IRJSE International Research Journal of Science & Engineering
Int. Res. J. of Science & Engineering, 2020; Special Issue A7: 834-838 SJIF Impact Factor 6.68
ISSN: 2322-0015
RESEARCH ARTICLE **OPEN ACCESS**

Convolutional Neural Network for Image Classification.

Ghori Md. Atheeq Sultan

Assistant Professor, Department of Computer Science & Engineering Telangana University, Nizamabad, Telangana State, India.

Manuscript Details
Available online on http://www.irjse.in
ISSN: 2322-0015

Cite this article as:
Ghori Md. Atheeq Sultan .Convolutional Neural Network for Image Classification., *Int. Res. Journal of Science & Engineering,* February, 2020, Special Issue A7 : 834-838.

ABSTRACT

We will be building a convolutional neural network that will be trained on few thousand images of cats and dogs, and later be able to predict if the given image is of a cat or a dog. In this it will be solving an image classification problem, where our goal will be to tell which class the input image belongs to. The way we are going to achieve it is by training an artificial neural network on few thousand images of cats and dogs and make the NN (Neural Network) learn to predict which class the image belongs to, next time it sees an image having a cat or dog in it.

Keywords: Convolutional Neural Network, Pooling Flattening.

INTRODUCTION

The key thing to understand while following this article is that the model we are building now can be trained on any type of class you want, i am using cat and dog only as a simple example for making you understand how convolutional neural networks work. For example, if there are any doctors reading this, after completing this article they will be able to build and train neural networks that can take a brain scan as an input and predict if the scan contains a tumour or not.

So coming to the coding part, we are going to use deep learning library in python to build our CNN (Convolutional Neural Network). Before we jump into building the model, i need you to download all the required training and test dataset by going into this drive by clicking,

download both the folders named "test set" and "training set" into your working directory, it may take a while as there are 10,000 images in both folders, which is the training data as well as the test dataset. Make sure to create a new directory and name it "whatever_you_want" and paste the above downloaded dataset folders into it.

Let's see what does the folders you just downloaded have in them. First, the folder "**training set**" contains two sub folders **cats** and **dogs**, each holding 8000 images of the respective category. Second, the folder "**test set**" contains two sub folders **cats** and **dogs**, each holding 2000 images of respective category.

The process of building a Convolutional Neural Network always involves four major steps.

Step - 1 : Convolution
Step - 2 : Pooling
Step - 3 : Flattening
Step - 4 : Full connection

We will be going through each of the above operations while coding our neural network. So first go to your working directory and create a new file and name it as "whatever_you_want".py , but I am going to refer to that file as cnn.py, where 'cnn' stands for Convolutional Neural Network and '.py' is the extension for a python file. You will be appending whatever code I write below to this file.

Let's Code:
First let us import all the required keras packages using which we are going to build our CNN, make sure that every package is installed properly in your machine, there is two ways os using keras, i.e Using Tensorflow backend and by Using Theano backend, but don't worry, all the code remains the same in either cases. I tested the below code using Tensorflow backend.

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Let us now see what each of the above packages are imported for :

In **line 1**, we've imported Sequential from keras.models, to initialise our neural network model as a sequential network. There are two basic ways of initialising a neural network, either by a sequence of layers or as a graph.

In **line 2**, we've imported Conv2D from keras.layers, this is to perform the convolution operation i.e the first step of a CNN, on the training images. Since we are working on images here, which a basically 2 Dimensional arrays, we're using Convolution 2-D, you may have to use Convolution 3-D while dealing with videos, where the third dimension will be time.

In **line 3**, we've imported MaxPooling2D from keras.layers, which is used for pooling operation, that is the step—2 in the process of building a cnn. For building this particular neural network, we are using a Maxpooling function, there exist different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling we need the maximum value pixel from the respective region of interest.

In **line 4**, we've imported Flatten from keras.layers, which is used for Flattening. Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

And finally in **line 5**, we've imported Dense from keras.layers, which is used to perform the full connection of the neural network, which is the step 4 in the process of building a CNN.

Now, we will create an object of the sequential class below: classifier = Sequential()

Let us now code the Convolution step, you will be surprised to see how easy it is to actually implement these complex operations in a single line of code in python, thanks to Keras. classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

Let's break down the above code function by function. We took the object which already has an idea of how our neural network is going to be(Sequential), then we

added a convolution layer by using the "Conv2D" function. The Conv2D function is taking 4 arguments, the first is the number of filters i.e 32 here, the second argument is the shape each filter is going to be i.e 3x3 here, the third is the input shape and the type of image(RGB or Black and White)of each image i.e the input image our CNN is going to be taking is of a 64x64 resolution and "3" stands for RGB, which is a colour img, the fourth argument is the activation function we want to use, here 'relu' stands for a rectifier function.

Now, we need to perform pooling operation on the resultant feature maps we get after the convolution operation is done on an image. The primary aim of a pooling operation is to reduce the size of the images as much as possible. In order to understand what happens in these steps in more detail you need to read few external resources. But the key thing to understand here is that we are trying to reduce the total number of nodes for the upcoming layers.

```
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

We start by taking our classifier object and add the pooling layer. We take a 2x2 matrix we'll have minimum pixel loss and get a precise region where the feature are located. Again, to understand the actual math behind Pooling, i suggest you to go learn from an external source, this tutorial concentrates more on the implementation part. We just reduced the complexity of the model without reducing it's performance.

It's time for us to now convert all the pooled images into a continuous vector through Flattening. Flattening is a very important step to understand. What we are basically doing here is taking the 2-D array, i.e pooled image pixels and converting them to a one dimensional single vector.

```
classifier.add(Flatten())
```

The above code is pretty self-explanatory. We've used flatten function to perform flattening, we no need to add any special parameters, keras will understand that the "classifier" object is already holding pooled image pixels and they need to be flattened.

In this step we need to create a fully connected layer, and to this layer we are going to connect the set of nodes we got after the flattening step, these nodes will act as an input layer to these fully-connected layers. As this layer will be present between the input layer and output layer, we can refer to it a hidden layer.

```
classifier.add(Dense(units = 128, activation = 'relu'))
```

As you can see, Dense is the function to add a fully connected layer, 'units' is where we define the number of nodes that should be present in this hidden layer, these units value will be always between the number of input nodes and the output nodes but the art of choosing the most optimal number of nodes can be achieved only through experimental tries. Though it's a common practice to use a power of 2. And the activation function will be a rectifier function.

Now it's time to initialise our output layer, which should contain only one node, as it is binary classification. This single node will give us a binary output of either a Cat or Dog.

```
classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

You can observe that the final layer contains only one node, and we will be using a sigmoid activation function for the final layer.

Now that we have completed building our CNN model, it's time to compile it.

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

From above :

- Optimizer parameter is to choose the stochastic gradient descent algorithm.
- Loss parameter is to choose the loss function.
- Finally, the metrics parameter is to choose the performance metric.

It's time to fit our CNN to the image dataset that you've downloaded.But before we do that, we are going to pre-process the images to prevent over-fitting. Overfitting is when you get a great training

accuracy and very poor test accuracy due to overfitting of nodes from one layer to another.

So before we fit our images to the neural network, we need to perform some image augmentations on them, which is basically synthesising the training data. We are going to do this using keras.preprocessing library for doing the synthesising part as well as to prepare the training set as well as the test test set of images that are present in a properly structured directories, where the directory's name is take as the label of all the images present in it. For example : All the images inside the 'cats' named folder will be considered as cats by keras.

```
train_datagen = ImageDataGenerator(rescale = 1./255,
shear_range= 0.2,
zoom_range = 0.2,
horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set =
train_datagen.flow_from_directory('training_set',
target_size = (64, 64),
batch_size = 32,
class_mode = 'binary')
test_set = test_datagen.flow_from_directory('test_set',
target_size = (64, 64),
batch_size = 32,
class_mode = 'binary')
```

You can find the explanation of what each of the above parameters do here**,** in the keras documentation page. Butwhat you need to understand as a whole of whats happening above is that we are creating synthetic data out of the same images by performing different type of operations on these images like flipping, rotating, blurring, etc.

Now lets fit the data to our model !

```
classifier.fit_generator(training_set,
steps_per_epoch = 8000,
epochs = 25,
validation_data = test_set,
validation_steps = 2000)
```

In the above code, 'steps_per_epoch' holds the number of training images, i.e the number of images the training_set folder contains.

And 'epochs', A single epoch is a single step in training a neural network; in other words when a neural network is trained on every training samples only in one pass we say that one epoch is finished. So training process should consist more than one epochs.In this case we have defined 25 epochs.

Making new predictions from our trained model :

```
import numpy as np
from keras.preprocessing import image
test_image =
image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
prediction = 'dog'
else:
prediction = 'cat'
```

CONCLUSION

The test_image holds the image that needs to be tested on the CNN. Once we have the test image, we will prepare the image to be sent into the model by converting its resolution to 64x64 as the model only excepts that resolution. Then we are using predict() method on our classifier object to get the prediction. As the prediction will be in a binary form, we will be receiving either a 1 or 0, which will represent a dog or a cat respectively.

Conflicts of interest: The authors stated that no conflicts of interest.

REFERENCES

1. Mansingh Gunjan, Reichgelt Han and Kweku-Muata Osei Bryson. CPEST: An expert system for the management of pests and diseases in the Jamaican coffee industry. Expert Systems with Applications, Vol. 32(1):184–192, January 2007.
2. Mahaman, B.D., Passam, H.C., Sideridis, A.B. and C.P Yialouris. DIARES-IPM: a diagnostic advisory

rule-based expert system for integrated pest management in *Solanaceous* crop systems.

3. Fink, M. and Scharpf, H.C. N-Expert - a decision support system for vegetable fertilization in the field. ISHS Acta Horticulturae, Vol. 339: 67-74, 1993.
4. Gonzalez-Diaz, L., Martínez-Jimenez, P., Bastida, F and J.L. Gonzalez-Andujar. Expert system for integrated plant protection in pepper (*Capsicum annuun* L.). Expert Systems with Applications, Vol. 36(5):8975–8979, July 2009.
5. Ghosh, I. and R. K. Samanta. TEAPEST: An expert system for insect pest management in tea. American Society of Agricultural and Biological Engineers, Vol. 19 (5):619–625, 2003.
6. Boulanger, A. G. The Expert System PLANT/CD: A Case Study in Applying the General Purpose Inference System ADVISE to Predicting Black Cutworm Damage in Corn. M. S. Thesis, Computer Science Dept., Univ. of Illinois at Champaign-Urbana, 1983.
7. Baker, J. and Lemmon, H. COMAX Expert Systems for Agriculture, Computers and Electronics in Agriculture, Vol.1: 31-40, 1985.
8. Durkin, J., Godine, R. and Y. Lu. CROPRO Expert System for Specialty Crop Management. The lnternational. Jnt. Conf. on Artificial Intelligence : 312-323, 1989.
9. John Durkin. Application of Expert Systems in the Sciences. OHIO J. SCI., Vol.90(5): 171-179,1990.
10. Yialouris, C.P., Passam, H.C., Sideridis, A.B. and C Métin. VEGES—A multilingual expert system for the diagnosis of pests, diseases and nutritional disorders of six greenhouse vegetables. Computers and Electronics in Agriculture, Vol.19(1):55-67, December 1997.Agricultural Systems, Vol. 76(3):1119–1135, June 2003.