All the Steps to Build your first Image Classifier (with code)

From creating datasets to testing your program accuracy

Arthur Arnx Mar 1, 2019 · 6 min read ★

Photo by Temple Cerulean on Unsplash

If you want to create an image classifier but have no idea where to start, follow this quick guide to understand the concepts and be able to train a convolutional neural network to recognize any image you want !

To achieve that, the code provided is written in Python (3.x), and we will mainly use the Keras library.

· · ·

First of all, if you have no idea what a neural network is, I can only encourage you to discover this concept with a quick guide. For example, here is the last one I wrote about that :

First neural network for beginners explained (with code)

Understand and create a Perceptron

towardsdatascience.com

Now, we will focus on the convolutional neural network, which keeps the same idea about columns and neurons, inputs and outputs, while simply adding a way to extract information in an image.

. . .

What is a convolutional neural network ?

This type of neural network consists of a deep neural network preceded by some operations.

> *Overall, keep in mind that an image is just a matrix of numbers, of dimension 2 if the image is only in gray level, and dimension 3 if it contains colors (the third dimension is for all RGB levels).*

First of all, when an image is given to the algorithm, it starts by applying a small filter on the initial image and takes it everywhere on it. This step is called Convolution.

Figure 1 — Convolution of a 5×5 image by a 3×3 filter

In Figure 1, the initial image is green, the filter is yellow and multiplies every number of the initial image by the corresponding filter's one.

After this operation, a new matrix (red) is obtained. By comparing pixels of the red matrix to a model, the program can determine if there is or not an object corresponding to a model on the first image.

For example, obtaining big numbers only on a line of pixels means that the initial image contains a line there.

The next step is called Pooling. It is about taking the highest value of each region and form a new matrix using only those values. It reduces the spatial dimension of the matrix and so helps the neural networks to operates quicker.

Figure 2 — Example of Pooling operation

In Figure 2, you can see that the dimension of the image is divided in 4 parts, with each one attributing its highest value. The new formed image is smaller.

Also, an activation function is used during the process to normalize all the values obtained. In the example below, we will be using ReLU.

Finally, a last step may be used to increase the accuracy, and is called Dropout. It forces a neural network to randomly disabling some neurons in the learning phase. We will implement this function in our example as well.

. . .

Now that you know the basics of the convolution, we can start building one !

Preparing the data

This part is useful only if you want to use your own data, or data that can't be found on the web easily, to build a convolutional neural network maybe more adapted to your needs. Otherwise, here is the code to directly use datasets from Keras :

```
from keras.datasets import mnist #replace mnist with any dataset

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Here, we simply call the function load_data to set the dataset for training and testing phase. You can replace "mnist" by any dataset you want to use (change it in both lines).

If you want to create your own dataset, here are the steps :

First of all, you will need to collect a lot of images. The more there are, the better. Remember to keep approximately the same amount of image for each class. For example, for my piece of 2D chess classifier, I had 160 images for each possible piece (and the empty case), so about **2,000 images** in total (which is not that much) but the size of the dataset depends on the projects (my 2D pieces always have the same aspects, while cats have a lot of breeds, different sizes, different postures, …).

There is also a powerful tool to help you creating more data, called **data augmentation**. It simply modifies an image and gives back plenty of new and unique images, all based on the first one, by flipping, rotating or cropping it.

Figure 3 — Example of a folder tree

In the end make sure that all your data is classified in a folder meant for that purpose, in which every class has its own subfolder.

Now in the main folder, we will create a python program to set up all the data.

Preparing the data

In line 14, you can change the list to any classes you need, but keep the same names that you used for the subfolders earlier.

Finally, after running the program, the data are setup in files and ready to be used.

. . .

Building the convolutional neural network

The convolutional neural network

If you decided to use an imported dataset, replace lines 9 & 10 by what we saw earlier, and the line 44 by :

```
model.fit(x_train, y_train, batch_size=32, epochs=40, verbose=1,
validation_data=(x_test, y_test))
```

In line 37, modify the parameter of **Dense()** to the number of classes you have. This is the number of possible output by the neural network.

For every convolutional layers, you can see that we always firstly add it with its number of neurons and filter size. Then, we involve the activation function, and finally use the Pooling method. We also added a Dropout in line 30 to see how to do it.

Also, before the first "normal" hidden layer, we added the function **Flatten()**, that transforms all information from previous convolutions into inputs for neurons. At this point, the rest simply contains layers and neurons from basic neural network.

Here it is, you built your own classifier !

. . .

Predicting an image class

Now, you can use your neural network to predict any image you want. Simply make a little script involving these few lines :

```
import cv2
import tensorflow as tf

CATEGORIES = ["bishopB", "bishopW", "empty", "kingB", "kingW",
                         "knightB", "knightW", "pawnB", "pawnW",
                         "queenB", "queenW", "rookB", "rookW"]

def prepare(file):
    IMG_SIZE = 50
    img_array = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

model = tf.keras.models.load_model("CNN.model")
image = "test.jpg" #your image path
prediction = model.predict([image])
prediction = list(prediction[0])
print(CATEGORIES[prediction.index(max(prediction))])
```

The function prepare(file) allows us to use an image of any size, since it automatically resize it to the image size we defined in the first program.

If you modified the image size in the data program, modify it here too.

. . .

That's it ! You just built your own image classifier adapted to your own images. Of course, do not hesitate to modify any line of code you see, since your neural network accuracy may vary a lot according to those parameters. Here is a non-exhaustive about those :

- The model : You can easily add or remove some layers in your neural network, change the number of neurons, or even the activation functions. You have a model for anything you would like to add.
- The data : The obtained accuracy isn't what you expected ? Maybe you could add more data and mainly verify that all your images are stored in their good folder.
- IMG_SIZE : defined in the program for the dataset, it characterizes the size of the images the network will work on. Don't try a too big number, since high quality images lead to a longer training phase. Moreover, even well-known databases such as MNIST contain very little images (28x28 for MNIST). Don't forget to also modify the IMG_SIZE of the reshaping function in the last program.
- New parameters such as callbacks used with Keras. The one called "EarlyStopping" may help you to improve the length of the training phase, and mainly avoid overfitting.

. . .

With this guide, we covered just enough for you to create and understand your first convolutional neural network. There are many other parameters or aspects that you could discover if you want, so don't hesitate to go further.

> Thanks for reading !
>
> I hope this little guide was useful, if you have any question and/or suggestion, let me know in the comments.

Machine Learning · Computer Vision · Neural Networks · Artificial Intelligence · Guides And Tutorials