Image Classification on Dogs vs Cats Dataset

Manish Gour
2019H1030025G
BITS GOA

1. Problem Statement:

Read about machine learning model calibration and implement this in the task of binary classification of images using a Convolutional Neural Network (CNN). Package: Use Keras or another framework to implement your CNN. Dataset: Cat/Dog classification problem (https://www.kaggle.com/c/ dogs-vs-cats). Results: On the same test set, compare the results of a calibrated model and a non-calibrated model. This could be just a comparison of the probabilities obtained at the output layer of the mode.

2. Setup:

For solving the famous Kaggle Challenge “Dogs vs. Cats” using Convolutional Neural Network (CNN). We will be using Keras Framework. Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow. We will be using Sequential model *from tensorflow.keras.models.* On top of it we will be using numpy(Linear algebra), os(for os related operation like making directory, extracting files), cv2(image processing) and matplotlib (for plotting graphs).

3. Implementation:

For implementation of machine leaning model which recognize cat and dog, we will be undergoing to below steps.

1. Loading the data

Data can be directly loaded from the Kaggle using api and unzipped into golab platform.

```
os.environ['KAGGLE_USERNAME'] = "cdr7299" # username from the json file
os.environ['KAGGLE_KEY'] = "0c0ef322468f028f71e0953b9bbc789a" # key from the json file
!kaggle competitions download -c dogs-vs-cats # api copied from kaggle
```

2. Preparation and Pre-processing of Dataset

The training archive contains 25,000 images of dogs and cats and testing archive contains 12,500 images of dogs and cats. Extracted data can be converted into uniform size using cv2.resize() function with gray as color channel. Also, data can be separated for testing and validation purpose.

```
[12] X = []
     y = []
     convert = lambda category : int(category == 'dog')
     def create_test_data(path):
         for p in os.listdir(path):
             category = p.split(".")[0]
             category = convert(category)
             img_array = cv2.imread(os.path.join(path,p),cv2.IMREAD_GRAYSCALE)
             new_img_array = cv2.resize(img_array, dsize=(80, 80))
             X.append(new_img_array)
             y.append(category)
```

3. Defining the model:
Implementing a Deep Convolutional Neural Network (CNN) using Keras is quite easy task. We define the model as the instance of Sequential() and then just define the layers (Conv2D, MaxPooling2D, Dense, Sigmoid). Our model architecture has 1 Convolutional layers followed by 2 Fully Connected Layer and 1 softmax layer with 10 outputs followed by a Sigmoid output.

Loss function used — binary_crossentropy
Optimizer used — Adam
Metric — accuracy

```
model = Sequential()
# Adds a densely-connected layer with 64 units to the model:
model.add(Conv2D(64,(3,3), activation = 'relu', input_shape = X.shape[1:]))
model.add(MaxPooling2D(pool_size = (2,2)))
# Add another:
model.add(Conv2D(64,(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
# Add a softmax layer with 10 output units:
model.add(Dense(1, activation='sigmoid'))

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

Model Summary:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 78, 78, 64)        640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 39, 39, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 37, 37, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 20736)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                1327168
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,364,801
Trainable params: 1,364,801
Non-trainable params: 0
_________________________________________________________________
```

4. Training of CNN model

Model can be trained with below

```
history = model.fit(X, y, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
625/625 [==============================] - 245s 392ms/step - loss: 0.6484 - accuracy: 0.6127 - val_loss: 0.5802 - val_accuracy: 0.6952
Epoch 2/10
625/625 [==============================] - 245s 392ms/step - loss: 0.5508 - accuracy: 0.7232 - val_loss: 0.5304 - val_accuracy: 0.7332
Epoch 3/10
625/625 [==============================] - 244s 391ms/step - loss: 0.5010 - accuracy: 0.7609 - val_loss: 0.4915 - val_accuracy: 0.7576
Epoch 4/10
625/625 [==============================] - 244s 391ms/step - loss: 0.4699 - accuracy: 0.7804 - val_loss: 0.4786 - val_accuracy: 0.7678
Epoch 5/10
625/625 [==============================] - 244s 390ms/step - loss: 0.4442 - accuracy: 0.7937 - val_loss: 0.4798 - val_accuracy: 0.7774
Epoch 6/10
625/625 [==============================] - 244s 390ms/step - loss: 0.4127 - accuracy: 0.8113 - val_loss: 0.4616 - val_accuracy: 0.7856
Epoch 7/10
625/625 [==============================] - 246s 393ms/step - loss: 0.3784 - accuracy: 0.8288 - val_loss: 0.4590 - val_accuracy: 0.7868
Epoch 8/10
625/625 [==============================] - 245s 392ms/step - loss: 0.3456 - accuracy: 0.8490 - val_loss: 0.4767 - val_accuracy: 0.7834
Epoch 9/10
625/625 [==============================] - 246s 393ms/step - loss: 0.3102 - accuracy: 0.8649 - val_loss: 0.4803 - val_accuracy: 0.7900
Epoch 10/10
625/625 [==============================] - 245s 392ms/step - loss: 0.2686 - accuracy: 0.8893 - val_loss: 0.5383 - val_accuracy: 0.7692
```

5. Testing of Trained Model

Model can be tested with provided data set as shown below.

```
[23] predictions = model.predict(X_test)
```

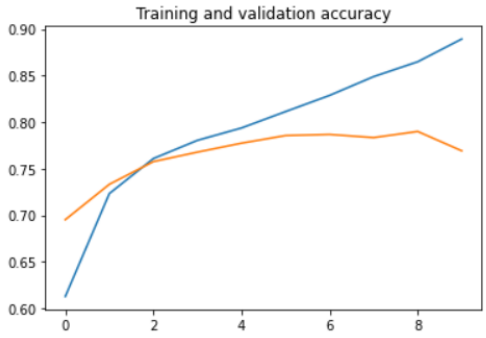
```
[24] predicted_val = [int(round(p[0])) for p in predictions]
```

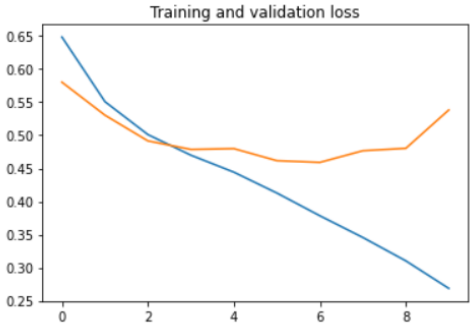
```
[25] print(predicted_val)
```

```
[1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
```

4. Result:

Our model can classify the dogs and cats' images with validation accuracy of 77% and validation loss 0.53.





5. Conclusion

- We are able to achieve a training accuracy of 88% and a validation accuracy of 76% using CNN Algorithm.
- We can further improve our accuracy by a significant amount using data augmentation and adding further layers to our model.