X_train, X_test: numpy arrays in float32.
#Y_train, Y_test: numpy arrays in uint8.

#Shuffle and split data between train and test sets
(X_train, y_train), (X_test, y_test)= mnist.load_data()

#Reshape the data
X_train=X_train.reshape(X_train.shape[0],1,img_rows,img_cols)
X_train=X_train.astype('float32')
X_train/=255
X_test=X_test.reshape(X_test.shape[0],1,img_rows,img_cols)
X_test=X_test.astype('float32')
X_test/=255

#Output: No_of_samples, no_of_channels(=1), imgage_dimensions
print ('X_train shape: ', X_train.shape)
print ('No of training samples', X_train.shape[0])
print ('No of testing samples', X_test.shape[0])

#Convert class vectors to binary class matrices
Y_train=np_utils.to_categorical(y_train, nb_classes)
Y_test=np_utils.to_categorical(y_test, nb_classes)

#Display the input image (4600th sample)
i=4600
plt.imshow(X_train[i,0], interpolation='nearest')
print ('Label: ', Y_train[i,:])
```

```
#%% Define Model

model=Sequential()  #Declare a sequential model

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                    input_shape=(1,img_rows,img_cols)))
convout1=Activation('relu')     #relu - Rectified Linear Unit
model.add(convout1)
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
```

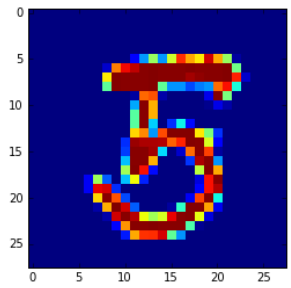
```
convout2=Activation('relu')
model.add(convout2)
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

#Flattens O/P from previous layer (i.e. make it fully connnected)
model.add(Flatten())
model.add(Dense(128))       #128 neurons
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))    #No of classes is no of neurons for final layer
#Final final layer (softmax fn.) makes output readable to us
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adadelta')
```

```
#%% Run the model
model.fit(X_train,Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
        show_accuracy=True, verbose=1, validation_data=(X_test, Y_test))
```

```
#%% Valididate
score=model.evaluate(X_test,Y_test,show_accuracy=True,verbose=2)
print('Test Score: ',score[0])
print('Test Accuracy: ',score[1])
print(model.predict_classes(X_test[1:5]))
print(Y_test[1:5])
```

Output:

```
Using Theano backend.
Using gpu device 0: GeForce 840M (CNMeM is enabled with initial size: 80.0% of memory, cuDNN 5004)
('X_train shape: ', (60000L, 1L, 28L, 28L))
('No of training samples', 60000L)
('No of testing samples', 10000L)
('Label: ', array([ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.]))
```




```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 24s - loss: 0.0234 - val_loss: 0.0242
Epoch 2/12
60000/60000 [==============================] - 23s - loss: 0.0218 - val_loss: 0.0264
Epoch 3/12
60000/60000 [==============================] - 23s - loss: 0.0215 - val_loss: 0.0271
Epoch 4/12
60000/60000 [==============================] - 24s - loss: 0.0212 - val_loss: 0.0253
Epoch 5/12
60000/60000 [==============================] - 23s - loss: 0.0197 - val_loss: 0.0269
Epoch 6/12
60000/60000 [==============================] - 24s - loss: 0.0165 - val_loss: 0.0290
Epoch 7/12
60000/60000 [==============================] - 23s - loss: 0.0173 - val_loss: 0.0282
```

```
Epoch 8/12
60000/60000 [==============================] - 23s - loss: 0.0160 - val_loss: 0.0293
Epoch 9/12
60000/60000 [==============================] - 24s - loss: 0.0165 - val_loss: 0.0260
Epoch 10/12
60000/60000 [==============================] - 24s - loss: 0.0156 - val_loss: 0.0293
Epoch 11/12
60000/60000 [==============================] - 24s - loss: 0.0155 - val_loss: 0.0271
Epoch 12/12
60000/60000 [==============================] - 24s - loss: 0.0152 - val_loss: 0.0275
Results:

Test Accuracy: 97.8%
4/4 [==============================] - 0s
[2 1 0 4]
[[ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]]
```