Assignment – 3

Flower Classification using CNN

Assignment Date	01 October 2022
Student Name	Mithrha R
Student Roll Number	2019504550
Maximum Marks	2 marks

1. Download the Dataset

Solution:

```
Download the dataset from Google Drive
```

2. Image Augmentation

Solution:

```
data_aug = Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(180, 180, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)

train_data = tf.keras.utils.image_dataset_from_directory(
  "drive/MyDrive/flowers",
  validation_split=0.25,
  subset="training",
  seed=120,
  image_size=(180, 180),
  batch_size=batch_size)

val_data_set = tf.keras.utils.image_dataset_from_directory(
  "drive/MyDrive/flowers",
  validation_split=0.25,
  subset="validation",
  seed=120,
  image_size=(180, 180),
  batch_size=batch_size)
```

Output:

In [1]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
import os
```

In [2]:
```python
batch_size = 16
```

IMAGE AUGMENTATION

In [3]:
```python
data_aug = Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(180, 180, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

In [6]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

> spliting dataset into training and test

In [13]:
```python
train_data = tf.keras.utils.image_dataset_from_directory(
  "drive/MyDrive/flowers",
  validation_split=0.25,
  subset="training",
  seed=120,
  image_size=(180, 180),
  batch_size=batch_size)
```

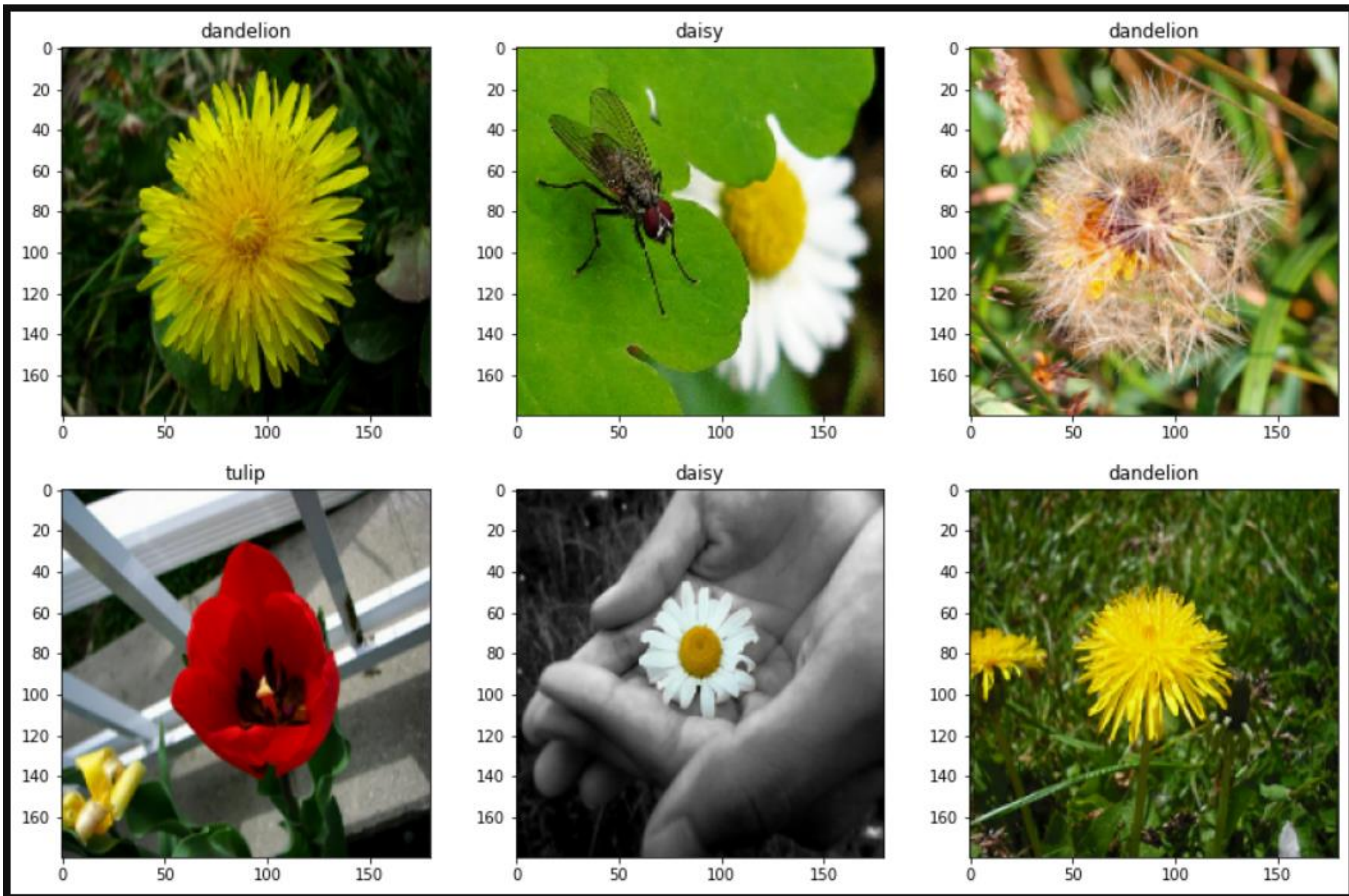
```
Found 4317 files belonging to 5 classes.
Using 3238 files for training.
```

In [23]:
```python
val_data_set = tf.keras.utils.image_dataset_from_directory(
 "drive/MyDrive/flowers",
  validation_split=0.25,
  subset="validation",
  seed=120,
  image_size=(180, 180),
  batch_size=batch_size)
```

```
Found 4317 files belonging to 5 classes.
Using 1079 files for validation.
```

In [14]:
```python
class_names = train_data.class_names
```

In [15]:
```python
plt.figure(figsize=(15, 15))
for images, labels in train_data.take(1):
  for i in range(6):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
```



Normalizing pixel value from 0 - 255 to 0 - 1

```
In [17]: normalization_layer = layers.Rescaling(1./255)
```

```
In [18]: dataset_normalized = train_data.map(lambda x, y: (normalization_layer(x), y))
         image_batch, labels_batch = next(iter(dataset_normalized))
         first_image = image_batch[0]
         print(np.min(first_image), np.max(first_image))

0.0 1.0
```

3. Creating models and Addition of Layers

Solution:

```
num_classes = len(class_names)

model = Sequential([
  data_aug,
  layers.Rescaling(1./255, input_shape=(180, 180, 3)),
  layers.Conv2D(16, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3,activation='relu'),
  layers.Conv2D(32, 3,activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, activation='relu'),
```

```
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Output:

```
num_classes = len(class_names)

model = Sequential([
  data_aug,
  layers.Rescaling(1./255, input_shape=(180, 180, 3)),
  layers.Conv2D(16, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3,activation='relu'),
  layers.Conv2D(32, 3,activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

4. Compilation of Model

Solution:

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategorical
Crossentropy(from_logits=True),metrics=['accuracy'])
```

Output:

```
# compiling model with categorical cross entropy and adam optimizer
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
```

5. Fitting the Model

Solution:

```
epochs=15
history =
model.fit(train_data,validation_data=val_data_set,epochs=epochs)
```

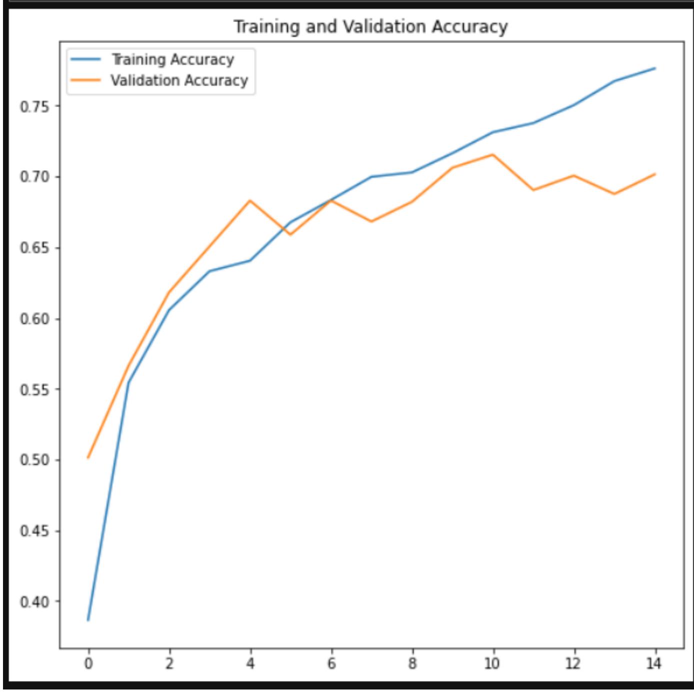
Output:

```
epochs=15
history = model.fit(train_data,validation_data=val_data_set,epochs=epochs)
```

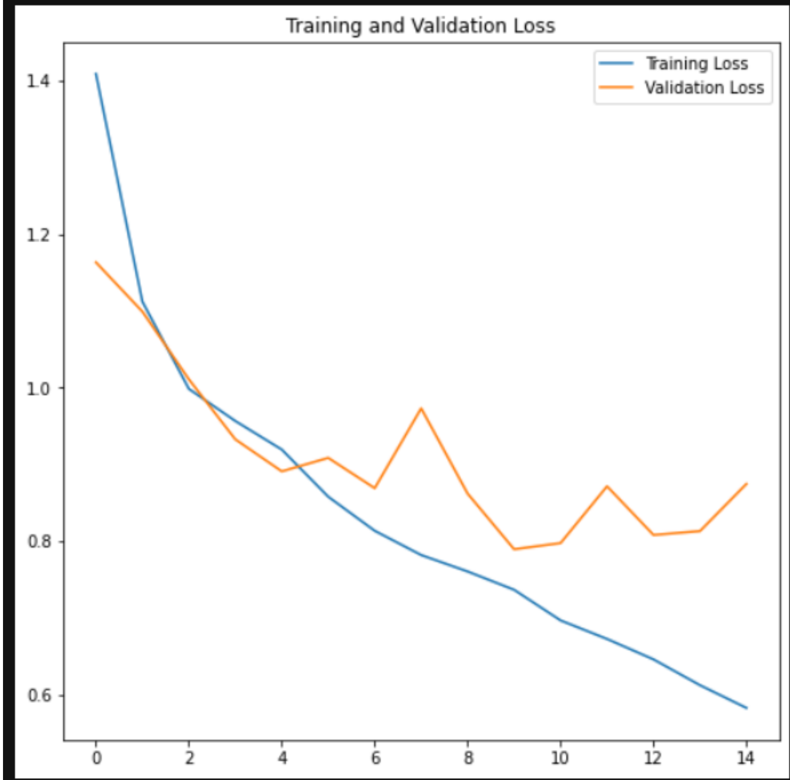
```
Epoch 1/15
203/203 [==============================] - 288s 1s/step - loss: 1.4088 - accuracy: 0.3867 - val_loss: 1.1632 - val_accuracy: 0.5014
Epoch 2/15
203/203 [==============================] - 175s 860ms/step - loss: 1.1120 - accuracy: 0.5544 - val_loss: 1.0990 - val_accuracy: 0.5663
Epoch 3/15
203/203 [==============================] - 174s 854ms/step - loss: 0.9984 - accuracy: 0.6056 - val_loss: 1.0108 - val_accuracy: 0.6182
Epoch 4/15
203/203 [==============================] - 171s 841ms/step - loss: 0.9565 - accuracy: 0.6331 - val_loss: 0.9326 - val_accuracy: 0.6506
Epoch 5/15
203/203 [==============================] - 172s 845ms/step - loss: 0.9193 - accuracy: 0.6405 - val_loss: 0.8909 - val_accuracy: 0.6830
Epoch 6/15
203/203 [==============================] - 171s 839ms/step - loss: 0.8578 - accuracy: 0.6677 - val_loss: 0.9085 - val_accuracy: 0.6589
Epoch 7/15
203/203 [==============================] - 173s 849ms/step - loss: 0.8135 - accuracy: 0.6834 - val_loss: 0.8690 - val_accuracy: 0.6830
Epoch 8/15
203/203 [==============================] - 172s 845ms/step - loss: 0.7819 - accuracy: 0.6998 - val_loss: 0.9730 - val_accuracy: 0.6682
Epoch 9/15
203/203 [==============================] - 172s 845ms/step - loss: 0.7604 - accuracy: 0.7029 - val_loss: 0.8619 - val_accuracy: 0.6821
Epoch 10/15
203/203 [==============================] - 172s 846ms/step - loss: 0.7366 - accuracy: 0.7165 - val_loss: 0.7894 - val_accuracy: 0.7062
Epoch 11/15
203/203 [==============================] - 176s 866ms/step - loss: 0.6967 - accuracy: 0.7313 - val_loss: 0.7974 - val_accuracy: 0.7155
Epoch 12/15
203/203 [==============================] - 175s 861ms/step - loss: 0.6725 - accuracy: 0.7378 - val_loss: 0.8714 - val_accuracy: 0.6905
Epoch 13/15
203/203 [==============================] - 175s 861ms/step - loss: 0.6459 - accuracy: 0.7505 - val_loss: 0.8080 - val_accuracy: 0.7006
Epoch 14/15
203/203 [==============================] - 173s 850ms/step - loss: 0.6123 - accuracy: 0.7674 - val_loss: 0.8130 - val_accuracy: 0.6877
Epoch 15/15
203/203 [==============================] - 174s 856ms/step - loss: 0.5827 - accuracy: 0.7764 - val_loss: 0.8745 - val_accuracy: 0.7016
```

```
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()
```



```
plt.figure(figsize=(8, 8))
plt.plot(epochs_range, history.history['loss'], label='Training Loss')
plt.plot(epochs_range, history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```



6. Save the Model

Solution:

```
model.save("./flowers.h5")
model.load_weights('./flowers.h5')
```

Output:

```
model.save("./flowers.h5")
```

```
model.load_weights('./flowers.h5')
```

7. Test the Model

Solution:

```
img = tf.keras.utils.load_img(
    "sun.jpg", target_size=(180, 180)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(class_names[np.argmax(score)],100 * np.max(score))
```

Output:

```
img = tf.keras.utils.load_img(
    "sun.jpg", target_size=(180, 180)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])


print(class_names[np.argmax(score)],100 * np.max(score))
```

```
sunflower 91.51520729064941
```