CLASSIFYING CAT AND DOG IMAGES USING DEEP LEARNING

IBM Machine Learning: Deep Learning

JULY 20, 2022

RATEEB YEHYA
Coursera

Table of Contents

1. About the Data Set

The dataset consists of 12501 images of cats and 12501 images of dogs. The dataset is extracted from Kaggle on the following link.

https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765

Microsoft Research provided Kaggle with the dataset for organizing a competition in 2012 to write an algorithm to classify whether images contain either a dog or a cat. "While random guessing is the easiest form of attack, various forms of image recognition can allow an attacker to make guesses that are better than random. There is enormous diversity in the photo database (a wide variety of backgrounds, angles, poses, lighting, etc.), making accurate automatic classification difficult." (Kaggle, 2012).

Web services are often protected with a challenge that is supposed to be easy for people to solve, but difficult for computers. Such a challenge is called CAPTCHA, which stands for Completely Automated Public Turing test to tell Computers and Humans Apart. You probably have been asked to identify images of bicycles, traffic lights, cats, dogs, etc. before having access to protected web services.

"In an informal poll conducted many years ago, computer vision experts posited that a classifier with better than 60% accuracy would be difficult without a major advance in the state of the art." (Kaggle, 2012)

With deep learning, it is suggested that machine classifiers can score above 80% accuracy in this task, leaving CAPTCHA in danger of being cracked by deep learning networks.

2. Objective

The objective of this project is to train a convolutional neural network on a variety of images of cats and dogs in order to achieve a high accuracy of correctly predicting whether an image is a cat or a dog.

3. Image Preprocessing

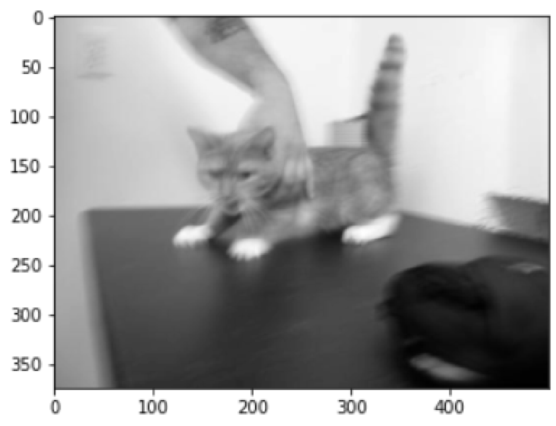
Before feeding our images to a convolutional neural network, it is important that we do some image preprocessing to ensure that our images are on the same scale and size.

First, import all the necessary libraries.

```python
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
from keras.models import Sequential
from keras.layers import Dense, Activation, MaxPooling2D, Conv2D, Flatten
```

Second, we import all the images into numpy arrays and change them using $CV2$ library into a gray scale. The reason why we do that is because the color is not an important factor here in differentiating between a cat and a dog. As a result, we can change the color of all images from colored to black and white before inputting them into our convolutional neural network. This increases our computational efficiency and memory storage as a gray scale image has one color channel whereas a colored image has three channels (Red, Blue, and Green).

```python
data_path = 'D:/Uni/Masters/Python/IBMMachineLearning/Course 5/FinalProject/PetImages'
categories = ['Cat', 'Dog']
for category in categories:
    path = os.path.join(data_path, category)
    for image in os.listdir(path):
        image_array = cv2.imread(os.path.join(path, image), cv2.IMREAD_GRAYSCALE)
        plt.imshow(image_array, cmap = 'gray')
        plt.show()
        break
    break
```



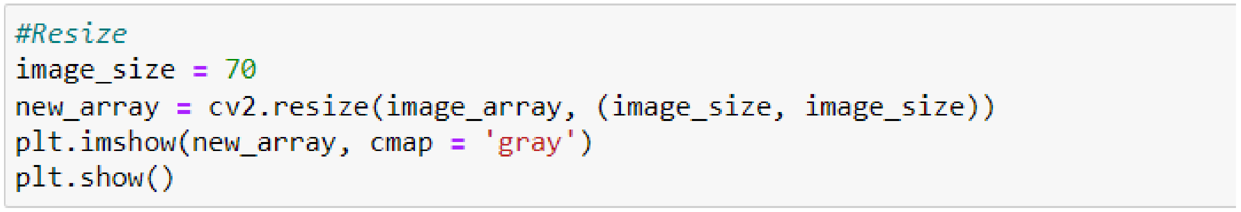
We can see above an example of how the color of an image has been changed to black and white (note that the original image is colored in the dataset).

If we print the shape of the image above, we get

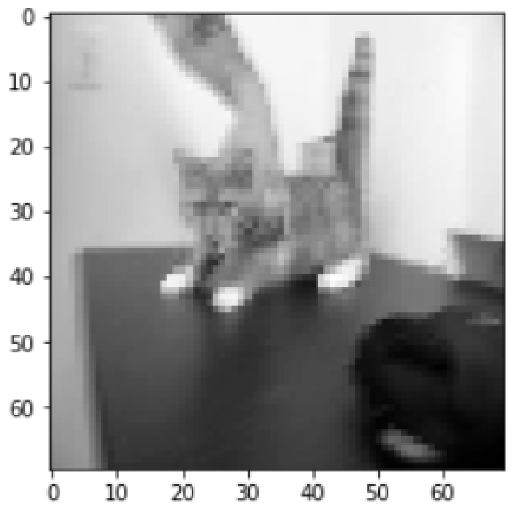
```python
print(image_array.shape)
```

```
(375, 500)
```

Now something to note here is that the images in our dataset are not all of the same size, so it is important to scale them all so that they have the same size before feeding them into our convolutional neural network. This is done in the code below.



```
#Resize
image_size = 70
new_array = cv2.resize(image_array, (image_size, image_size))
plt.imshow(new_array, cmap = 'gray')
plt.show()
```



We can see in the picture above how the size of the image was reduced to (70x70) with a price of lower quality. However, we can still tell that the image represents a cat rather than a dog.

4. Training Set

In this section, we create our training data using a function which we create. The function's code is represented below:

```
training_data = []

def create_training_data():
    for category in categories:
        path = os.path .join(data_path, category)
        class_num = categories.index(category)
        for image in os.listdir(path):
            try:
                image_array = cv2.imread(os.path.join(path, image), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(image_array, (image_size, image_size))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

create_training_data()
```

There are somethings to note about the function above. First, we use try and except because some images might be crashed and therefore we might not be able to import them properly. Second, notice that we changed the categories from strings $['Cats', 'Dogs']$ as defined earlier into binary digits of $0\ and\ 1$, by calling $categories.index(category)$. This is important because machine learning models cannot take in strings. Therefore, we use binary digits instead. Third, notice how all images are resized to $(70\ by\ 70)$ before appending them to our training data.

5. Imbalanced Classes

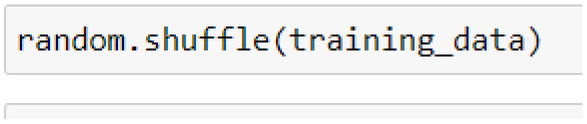
Before we proceed with the CNN, it is important to check if our classes in the training set are balanced or not. If the classes are imbalanced, it is important that we use class weighting in our CNN model. Otherwise, the model will not do a good job in predicting the minor class.

```
In [46]: df=pd.DataFrame(training_data)
         df[1].value_counts(normalize = True)

Out[46]: 0    0.50012
         1    0.49988
         Name: 1, dtype: float64
```

We can see here that the classes are balanced, as required, with a 50-50 split.

6. Randomization

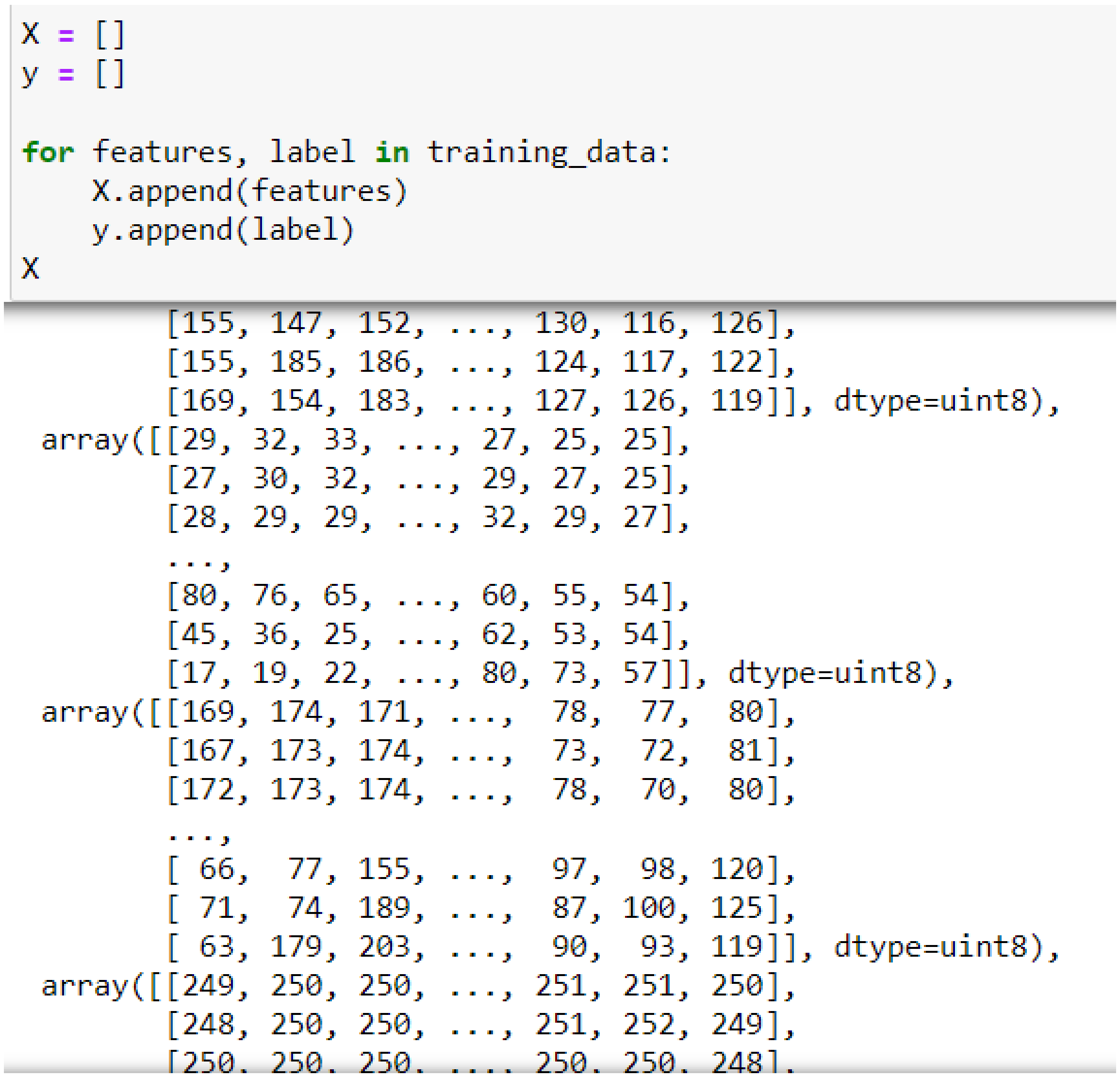
The data is provided in two files: the first file contains all the images of cats and the second file contains all the images of dogs. Until now, our training set consists of all the images of cats, followed by all the images of dogs. If we leave the dataset as it is without any randomization and feed it to the convolutional neural network, our model will be confused in predicting the desired outcomes because all images would be ordered. To prevent the model from learning the order of images, we include randomization before feeding the images into our network. In that way, our model would learn purely from the content of the images and not their order.



```
random.shuffle(training_data)
```

7. Splitting Features from Target Label

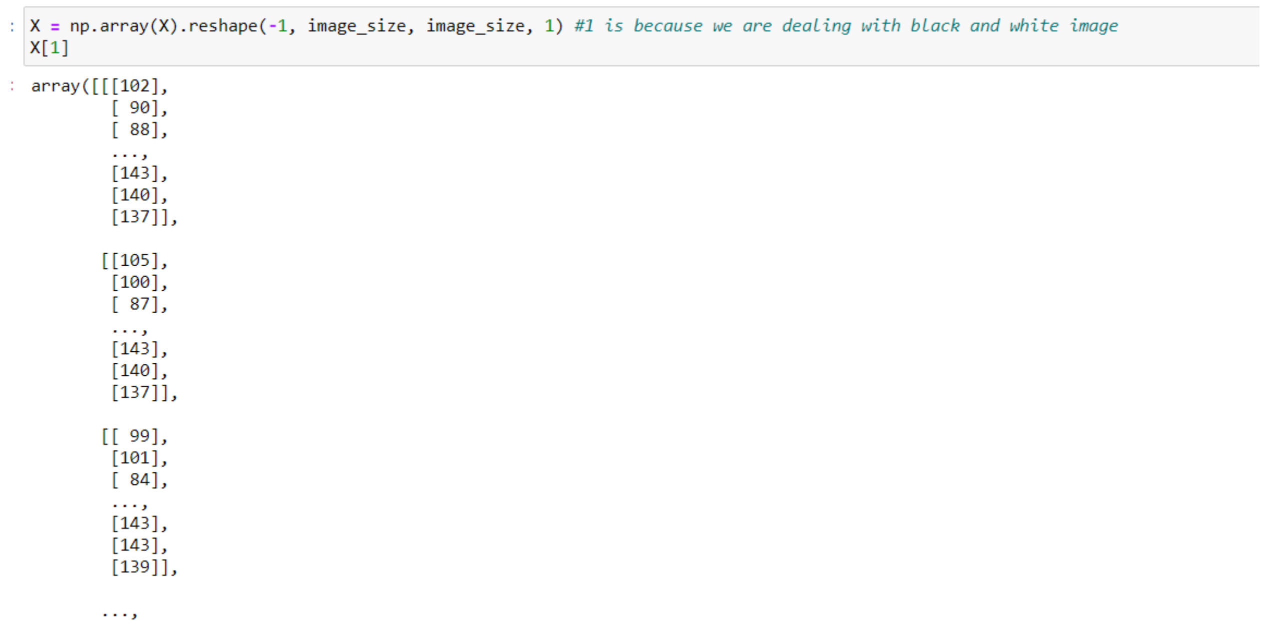
In this section, we split the features (X) from the target label (y) in order to make our data ready for training.



```
X = []
y = []

for features, label in training_data:
    X.append(features)
    y.append(label)
X
```

```
        [155, 147, 152, ..., 130, 116, 126],
        [155, 185, 186, ..., 124, 117, 122],
        [169, 154, 183, ..., 127, 126, 119]], dtype=uint8),
 array([[29, 32, 33, ..., 27, 25, 25],
        [27, 30, 32, ..., 29, 27, 25],
        [28, 29, 29, ..., 32, 29, 27],
        ...,
        [80, 76, 65, ..., 60, 55, 54],
        [45, 36, 25, ..., 62, 53, 54],
        [17, 19, 22, ..., 80, 73, 57]], dtype=uint8),
 array([[169, 174, 171, ...,  78,  77,  80],
        [167, 173, 174, ...,  73,  72,  81],
        [172, 173, 174, ...,  78,  70,  80],
        ...,
        [ 66,  77, 155, ...,  97,  98, 120],
        [ 71,  74, 189, ...,  87, 100, 125],
        [ 63, 179, 203, ...,  90,  93, 119]], dtype=uint8),
 array([[249, 250, 250, ..., 251, 251, 250],
        [248, 250, 250, ..., 251, 252, 249],
        [250, 250, 250, ..., 250, 250, 248],
```

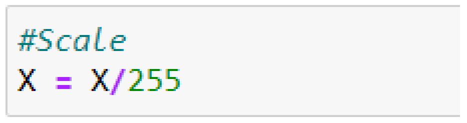
We can see here that the variable X contains a set of arrays, each array representing an image. We change X into a numpy array to make it ready for training.



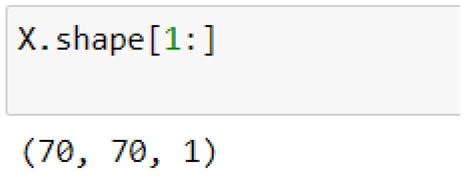
```
X = np.array(X).reshape(-1, image_size, image_size, 1) #1 is because we are dealing with black and white image
X[1]
```

```
array([[[102],
        [ 90],
        [ 88],
        ...,
        [143],
        [140],
        [137]],

       [[105],
        [100],
        [ 87],
        ...,
        [143],
        [140],
        [137]],

       [[ 99],
        [101],
        [ 84],
        ...,
        [143],
        [143],
        [139]],

       ...,
```

Since the pixel value can range between 0 and 255, we can scale all our features by dividing by 255.



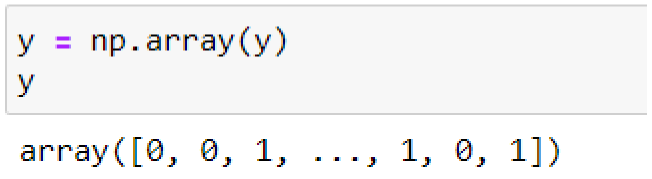
```
#Scale
X = X/255
```



```
X.shape[1:]
```

```
(70, 70, 1)
```

Now we create our target label and store it in an array.



```
y = np.array(y)
y
```

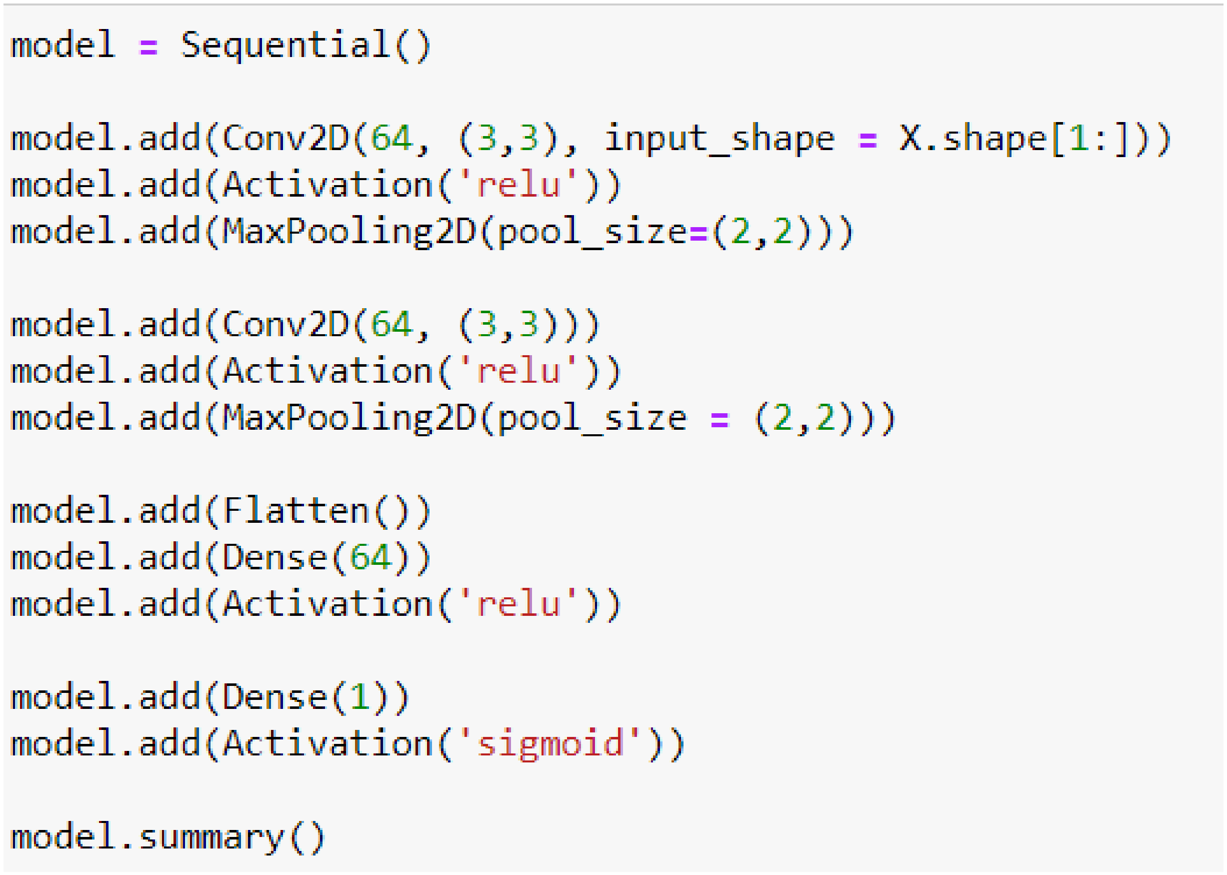
```
array([0, 0, 1, ..., 1, 0, 1])
```

8. Convolutional Neural Network (CNN)

Now that we have our data ready for training, it is time we build and train our convolutional neural network. We will use the following sequence for our CNN model.

$$Conv2D(64) - Activation\ (relu) - MaxPooling2D - Conv2D(64) - Activation(relu) - MaxPooling2D - Flatten - Dense(64) - Activation(relu) - Dense(1) - Activation(sigmoid)$$

We will use three by three kernels for our convolutional neural network and a two by two max pool size. We will also use 64 nodes for our $Conv2D$ layers as well as our dense layer at the end.



```
model = Sequential()

model.add(Conv2D(64, (3,3), input_shape = X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary()
```

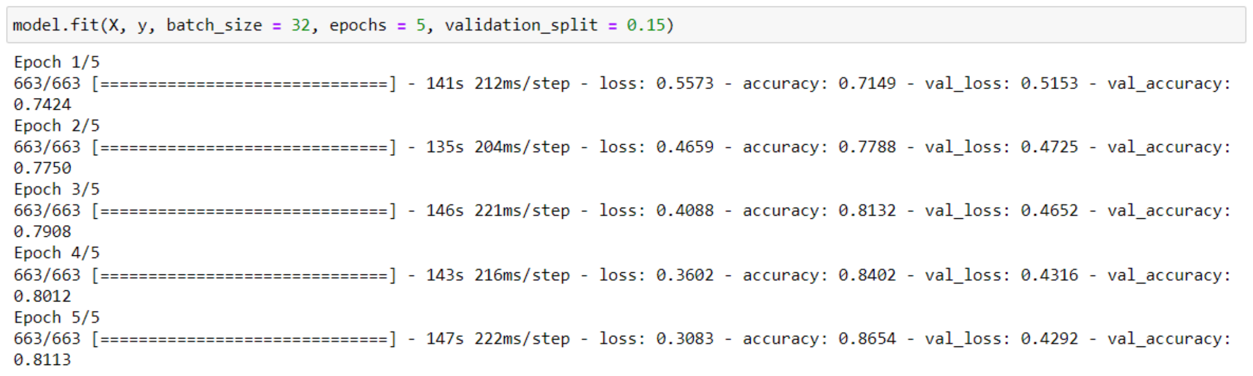
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_10 (Conv2D)          (None, 68, 68, 64)        640       
                                                                 
 activation_18 (Activation)  (None, 68, 68, 64)        0         
                                                                 
 max_pooling2d_10 (MaxPoolin  (None, 34, 34, 64)       0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (None, 32, 32, 64)        36928     
                                                                 
 activation_19 (Activation)  (None, 32, 32, 64)        0         
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 16, 16, 64)       0         
 g2D)                                                            
                                                                 
 flatten_5 (Flatten)         (None, 16384)             0         
                                                                 
 dense_9 (Dense)             (None, 64)                1048640   
                                                                 
 activation_20 (Activation)  (None, 64)                0         
                                                                 
 dense_10 (Dense)            (None, 1)                 65        
                                                                 
 activation_21 (Activation)  (None, 1)                 0         
                                                                 
=================================================================
Total params: 1,086,273
Trainable params: 1,086,273
Non-trainable params: 0
_________________________________________________________________
```

The summary of our CNN model is represented above, with a total of around one million trainable parameters.

We now complie our model with a binary loss since we are dealing with a binary classification. We will use the Adam optimizer and we will track accuracy.

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

We will now fit our model to $X\ and\ y$, with a batch size of 32 and 5 $epochs$. We will also use a validation split of 15%. This means that our out of sample data represents 15% of our total data.



```
model.fit(X, y, batch_size = 32, epochs = 5, validation_split = 0.15)
```

```
Epoch 1/5
663/663 [==============================] - 141s 212ms/step - loss: 0.5573 - accuracy: 0.7149 - val_loss: 0.5153 - val_accuracy:
0.7424
Epoch 2/5
663/663 [==============================] - 135s 204ms/step - loss: 0.4659 - accuracy: 0.7788 - val_loss: 0.4725 - val_accuracy:
0.7750
Epoch 3/5
663/663 [==============================] - 146s 221ms/step - loss: 0.4088 - accuracy: 0.8132 - val_loss: 0.4652 - val_accuracy:
0.7908
Epoch 4/5
663/663 [==============================] - 143s 216ms/step - loss: 0.3602 - accuracy: 0.8402 - val_loss: 0.4316 - val_accuracy:
0.8012
Epoch 5/5
663/663 [==============================] - 147s 222ms/step - loss: 0.3083 - accuracy: 0.8654 - val_loss: 0.4292 - val_accuracy:
0.8113
```

We can see here that using our CNN model with 5 epochs, we are able to obtain an accuracy of 86.5% and a validation accuracy of 81%.

9. Summary and Key Insights

In this project, we train a convolutional neural network on a variety of cats and dogs images. The objective of the neural network is to classify untrained cat and dog images properly.

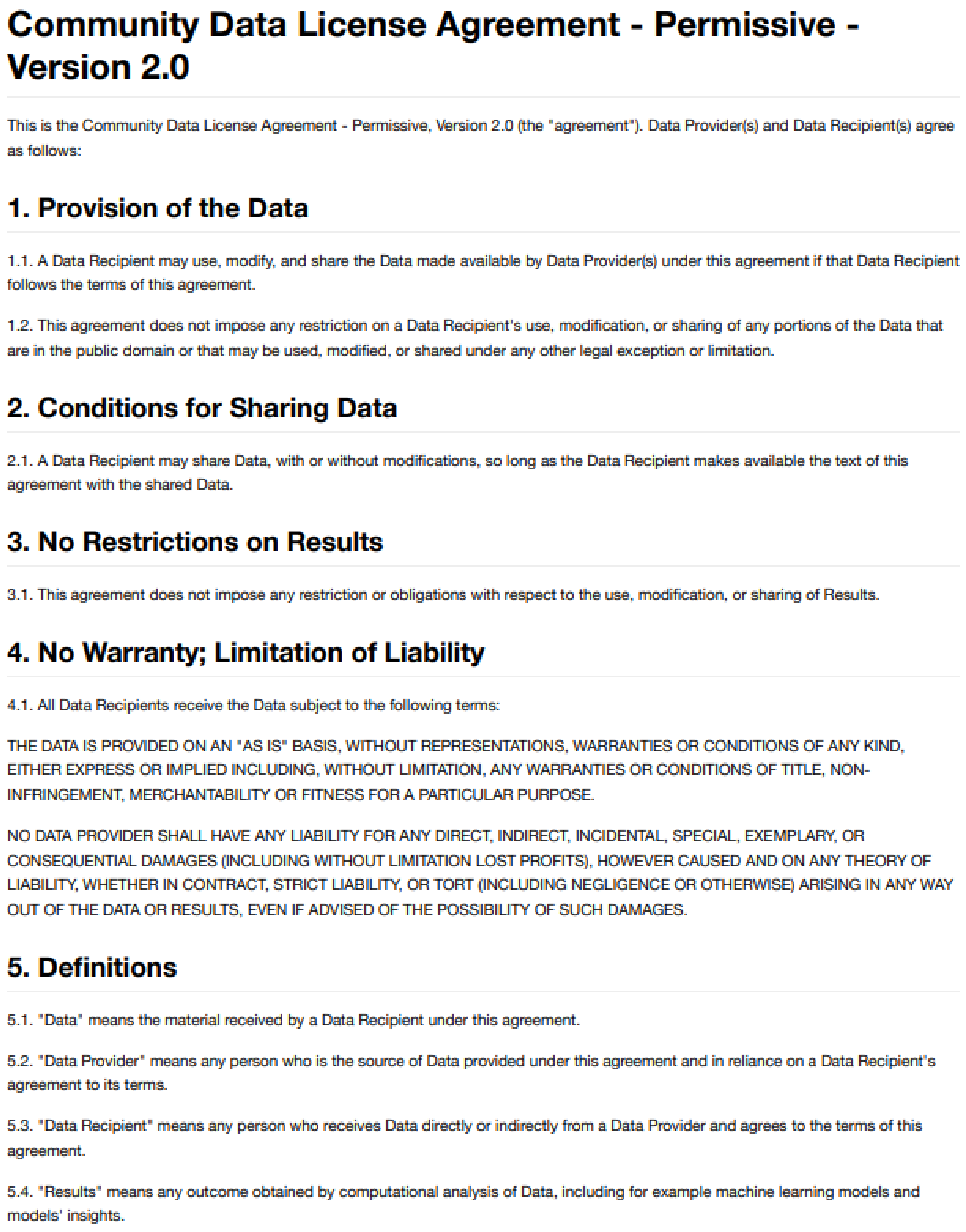
As suggested in the introduction, our model is able to crack CAPTCH with an out of sample accuracy of 81%. This is fascinating and dangerous at the same time. The reason is that if such model is handy to web service attackers, it can be used to crack CAPTCHA which is supposed to protect web pages against possible machine attacks.

Moreover, notice that the model's accuracy can still be increased by adding more epochs since in all the epochs that we ran, the validation accuracy continues to decrease and therefore might increase with additional epochs. However, we see that the validation loss in epochs 4 and 5 is pretty much the same (around 43%), suggesting that the validation loss is beginning to plateau and it would not be long enough before it starts to increase again.

10.Future Suggestions

We can try different configurations of our neural network in an attempt to obtain better results. For example, we can add more hidden layers, increase the number of nodes, change the activation or loss functions, or increase the number of epochs.

11. Data Agreement for Using the Dataset



Community Data License Agreement - Permissive - Version 2.0

This is the Community Data License Agreement - Permissive, Version 2.0 (the "agreement"). Data Provider(s) and Data Recipient(s) agree as follows:

1. Provision of the Data

1.1. A Data Recipient may use, modify, and share the Data made available by Data Provider(s) under this agreement if that Data Recipient follows the terms of this agreement.

1.2. This agreement does not impose any restriction on a Data Recipient's use, modification, or sharing of any portions of the Data that are in the public domain or that may be used, modified, or shared under any other legal exception or limitation.

2. Conditions for Sharing Data

2.1. A Data Recipient may share Data, with or without modifications, so long as the Data Recipient makes available the text of this agreement with the shared Data.

3. No Restrictions on Results

3.1. This agreement does not impose any restriction or obligations with respect to the use, modification, or sharing of Results.

4. No Warranty; Limitation of Liability

4.1. All Data Recipients receive the Data subject to the following terms:

THE DATA IS PROVIDED ON AN "AS IS" BASIS, WITHOUT REPRESENTATIONS, WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

NO DATA PROVIDER SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE DATA OR RESULTS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

5. Definitions

5.1. "Data" means the material received by a Data Recipient under this agreement.

5.2. "Data Provider" means any person who is the source of Data provided under this agreement and in reliance on a Data Recipient's agreement to its terms.

5.3. "Data Recipient" means any person who receives Data directly or indirectly from a Data Provider and agrees to the terms of this agreement.

5.4. "Results" means any outcome obtained by computational analysis of Data, including for example machine learning models and models' insights.