Task 2

Develop a deep learning model that categorizes images of cats and dogs into 2 categories (cats, dogs). Use the Kaggle dataset. Aim for an accuracy level above 85%. Justify all your decisions. You can use the TensorFlow library (the dataset is provided ready-made by this library) or PyTorch of Python or any other programming language you desire.

Solution

Note: The following documentation follows the flow of the code.

To complete the above task, I decided to use Jupyter Notebook (embedded in VS Code), as well as the TensorFlow library, which is used for developing deep learning, a technique of machine learning models, through neural networks.

After installing the necessary libraries: TensorFlow, NumPy for mathematical calculations, cv2 for image modification and management, Matplotlib for statistical graphs, os for file management, I defined the directory where the photos of the Kaggle dataset (cats and dogs) to be used for training the model are located.

Continuing, I limited the acceptable types of photos to jpg, jpeg, bmp, png through an array.

As for loading the files, I used the image_dataset_from_directory function of Keras, which automatically creates the classes of the data based on the folders we have in the file we defined (here we have test1 and train), as well as performs the necessary pre-processing of the data before loading them (the data is not loaded from the start, but on-the-fly).

After creating the data iterator that takes data from folders (one batch at a time), I define the batch as the set of data taken each time from the data iterator (each batch is expressed as a numpy array of 0s and 1s, where 0s represent images and 1s represent labels that are arrays of 0s and 1s, where 0s represent one class and 1s represent the other class).

The next step involves pre-processing the data where I scale the data to save resources. We divide images by 255 to create coordinates with a range of values [0, 1] using a lambda function in the map function, which helps us perform this calculation when loading the data.

Continuing with deep learning, I used the Sequential API of Keras (1 input and 1 output, with layers added sequentially), along with some layers including Conv2D (spatial convolution over images), MaxPooling2D (condensing images), Dense, and Flatten. I defined a model and then added the layers that I will describe below. We use the add function to add the desired layers to the model. First, we add a convolution with 16 filters of 3x3 pixels, examining 1 pixel at a time with relu activation (keeping only positive values and setting negative values to 0, thus including non-linear patterns in the output) and a size of 256x256 pixels in 3 channels. In the second step, MaxPooling2D will take the maximum value of a 2x2 area from the output returned by the first layer. We repeat this process two more times before flattening the output so that we have 256 values and finally have 1 output through Dense() with sigmoid activation (converting any output to a range between 0 and 1).

Additionally, we will need to compile the model with the adam optimizer, specifying the loss and metrics that will indicate its accuracy. As for training the model, I created a log file to monitor the model's progress during training. Continuing with the fit function, I trained the model for 4 epochs.

- Bellow, a snippet of the code is provided,

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
```

Python

```python
model = Sequential()
```

Python

```python
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Python

```python
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

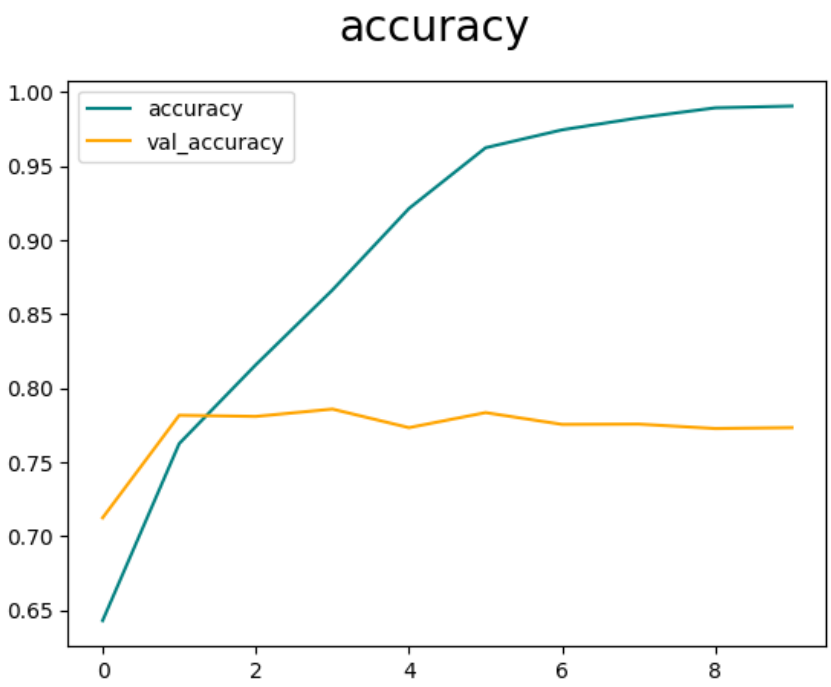
Python

```python
hist = model.fit(train, epochs=4, validation_data=val, callbacks=[tensorboard_callback])
```

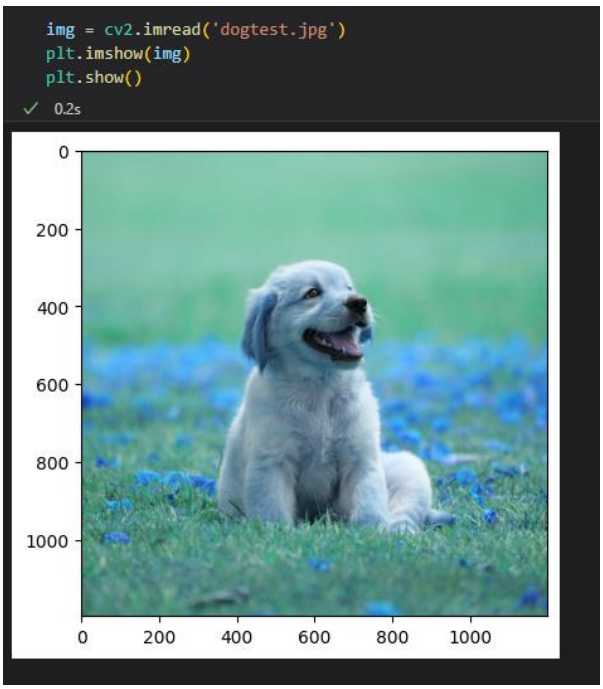
Python

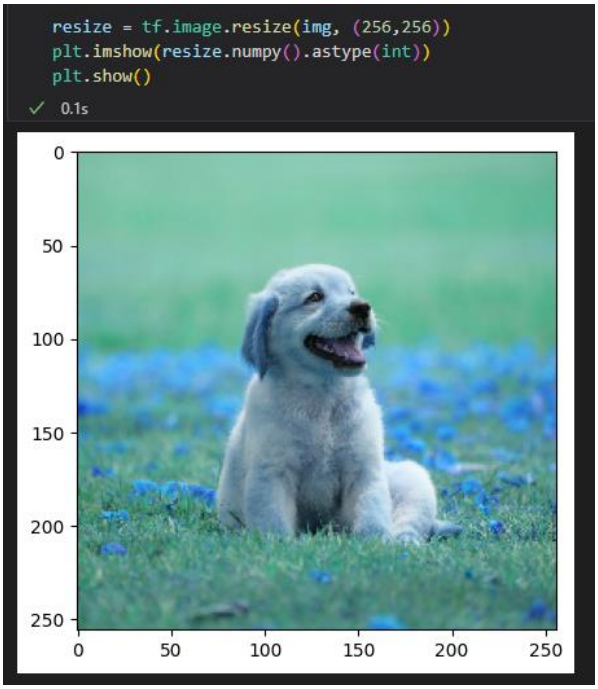
```
Epoch 1/4
547/547 [==============================] - 388s 707ms/step - loss: 0.0440 - accuracy: 0.9860 - val_loss: 1.2663 - '
Epoch 2/4
547/547 [==============================] - 374s 684ms/step - loss: 0.0312 - accuracy: 0.9899 - val_loss: 1.4489 - '
Epoch 3/4
547/547 [==============================] - 330s 603ms/step - loss: 0.0191 - accuracy: 0.9940 - val_loss: 1.4132 - '
Epoch 4/4
547/547 [==============================] - 354s 647ms/step - loss: 0.0227 - accuracy: 0.9929 - val_loss: 1.4757 - '
```

- We can also see the accuracy,



- I also tested the model with a random dog image (form outside the dataset),





```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

✓ 0.1s

```
1/1 [==============================] - 0s 104ms/step
```

```
yhat
```

✓ 0.0s

```
array([[0.7046361]], dtype=float32)
```

```
if (yhat > 0.5):
    print("dog")
else:
    print("cat")
```

✓ 0.0s

```
dog
```

Sources: As this project is my first interaction with deep learning CNN models my main source was the following video: https://www.youtube.com/watch?v=jztwpsIzEGc