Assignment 3: Build CNN Model for Classification Of flowers.

```
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
```

2. 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\\manok\\Documents\\Sem_7\\HX5001-HX6001\\Assignment\\Assignment_3\\flowers'
```

In [5]:

```
splitfolders.ratio(input_folder,output="C:\\Users\\manok\\Documents\\Sem_7\\HX5001-HX6001\\Assignment\\Assignment_3\\flowersdataset",ratio=(.8,0,.2),group_prefix=None)
```

```
Copying files: 4317 files [00:26, 166.01 files/s]
```

In [6]:

```
x_train=train_datagen.flow_from_directory(r"C:\Users\manok\Documents\Sem_7\HX5001-HX6001\Assignment\Assignment_3\flowersdataset\train",target_size=(64,64),class_mode='categorical',batch_size=24)
```

```
Found 3452 images belonging to 5 classes.
```

In [7]:

```
x_test=test_datagen.flow_from_directory(r"C:\Users\manok\Documents\Sem_7\HX5001-HX6001\Assignment\Assignment_3\flowersdataset\test",target_size=(64,64),class_mode='categorical',batch_size=24)
```

```
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}
```

3. Create Model

In [9]:

```
model=Sequential()
```

4. Add Layers

4.1. Convolution Layer

In [10]:

```
model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
```

4.2. MaxPooling Layer

In [11]:

```
model.add(MaxPooling2D(pool_size=(2,2)))
```

4.3. Flatten Layer

In [12]:

```
model.add(Flatten())
```

4.4. Dense Layer

In [13]:

```
model.add(Dense(300,activation='relu'))
model.add(Dense(150,activation='relu'))
```

In [14]:

```
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
_________________________________________________________________
_
```

4.5. Output Layer

In [15]:

```
model.add(Dense(5,activation='softmax'))
```

In [16]:

```
 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
_________________________________________________________________
_
```

5. Compile The Model

In [17]:

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accu
racy'])
len(x_train)
```

Out[17]:

```
144
```

6. Fit The Model

In [18]:

```
epo=20
history =
model.fit(x_train,steps_per_epoch=len(x_train),validation_data=x_test,validatio
n_steps=len(x_test),epochs=epo)
```

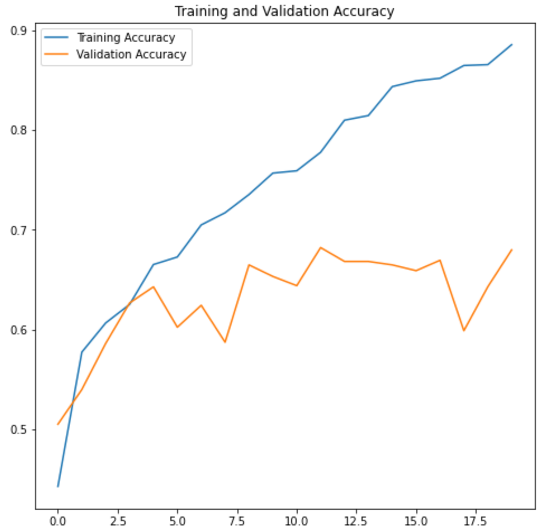
```
Epoch 1/20
144/144 [==============================] - 81s 552ms/step - loss: 1.
4602 - accuracy: 0.4429 - val_loss: 1.2587 - val_accuracy: 0.5052
Epoch 2/20
144/144 [==============================] - 38s 264ms/step - loss: 1.
0639 - accuracy: 0.5773 - val_loss: 1.1310 - val_accuracy: 0.5399
Epoch 3/20
144/144 [==============================] - 38s 263ms/step - loss: 0.
9872 - accuracy: 0.6066 - val_loss: 1.0271 - val_accuracy: 0.5861
Epoch 4/20
144/144 [==============================] - 38s 263ms/step - loss: 0.
9298 - accuracy: 0.6251 - val_loss: 1.0208 - val_accuracy: 0.6266
Epoch 5/20
```

```
144/144 [==============================] - 38s 264ms/step - loss: 0.
8497 - accuracy: 0.6651 - val_loss: 0.9911 - val_accuracy: 0.6428
Epoch 6/20
144/144 [==============================] - 38s 263ms/step - loss: 0.
8255 - accuracy: 0.6727 - val_loss: 1.1223 - val_accuracy: 0.6023
Epoch 7/20
144/144 [==============================] - 36s 253ms/step - loss: 0.
7639 - accuracy: 0.7048 - val_loss: 1.0702 - val_accuracy: 0.6243
Epoch 8/20
144/144 [==============================] - 37s 254ms/step - loss: 0.
7179 - accuracy: 0.7170 - val_loss: 1.1313 - val_accuracy: 0.5873
Epoch 9/20
144/144 [==============================] - 37s 254ms/step - loss: 0.
6676 - accuracy: 0.7352 - val_loss: 0.9532 - val_accuracy: 0.6647
Epoch 10/20
144/144 [==============================] - 36s 252ms/step - loss: 0.
6323 - accuracy: 0.7567 - val_loss: 0.9810 - val_accuracy: 0.6532
Epoch 11/20
144/144 [==============================] - 37s 253ms/step - loss: 0.
6231 - accuracy: 0.7590 - val_loss: 1.0481 - val_accuracy: 0.6439
Epoch 12/20
144/144 [==============================] - 36s 254ms/step - loss: 0.
5839 - accuracy: 0.7775 - val_loss: 1.0738 - val_accuracy: 0.6821
Epoch 13/20
144/144 [==============================] - 36s 253ms/step - loss: 0.
5251 - accuracy: 0.8097 - val_loss: 0.9613 - val_accuracy: 0.6682
Epoch 14/20
144/144 [==============================] - 35s 245ms/step - loss: 0.
4838 - accuracy: 0.8143 - val_loss: 1.0360 - val_accuracy: 0.6682
Epoch 15/20
144/144 [==============================] - 96s 667ms/step - loss: 0.
4308 - accuracy: 0.8433 - val_loss: 1.1060 - val_accuracy: 0.6647
Epoch 16/20
144/144 [==============================] - 26s 180ms/step - loss: 0.
4230 - accuracy: 0.8491 - val_loss: 1.2172 - val_accuracy: 0.6590
Epoch 17/20
144/144 [==============================] - 28s 192ms/step - loss: 0.
4122 - accuracy: 0.8517 - val_loss: 1.0914 - val_accuracy: 0.6694
Epoch 18/20
144/144 [==============================] - 29s 199ms/step - loss: 0.
3877 - accuracy: 0.8644 - val_loss: 1.4504 - val_accuracy: 0.5988
Epoch 19/20
```

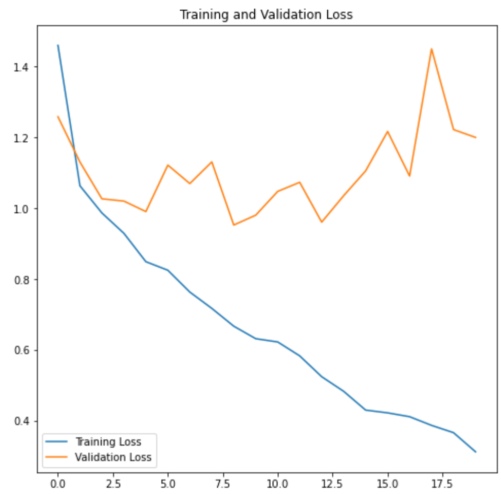
```
144/144 [==============================] - 39s 271ms/step - loss: 0.
3670 - accuracy: 0.8653 - val_loss: 1.2226 - val_accuracy: 0.6428
Epoch 20/20
144/144 [==============================] - 39s 272ms/step - loss: 0.
3131 - accuracy: 0.8853 - val_loss: 1.2005 - val_accuracy: 0.6798
```

In [19]:

```
epochs_range = range(epo)
plt.figure(figsize=(8, 8))
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy')
p lt. plot (epochs_ range, history .history[ 'v al 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()
```



7. Save the Model

In [21]:

```
model.save('flowers.h5')
```

8. Test the Model

In [22]:

```
img=image.load_img(r"C:\Users\manok\Documents\Sem_7\HX5001-HX6001\Assignment\Assignment_3\flowersdataset\test\daisy\3706420943_66f3214862_n.jpg",target_size=(64,64))
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','dandellion','rose','sunflower','tulip']
index[y[0]]
```

```
1/1 [==============================] - 1s 661ms/step
```

Out[22]:

```
'daisy'
```

In [23]:

```
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])
```

```
1/1 [==============================] - 2s 2s/step
[('n11939491', 'daisy', 0.57757616), ('n02206856', 'bee', 0.249383), ('n03991062
', 'pot', 0.01181932)]
```