DL0101EN-4-1-Convolutional-Neural-Networks-with-Keras-py-v1.0

January 16, 2020

Convolutional Neural Networks with Keras

In this lab, we will learn how to use the Keras library to build convolutional neural networks. We will also use the popular MNIST dataset and we will compare our results to using a conventional neural network.

0.1 Table of Contents

0.2 Import Keras and Packages

Let's start by importing the keras libraries and the packages that we would need to build a neural network.

[1]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

When working with convolutional neural networks in particular, we will need additional packages.

[2]:
```
from keras.layers.convolutional import Conv2D # to add convolutional layers
from keras.layers.convolutional import MaxPooling2D # to add pooling layers
from keras.layers import Flatten # to flatten data for fully connected layers
```

0.3 Convolutional Layer with One set of convolutional and pooling layers

[3]:
```
# import data
from keras.datasets import mnist

# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
# reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Let's normalize the pixel values to be between 0 and 1

[4]:
```
X_train = X_train / 255 # normalize training data
X_test = X_test / 255 # normalize test data
```

Next, let's convert the target variable into binary categories

[5]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

num_classes = y_test.shape[1] # number of categories
```

Next, let's define a function that creates our model. Let's start with one set of convolutional and pooling layers.

[6]:
```
def convolutional_model():

    # create model
    model = Sequential()
    model.add(Conv2D(16, (5, 5), strides=(1, 1), activation='relu',␣
 ↪input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    # compile model
    model.compile(optimizer='adam', loss='categorical_crossentropy', ␣
 ↪metrics=['accuracy'])
    return model
```

Finally, let's call the function to create the model, and then let's train it and evaluate it.

[7]:
```
# build the model
model = convolutional_model()

# fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,␣
 ↪batch_size=200, verbose=2)
```

```
# evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: {} \n Error: {}".format(scores[1], 100-scores[1]*100))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 18s - loss: 0.2901 - acc: 0.9194 - val_loss: 0.1087 - val_acc: 0.9679
Epoch 2/10
 - 19s - loss: 0.0872 - acc: 0.9751 - val_loss: 0.0624 - val_acc: 0.9804
Epoch 3/10
 - 18s - loss: 0.0583 - acc: 0.9830 - val_loss: 0.0504 - val_acc: 0.9832
Epoch 4/10
 - 17s - loss: 0.0446 - acc: 0.9867 - val_loss: 0.0414 - val_acc: 0.9866
Epoch 5/10
 - 17s - loss: 0.0361 - acc: 0.9891 - val_loss: 0.0369 - val_acc: 0.9881
Epoch 6/10
 - 18s - loss: 0.0287 - acc: 0.9915 - val_loss: 0.0373 - val_acc: 0.9876
Epoch 7/10
 - 17s - loss: 0.0247 - acc: 0.9924 - val_loss: 0.0392 - val_acc: 0.9872
Epoch 8/10
 - 17s - loss: 0.0204 - acc: 0.9939 - val_loss: 0.0382 - val_acc: 0.9872
Epoch 9/10
 - 17s - loss: 0.0169 - acc: 0.9949 - val_loss: 0.0396 - val_acc: 0.9879
Epoch 10/10
 - 17s - loss: 0.0138 - acc: 0.9960 - val_loss: 0.0409 - val_acc: 0.9871
Accuracy: 0.9871
 Error: 1.2900000000000063
```

0.4 Convolutional Layer with two sets of convolutional and pooling layers

Let's redefine our convolutional model so that it has two convolutional and pooling layers instead of just one layer of each.

[8]:
```
def convolutional_model():

    # create model
    model = Sequential()
    model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Conv2D(8, (2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(optimizer='adam', loss='categorical_crossentropy', ␣
↪metrics=['accuracy'])
    return model
```

Now, let's call the function to create our new convolutional neural network, and then let's train it and evaluate it.

[9]:
```
# build the model
model = convolutional_model()

# fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,␣
↪batch_size=200, verbose=2)

# evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: {} \n Error: {}".format(scores[1], 100-scores[1]*100))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 19s - loss: 0.5135 - acc: 0.8515 - val_loss: 0.1447 - val_acc: 0.9580
Epoch 2/10
 - 19s - loss: 0.1235 - acc: 0.9631 - val_loss: 0.0764 - val_acc: 0.9772
Epoch 3/10
 - 18s - loss: 0.0860 - acc: 0.9747 - val_loss: 0.0615 - val_acc: 0.9816
Epoch 4/10
 - 18s - loss: 0.0673 - acc: 0.9797 - val_loss: 0.0535 - val_acc: 0.9824
Epoch 5/10
 - 19s - loss: 0.0588 - acc: 0.9824 - val_loss: 0.0475 - val_acc: 0.9854
Epoch 6/10
 - 18s - loss: 0.0518 - acc: 0.9838 - val_loss: 0.0445 - val_acc: 0.9865
Epoch 7/10
 - 18s - loss: 0.0456 - acc: 0.9861 - val_loss: 0.0399 - val_acc: 0.9880
Epoch 8/10
 - 18s - loss: 0.0419 - acc: 0.9869 - val_loss: 0.0357 - val_acc: 0.9881
Epoch 9/10
 - 19s - loss: 0.0387 - acc: 0.9885 - val_loss: 0.0378 - val_acc: 0.9865
Epoch 10/10
 - 19s - loss: 0.0357 - acc: 0.9892 - val_loss: 0.0369 - val_acc: 0.9886
Accuracy: 0.9886
 Error: 1.1400000000000006
```

0.4.1 Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!