

Artificial Intelligence and machine learning

Chapter 5 : Training your own classification model using Keras

I. Introduction

Keras is a high-level neural networks API, written in Python and capable of running on top of [TensorFlow](#), [CNTK](#), or [Theano](#). It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

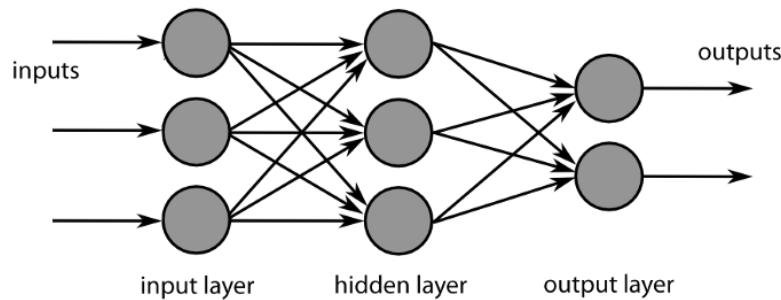
Read the documentation at [Keras.io](#).

Keras is compatible with: **Python 2 and python 3**

Guiding principles

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python.** No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

II. A Simple Animal classifier from scratch using Keras



In this chapter, we will show you how to make a simple animal classifier using keras.

Prerequisite

Basics of convolutional neural network

Step 1 — Install libraries

For this project you will need following python libraries installed on your machine-

Keras, Open cv, Pillow (For image manipulation), Numpy

Step 2 — Prepare Dataset

I downloaded nearly 10000 photos each for cat, dog, categories. You can download photos in batch using some great chrome extensions. In my case I used Download All Images extension in Chrome.

Now I will tell you about the programming part that I used to make the images in the right format as required by model.

Import Image and numpy library

```
from PIL import Image
```

```
import numpy as np
```

Code for making images into array -

```
data=[]
labels=[]
cats=os.listdir("cats")
for cat in cats:
    imag=cv2.imread("cats/"+cat)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
```

```
data.append(np.array(resized_image))
labels.append(0)
dogs=os.listdir("dogs")
for dog in dogs:
    imag=cv2.imread("dogs/"+dog)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(1)
```

“data” is the array of all images converted to numpy array and “labels” is the array of corresponding labels.

How it works —

We go into each directory and go through all the files inside it. Then we read the images using cv2.imread. After that using Image object from PIL we convert the image into array. Since while training the convolutional neural network it is required that you have images of same size we resize the images to width and height of 50px. Then we convert it to numpy array just by passing the image array in the function np.array() and we append the numpy array in our data array. Also add corresponding label to the image. Eg for cat label is 0, for dog label is 1 and so on. Labels are required as the training is done in supervised manner.

Since the “data” and “labels” are normal array , convert them to numpy arrays-

```
animals=np.array(data)
labels=np.array(labels)
```

Now save these numpy arrays so that you dont need to do this image manipulation again.

```
np.save("animals",animals)
np.save("labels",labels)
```

Load the arrays (Optional : Required only if you have closed your jupyter notebook after saving numpy array)

```
animals=np.load("animals.npy")
labels=np.load("labels.npy")
```

Now shuffle the “animals” and “labels” set so that you get good mixture when you separate the dataset into train and test

```
s=np.arange(animals.shape[0])
np.random.shuffle(s)
animals=animals[s]
labels=labels[s]
```

Make a variable num_classes which is the total number of animal categories and a variable data_length which is size of dataset

```
num_classes=len(np.unique(labels))
data_length=len(animals)
```

Divide data into test and train

Take 90% of data in train set and 10% in test set

```
(x_train,x_test)=animals[(int)(0.1*data_length):],animals[::(int)(0.1*data_length)]x_train =
x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
train_length=len(x_train)
test_length=len(x_test)
```

Divide labels into test and train

```
(y_train,y_test)=labels[(int)(0.1*data_length):],labels[::(int)(0.1*data_length)]
```

Make labels into One Hot Encoding

```
import keras
from keras.utils import np_utils
#One hot encoding
y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
```

Conratulations !! Your data preparation part is over

Step 3 — Making Keras model

It is a very easy process to make your deep learning model. Thanks to the great Keras library. All you have to do is think about hyper parameters like Filter size, number of filters, which type of padding to use, which activation functions to use etc.

Here is my Keras model-

```
# import sequential model and all the required layers
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
#make model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(50,50,3)))model.a
dd(MaxPooling2D(pool_size=2))model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu
"))model.add(MaxPooling2D(pool_size=2))model.add(Conv2D(filters=64,kernel_size=2,padding="same",activa
tion="relu"))model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(4,activation="softmax"))
model.summary()
```

I added 3 pairs of Conv2D layer and Maxpool2D layer with increasing filter sizes (16,32 ,64) . This helps to make image grow more in depthwise and become more flatten. Maxpool layers

are great as they optimize the training time. I have also add Dropout layers to reduce overfitting.

In final Dense layer there are 4 nodes because we have 4 categories of animals (cat,dog,bird,fish). Softmax activation is used to give scores to these categories which lie between 0 and 1.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 50, 50, 16)	208
max_pooling2d_1 (MaxPooling2D)	(None, 25, 25, 16)	0
conv2d_2 (Conv2D)	(None, 25, 25, 32)	2080
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	8256
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 500)	1152500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 4)	2004
Total params: 1,165,048		
Trainable params: 1,165,048		
Non-trainable params: 0		

You will get following model summary after running above code

Compile the model

We use loss function as categorical_crossentropy and Adam optimizer

compile the model

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

Step 4 — Train the model

Congrats on reaching upto this step. Your all work is over. Now just train the model and wait patiently.

```
model.fit(x_train,y_train,batch_size=50
        ,epochs=100,verbose=1)
```

“epochs” is the number of steps you want to train the model. Batch size is the size of dataset that is thrown to the model at a time. It is really important to set this because if you have a lower memory problem in your computer you cant train the model by throwing all the data at once.

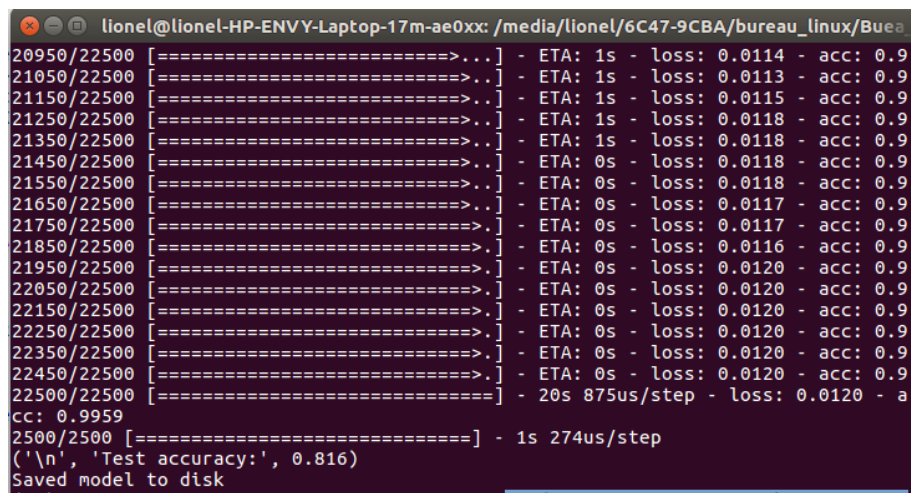
After running above snippet you can it running —

Step 5 — Test the model

Use model.evaluate to see how model work on test set

```
score = model.evaluate(x_test, y_test, verbose=1)
print('\n', 'Test accuracy:', score[1])
```

I got an accuracy of 81.6% on the test set



```
lionel@lionel-HP-ENVY-Laptop-17m-ae0xx: /media/lionel/6C47-9CBA/bureau_linux/Buea
20950/22500 [=====>...] - ETA: 1s - loss: 0.0114 - acc: 0.9
21050/22500 [=====>...] - ETA: 1s - loss: 0.0113 - acc: 0.9
21150/22500 [=====>...] - ETA: 1s - loss: 0.0115 - acc: 0.9
21250/22500 [=====>...] - ETA: 1s - loss: 0.0118 - acc: 0.9
21350/22500 [=====>...] - ETA: 1s - loss: 0.0118 - acc: 0.9
21450/22500 [=====>...] - ETA: 0s - loss: 0.0118 - acc: 0.9
21550/22500 [=====>...] - ETA: 0s - loss: 0.0118 - acc: 0.9
21650/22500 [=====>...] - ETA: 0s - loss: 0.0117 - acc: 0.9
21750/22500 [=====>...] - ETA: 0s - loss: 0.0117 - acc: 0.9
21850/22500 [=====>...] - ETA: 0s - loss: 0.0116 - acc: 0.9
21950/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22050/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22150/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22250/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22350/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22450/22500 [=====>...] - ETA: 0s - loss: 0.0120 - acc: 0.9
22500/22500 [=====] - 20s 875us/step - loss: 0.0120 - a
cc: 0.9959
2500/2500 [=====] - 1s 274us/step
('\n', 'Test accuracy:', 0.816)
Saved model to disk
```

Step 6 — Predicting on single images

If you want to predict on a single image use this code-

```
def convert_to_array(img):
    im = cv2.imread(img)
    img = Image.fromarray(im, 'RGB')
    image = img.resize((50, 50))
    return np.array(image)
def get_animal_name(label):
    if label==0:
        return "cat"
    if label==1:
        return "dog"

def predict_animal(file):
    print("Predicting .....")
    ar=convert_to_array(file)
    ar=ar/255
    label=1
    a=[]
    a.append(ar)
```

```
a=np.array(a)
score=model.predict(a,verbose=1)
print(score)
label_index=np.argmax(score)
print(label_index)
acc=np.max(score)
animal=get_animal_name(label_index)
print(animal)
print("The predicted Animal is a "+animal+" with accuracy = "+str(acc))
```

Now run the predict_animal function on the image. Make sure your image is in same directory as your code.

```
predict_animal("animal1.jpg")
```

```
Loaded model from disk
Predicting .....
1/1 [=====] - 0s 43ms/step
[[1.5504133e-04 9.9984491e-01]]
1
dog
The predicted Animal is a dog with accuracy = 0.9998449
```

```
predict_animal("animal3.jpg")
```

```
Loaded model from disk
Predicting .....
1/1 [=====] - 0s 42ms/step
[[0.9948887 0.00511135]]
0
cat
The predicted Animal is a cat with accuracy = 0.9948887
```

Congratulations you have successfully build your image classifier !!

Step 7 — Save the model

It is important to save the trained model as if you dont you have to train it again. So save it and then use it when required. It will basically save the weights in a file. Use below code to save-

```
from keras.models import model_from_json# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

