APPLYING DEEP LEARNING TO CLASSIFY DOG BREEDS

Camilo Andrés Cortés Garzón
Daniel Steven Betancourt Correa
cacortesg@academia.usbbog.edu.co
dbetancourt@academia.usbbog.edu.co
Universidad de San Buenaventura Bogotá, Colombia

***Abstract*—This document has as purpose to explain how to classify twelve dog breeds with a convolutional neural network.The desire way, its make a categorical classification. In order to do this, a dataset was built using an application called flickr. Additionaly, it describes how to get a better accuracy, making use of different keras models.**

Keywords- Convolution, Flickr, Classification, Accuracy.

I. Introduction

Convolutionary neural networks are neural networks used for image processing. The advantage of this technique over a conventional neural network is that a convolution can reduce the number of parameters by sectioning the image in different areas. Additionally, apply techniques such as pooling, pading, dropoff and a layers connection.

II. Objetives

- Configure a virtual machine in google cloud databases.
- Develop a convolutional neuronal network to classify twelve dog breeds.
- Explaine how was made each part of process, what is it for.

III. Dataset

The download flickr photos code explained below was developed by Bert Carremans and its absolutely used for educational purposes.

The breeds that it aims to classify, are the followings:

- Yorkshire Terrier
- Pug
- Cocker spaniel
- Jack Russell Terrier
- Beagle
- Husky
- Samoyed
- Labrador
- Tosa
- Dachshund
- French Bulldog
- Chihuahua

To aquire this dataset, it makes use of an Flickr API which saves the images in the folder where the main program is running. The program can be evidenced in annex 1.

A. Input parámeters

Firstly, size parameters are defined to download the image. Possible options are:

- Thumbnail
- Square
- Medium
- Original

Next, it connects to the Flickr API using the line of code shown below:

```
flickr = FlickrAPI(config.API_KEY, config.API_SECRET)
```

After that, it creates the subfolders to save the images how is shown below:

```
results_folder = config.IMG_FOLDER + keyword.replace(" ", "_") + "/"
if not os.path.exists(results_folder):
    os.makedirs(results_folder)
```

Using walk from Flickr API it searches images based on the keyword selected:

```
for photo in photos:
    try:
        url=photo.get('url_m')
        print(url)
        count += 1
        urllib.request.urlretrieve(url,  results_folder + str(count) +".jpg")
    except Exception as e:
        print(e, 'Download failure')
```

As shown in the code, the first parameter defines the keyword, the second refers to the size and finally the license to access the library is specified, in this case it is a non-commercial license.

Finally, the images are saved specifying those that have been saved successfully, a possible error in the acquisition of the image and the amount of data needed.

B. Application of method

For the purpose classification, each keyword was written with the prefix ”dog_” and the name of the breed.

For the size, option ’square’ was taken for the procedure, that is, 150x150 pixel images because the convolutional neural network allows to analyze this scale efficiently.

In terms of amount of data, 700 were selected in order to have 600 training and 100 for testing.

```
dog = ['dog Yorkshire Terrier',
       'dog pug',
       'dog Cocker spaniel',
       'dog Jack Russell Terrier',
       'dog Beagle','dog Husky',
       'dog Samoyed',
       'dog Labrador',
       'dog Tosa',
       'dog Dachshund',
       'dog French Bulldog',
       'dog Chihuahua']
for dogs in dog:
    download_flickr_photos(dogs, size='medium', max_nb_img=700)
```

IV. Image Transformation

Overfitting is a case in which the database is made up of images of the same type, however, all of them have many differences, which reduces classification accuracy. Thats why we need to replicate some similar images to expand the dataset and adapt it better to the learning process.

A. Categorization

To perform the replication process, was needed to convert the image into numbers to treat them with some of the functions that will be explained below:

```
img = load_img('image_direction.jpg')
x = img_to_array(img)
x = x.reshape((1,) + x.shape)
```

It aplays a reshape to convert the image which is initialy 150x150x1 at 150x150x3 because of the RGB representation.

The parameters that will be midified, will be made by a keras module called ”ImageDataGenerator”. Below is the way in which it was applied for this classification

```
train_datagen = ImageDataGenerator(
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    rescale = 1./255,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True)
```

- Rotation_range: it allows to rotate the image
- Width_shift_range: move the image the selected percentage and fill in the lost area with the value at the lateral end.
- Height_shift_range: move the image the selected percentage and fill in the lost area with the value at the vertical end.
- Rescale: rescaling an image will multiply the RGB values of each pixel by a chosen value before any other preprocessing.
- Shear_range: it allows how the shearing transformations must be applied.
- Zoom_range: expand the image in a purposed percentage.
- Horizontal_flip: Rotates in a horizontal way if its true, else dont.

Using the module of keras flow_from_directory its acquired the data stream of 32 transformed images. The application is shown below:

```
train_generator = train_datagen.flow_from_directory(
    'C:/Users/camic/Flikr/Flickr_classifier/flickr/dog_test',
    target_size=(IMG_SIZE,IMG_SIZE),
    class_mode='categorical',
    batch_size = BATCH_SIZE)

validation_generator = validation_datagen.flow_from_directory(
    'C:/Users/camic/Flikr/Flickr_classifier/flickr/dog_train',
    target_size=(IMG_SIZE,IMG_SIZE),
    class_mode='categorical',
    batch_size = BATCH_SIZE)
```

V. Convolutional Neural Network

A CNN its a set of layers of processes within which we find a input layer, convolution layer, grouping, dropout, complete connection of layers and activation type. These layers can be repeated as many times as desired in order to improve the classification.
As input we have each image transformed to a 3-layer matrix corresponding to the RGB.

A. Convolution

```
cnn = Sequential()

cnn.add(Conv2D(filters=32,
               kernel_size=(2,2),
               strides=(1,1),
               padding='same',
               input_shape=(IMG_SIZE,IMG_SIZE,NB_CHANNELS),
               data_format='channels_last'))
```

- kernnel: Allows to determine the size of the convolution filter.
- Strides: Determines the size of the step that the filter will pass through.
- Padding: adds additional pixels around the image. In this case it is the same which means that no type of filler will be applied.
- Input_shape: Defines the input parameters of the image, these parameters have been defined previously.
- Data_format: Sets the order of the image characteristics.

B. Activation

For the proposed sorting program, a ReLu layer (rectified linear unit) adds a degree of non-linearity. For ReLu the result of the values x ¡=0 is 0.

```
cnn.add(Activation('relu'))
```

C. Pooling

Used to reduce dimensions based on the highest value detected by the convolution filter.

```
cnn.add(MaxPooling2D(pool_size=(2,2),
                     strides=2))
```

D. Dropout

It is the rate of desertion in the taking of images to avoid overfitting in the weights and it is said that it works well between 20% and 50%. In this case is 40%.

```
cnn.add(Dropout(0.4))
```

E. Fully connected layer

Determines the characteristics of higher levels. It goes in the last layer of ReLu converting it into a vector and connecting it to all neurons in the completely connected layer.

```
cnn.add(Flatten())
cnn.add(Dense(64))
```

F. Compiling

In the compilation it has to take into account that its working with different classes so the loss function must be in 'categorical_crossentropy'.

```
cnn.compile(loss = 'categorical_crossentropy',
            optimizer = 'rmsprop', metrics = ['accuracy'])
cnn.summary()
```

VI. Training

Once the CNN has been parameterized, the data group is trained with the code shown below:

```
start = time.time()
cnn.fit_generator(
    train_generator,
    steps_per_epoch=NB_TRAIN_IMG//BATCH_SIZE,
    epochs=22,
    validation_data=validation_generator,
    validation_steps=NB_VALID_IMG//BATCH_SIZE)
end = time.time()
print('Processing time:',(end - start)/60)
cnn.save_weights('cnn.h5')
```