**Keras Overview**

A sign on a beach near a body of water

Description automatically generated

[Keras](https://faroit.github.io/keras-docs/1.2.1/) makes coding deep neural networks simpler. To demonstrate just how easy it is, you're going to build a simple fully-connected network in a few dozen lines of code.

We’ll be connecting the concepts that you’ve learned in the previous lessons to the methods that Keras provides.

The network you will build is similar to Keras’s [sample network](https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py) that builds out a convolutional neural network for [MNIST](http://yann.lecun.com/exdb/mnist/). However for the network you will build you're going to use a small subset of the [German Traffic Sign Recognition Benchmark](http://benchmark.ini.rub.de/?section=gtsrb&subsection=news) dataset that you've used previously.

The general idea for this example is that you'll first load the data, then define the network, and then finally train the network.

# Neural Networks in Keras

Here are some core concepts you need to know for working with Keras. All Keras exercises will be performed in JUPYTER workspaces, using python 3.5, Tensorflow 1.3, and [**Keras**](https://keras.io/) 2.09. More information on using JUPYTER in workspaces, can be found earlier in the term in the Workspaces lesson.

## Sequential Model

**from** keras.models **import** Sequential

*# Create the Sequential model*

model = Sequential()

The [**keras.models.Sequential**](https://keras.io/models/sequential/) class is a wrapper for the neural network model. It provides common functions like fit(), evaluate(), and compile(). We'll cover these functions as we get to them. Let's start looking at the layers of the model.

**See the documentation for keras.models.Sequential in Keras 2.09**[**here**](https://faroit.github.io/keras-docs/2.0.9/models/sequential/)**.**

## Layers

A Keras layer is just like a neural network layer. There are fully connected layers, max pool layers, and activation layers. You can add a layer to the model using the model's add() function. For example, a simple model would look like this:

**from** keras.models **import** Sequential

**from** keras.layers.core **import** Dense, Activation, Flatten

*# Create the Sequential model*

model = Sequential()

*#1st Layer - Add a flatten layer*

model.add(Flatten(input\_shape=(32, 32, 3)))

*#2nd Layer - Add a fully connected layer*

model.add(Dense(100))

*#3rd Layer - Add a ReLU activation layer*

model.add(Activation('relu'))

*#4th Layer - Add a fully connected layer*

model.add(Dense(60))

*#5th Layer - Add a ReLU activation layer*

model.add(Activation('relu'))

Keras will automatically infer the shape of all layers after the first layer. This means you only have to set the input dimensions for the first layer.

The first layer from above, model.add(Flatten(input\_shape=(32, 32, 3))), sets the input dimension to (32, 32, 3) and output dimension to (3072=32 x 32 x 3). The second layer takes in the output of the first layer and sets the output dimensions to (100). This chain of passing output to the next layer continues until the last layer, which is the output of the model.

## Quiz

In this quiz you will build a multi-layer feedforward neural network to classify traffic sign images using Keras.

1. Set the first layer to a Flatten() layer with the input\_shape set to (32, 32, 3).
2. Set the second layer to a Dense() layer with an output width of 128.
3. Use a ReLU activation function after the second layer.
4. Set the output layer width to 5, because for this data set there are only 5 classes.
5. Use a softmax activation function after the output layer.
6. Train the model for 3 epochs. You should be able to get over 50% training accuracy.

To get started, review the Keras documentation about models and layers. The Keras example of a [**Multi-Layer Perceptron**](https://github.com/fchollet/keras/blob/master/examples/mnist_mlp.py) network is similar to what you need to do here. Use that as a guide, but keep in mind that there are a number of differences.

## Convolutions

1. Build from the previous network.
2. Add a [**convolutional layer**](https://keras.io/layers/convolutional/#convolution2d) with 32 filters, a 3x3 kernel, and valid padding before the flatten layer.
3. Add a ReLU activation after the convolutional layer.
4. Train for 3 epochs again, should be able to get over 50% accuracy.

Hint: The Keras example of a [**convolutional neural**](https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py) network for MNIST would be a good example to review.

## Pooling

1. Build from the previous network
2. Add a 2x2 [**max pooling layer**](https://keras.io/layers/pooling/#maxpooling2d) immediately following your convolutional layer.
3. Train for 3 epochs again. You should be able to get over 50% training accuracy.