

Assignment 3 - Build CNN Model for Classification Of Flowers

TEAM MEMBER- Sanjay K (Roll No:911519104032)

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.layers import load_model
from tensorflow.keras.layers import Dense, Convolution2D, MaxPooling2D, Flatten from
tensorflow.keras.applications.resnet50 import preprocess_input, decode_predicti
from tensorflow.keras.preprocessing import image import matplotlib.pyplot as plt
```

```
train_datagen =
ImageDataGenerator(rescale=1./255, zoom_range=0.2, horizontal_flip=True
```

2. Image Augmentation

In [2]:

In [3]:

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [4]:

```
input_folder = './Flowers-Dataset\flowers'
```

In [5]:

```
splitfolders.ratio(input_folder, output="flowers", ratio=(.8, 0.2), group_prefix=None)
```

Copying files: 4317 files [00:03, 1292.11 files/s]

In [6]:

```
x_train=train_datagen.flow_from_directory(r'./flowers\train', target_size=(64, 64), cla
```

Found 3452 images belonging to 5 classes.

In [7]:

```
x_test=test_datagen.flow_from_directory(r'./flowers\test', target_size=(64, 64), class_
```

Found 865 images belonging to 5 classes.

In [8]:

```
x_train.class_indices
```

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

Out[8]:

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	

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

```
model.summary()
```

4.5. Output Layer

In [15]:

In [16]:

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
=====

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

5. Compile The Model

In [17]:

144

Out[17]:

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

6. Fit The Model

In [18]:

Epoch 1/20
144/144 [=====] - 29s 202ms/step - loss: 1.4725 -
accuracy:

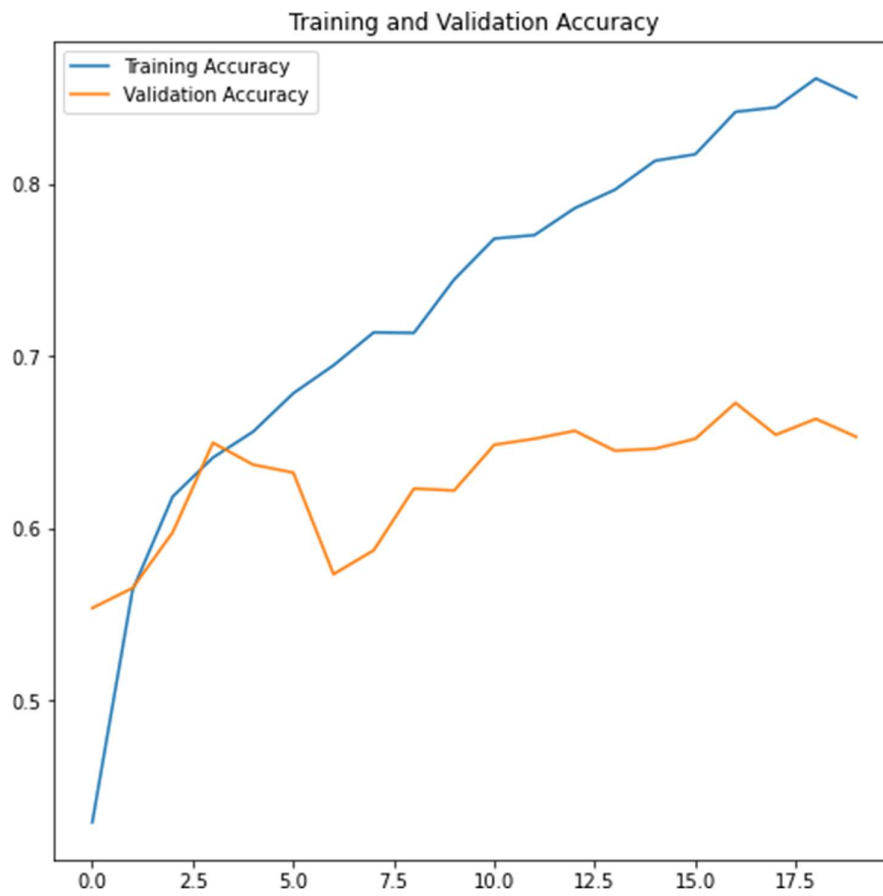
0.4293 - val_loss: 1.1148 - val_accuracy: 0.5538
Epoch 2/20
144/144 [=====] - 15s 101ms/step - loss: 1.0813 - accuracy:
0.5640 - val_loss: 1.0807 - val_accuracy: 0.5653
Epoch 3/20
144/144 [=====] - 15s 102ms/step - loss: 0.9676 - accuracy:
0.6185 - val_loss: 1.0689 - val_accuracy: 0.5977
Epoch 4/20
144/144 [=====] - 15s 101ms/step - loss: 0.9144 - accuracy:
0.6411 - val_loss: 0.9561 - val_accuracy: 0.6497
Epoch 5/20
144/144 [=====] - 17s 116ms/step - loss: 0.8731 - accuracy:
0.6561 - val_loss: 0.9766 - val_accuracy: 0.6370
Epoch 6/20
144/144 [=====] - 15s 107ms/step - loss: 0.8303 - accuracy:
0.6784 - val_loss: 1.0373 - val_accuracy: 0.6324
Epoch 7/20
144/144 [=====] - 16s 108ms/step - loss: 0.7858 - accuracy:
0.6947 - val_loss: 1.1446 - val_accuracy: 0.5734
Epoch 8/20
144/144 [=====] - 15s 105ms/step - loss: 0.7539 - accuracy:
0.7138 - val_loss: 1.1979 - val_accuracy: 0.5873
Epoch 9/20
144/144 [=====] - 15s 107ms/step - loss: 0.7262 - accuracy:
0.7135 - val_loss: 1.0924 - val_accuracy: 0.6231
Epoch 10/20
144/144 [=====] - 15s 101ms/step - loss: 0.6684 - accuracy:
0.7445 - val_loss: 1.1218 - val_accuracy: 0.6220
Epoch 11/20
144/144 [=====] - 15s 106ms/step - loss: 0.6142 - accuracy:
0.7683 - val_loss: 1.0576 - val_accuracy: 0.6486
Epoch 12/20
144/144 [=====] - 15s 106ms/step - loss: 0.6006 - accuracy:
0.7703 - val_loss: 1.0454 - val_accuracy: 0.6520
Epoch 13/20
144/144 [=====] - 15s 105ms/step - loss: 0.5584 - accuracy:
0.7859 - val_loss: 1.0735 - val_accuracy: 0.6566
Epoch 14/20
144/144 [=====] - 15s 102ms/step - loss: 0.5387 - accuracy:
0.7966 - val_loss: 1.1083 - val_accuracy: 0.6451
Epoch 15/20
144/144 [=====] - 15s 103ms/step - loss: 0.4935 - accuracy:
0.8134 - val_loss: 1.0815 - val_accuracy: 0.6462
Epoch 16/20
144/144 [=====] - 14s 100ms/step - loss: 0.4961 - accuracy:

0.8172 - val_loss: 1.0991 - val_accuracy: 0.6520
Epoch 17/20
144/144 [=====] - 15s 103ms/step - loss: 0.4373 -
accuracy:
0.8418 - val_loss: 1.2605 - val_accuracy: 0.6728
Epoch 18/20
144/144 [=====] - 15s 102ms/step - loss: 0.4228 -
accuracy:
0.8444 - val_loss: 1.1316 - val_accuracy: 0.6543
Epoch 19/20
144/144 [=====] - 15s 104ms/step - loss: 0.3853 -
accuracy:
0.8612 - val_loss: 1.1264 - val_accuracy: 0.6636
Epoch 20/20
144/144 [=====] - 14s 100ms/step - loss: 0.3900 -
accuracy:
0.8502 - val_loss: 1.1911 - val_accuracy: 0.6532

In [19]:

```
epochs_range = range(epo)

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()
```



In [20]:

```
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')
```

```
img=image.load_img(r".\flowers\test\daisy\3706420943_66f3214862_n.jpg",target_size=(
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]]
```

8. Test the Model

In [22]:

Out[22]: 1/1 [=====] - 0s 77ms/step
'daisy'


```

In [23]: Downloading data from
https://storage.googleapis.com/download.tensorflow.org/example_images/592px-
Red_sunflower.jpg
img_url = 117948/117948 [=====] - 0s 0us/step
"Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/imagenet50_weights_tf_dim_ordering_tf_kernels.h5
img_path = tf.keras.utils.get_file('Red_sunflower', origin=img_url)
102967424/102967424 [=====] - 3s 0us/step
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)
agenet_class_index.json
img_preprocessed = preprocess_input(img_batch)
model = [(('n11939491', 'daisy', 0.5775759), ('n02206856', 'bee', 0.24938338),
tf.keras.applications.resnet50.ResNet50(include_top=True))]
prediction = model.predict(img_preprocessed)
print(decode_predictions(prediction, top=3)[0])

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

```