Chapter 9 : Part 6

**Deep Learning**

**CNN:** Predict new single Data-point

**9.6.1 Predicting Cat or Dog**

Before proceed, do some online research on what tools to use to make single predictions with CNN. The deep learning scientist spends a lot of his time doing research on how to implement models, or even sometimes on how to use them.

* Add a new sub-folder in the ***dataset*** folder called ***single\_prediction***. This new folder contains two images, a cat and a dog.
* We need ***NumPy***. We will actually use a function by ***NumPy*** to ***pre-process*** the image that we are going to ***load***, so that it can be accepted by the ***predict()*** method.

# *Part 3 - Making new predictions*

**import** numpy **as** np

**from** tensorflow.keras.preprocessing **import** image

# *from keras.preprocessing import image # for old versions*

# *we can also use 'utils ' module*

# *from keras import utils*

# *test\_image = utils.load\_img('dataset/single\_prediction/cat\_or\_dog\_2.jpg',target\_size=(64, 64))*

# *test\_image = utils.img\_to\_array(test\_image)*

test\_image = **image.load\_img**('dataset/single\_prediction/cat\_or\_dog\_1.jpg', target\_size=(64, 64))

# *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 = **cnn\_classifier.predict**(test\_image)

idx = training\_set.class\_indices

**if** result[0][0]**==** 1:

    prediction = 'dog'

**else**:

    prediction = 'cat'

* ***image*** module: Is the image module from Keras. (***from*** keras.preprocessing ***import*** ***image*** *# used in old version*).

**from** tensorflow.keras.preprocessing **import** image

* We can also use:

**from** keras **import** utils

It can do the same job.

* **load\_img()**: To load our image on which we wanna make our prediction, we use **load\_img()**. Here we specify the destination folder and the size for the image. Since we used image size to train our model, we have to set the same size for the new image.

test\_image = **image.load\_img**('dataset/single\_prediction/cat\_or\_dog\_1.jpg',target\_size=(64, 64))

or

test\_image = **utils.load\_img**('dataset/single\_prediction/cat\_or\_dog\_2.jpg',target\_size=(64, 64))

* Do not forget to ***specify*** the ***extension***, We have to include it, which is ***JPG***.
* Conversion to array: To convert our image into a 3D array we use

test\_image = **image.img\_to\_array**(test\_image)

or

test\_image = **utils.img\_to\_array**(test\_image)

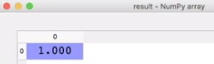
* Remember, the input shape in the input layer of our CNN has three dimensions, each of layer, because it's a colored image **img\_to\_array()** will allow to create this 3D array that will have the same format as the input shape in the input layer of our CNN.
* Using NumPy to add extra dimension: We have to add a new dimension to our test image using ***expand\_dims()*** function. Using 3-dimension, will get an error saying that we will need four dimensions instead of three dimensions.

test\_image= **np.expand\_dims**(test\_image, axis = 0)

* What this dimension corresponds to?
* It corresponds to the batch because, in general, the functions of neural networks, like the *predict* *function*, cannot accept a *single* *input* by itself, like the image we have here.
* It only accepts ***inputs*** in a batch. Even if the ***batch*** contains ***one*** ***input***. The ***input*** must be in the ***batch***, and this ***new dimension*** that we are creating right now corresponds to the ***batch***. Whether there is *one* *input* or *several* *inputs*.
* So here, we will have *one batch* of *one input*, but then in general, we can have *several batches* of *several inputs*, and we can apply the predict method on that.
* ***axis***: ***axis*** is to specify the position of the index of the dimension that we are adding.
* We need to add this *dimension* in the *first* *position*, therefore, we will specify ***axis = 0***. because ***axis = 0*** means that that *index* of this new dimension we are adding is gonna have the *first* *index*, that is *index* *zero*.
* Next we predict the image:

result = **cnn\_classifier.predict**(test\_image)

* We put the result of that single prediction in a new variable ***result***, The new single prediction, will be ***1*** or ***0*** (binary-classification).



* Next we check the indices of our training set: Does ***1*** correspond to cat or to dog? To check that we need to use following code:

idx = training\_set.class\_indices



* We can clearly see that cats correspond to ***0*** and dogs correspond to ***1***. Perfect, so that means that the prediction by our CNN model for this first image, ***cat\_or\_dog\_1.jpg***, is correct because this image contains a dog.
* If you want to make it even more simple you can add following codes. Remember you need to first check out the ***class\_indices*** indices, then you can use the following conditions:

idx = training\_set.class\_indices

**if** result[0][0]**==** 1:

    prediction = 'dog'

**else**:

    prediction = 'cat'

* Notice, this ***result*** is an array of two dimensions. **result[0][0]**, Used to get the first row and the first column.
* When we look at the test-set validation accuracy ***val\_accuracy***:, well, remember we obtained between ***81*** and ***83%***, so you know our model had ***82%*** chance to make ***correct*** ***predictions***, and that's how we got these good results.

**All code at once (practiced)**

# *=========== ====  Convolutional Neural Network : CNN  ==== ============*

# *----------- Install following  packages -------------*

    # *Install Theano*

    # *Install Tensorflow*

    # *Install Tensorflow*

    # *Install Keras*

# *---------- Part 1 - Building the CNN ----------------*

# *Importing the Keras libraries and packages*

**from** keras.models **import** Sequential     # *to initialize as sequence-of-layers*

**from** keras.layers **import** Convolution2D  # *Convolution step for images*

**from** keras.layers **import** MaxPooling2D   # *not "MaxPool2D". Pooling step for images*

**from** keras.layers **import** Flatten        # *Flatenning step*

**from** keras.layers **import** Dense          # *ads fully-connected-layers to classic ANN*

# *initializing the CNN*

cnn\_classifier = **Sequential**()

# *step 1 : Convolution - layer*

**cnn\_classifier.add**(**Convolution2D**(32, 3, 3, input\_shape = (64, 64, 3), activation= "relu"))

# *step 2 : Pooling  - layer*

**cnn\_classifier.add**(**MaxPooling2D**(pool\_size = (2, 2)))

# *improving step : Adding 2nd-Convolution and  2nd-Pooling layers*

**cnn\_classifier.add**(**Convolution2D**(32, 3, 3, activation= "relu"))

**cnn\_classifier.add**(**MaxPooling2D**(pool\_size = (2, 2)))

# *step 3 : Flattening*

**cnn\_classifier.add**(**Flatten**())

# *step 4 : ANN - full connection*

**cnn\_classifier.add**(**Dense**(units= 128, activation= "relu")) # *fully connected layers*

**cnn\_classifier.add**(**Dense**(units= 1, activation= "sigmoid")) # *output layer*

# *compile the NN*

**cnn\_classifier.compile**(optimizer= "adam", loss="binary\_crossentropy", metrics = ["accuracy"])

# *---------- Part 2 - Image Preprocessing & fit CNN to our images ----------------*

**from** keras.preprocessing.image **import** ImageDataGenerator

**import** math

# *creating two data-generator/augmentation obgects for train and test data*

        # *here we specify the transform parameters*

train\_datagen = **ImageDataGenerator**(

                                    rescale=1./255,

                                    shear\_range=0.2,

                                    zoom\_range=0.2,

                                    horizontal\_flip=**True** )

test\_datagen = **ImageDataGenerator**(rescale=1./255)

# *applying augmentation on training data : training folder path needed*

                                        # *target\_size = input\_shape (dimension of expected resized images)*

                                        # *batch\_size is not related to no. of filters*

training\_set = **train\_datagen.flow\_from\_directory**('dataset/training\_set',

                                                target\_size=(64, 64),

                                                batch\_size=32,

                                                class\_mode='binary')

# *applying augmentation on test data : test folder path needed*

test\_set = **test\_datagen.flow\_from\_directory**('dataset/test\_set',

                                            target\_size=(64, 64),

                                            batch\_size=32,

                                            class\_mode='binary')

"""

cnn\_classifier.fit\_generator(training\_set,

 # due to incompatibility between Tensorflow and Keras version, "samples\_per\_epoch", "nb\_epoch", "nb\_val\_samples" may not work

 # Then use "steps\_per\_epoch", "epochs",  "validation\_steps" instead

                            # samples\_per\_epoch = 8000,

                            # nb\_epoch=25,

                            # nb\_val\_samples = 2000,

                            steps\_per\_epoch = 250,  # training\_set\_size/batch\_size

                            epochs = 25,

                            validation\_data=test\_set,

                            validation\_steps = 62   # test\_set\_size/batch\_size

                            )

"""

# *In this case no need to explicitly specify training\_set's or test\_set's sample-size: i.e 8000, 2000 or 800, 200*

**cnn\_classifier.fit\_generator**(

                            training\_set,

                            steps\_per\_epoch=**math.floor**((training\_set.samples)/(training\_set.batch\_size)),

                            epochs=25,

                            validation\_data=test\_set,

                            validation\_steps=**math.floor**((test\_set.samples)/(test\_set.batch\_size))

                            )

# *history = model.fit\_generator(train\_gen,*

# *steps\_per\_epoch=(train\_gen.samples/batch\_size),  # len(train\_gen)*

# *epochs=100,*

# *validation\_data=validation\_gen,*

# *validation\_steps=(validation\_gen.samples/batch\_size),*

# *callbacks=[checkpointer],*

# *workers=4*

# *)*

# *Part 3 - Making new predictions*

**import** numpy **as** np

**from** tensorflow.keras.preprocessing **import** image

# *from keras.preprocessing import image # for old versions*

# *we can also use 'utils ' module*

# *from keras import utils*

# *test\_image = utils.load\_img('dataset/single\_prediction/cat\_or\_dog\_2.jpg',target\_size=(64, 64))*

# *test\_image = utils.img\_to\_array(test\_image)*

test\_image = **image.load\_img**('dataset/single\_prediction/cat\_or\_dog\_2.jpg',target\_size=(64, 64))

# *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 = **cnn\_classifier.predict**(test\_image)

idx = training\_set.class\_indices

**if** result[0][0]**==** 1:

    prediction = 'dog'

**else**:

    prediction = 'cat'

# *python prctc\_cnn.py*

* We can also use '**utils**' module

**from** keras **import** utils

test\_image = **utils.load\_img**('dataset/single\_prediction/cat\_or\_dog\_2.jpg',target\_size=(64, 64))

test\_image = **utils.img\_to\_array**(test\_image)

|  |  |
| --- | --- |
|  |  |