flower prediction

February 27, 2022

1 Libraries

[3]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from keras.preprocessing import image
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

2 Dataset

2.1 Preprocessing the Training set

[4]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 3670 images belonging to 5 classes.
```

[5]:
```
# target_size 150, 150 or 256, 256
```

2.2 Preprocessing the Test set

[6]:
```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 3670 images belonging to 5 classes.
```

[7]:
```
# target size 150,150 or 256, 256 (same training set)
```

3 Building a Convolutional Neural Network (CNN)

[8]:
```
cnn = tf.keras.models.Sequential()
```

3.1 Convolution

[9]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',␣
 ↪input_shape=[64, 64, 3]))
```

[10]:
```
# 64,64 because we used earlier in processing in test and train dataset
# has to match
# for black images change the last digit 3 into 1
# kerner_size = 3 or 5 or 7
```

3.2 Pooling

[11]:
```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

3.3 Second Convolutional Layer

[12]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

3.4 Flattening

[13]:
```
cnn.add(tf.keras.layers.Flatten())
```

3.5 Full Connection

[14]:
```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

[15]:
```
# hidden neuron = 128
```

3.6 Output Layer

[16]:
```
cnn.add(tf.keras.layers.Dense(units=5, activation='softmax'))
```

4 Training CNN

4.1 Compiling the CNN

[25]:
```
cnn.compile(optimizer = 'adam', loss = tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics = ['accuracy'])
```

4.2 Training the CNN on the Training set and evaluating it on the Test set

[27]:
```
cnn.fit(x = training_set, validation_data = test_set, epochs=25)
```

```
Epoch 1/25
115/115 [==============================] - ETA: 0s - loss: 1.1691 - accuracy:
0.5166
```

```
C:\Users\jakar\AppData\Roaming\Python\Python38\site-
packages\tensorflow\python\util\dispatch.py:1096: UserWarning:
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `output`
argument was produced by a sigmoid or softmax activation and thus does not
represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
115/115 [==============================] - 49s 433ms/step - loss: 1.1691 -
accuracy: 0.5166 - val_loss: 1.0349 - val_accuracy: 0.5959
Epoch 2/25
115/115 [==============================] - 27s 235ms/step - loss: 1.0390 -
accuracy: 0.5926 - val_loss: 0.9665 - val_accuracy: 0.6302
Epoch 3/25
115/115 [==============================] - 28s 247ms/step - loss: 0.9407 -
accuracy: 0.6297 - val_loss: 0.9111 - val_accuracy: 0.6608
Epoch 4/25
115/115 [==============================] - 27s 236ms/step - loss: 0.8892 -
accuracy: 0.6657 - val_loss: 0.7944 - val_accuracy: 0.7035
Epoch 5/25
115/115 [==============================] - 27s 235ms/step - loss: 0.7981 -
accuracy: 0.6937 - val_loss: 0.7386 - val_accuracy: 0.7193
Epoch 6/25
115/115 [==============================] - 27s 236ms/step - loss: 0.7630 -
accuracy: 0.7101 - val_loss: 0.9721 - val_accuracy: 0.6499
Epoch 7/25
115/115 [==============================] - 27s 234ms/step - loss: 0.7598 -
accuracy: 0.7065 - val_loss: 0.7711 - val_accuracy: 0.7125
Epoch 8/25
115/115 [==============================] - 27s 235ms/step - loss: 0.6998 -
accuracy: 0.7354 - val_loss: 0.6347 - val_accuracy: 0.7599
Epoch 9/25
115/115 [==============================] - 27s 233ms/step - loss: 0.6643 -
accuracy: 0.7523 - val_loss: 0.6408 - val_accuracy: 0.7613
Epoch 10/25
115/115 [==============================] - 27s 235ms/step - loss: 0.6492 -
accuracy: 0.7490 - val_loss: 0.5510 - val_accuracy: 0.7869
Epoch 11/25
115/115 [==============================] - 27s 235ms/step - loss: 0.6075 -
accuracy: 0.7768 - val_loss: 0.5831 - val_accuracy: 0.7831
Epoch 12/25
115/115 [==============================] - 27s 234ms/step - loss: 0.5815 -
```

```
accuracy: 0.7817 - val_loss: 0.4853 - val_accuracy: 0.8172
Epoch 13/25
115/115 [==============================] - 27s 235ms/step - loss: 0.5508 -
accuracy: 0.7896 - val_loss: 0.4477 - val_accuracy: 0.8286
Epoch 14/25
115/115 [==============================] - 27s 235ms/step - loss: 0.5242 -
accuracy: 0.8033 - val_loss: 0.4229 - val_accuracy: 0.8395
Epoch 15/25
115/115 [==============================] - 27s 233ms/step - loss: 0.4954 -
accuracy: 0.8174 - val_loss: 0.3828 - val_accuracy: 0.8654
Epoch 16/25
115/115 [==============================] - 27s 234ms/step - loss: 0.4795 -
accuracy: 0.8185 - val_loss: 0.3745 - val_accuracy: 0.8583
Epoch 17/25
115/115 [==============================] - 27s 234ms/step - loss: 0.4647 -
accuracy: 0.8237 - val_loss: 0.4693 - val_accuracy: 0.8240
Epoch 18/25
115/115 [==============================] - 27s 233ms/step - loss: 0.4487 -
accuracy: 0.8327 - val_loss: 0.3492 - val_accuracy: 0.8777
Epoch 19/25
115/115 [==============================] - 27s 236ms/step - loss: 0.4129 -
accuracy: 0.8488 - val_loss: 0.3418 - val_accuracy: 0.8837
Epoch 20/25
115/115 [==============================] - 28s 241ms/step - loss: 0.3987 -
accuracy: 0.8569 - val_loss: 0.2784 - val_accuracy: 0.9000
Epoch 21/25
115/115 [==============================] - 28s 241ms/step - loss: 0.3640 -
accuracy: 0.8706 - val_loss: 0.2574 - val_accuracy: 0.9079
Epoch 22/25
115/115 [==============================] - 28s 242ms/step - loss: 0.3428 -
accuracy: 0.8766 - val_loss: 0.2362 - val_accuracy: 0.9169
Epoch 23/25
115/115 [==============================] - 28s 245ms/step - loss: 0.3310 -
accuracy: 0.8809 - val_loss: 0.2022 - val_accuracy: 0.9360
Epoch 24/25
115/115 [==============================] - 28s 243ms/step - loss: 0.3166 -
accuracy: 0.8872 - val_loss: 0.2011 - val_accuracy: 0.9406
Epoch 25/25
115/115 [==============================] - 28s 242ms/step - loss: 0.2880 -
accuracy: 0.8984 - val_loss: 0.2819 - val_accuracy: 0.8954
```

```
[27]: <keras.callbacks.History at 0x1be1bf2b6a0>
```

5 Prediction

[79]:
```
test_image = image.load_img('dataset/single_prediction/537207677_f96a0507bb.
↪jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] > 0.5:
    prediction = 'daisy'
elif result[0][1] > 0.5:
    prediction='dandelion'
elif result[0][2] > 0.5:
    prediction='rose'
elif result[0][3] > 0.5:
    prediction='sun flower'
else:
    prediction='tulips'

print(prediction)
```

```
rose
```

[76]:
```
result[0][3]
```

[76]:
```
1.0
```

[32]:
```
# train and set er sime same same hote hbe 64, 64
# result first 0 means batch
# scond 0 pic er index
```

[78]:
```
img=mpimg.imread('dataset/single_prediction/537207677_f96a0507bb.jpg')
plt.figure(figsize=(20, 20))
plt.axis('off')
plt.imshow(img)
```

[78]:
```
<matplotlib.image.AxesImage at 0x1be23fd70d0>
```

```
[ ]:
```