Assignment – 3

Build CNN Model for Classification of Flowers

Assignment Date	02 October 2022
Student Name	Mr. R. Nandhini
Student Roll Number	142219106053
Maximum Marks	2 Marks

TASKS:

1. Download the dataset

2. Image Augmentation

In [7]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [5]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
```

In [6]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [8]:
```
xtrain = train_datagen.flow_from_directory('/content/flowers',
                                           target_size=(64,64),
                                           class_mode='categorical',
                                           batch_size=100)
```

```
Found 4317 images belonging to 5 classes.
```

In [10]:
```
xtest = test_datagen.flow_from_directory('/content/flowers',
                                         target_size=(64,64),
                                         class_mode='categorical',
                                         batch_size=100)
```

```
Found 4317 images belonging to 5 classes.
```

3. Create model

In [11]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense
```

In [12]:
```
model = Sequential()
```

4. Adding Layers

In [13]:
```
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
```

MaxPooling

In [14]:
```
model.add(MaxPooling2D(pool_size=(2,2)))
```

Flatten

In [15]:
```
model.add(Flatten())
```

Dense Layer

In [16]:
```
model.add(Dense(300,activation='relu')) #hiddenlayer 1
model.add(Dense(150,activation='relu')) #hiddenlayer 2
```

5. Compile the model

In [18]:
```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

6. Fit the model

In [19]:
```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

In [20]:
```
early_stopping = EarlyStopping(monitor='val_accuracy',
                               patience=5)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                               patience=5,
                               factor=0.5,min_lr=0.00001)

callback = [reduce_lr,early_stopping]
```

In [21]:
```
# Train model

model.fit_generator(xtrain,
                    steps_per_epoch=len(xtrain),
                    epochs=100,
                    callbacks=callback,
                    validation_data=xtest,
                    validation_steps=len(xtest))
```

```
Epoch 1/100
44/44 [==============================] - 40s 894ms/step - loss: 1.4975 -
accuracy: 0.4003 - val_loss: 1.2238 - val_accuracy: 0.4964 - lr: 0.0010
Epoch 2/100
44/44 [==============================] - 39s 883ms/step - loss: 1.1079 -
accuracy: 0.5548 - val_loss: 1.1712 - val_accuracy: 0.5395 - lr: 0.0010
Epoch 3/100
44/44 [==============================] - 40s 907ms/step - loss: 1.0301 -
accuracy: 0.5956 - val_loss: 0.9753 - val_accuracy: 0.6284 - lr: 0.0010
Epoch 4/100
44/44 [==============================] - 39s 886ms/step - loss: 0.9719 -
accuracy: 0.6206 - val_loss: 0.9336 - val_accuracy: 0.6275 - lr: 0.0010
Epoch 5/100
44/44 [==============================] - 39s 878ms/step - loss: 0.8994 -
accuracy: 0.6518 - val_loss: 0.8369 - val_accuracy: 0.6919 - lr: 0.0010
Epoch 6/100
44/44 [==============================] - 39s 886ms/step - loss: 0.8470 -
accuracy: 0.6750 - val_loss: 0.8504 - val_accuracy: 0.6889 - lr: 0.0010
Epoch 7/100
44/44 [==============================] - 39s 884ms/step - loss: 0.8215 -
accuracy: 0.6891 - val_loss: 0.7804 - val_accuracy: 0.7100 - lr: 0.0010
Epoch 8/100
44/44 [==============================] - 40s 918ms/step - loss: 0.7763 -
accuracy: 0.7074 - val_loss: 0.7501 - val_accuracy: 0.7206 - lr: 0.0010
Epoch 9/100
44/44 [==============================] - 39s 887ms/step - loss: 0.7232 -
accuracy: 0.7301 - val_loss: 0.7413 - val_accuracy: 0.7285 - lr: 0.0010
Epoch 10/100
44/44 [==============================] - 39s 884ms/step - loss: 0.6905 -
accuracy: 0.7352 - val_loss: 0.6529 - val_accuracy: 0.7607 - lr: 0.0010
Epoch 11/100
44/44 [==============================] - 39s 885ms/step - loss: 0.6785 -
accuracy: 0.7461 - val_loss: 0.7277 - val_accuracy: 0.7246 - lr: 0.0010
Epoch 12/100
44/44 [==============================] - 40s 911ms/step - loss: 0.6417 -
accuracy: 0.7626 - val_loss: 0.6243 - val_accuracy: 0.7688 - lr: 0.0010
Epoch 13/100
44/44 [==============================] - 39s 890ms/step - loss: 0.6232 -
accuracy: 0.7642 - val_loss: 0.5709 - val_accuracy: 0.7869 - lr: 0.0010
Epoch 14/100
44/44 [==============================] - 39s 882ms/step - loss: 0.5917 -
accuracy: 0.7741 - val_loss: 0.6153 - val_accuracy: 0.7772 - lr: 0.0010
Epoch 15/100
44/44 [==============================] - 39s 884ms/step - loss: 0.5703 -
accuracy: 0.7878 - val_loss: 0.5209 - val_accuracy: 0.8050 - lr: 0.0010
Epoch 16/100
44/44 [==============================] - 39s 881ms/step - loss: 0.5262 -
accuracy: 0.8087 - val_loss: 0.5211 - val_accuracy: 0.8117 - lr: 0.0010
Epoch 17/100
44/44 [==============================] - 40s 907ms/step - loss: 0.5024 -
accuracy: 0.8156 - val_loss: 0.3861 - val_accuracy: 0.8622 - lr: 0.0010
Epoch 18/100
44/44 [==============================] - 39s 889ms/step - loss: 0.4733 -
accuracy: 0.8288 - val_loss: 0.3981 - val_accuracy: 0.8536 - lr: 0.0010
Epoch 19/100
44/44 [==============================] - 39s 887ms/step - loss: 0.4625 -
accuracy: 0.8309 - val_loss: 0.3904 - val_accuracy: 0.8582 - lr: 0.0010
Epoch 20/100
```

```
44/44 [==============================] - 39s 889ms/step - loss: 0.4534 -
accuracy: 0.8309 - val_loss: 0.5840 - val_accuracy: 0.7802 - lr: 0.0010
Epoch 21/100
44/44 [==============================] - 39s 887ms/step - loss: 0.4899 -
accuracy: 0.8251 - val_loss: 0.4176 - val_accuracy: 0.8464 - lr: 0.0010
Epoch 22/100
44/44 [==============================] - 39s 885ms/step - loss: 0.3994 -
accuracy: 0.8543 - val_loss: 0.3450 - val_accuracy: 0.8728 - lr: 0.0010
Epoch 23/100
44/44 [==============================] - 39s 896ms/step - loss: 0.4214 -
accuracy: 0.8434 - val_loss: 0.3122 - val_accuracy: 0.8955 - lr: 0.0010
Epoch 24/100
44/44 [==============================] - 39s 880ms/step - loss: 0.3556 -
accuracy: 0.8740 - val_loss: 0.3274 - val_accuracy: 0.8795 - lr: 0.0010
Epoch 25/100
44/44 [==============================] - 39s 882ms/step - loss: 0.3834 -
accuracy: 0.8608 - val_loss: 0.2577 - val_accuracy: 0.9099 - lr: 0.0010
Epoch 26/100
44/44 [==============================] - 40s 915ms/step - loss: 0.3258 -
accuracy: 0.8870 - val_loss: 0.2300 - val_accuracy: 0.9187 - lr: 0.0010
Epoch 27/100
44/44 [==============================] - 39s 886ms/step - loss: 0.3285 -
accuracy: 0.8819 - val_loss: 0.2780 - val_accuracy: 0.8969 - lr: 0.0010
Epoch 28/100
44/44 [==============================] - 39s 881ms/step - loss: 0.3346 -
accuracy: 0.8809 - val_loss: 0.2399 - val_accuracy: 0.9166 - lr: 0.0010
Epoch 29/100
44/44 [==============================] - 39s 884ms/step - loss: 0.2992 -
accuracy: 0.8911 - val_loss: 0.2409 - val_accuracy: 0.9085 - lr: 0.0010
Epoch 30/100
44/44 [==============================] - 39s 882ms/step - loss: 0.3078 -
accuracy: 0.8883 - val_loss: 0.2281 - val_accuracy: 0.9155 - lr: 0.0010
Epoch 31/100
44/44 [==============================] - 40s 910ms/step - loss: 0.2466 -
accuracy: 0.9155 - val_loss: 0.2137 - val_accuracy: 0.9266 - lr: 0.0010
Epoch 32/100
44/44 [==============================] - 39s 886ms/step - loss: 0.2508 -
accuracy: 0.9148 - val_loss: 0.2318 - val_accuracy: 0.9192 - lr: 0.0010
Epoch 33/100
44/44 [==============================] - 39s 898ms/step - loss: 0.2238 -
accuracy: 0.9201 - val_loss: 0.1724 - val_accuracy: 0.9358 - lr: 0.0010
Epoch 34/100
44/44 [==============================] - 39s 883ms/step - loss: 0.2174 -
accuracy: 0.9247 - val_loss: 0.1982 - val_accuracy: 0.9314 - lr: 0.0010
Epoch 35/100
44/44 [==============================] - 40s 911ms/step - loss: 0.1841 -
accuracy: 0.9375 - val_loss: 0.1722 - val_accuracy: 0.9405 - lr: 0.0010
Epoch 36/100
44/44 [==============================] - 39s 885ms/step - loss: 0.1896 -
accuracy: 0.9361 - val_loss: 0.1426 - val_accuracy: 0.9502 - lr: 0.0010
Epoch 37/100
44/44 [==============================] - 39s 888ms/step - loss: 0.1942 -
accuracy: 0.9349 - val_loss: 0.1617 - val_accuracy: 0.9442 - lr: 0.0010
Epoch 38/100
44/44 [==============================] - 39s 886ms/step - loss: 0.2163 -
accuracy: 0.9229 - val_loss: 0.1500 - val_accuracy: 0.9470 - lr: 0.0010
Epoch 39/100
44/44 [==============================] - 39s 883ms/step - loss: 0.1751 -
accuracy: 0.9363 - val_loss: 0.1106 - val_accuracy: 0.9622 - lr: 0.0010
```

```
Epoch 40/100
44/44 [==============================] - 40s 912ms/step - loss: 0.1849 -
accuracy: 0.9338 - val_loss: 0.2038 - val_accuracy: 0.9266 - lr: 0.0010
Epoch 41/100
44/44 [==============================] - 39s 883ms/step - loss: 0.1617 -
accuracy: 0.9486 - val_loss: 0.1293 - val_accuracy: 0.9560 - lr: 0.0010
Epoch 42/100
44/44 [==============================] - 39s 886ms/step - loss: 0.1336 -
accuracy: 0.9583 - val_loss: 0.1023 - val_accuracy: 0.9641 - lr: 0.0010
Epoch 43/100
44/44 [==============================] - 39s 890ms/step - loss: 0.1275 -
accuracy: 0.9590 - val_loss: 0.0941 - val_accuracy: 0.9720 - lr: 0.0010
Epoch 44/100
44/44 [==============================] - 40s 912ms/step - loss: 0.1351 -
accuracy: 0.9581 - val_loss: 0.1591 - val_accuracy: 0.9456 - lr: 0.0010
Epoch 45/100
44/44 [==============================] - 39s 891ms/step - loss: 0.1275 -
accuracy: 0.9574 - val_loss: 0.1165 - val_accuracy: 0.9625 - lr: 0.0010
Epoch 46/100
44/44 [==============================] - 39s 885ms/step - loss: 0.1260 -
accuracy: 0.9574 - val_loss: 0.0675 - val_accuracy: 0.9773 - lr: 0.0010
Epoch 47/100
44/44 [==============================] - 39s 882ms/step - loss: 0.1650 -
accuracy: 0.9423 - val_loss: 0.1186 - val_accuracy: 0.9618 - lr: 0.0010
Epoch 48/100
44/44 [==============================] - 39s 885ms/step - loss: 0.1151 -
accuracy: 0.9627 - val_loss: 0.0573 - val_accuracy: 0.9822 - lr: 0.0010
Epoch 49/100
44/44 [==============================] - 40s 912ms/step - loss: 0.0819 -
accuracy: 0.9743 - val_loss: 0.0733 - val_accuracy: 0.9764 - lr: 0.0010
Epoch 50/100
44/44 [==============================] - 39s 878ms/step - loss: 0.1102 -
accuracy: 0.9627 - val_loss: 0.1269 - val_accuracy: 0.9578 - lr: 0.0010
Epoch 51/100
44/44 [==============================] - 39s 882ms/step - loss: 0.1004 -
accuracy: 0.9666 - val_loss: 0.0730 - val_accuracy: 0.9778 - lr: 0.0010
Epoch 52/100
44/44 [==============================] - 39s 883ms/step - loss: 0.0952 -
accuracy: 0.9701 - val_loss: 0.0715 - val_accuracy: 0.9787 - lr: 0.0010
Epoch 53/100
44/44 [==============================] - 40s 912ms/step - loss: 0.0953 -
accuracy: 0.9683 - val_loss: 0.0742 - val_accuracy: 0.9761 - lr: 0.0010
```

Out[21]:

```
<keras.callbacks.History at 0x7f438af89490>
```

7. Save the model

In [22]:
```
model.save('Flowers.h5')
```

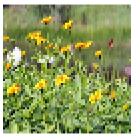
8. Test the model

In [23]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
```

In [24]:
```
img = image.load_img('/content/flowers/daisy/10300722094_28fa978807_n.jpg',target_size=(64,64))
```

In [25]:
```
img
```

Out[25]:



In [26]:
```
x = image.img_to_array(img)
x
```

```
array([[[ 35.,  12.,  56.],
        [ 52.,  32.,  60.],
        [ 59.,  46.,  63.],
        ...,
        [151., 156., 124.],
        [109., 133.,  73.],
        [162., 166., 141.]],

       [[ 65.,  54.,  68.],
        [ 92.,  88.,  77.],
        [ 89.,  85.,  74.],
        ...,
        [158., 165., 132.],
        [104., 126.,  77.],
        [140., 153., 125.]],

       [[123., 128.,  88.],
        [135., 143., 106.],
        [132., 136.,  99.],
        ...,
        [148., 158., 121.],
        [140., 163., 111.],
        [138., 152., 117.]],

       ...,

       [[  3.,   1.,  14.],
        [101., 122.,  83.],
        [ 78., 103.,  63.],
        ...,
        [ 79., 122.,   6.],
        [ 83., 113.,  17.],
        [ 98., 135.,  39.]],

       [[147., 172., 140.],
        [145., 173., 135.],
        [152., 175., 133.],
        ...,
        [ 61.,  99.,  38.],
        [133., 166., 113.],
        [  0.,  10.,   7.]],

       [[149., 171., 135.],
        [137., 156., 124.],
        [147., 170., 126.],
        ...,
        [ 97., 123.,  60.],
        [145., 182., 105.],
```

```
        [105., 128.,  58.]]], dtype=float32)
```

In [27]:

```python
x = np.expand_dims(x,axis=0)
x
```

```
array([[[[ 35.,  12.,  56.],
         [ 52.,  32.,  60.],
         [ 59.,  46.,  63.],
         ...,
         [151., 156., 124.],
         [109., 133.,  73.],
         [162., 166., 141.]],

        [[ 65.,  54.,  68.],
         [ 92.,  88.,  77.],
         [ 89.,  85.,  74.],
         ...,
         [158., 165., 132.],
         [104., 126.,  77.],
         [140., 153., 125.]],

        [[123., 128.,  88.],
         [135., 143., 106.],
         [132., 136.,  99.],
         ...,
         [148., 158., 121.],
         [140., 163., 111.],
         [138., 152., 117.]],

        ...,

        [[  3.,   1.,  14.],
         [101., 122.,  83.],
         [ 78., 103.,  63.],
         ...,
         [ 79., 122.,   6.],
         [ 83., 113.,  17.],
         [ 98., 135.,  39.]],

        [[147., 172., 140.],
         [145., 173., 135.],
         [152., 175., 133.],
         ...,
         [ 61.,  99.,  38.],
         [133., 166., 113.],
         [  0.,  10.,   7.]],

        [[149., 171., 135.],
         [137., 156., 124.],
         [147., 170., 126.],
         ...,
         [ 97., 123.,  60.],
         [145., 182., 105.],
         [105., 128.,  58.]]]], dtype=float32)
```

In [28]:

```
model.predict(x)
```

Out[28]:

```
array([[0., 0., 0., 1., 0.]], dtype=float32)
```

In [29]:

```
xtrain.class_indices
```

Out[29]:

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

In [31]:

```
op = ['daisy','sunfolower','rose','tulip','dandelion']
pred = np.argmax(model.predict(x))
op[pred]
```

Out[31]:

```
'tulip'
```