CNN Examples Using Keras

CNN Examples Using Keras

What to expect?

In the previous slides, we learned about convolution and convolutional neural networks (CNNs). Now, we will practice building a CNN from scratch using Keras. We will solve two image classification problems. In the first example, we will use the MNIST dataset, which is a simple dataset containing images of handwritten digits from 0 to 9. In the second example, we will use the CIFAR-10 dataset, which is a slightly more complex dataset featuring images of various objects, including birds, ships, deer, and automobiles.

In each example, we will examine some samples from the dataset. Then, we will load the dataset and preprocess it. Next, we will define our model. It is essential to understand what each line of code does in this section. After that, we will compile the model and train it. Finally, we will evaluate the model on the testing dataset. We have encountered this process many times when solving problems using artificial neural networks (ANNs). The main difference here lies in defining the convolutional neural network (CNN).

CNN Examples Using Keras

MaxPooling2d & Conv2d Layers

We know that convolutional layers and max pooling layers are essential for defining a convolutional neural network (CNN). We can add these layers using the Conv2D and MaxPooling2D methods in Keras.
It is important to understand the arguments of these methods, as they are the core components of our CNN model.

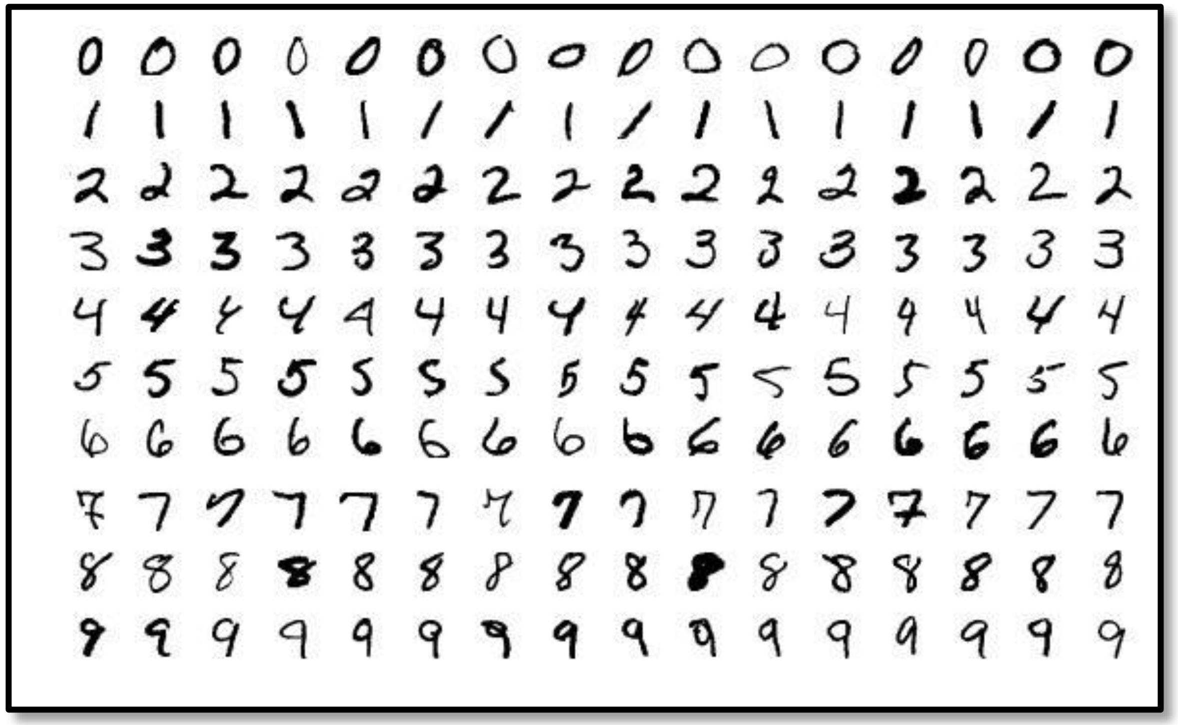
The first number in the Conv2D layer defines the number of filters, followed by the size of the filters. We have set the padding to "same," which means that we will add zero padding to ensure that the spatial dimensions of the output of the layer remain consistent with the input dimensions. We could have used "valid" if we did not want to add zero padding.
The pool size of the MaxPooling2D layer determines the size of the kernel.

```
model.add(Conv2D(32, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

CNN Examples Using Keras

The MNIST dataset

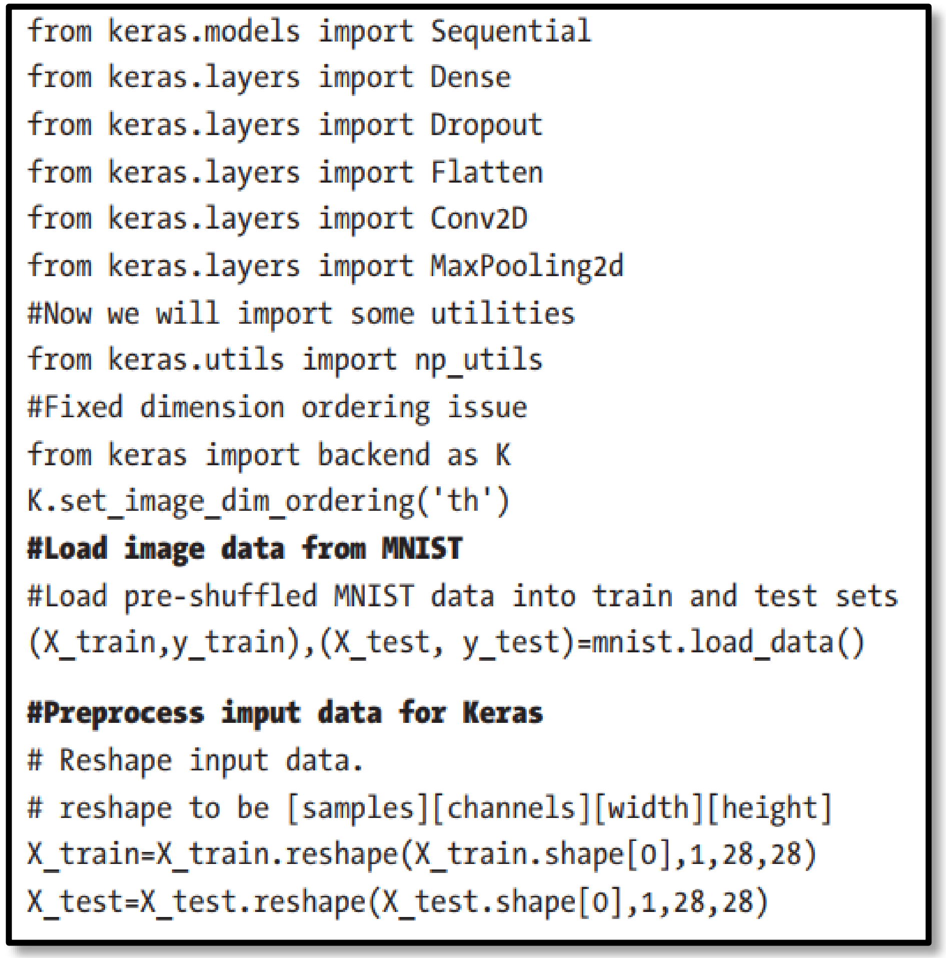
The MNIST dataset contains 60,000 training images and 10,000 testing images. Each image is a 28 by 28 pixel, grayscale picture of a handwritten digit between 0 and 9.
The dataset is fairly large, and the images are simple. Therefore, models usually perform very well on this dataset.



CNN Examples Using Keras

Importing the dataset and preprocessing

First, we need to import the essential modules that we will use to build and test our model. Among the modules we have imported, we can see Conv2D and MaxPooling2D. These are the modules that we will use to build our CNN.
Next, it is time to load the dataset. We will load the training and testing datasets and reshape them.

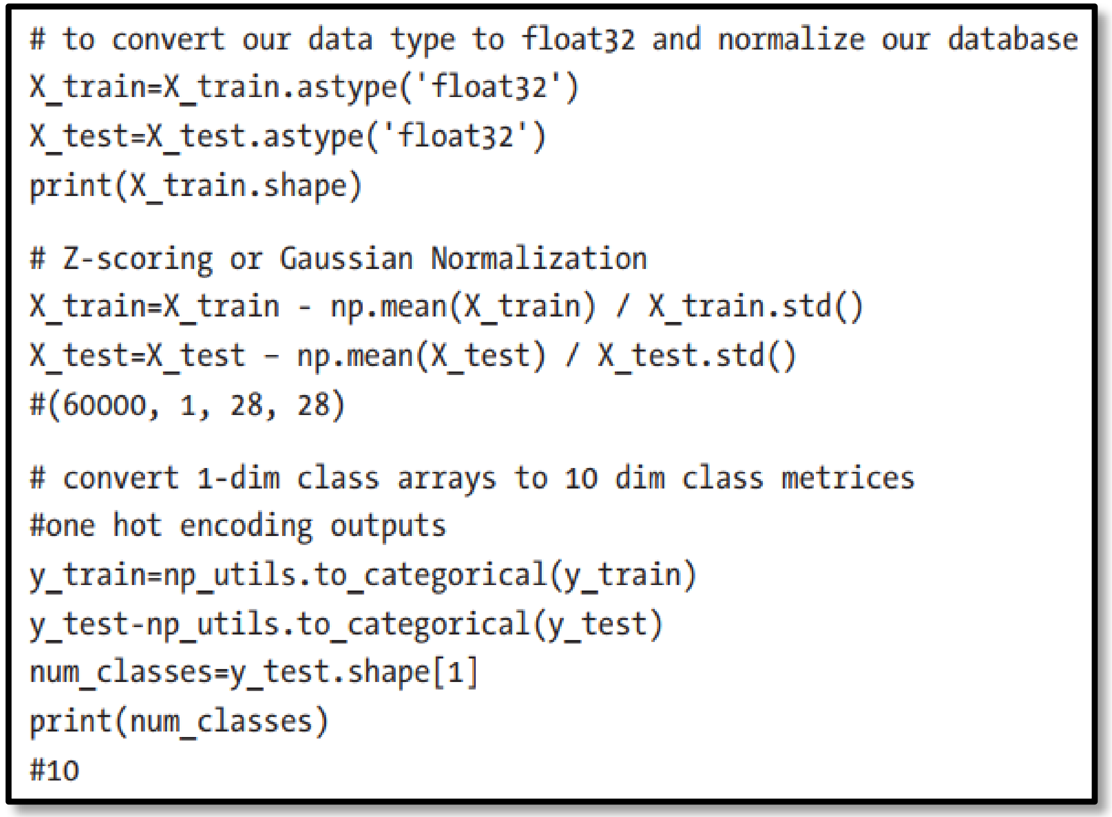


```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2d
#Now we will import some utilities
from keras.utils import np_utils
#Fixed dimension ordering issue
from keras import backend as K
K.set_image_dim_ordering('th')
#Load image data from MNIST
#Load pre-shuffled MNIST data into train and test sets
(X_train,y_train),(X_test, y_test)=mnist.load_data()

#Preprocess imput data for Keras
# Reshape input data.
# reshape to be [samples][channels][width][height]
X_train=X_train.reshape(X_train.shape[0],1,28,28)
X_test=X_test.reshape(X_test.shape[0],1,28,28)
```

CNN Examples Using Keras

Importing the data set and preprocessing

Now, we need to normalize the inputs. First, we convert their type to float to ensure they are in the correct format. Then, we subtract the mean from the inputs and divide them by the standard deviation.
We know that there are 10 classes in the MNIST dataset. Therefore, we need to use one-hot encoding to prepare the labels.
To verify that everything has gone as expected, we print the number of classes after all the preprocessing.



```
# to convert our data type to float32 and normalize our database
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')
print(X_train.shape)

# Z-scoring or Gaussian Normalization
X_train=X_train - np.mean(X_train) / X_train.std()
X_test=X_test - np.mean(X_test) / X_test.std()
#(60000, 1, 28, 28)

# convert 1-dim class arrays to 10 dim class metrices
#one hot encoding outputs
y_train=np_utils.to_categorical(y_train)
y_test-np_utils.to_categorical(y_test)
num_classes=y_test.shape[1]
print(num_classes)
#10
```

CNN Examples Using Keras

Building the model

In this section, we will build our model. We will have one convolutional layer with 32 filters, and the size of the filter we are using is 5 by 5. We will use ReLU as the activation function. This layer is followed by a max pooling layer, and we will also add dropout as a regularization method.

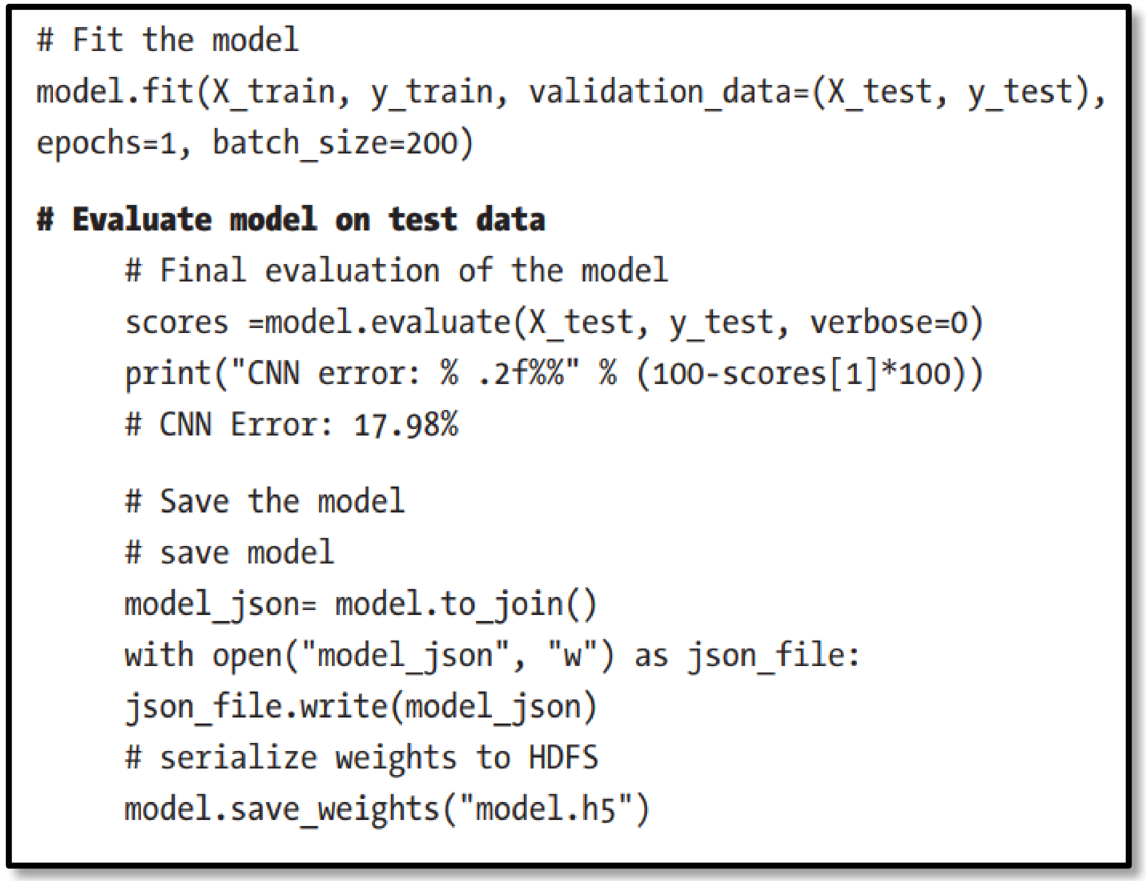
Next, we flatten the layer and add our fully connected layers on top of it. We will keep it simple and only add one layer with 240 neurons. Finally, we will have our output layer, which will use SoftMax as its activation function. Now, we compile the model, using Adagrad as our optimizer.

```
# create a model
     model=Sequential()
     model.add(Conv2D(32, (5,5), input_shape=(1,28,28),
     activation='relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(Dropout(0.3))      # Dropout, one form of
     regularization
     model.add(Flatten())
     model.add(Dense(240,activation='elu'))
     model.add(Dense(num_classes, activation='softmax'))
     print(model.output_shape)
     (None, 10)

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adagrad',
matrices=['accuracy'])
```

CNN Examples Using Keras

Evaluating the model

The model has been compiled with random weights and biases. We need to train it using our training data. We will set the batch size to 200 and train the model for one epoch, just for demonstration purposes. We can use a higher number of epochs to achieve better results.
Next, it is time to evaluate the model. We will use the testing dataset to evaluate the model and print the misclassification percentage. Finally, we can save the model to load it whenever we need it.

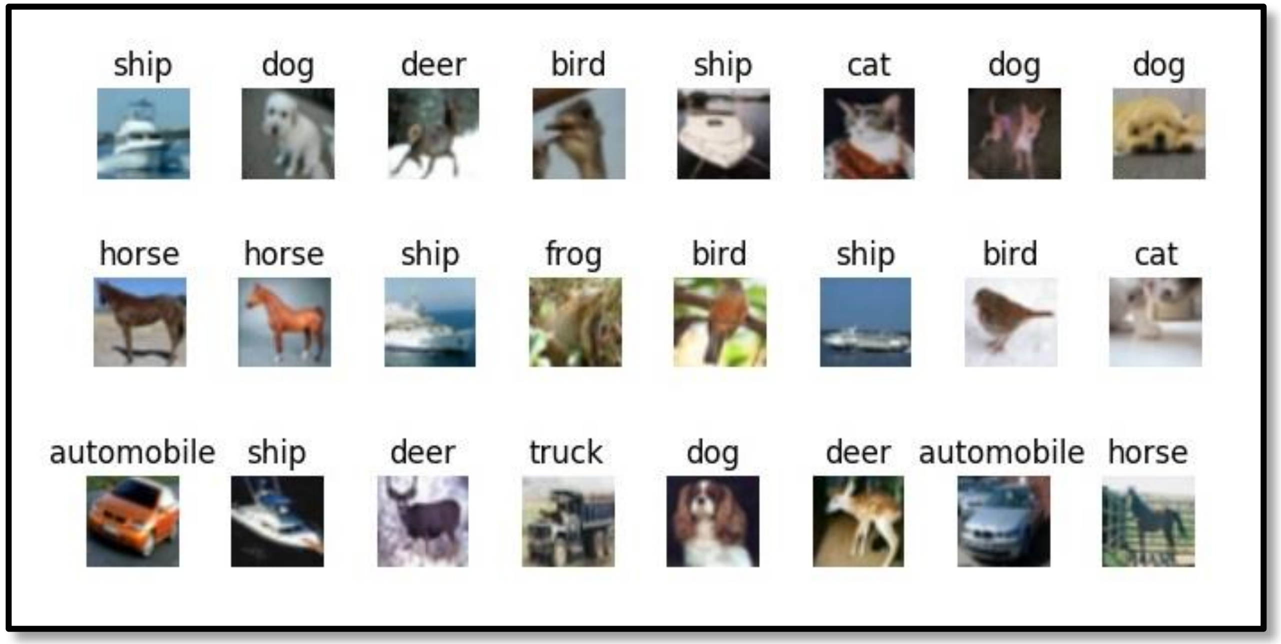


```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=1, batch_size=200)

# Evaluate model on test data
    # Final evaluation of the model
    scores =model.evaluate(X_test, y_test, verbose=0)
    print("CNN error: % .2f%%" % (100-scores[1]*100))
    # CNN Error: 17.98%

    # Save the model
    # save model
    model_json= model.to_join()
    with open("model_json", "w") as json_file:
    json_file.write(model_json)
    # serialize weights to HDFS
    model.save_weights("model.h5")
```

CNN Examples Using Keras

The CIFAR-10 data set

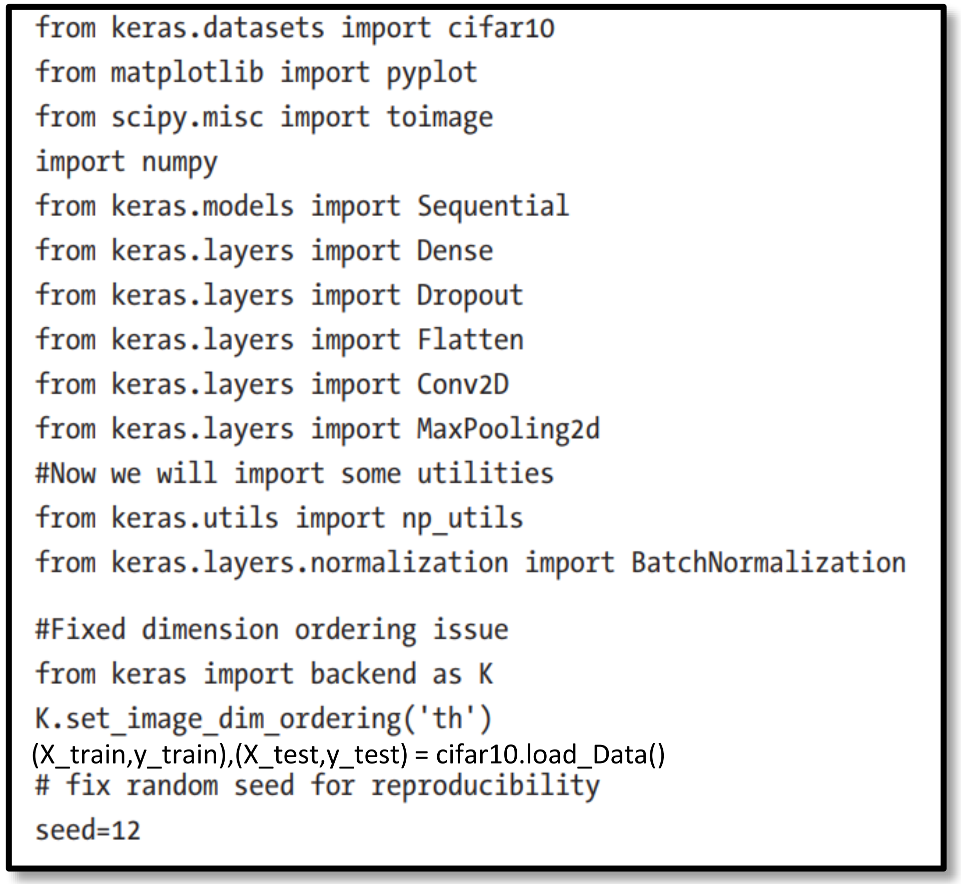
The CIFAR-10 dataset contains 50,000 training images and 10,000 testing images. Each image is a 32 by 32 pixel, RGB picture of various objects, including animals such as birds and deer, as well as means of transportation like ships and automobiles. In total, there are 10 classes in this dataset.
This dataset is more sophisticated than the MNIST dataset; therefore, we will use a slightly more complicated convolutional neural network (CNN).



CNN Examples Using Keras

Importing the data set and preprocessing

First, we need to import the essential modules that we are going to use to build and test our model. We are going to import Conv2d and the MaxPooling2d again as these are very useful when it comes to defining a CNN in Keras.
We are going to import the CIFAR-10 data set. The process is exactly like the previous example where we imported the MNIST data set. Then, we are going to set the random seed for reproducibility.

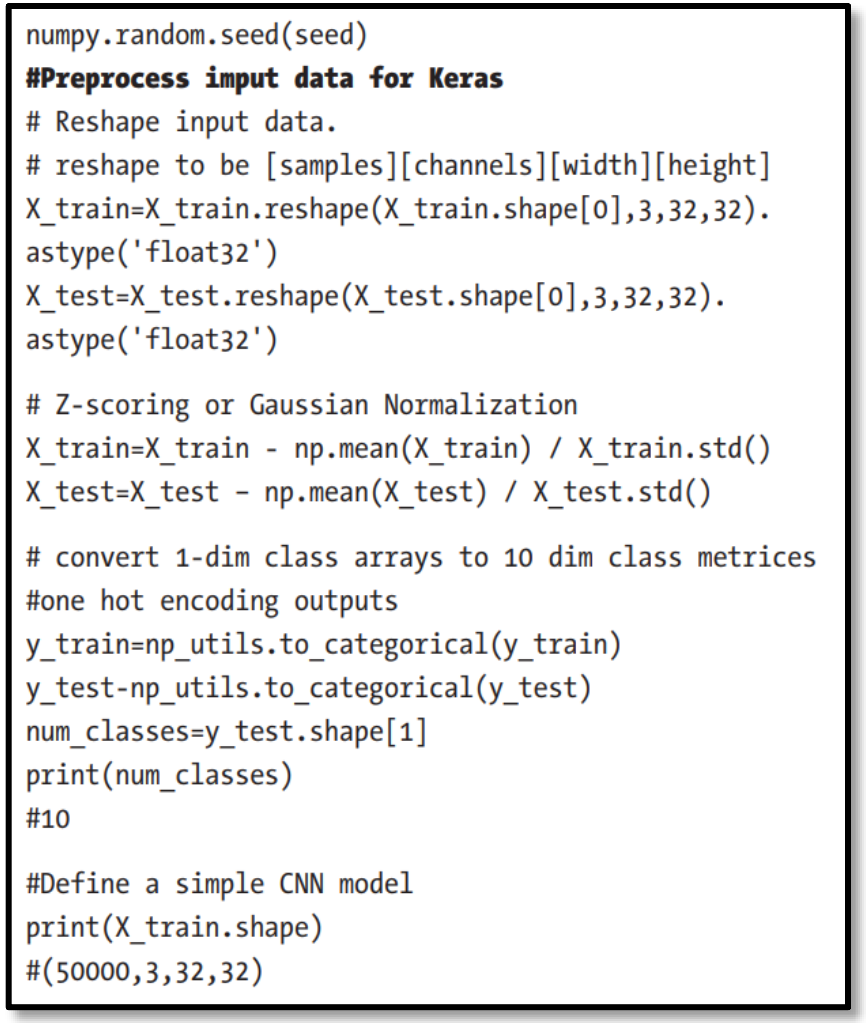


```
from keras.datasets import cifar10
from matplotlib import pyplot
from scipy.misc import toimage
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2d
#Now we will import some utilities
from keras.utils import np_utils
from keras.layers.normalization import BatchNormalization

#Fixed dimension ordering issue
from keras import backend as K
K.set_image_dim_ordering('th')
(X_train,y_train),(X_test,y_test) = cifar10.load_Data()
# fix random seed for reproducibility
seed=12
```

CNN Examples Using Keras

Importing the data set and preprocessing

We are going to reshape the inputs to the appropriate shape. Then, we are going to Normalize the inputs. So, we subtract the mean from the inputs and divide them by the standard deviation.
Now, we need to prepare our labels. We have 10 classes in the CIFAR-10 data set. We are going to use one hot encoding to prepare the labels for the training and testing process. Again, just like what we did for the MNIST data set, we are going to print the number of classes to make sure everything has gone as we expected.

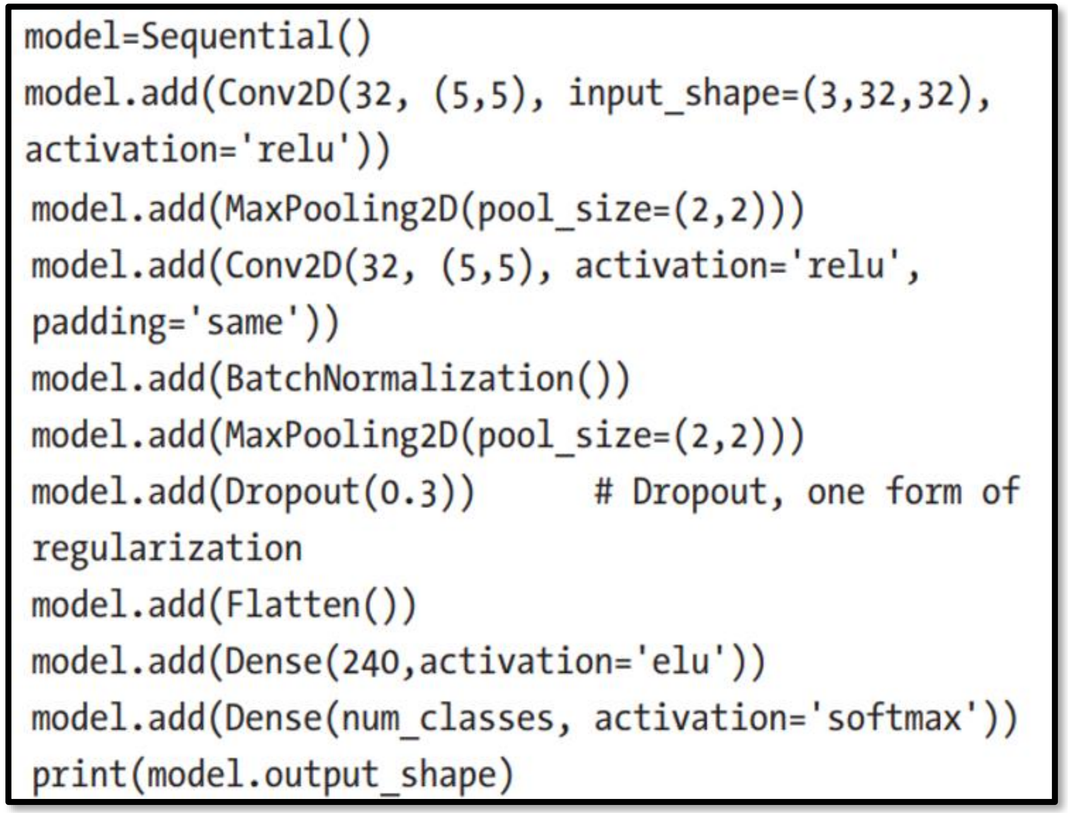


```
numpy.random.seed(seed)
#Preprocess imput data for Keras
# Reshape input data.
# reshape to be [samples][channels][width][height]
X_train=X_train.reshape(X_train.shape[0],3,32,32).
astype('float32')
X_test=X_test.reshape(X_test.shape[0],3,32,32).
astype('float32')

# Z-scoring or Gaussian Normalization
X_train=X_train - np.mean(X_train) / X_train.std()
X_test=X_test - np.mean(X_test) / X_test.std()

# convert 1-dim class arrays to 10 dim class metrices
#one hot encoding outputs
y_train=np_utils.to_categorical(y_train)
y_test-np_utils.to_categorical(y_test)
num_classes=y_test.shape[1]
print(num_classes)
#10

#Define a simple CNN model
print(X_train.shape)
#(50000,3,32,32)
```

CNN Examples Using Keras

Building the model

In this example, we will use two convolutional layers. Each of these layers will have 32 filters of size 5 by 5. Both convolutional layers are followed by a max pooling layer. This will result in a slightly more complex model compared to our previous model, allowing the new model to extract features more effectively.
To avoid overfitting, we will use a batch normalization layer before the fully connected layers, and we will also use dropout.
Finally, we will add a fully connected layer with 240 neurons, followed by a layer with 10 neurons that uses SoftMax activation.



```
model=Sequential()
model.add(Conv2D(32, (5,5), input_shape=(3,32,32),
activation='relu'))
 model.add(MaxPooling2D(pool_size=(2,2)))
 model.add(Conv2D(32, (5,5), activation='relu',
 padding='same'))
 model.add(BatchNormalization())
 model.add(MaxPooling2D(pool_size=(2,2)))
 model.add(Dropout(0.3))      # Dropout, one form of
 regularization
 model.add(Flatten())
 model.add(Dense(240,activation='elu'))
 model.add(Dense(num_classes, activation='softmax'))
 print(model.output_shape)
```

CNN Examples Using Keras

Evaluating the model

After defining the model, we will compile it, using Adagrad as our optimizer.
We will train our model with the training data for 1 epoch. Training the model for more epochs will likely result in better accuracy. When the training is complete, it is time to evaluate our model.

```
print(model.output_shape)
model.compile(loss='binary_crossentropy', optimizer='adagrad')
# fit model
model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=1, batch_size=200)

# Final evaluation of the model
scores =model.evaluate(X_test, y_test, verbose=0)
print("CNN error: % .2f%%" % (100-scores[1]*100
```