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.

Convolutional Neural Networks with Keras

Objective for this Notebook

1. How to use the Keras library to build convolutional neural networks.

2. Convolutional Neural Network with One Convolutional and Pooling Layers.

3. Convolutional Neural Network with Two Convolutional and Pooling Layers.

Table of Contents

Import Keras and Packages

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

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

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

In [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
```

Convolutional Layer with One set of convolutional and pooling layers

In [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://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 3s 0us/step
```

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

In [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

In [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.

In [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.

In [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))
```

```
Epoch 1/10
300/300 - 10s - loss: 0.2980 - accuracy: 0.9157 - val_loss: 0.1081 - val_accuracy: 0.9689
Epoch 2/10
300/300 - 8s - loss: 0.0902 - accuracy: 0.9741 - val_loss: 0.0666 - val_accuracy: 0.9791
Epoch 3/10
300/300 - 8s - loss: 0.0597 - accuracy: 0.9826 - val_loss: 0.0482 - val_accuracy: 0.9843
Epoch 4/10
300/300 - 8s - loss: 0.0459 - accuracy: 0.9863 - val_loss: 0.0432 - val_accuracy: 0.9864
Epoch 5/10
300/300 - 7s - loss: 0.0377 - accuracy: 0.9888 - val_loss: 0.0393 - val_accuracy: 0.9869
Epoch 6/10
300/300 - 7s - loss: 0.0309 - accuracy: 0.9906 - val_loss: 0.0415 - val_accuracy: 0.9853
Epoch 7/10
300/300 - 7s - loss: 0.0266 - accuracy: 0.9922 - val_loss: 0.0376 - val_accuracy: 0.9871
Epoch 8/10
300/300 - 7s - loss: 0.0218 - accuracy: 0.9927 - val_loss: 0.0376 - val_accuracy: 0.9874
Epoch 9/10
300/300 - 7s - loss: 0.0187 - accuracy: 0.9946 - val_loss: 0.0342 - val_accuracy: 0.9901
Epoch 10/10
300/300 - 8s - loss: 0.0146 - accuracy: 0.9955 - val_loss: 0.0438 - val_accuracy: 0.9857
Accuracy: 0.9857000112533569
 Error: 1.4299988746643066
```

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.

In [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.

In [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))
```

```
Epoch 1/10
300/300 - 11s - loss: 0.4805 - accuracy: 0.8661 - val_loss: 0.1409 - val_accuracy: 0.9582
Epoch 2/10
300/300 - 9s - loss: 0.1167 - accuracy: 0.9659 - val_loss: 0.0790 - val_accuracy: 0.9771
Epoch 3/10
300/300 - 9s - loss: 0.0786 - accuracy: 0.9764 - val_loss: 0.0607 - val_accuracy: 0.9801
Epoch 4/10
300/300 - 9s - loss: 0.0626 - accuracy: 0.9818 - val_loss: 0.0514 - val_accuracy: 0.9826
Epoch 5/10
300/300 - 9s - loss: 0.0516 - accuracy: 0.9842 - val_loss: 0.0487 - val_accuracy: 0.9857
Epoch 6/10
300/300 - 9s - loss: 0.0446 - accuracy: 0.9863 - val_loss: 0.0435 - val_accuracy: 0.9851
Epoch 7/10
300/300 - 9s - loss: 0.0403 - accuracy: 0.9878 - val_loss: 0.0530 - val_accuracy: 0.9810
Epoch 8/10
300/300 - 9s - loss: 0.0368 - accuracy: 0.9889 - val_loss: 0.0498 - val_accuracy: 0.9834
Epoch 9/10
300/300 - 9s - loss: 0.0331 - accuracy: 0.9895 - val_loss: 0.0440 - val_accuracy: 0.9861
Epoch 10/10
300/300 - 9s - loss: 0.0296 - accuracy: 0.9909 - val_loss: 0.0361 - val_accuracy: 0.9880
Accuracy: 0.9879999756813049
 Error: 1.2000024318695068
```

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!

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-21	2.0	Srishti	Migrated Lab to Markdown and added to course repo in GitLab

---

This notebook is part of a course on **Coursera** called *Introduction to Deep Learning & Neural Networks with Keras*. If you accessed this notebook outside the course, you can take this course online by clicking here.
