

## Assignment – 2

### Flower Classification using CNN

Assignment Date	01 October 2022
Student Name	Charaan S
Student Roll Number	2019504511
Maximum Marks	2 Marks

### Task - 1: Import the necessary libraries

#### 1. Import the necessary libraries

```
In [1]: import splitfolders
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import Dense, Convolution2D, MaxPooling2D, Flatten
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

### Task - 2: Download the dataset and perform image augmentation

#### 2. Download the dataset and perform image augmentation

```
In [2]: train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.2, horizontal_flip=True, vertical_flip=False)

In [3]: test_datagen = ImageDataGenerator(rescale=1./255)

In [4]: input_folder = 'C:\\Users\\kumar\\OneDrive\\Documents\\IBM\\assignment_3\\flowers'

In [5]: splitfolders.ratio(input_folder, output="C:\\Users\\kumar\\OneDrive\\Documents\\IBM\\assignment_3\\flowers\\flowersdataset", ratio=(
<
Copying files: 4317 files [00:25, 170.99 files/s]

In [6]: x_train=train_datagen.flow_from_directory(r"C:\\Users\\kumar\\OneDrive\\Documents\\IBM\\assignment_3\\flowers\\flowersdataset\\train", target_s
<
Found 3452 images belonging to 5 classes.

In [7]: x_test=test_datagen.flow_from_directory(r"C:\\Users\\kumar\\OneDrive\\Documents\\IBM\\assignment_3\\flowers\\flowersdataset\\test", target_s
<
Found 865 images belonging to 5 classes.

In [8]: x_train.class_indices

Out[8]: {'daisy': 0, 'dandelion': 1, 'rose': 2, 'sunflower': 3, 'tulip': 4}
```

## Task - 3: Create the model

Model should contain a Convolution Layer, MaxPooling Layer, Flatten Layer, Dense Layer and Output layer

### 3. Create the model

Model should contain a Convolution Layer, MaxPooling Layer, Flatten Layer, Dense Layer and Output layer

```
In [9]: model=Sequential()
model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(300,activation='relu'))
model.add(Dense(150,activation='relu'))
```

```
In [10]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
flatten (Flatten)	(None, 30752)	0
dense (Dense)	(None, 300)	9225900
dense_1 (Dense)	(None, 150)	45150
Total params: 9,271,946		
Trainable params: 9,271,946		
Non-trainable params: 0		

```
In [11]: model.add(Dense(5,activation='softmax'))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
flatten (Flatten)	(None, 30752)	0
dense (Dense)	(None, 300)	9225900
dense_1 (Dense)	(None, 150)	45150
dense_2 (Dense)	(None, 5)	755
Total params: 9,272,701		
Trainable params: 9,272,701		
Non-trainable params: 0		

## Task – 4: Compile the model and train it for 20 epochs using model.fit() method

### 4. Compile the model and train it for 20 epochs using model.fit() method

```
In [12]: model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
         len(x_train)
```

Out[12]: 144

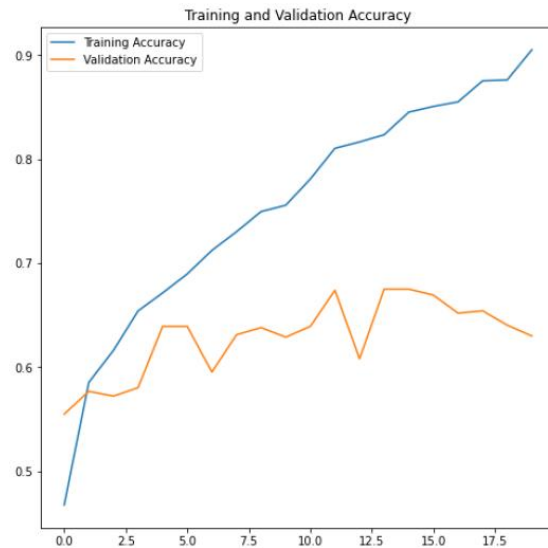
```
In [13]: epoch=20
         history = model.fit(x_train,steps_per_epoch=len(x_train),validation_data=x_test,validation_steps=len(x_test),epochs=epoch)

Epoch 1/20
144/144 [=====] - 44s 158ms/step - loss: 1.2757 - accuracy: 0.4676 - val_loss: 1.1261 - val_accuracy: 0.5549
Epoch 2/20
144/144 [=====] - 8s 57ms/step - loss: 1.0409 - accuracy: 0.5855 - val_loss: 1.1483 - val_accuracy: 0.5769
Epoch 3/20
144/144 [=====] - 7s 52ms/step - loss: 0.9904 - accuracy: 0.6162 - val_loss: 1.1175 - val_accuracy: 0.5723
Epoch 4/20
144/144 [=====] - 7s 49ms/step - loss: 0.8925 - accuracy: 0.6541 - val_loss: 1.0778 - val_accuracy: 0.5803
Epoch 5/20
144/144 [=====] - 7s 51ms/step - loss: 0.8504 - accuracy: 0.6715 - val_loss: 1.0019 - val_accuracy: 0.6393
Epoch 6/20
144/144 [=====] - 7s 51ms/step - loss: 0.8065 - accuracy: 0.6897 - val_loss: 1.0034 - val_accuracy: 0.6393
Epoch 7/20
144/144 [=====] - 7s 50ms/step - loss: 0.7544 - accuracy: 0.7123 - val_loss: 1.1677 - val_accuracy: 0.5954
Epoch 8/20
144/144 [=====] - 7s 51ms/step - loss: 0.7005 - accuracy: 0.7303 - val_loss: 1.0580 - val_accuracy: 0.6312
Epoch 9/20
144/144 [=====] - 7s 51ms/step - loss: 0.6536 - accuracy: 0.7497 - val_loss: 1.0528 - val_accuracy: 0.6382
Epoch 10/20
144/144 [=====] - 7s 51ms/step - loss: 0.6092 - accuracy: 0.7558 - val_loss: 1.1169 - val_accuracy: 0.6289
Epoch 11/20
144/144 [=====] - 8s 56ms/step - loss: 0.5679 - accuracy: 0.7810 - val_loss: 1.1217 - val_accuracy: 0.6393
Epoch 12/20
144/144 [=====] - 8s 55ms/step - loss: 0.5288 - accuracy: 0.8105 - val_loss: 1.0372 - val_accuracy: 0.6740
Epoch 13/20
144/144 [=====] - 7s 51ms/step - loss: 0.4846 - accuracy: 0.8166 - val_loss: 1.1872 - val_accuracy: 0.6081
Epoch 14/20
144/144 [=====] - 7s 51ms/step - loss: 0.4804 - accuracy: 0.8236 - val_loss: 1.1117 - val_accuracy: 0.6751
Epoch 15/20
144/144 [=====] - 7s 51ms/step - loss: 0.4333 - accuracy: 0.8453 - val_loss: 1.1388 - val_accuracy: 0.6751
Epoch 16/20
144/144 [=====] - 7s 51ms/step - loss: 0.4119 - accuracy: 0.8508 - val_loss: 1.1882 - val_accuracy: 0.6694
Epoch 17/20
144/144 [=====] - 7s 51ms/step - loss: 0.3981 - accuracy: 0.8552 - val_loss: 1.3760 - val_accuracy: 0.6520
Epoch 18/20
144/144 [=====] - 9s 62ms/step - loss: 0.3456 - accuracy: 0.8754 - val_loss: 1.3123 - val_accuracy: 0.6543
Epoch 19/20
144/144 [=====] - 10s 66ms/step - loss: 0.3341 - accuracy: 0.8763 - val_loss: 1.4009 - val_accuracy: 0.6405
Epoch 20/20
144/144 [=====] - 8s 52ms/step - loss: 0.2888 - accuracy: 0.9053 - val_loss: 1.4764 - val_accuracy: 0.6301
```

## Task – 5: Plot the training and validation accuracy along with training and validation loss

```
In [15]: epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(epochs_range, history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```



## Task – 6: Save the model

### 6. Save the Model

```
In [17]: model.save('flowers.h5')
```

## Task – 7: Test the model

### 7. Test the model

```
In [18]: img=image.load_img(r"C:\Users\kumar\OneDrive\Documents\IBM\assignment_3\flowers\flowersdataset\test\daisy\3706420943_66f3214862_n.
x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
y=np.argmax(model.predict(x),axis=1)
x_train.class_indices
index=['daisy','dandelion','rose','sunflower','tulip']
index[y[0]]
```

```
Out[18]: 'daisy'
```

```
In [19]: img_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
img_path = tf.keras.utils.get_file('Red_sunflower', origin=img_url)

img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_batch = np.expand_dims(img_array, axis=0)

img_preprocessed = preprocess_input(img_batch)
model = tf.keras.applications.resnet50.ResNet50()
prediction = model.predict(img_preprocessed)

print(decode_predictions(prediction, top=3)[0])

score = tf.nn.softmax(prediction[0])
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg (https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg)
122880/117948 [=====] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5 (https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5)
102973440/102967424 [=====] - 9s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json (https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json)
40960/35363 [=====] - 0s 0us/step
[('n11939491', 'daisy', 0.58117616), ('n02206856', 'bee', 0.24989876), ('n03991062', 'pot', 0.01135167)]
```