A

Machine Learning Project

IMAGE CLASIFICATION TENSORFLOW

(DOG AND CAT IMAGE PREDECTION)

Submitted

By

MALAY MALAKAR & BIBHAKAR PAUL

(AI, MACHINE LEARNING DATA SCIENCE)

Under Guidance of

MAHENDRA DUTTA

Teacher Department of Computer Science

Department of Machine Learning

Sector V, Bidhannagar, Kolkata, West Bengal 70091

July 2022

CANDIDATE'S DECLEARATION

We hereby declare that the Machine learning project work being presented in this report entitled "IMAGE CLASIFICATION OF TENSORFLOW" submitted in the Department of Machine Learning, Ardent Computech, Bidhannagar, Kolkata is the authentic work carried out by Malay Malakar & Bibhakar Paul under the guidance of respective teacher Mahendra Dutta , Teacher, Department of Machine Learning, Ardent Computech, Bidhannagar, Kolkata.

Date .... /.... / .....

MALAY MALAKAR

BIBHAKAR PAUL

AI, MACHINE LEARNING DATA SCIENCE

Department of Machine Learning

Ardent Computech,

Bidhannagar,Kolkata, West Bengal 70091

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to my teacher Mahendra Dutta who Gave our the golden opportunity to do this wonderful project on the topic **IMAGE CLASIFICATION TENSORFLOW** (DOG AND CAT IMAGE PREDECTION) , which also helped me in doing a lot of Research and we came to know about so many new things we are really thankful to them .

Date .... /.... / .....

MALAY MALAKAR

BIBHAKAR PAUL

AI, MACHINE LEARNING DATA SCIENCE

Department of Machine Learning

Ardent Computech,

Bidhannagar,Kolkata, West Bengal 70091

PROJECT INDEX

1.Preparing the training data

2.Reading the data from the images

3.Distributing the training images

4.Preparing the Image Data Generator

5.Training the model

6.Visualizing the training result

7.Output

8.Conclution

Using Tensor Flow to recognize Cats and Dogs

I like to use practical examples and projects to help me memorize the theory during my study of Deep Neural Networks. An excellent resource for finding these practical projects is Kaggle. Kaggle is an online community of data scientists and machine learning practitioners. Kaggle allows you to search and publish data sets, explore, and build models. You can perform these functions in a web-based environment. Kaggle also offers machine learning competitions with actual problems and provides prizes to the winners.

I am currently studying Deep Learning with TensorFlow. One of the subjects I want to learn is image recognition. This article describes my attempt to solve a former Kaggle competition from 2022, called “Dogs vs. Cats.” For implementing the solution I used Python 3.9 and TensorFlow 2.3.0. The original “Dogs vs. Cats” competition’s goal was to write an algorithm to classify whether images contain either a dog or a cat. Note that in 2022 there was no TensorFlow or another framework such as PyTorch to help. Although the competition is finished, it is still possible to upload and let Kaggle score your predictions.

1.Preparing the training data :

Before we can create and train a model, we must prepare the training data. The training data for this competition involves 4000 images of dogs and cats. The filename of each image specifies if the image is a dog or a cat.

To create a submission for the competition, you must create a prediction of each of the 2000 images from the test set. You have to predict if a dog or a cat is in the image. You can score the submission by uploading a CSV that contains a row for each of the 2000 images. Each row must contain the id of the image and the prediction of whether it is a dog or a cat (1 = dog, 0 = cat).

2.Reading the data from the images :

Kaggle provides the training data through a single zip file that contains all the cats' and dogs' images. To enter the image data into the model during training, we first have to load an image from disk and transform it into an array of bytes. The training program then feeds this byte array together with the label "cat" or "dog" into the neural network to learn if it is a cat or a dog.

```
.
├── cats
│   ├── cat000001.jpg
│   └── cat000003.jpg
└── dogs
    ├── dog000111.jpg
    └── dog000112.jpg
```

Instead of writing a Python program to read the files from disk, I use `Image Data Generator` from the Tensorflow Keras Preprocessing module. This class can load images from disk and generate batches of image data that the training process can use directly. It can also do other things, which I will show you later when we optimize the model.

To use the ImageDataGenerator, you must structure your data a certain way on disk. For each label or category that you want to recognize, you have to create a subdirectory with the same name as the label. In that directory, you place the images of that specific category.

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

3.Distributing the training images :

We have to copy the images from the zip from Kaggle into the directory structure, as discussed before. Also, to get an indication of how good our model is, we split the

training images we got from Kaggle into a training and validation set. For example, we reserve 2,000 photos from the 4000 for validating the trained model.

The following Python function distributes the cat and dog images from the folder where you unpacked the zip with pictures from Kaggle. The process creates a structure that can directly be used by the ImageDataGenerator.

4.Preparing the Image Data Generator :

We will use the `flow from directory` method from the Image Data Generator. This method creates an iterator that retrieves and returns images from the specified directory. I use the rescale parameter when creating an instance to rescale the pixels of the image automatically. Instead of using RGB values that range from 0 to 255, we rescale them into a floating-point number from 0 to 1. This rescale makes it easier to train the deep learning model.

We use the following parameters with the `flow_from_directory` method:

- directory — The directory that contains the subdirectories with the categories.
- target_size — Each image must be of the same size before submitting it to the deep learning pipeline. The ImageDataGenerator can perform this on the fly.
- batch_size — We submit the images to the deep learning pipeline in batches. This sets the size of the individual batches.
- class_mode — A hint of the type of data we read. In our case, ‘binary’ to show that we will use two categories.

When we run the code, the flow_from_directory reports the number of images found in what categories. In our case, the training generator reports 20000 images, and the validation generator reports 5000 images.

As the ImageDataGenerator is ready we can start creating and training the Deep Learning model.

```
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Convolution2D(32, 3, 3,
        input_shape = (64, 64, 3), activation = 'relu'))
# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(1, activation = 'relu'))
classifier.add(Dense(1, activation = 'sigmoid'))

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

5.Training the model :

After we compiled the model, we can start training the model. We can start the training by calling the `fit` method on the model instance.

The first parameter is the iterator that we created by calling the `flow_from_directory` of the `ImageDataGenerator` for the training data. The second parameter is the iterator from the validation data.

The epoch parameter shows how many times the model will process the entire training set. Here, I want to process all 20.000 training images 10 times.

The `steps_per_epoch` show how many batches it should process before it finishes the epoch. We established a batch size of 200 previously on the `flow_from_directory`, which gives 20 * 100 = 2000 — the number of images of our training set.

The same goes for the `validation_steps`, 50 * 100 = 5,000, the number of images in our validation set.

```
from keras.preprocessing.image import ImageDataGenerator

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('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = (1792/32),
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = (2000/32))


class_labels = {v: k for k, v in training_set.class_indices.items()}
print(class_labels)
```

During training, the `fit` method prints various metrics. Below you see that the accuracy on our training set at the end of the training is 0.99. The accuracy on the validation set is 0.79.

The `model.fit` method returns a history object which contains all the training data once the training is finished. This history object can be used to visualize the metrics.

```
Found 1792 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.

C:\Users\avima\AppData\Local\Temp\ipykernel_2052\1450622249.py:21: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  classifier.fit_generator(training_set,

Epoch 1/8
56/56 [==============================] - 10s 168ms/step - loss: 0.6926 - accuracy: 0.5525 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 2/8
56/56 [==============================] - 9s 163ms/step - loss: 0.6912 - accuracy: 0.5580 - val_loss: 0.6935 - val_accuracy: 0.5000
Epoch 3/8
56/56 [==============================] - 9s 154ms/step - loss: 0.6900 - accuracy: 0.5580 - val_loss: 0.6939 - val_accuracy: 0.5000
Epoch 4/8
56/56 [==============================] - 9s 159ms/step - loss: 0.6891 - accuracy: 0.5580 - val_loss: 0.6943 - val_accuracy: 0.5000
Epoch 5/8
56/56 [==============================] - 9s 159ms/step - loss: 0.6884 - accuracy: 0.5580 - val_loss: 0.6949 - val_accuracy: 0.5000
Epoch 6/8
56/56 [==============================] - 9s 154ms/step - loss: 0.6878 - accuracy: 0.5580 - val_loss: 0.6953 - val_accuracy: 0.5000
Epoch 7/8
56/56 [==============================] - 9s 158ms/step - loss: 0.6875 - accuracy: 0.5580 - val_loss: 0.6959 - val_accuracy: 0.5000
Epoch 8/8
56/56 [==============================] - 9s 155ms/step - loss: 0.6872 - accuracy: 0.5580 - val_loss: 0.6964 - val_accuracy: 0.5000
{0: 'cats', 1: 'dogs'}
```

6.Visualizing the training result :

We can use the history object in combination with the Matplotlib library to visualize the loss and accuracy per epoch. The function below creates two graphs, one that shows the training and validation accuracy, and another that shows the training and validation loss.

The `plot_result` function shows the following two graphs if you train the model for 10 epochs.

Before I create and submit a prediction to Kaggle, I want to see if we can optimize the model.

```
######Predicting the Image#############
from skimage.io import imread
from skimage.transform import resize
import numpy as np


img = imread('download.jpeg')
img = resize(img,(64,64))
img = np.expand_dims(img,axis=0)
#prediction = classifier.predict_classes(img)
prediction = (classifier.predict(img) > 0.5).astype("int32")
#print(prediction)
if prediction[0][0]==1:
    print("It is a dog")
elif prediction[0][0]==0:
    print("It is a cat")
```

7.Output :

Input a cat image (‘download.jpge’) and the output is show bellow the image

```
img = imread('download.jpeg')
img = resize(img,(64,64))
img = np.expand_dims(img,axis=0)
#prediction = classifier.predict_classes(img)
prediction = (classifier.predict(img) > 0.5).astype("int32")
#print(prediction)
if prediction[0][0]==1:
    print("It is a dog")
elif prediction[0][0]==0:
    print("It is a cat")
```

✓ 0.2s

```
1/1 [==============================] - 0s 79ms/step
It is a cat
```

8.Conclusion

In this report, I described how to create and use a Convolution Neural Network using TensorFlow. This Neural Network was used to recognize Cats and Dogs from images. The images of the cats and dogs were taken from this old Kaggle competition Cats vs Dogs.

Although the competition is not running anymore, you can still upload and score your predictions of the test set.

In an upcoming article, I will describe how we can further increase the accuracy using transfer learning.

The source-code for this article can be found on GitHub. The repository is pretty large as it includes all the training and test images.

Source code link : -- https://github.com/BIB-HACKER/Machine_Learning-Library/blob/main/PROJECT/IMAGE%20CLASSIFICATION%20WITH%20TENSORFLOW/image_classification.ipynb