ASSIGNMENT-3

Built CNN Model for Classification Of Flowers

QUESTION-1

1. Download the Dataset ?

```
import numpy as np

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt

batch_size = 32

img_height = 180

data_dir ="/content/flowers"


import ImageDataGenerator

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

x_train = train_datagen.flow_from_directory(r"/content/flowers", target_size = (64,64) ,
class_mode = "categorical", batch_size = 100)

Found 4317 images belonging to 5 classes.
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
batch_size = 32
img_height = 180
img_width = 180
data_dir = "/content/flowers"
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

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

```
x_train = train_datagen.flow_from_directory(r"/content/flowers", target_size = (64,64) , class_mode = "categorical", batch_s
```

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

2.Image Argumentation ?

#Image Augumentation accuracy

data_augmentation = Sequential(

[

layers.RandomFlip("horizontal",input_shape=(img_height, img_width, 3)),

layers.RandomRotation(0.1),

layers.RandomZoom(0.1),

]

)

```
#Image Augumentation accuracy
data_augmentation = Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(img_height, img_width, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

3.Create the Model ?

```
from tensorflow.keras.models import Sequential from tensorflow.keras.layers import
Convolution2D,MaxPooling2D,Flatten,Dense

model = Sequential()

train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

Found 4317 files belonging to 5 classes.

Using 3454 files for training.

val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

Found 4317 files belonging to 5 classes.
```

Using 863 files for validation.

class_names = train_ds.class_names

print(class_names)

['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

plt.figure(figsize=(10, 10))

for images, labels in train_ds.take(1):

for i in range(9):

ax = plt.subplot(3, 3, i + 1)

plt.imshow(images[i].numpy().astype("uint8"))

plt.title(class_names[labels[i]])

plt.axis("off")

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense
model = Sequential()
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 4317 files belonging to 5 classes.
Using 3454 files for training.
```

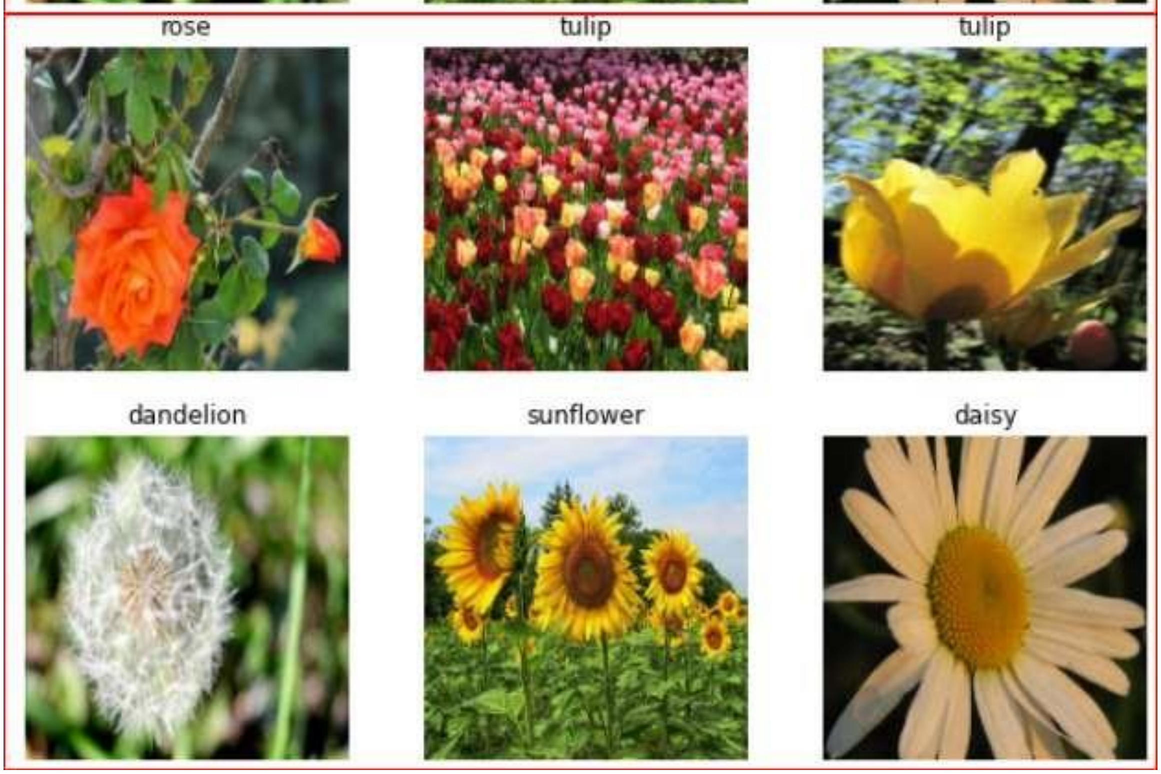
```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 4317 files belonging to 5 classes.
Using 863 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
```

```
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



4.Add Layers (Convolution,Maxpooling,Flatten,Dense-(Hidden Layer),Output)

```
model.add(Convolution2D(32, (3,3), activation = "relu", input_shape = (64,64,3) ))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(300, activation = "relu"))
model.add(Dense(150, activation = "relu")) #mulitple dense layers
model.add(Dense(5, activation = "softmax")) #output layer
#Adding the layers for accuracy
num_classes = len(class_names)

model = Sequential([
data_augmentation,
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
```

layers.Flatten(),

layers.Dense(128, activation='relu'),

)]

```
model.add(Convolution2D(32, (3,3), activation = "relu", input_shape = (64,64,3) ))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(300, activation = "relu"))
model.add(Dense(150, activation = "relu")) #mulitple dense layers
model.add(Dense(5, activation = "softmax")) #output layer
```

```
#Adding the layers for accuracy
num_classes = len(class_names)

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

3.Compile The Model

```
model.compile(loss = "categorical_crossentropy", metrics = ["accuracy"], optimizer = "adam")
len(x_train)
#Compile the model for further accuracy
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

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

```
44
```

```
#Compile the model for further accuracy
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
108/108 [==============================] - 128s 1s/step - loss: 1.3816 - accuracy: 0.4091 - val_loss: 1.2008 - val_accurac
y: 0.4913
Epoch 2/10
108/108 [==============================] - 125s 1s/step - loss: 1.0935 - accuracy: 0.5608 - val_loss: 1.0211 - val_accurac
y: 0.5794
Epoch 3/10
108/108 [==============================] - 126s 1s/step - loss: 0.9751 - accuracy: 0.6167 - val_loss: 0.9680 - val_accurac
y: 0.6130
Epoch 4/10
108/108 [==============================] - 126s 1s/step - loss: 0.9249 - accuracy: 0.6372 - val_loss: 0.8913 - val_accurac
y: 0.6512
Epoch 5/10
108/108 [==============================] - 125s 1s/step - loss: 0.8490 - accuracy: 0.6859 - val_loss: 0.8196 - val_accurac
y: 0.6744
Epoch 6/10
108/108 [==============================] - 126s 1s/step - loss: 0.8293 - accuracy: 0.6737 - val_loss: 0.9374 - val_accurac
y: 0.6477
Epoch 7/10
108/108 [==============================] - 125s 1s/step - loss: 0.7899 - accuracy: 0.7006 - val_loss: 0.7637 - val_accurac
y: 0.6871
Epoch 8/10
108/108 [==============================] - 125s 1s/step - loss: 0.7297 - accuracy: 0.7290 - val_loss: 0.7591 - val_accurac
y: 0.7196
Epoch 9/10
108/108 [==============================] - 126s 1s/step - loss: 0.7160 - accuracy: 0.7279 - val_loss: 0.8055 - val_accurac
y: 0.7115
Epoch 10/10
108/108 [==============================] - 125s 1s/step - loss: 0.6868 - accuracy: 0.7276 - val_loss: 0.8471 - val_accurac
y: 0.7022
```

5.Compile The Model

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

len(x_train)

44

#Compile the model for further accuracy

model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])

epochs=10

history = model.fit(

  train_ds,

  validation_data=val_ds,

  epochs=epochs

)

Epoch 1/10
```

108/108 [==============================] - 128s 1s/step - loss: 1.3816 - accuracy: 0.4091 - val_loss: 1.2008 - val_accuracy: 0.4913

Epoch 2/10

108/108 [==============================] - 125s 1s/step - loss: 1.0935 - accuracy: 0.5608 - val_loss: 1.0211 - val_accuracy: 0.5794

Epoch 3/10

108/108 [==============================] - 126s 1s/step - loss: 0.9751 - accuracy: 0.6167 - val_loss: 0.9680 - val_accuracy: 0.6130

Epoch 4/10

108/108 [==============================] - 126s 1s/step - loss: 0.9249 - accuracy: 0.6372 - val_loss: 0.8913 - val_accuracy: 0.6512

Epoch 5/10

108/108 [==============================] - 125s 1s/step - loss: 0.8490 - accuracy: 0.6859 - val_loss: 0.8196 - val_accuracy: 0.6744

Epoch 6/10

108/108 [==============================] - 126s 1s/step - loss: 0.8293 - accuracy: 0.6737 - val_loss: 0.9374 - val_accuracy: 0.6477

Epoch 7/10

108/108 [==============================] - 125s 1s/step - loss: 0.7899 - accuracy: 0.7006 - val_loss: 0.7637 - val_accuracy: 0.6871

Epoch 8/10

108/108 [==============================] - 125s 1s/step - loss: 0.7297 - accuracy: 0.7290 - val_loss: 0.7591 - val_accuracy: 0.7196

Epoch 9/10

108/108 [==============================] - 126s 1s/step - loss: 0.7160 - accuracy: 0.7279 - val_loss: 0.8055 - val_accuracy: 0.7115

Epoch 10/10

108/108 [==============================] - 125s 1s/step - loss: 0.6868 - accuracy: 0.7276 - val_loss: 0.8471 - val_accuracy: 0.7022

#To find the Training and Validation- Accuracy & Loss (Visualization)

acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

```
loss  = history.history['loss']

val_loss = history.history['val_loss']


epochs_range = range(epochs)


plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs_range, acc, label='Training Accuracy')

plt.plot(epochs_range, val_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')


plt.subplot(1, 2, 2)

plt.plot(epochs_range, loss, label='Training Loss')

plt.plot(epochs_range, val_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

model.save("flowers.h1")

model.save("flowers.m5")
```

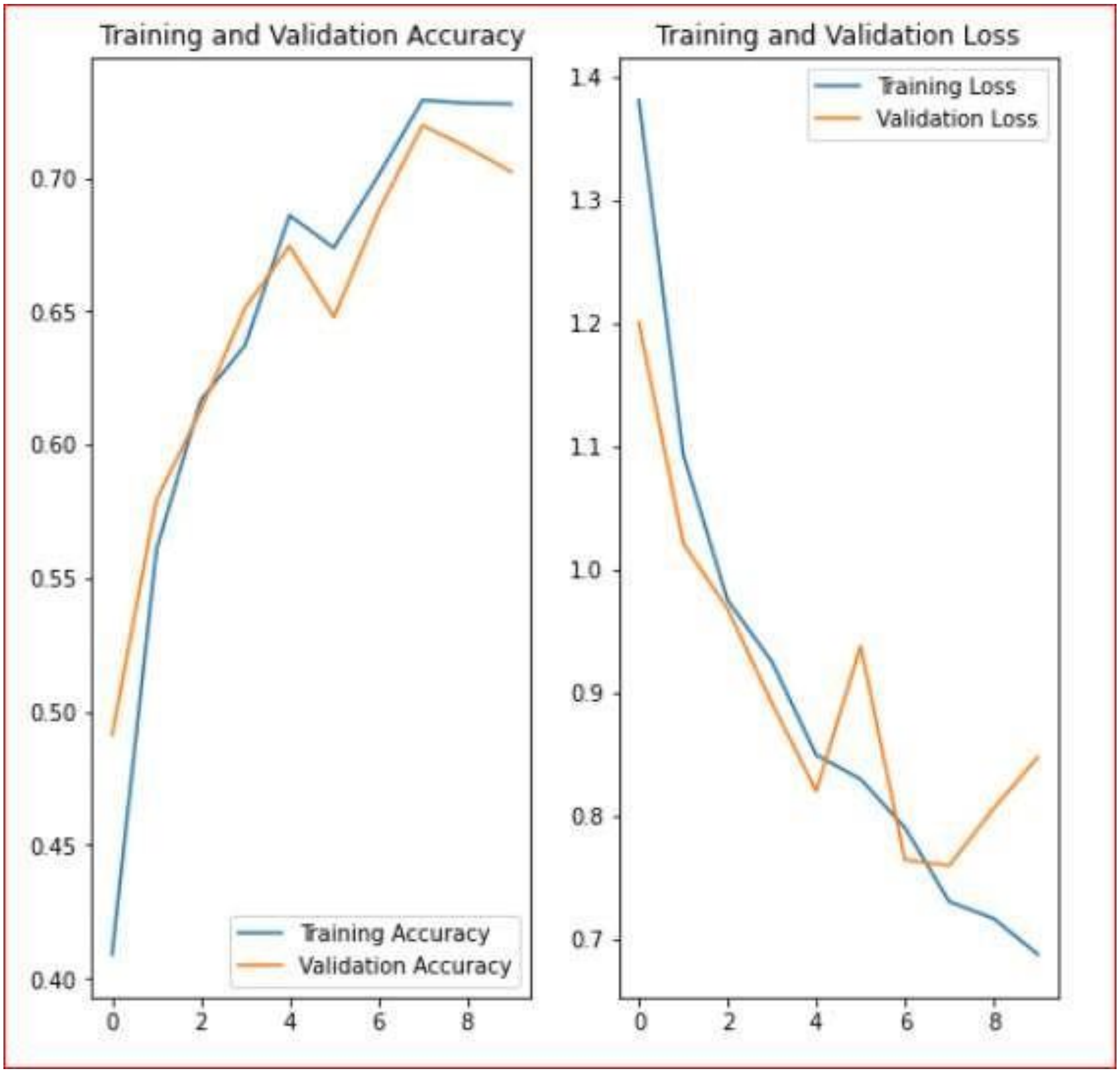
```
model.compile(loss = "categorical_crossentropy", metrics = ["accuracy"], optimizer = "adam")
len(x_train)
```

```
44
```

```
#Compile the model for further accuracy
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
108/108 [==============================] - 128s 1s/step - loss: 1.3816 - accuracy: 0.4091 - val_loss: 1.2008 - val_accurac
y: 0.4913
Epoch 2/10
108/108 [==============================] - 125s 1s/step - loss: 1.0935 - accuracy: 0.5608 - val_loss: 1.0211 - val_accurac
y: 0.5794
Epoch 3/10
108/108 [==============================] - 126s 1s/step - loss: 0.9751 - accuracy: 0.6167 - val_loss: 0.9680 - val_accurac
y: 0.6130
Epoch 4/10
108/108 [==============================] - 126s 1s/step - loss: 0.9249 - accuracy: 0.6372 - val_loss: 0.8913 - val_accurac
y: 0.6512
Epoch 5/10
108/108 [==============================] - 125s 1s/step - loss: 0.8490 - accuracy: 0.6859 - val_loss: 0.8196 - val_accurac
y: 0.6744
Epoch 6/10
108/108 [==============================] - 126s 1s/step - loss: 0.8293 - accuracy: 0.6737 - val_loss: 0.9374 - val_accurac
y: 0.6477
Epoch 7/10
108/108 [==============================] - 125s 1s/step - loss: 0.7899 - accuracy: 0.7006 - val_loss: 0.7637 - val_accurac
y: 0.6871
Epoch 8/10
108/108 [==============================] - 125s 1s/step - loss: 0.7297 - accuracy: 0.7290 - val_loss: 0.7591 - val_accurac
y: 0.7196
Epoch 9/10
108/108 [==============================] - 126s 1s/step - loss: 0.7160 - accuracy: 0.7279 - val_loss: 0.8055 - val_accurac
y: 0.7115
Epoch 10/10
108/108 [==============================] - 125s 1s/step - loss: 0.6868 - accuracy: 0.7276 - val_loss: 0.8471 - val_accurac
y: 0.7022
```

```python
#To find the Training and Validation- Accuracy & Loss (Visualization)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
model.save("flowers.h1")
```

```
model.save("flowers.m5")
```

8. Test The Model

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
model = load_model("/content/flowers.h1")
#Testing with a random rose image from Google
img = image.load_img("/content/daisy.gif", target_size = (64,64) ) img
x = image.img_to_array(img) x.ndim
3
x = np.expand_dims(x,axis = 0) x.ndim
4

labels = ['daisy','dandelion','roses','sunflowers','tulips'] x_pred=model.predict
x_pred

<bound method Model.predict of <keras.engine.sequential.Sequential object at 0x7f3901f2d850>>
labels[np.argmax(x_pred)]
'daisy'

#Testing the model with accuracy

daisy_url = "http://m.gettywallpapers.com/wp-content/uploads/2022/07/Daisy-Wallpaper-Images.jpg"

daisy_path = tf.keras.utils.get_file('Daisy-Wallpaper-Images', origin=daisy_url)

img = tf.keras.utils.load_img(

daisy_path, target_size=(img_height, img_width)

)
```

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("This image most likely belongs to {} with a {:.2f} percent confidence."

.format(class_names[np.argmax(x_pred)], 100 * np.max(score))

)

Downloading data from

http://m.gettywallpapers.com/wp-content/uploads/2022/07/Daisy-Wallpaper-Images.jpg

901120/897455 [==============================] - 0s 0us/step

909312/897455 [==============================] - 0s 0us/step

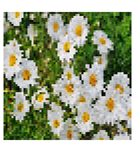
This image most likely belongs to daisy with a 94.37 percent confidence.

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
```

```
model = load_model("/content/flowers.h1")
```

```
#Testing with a random rose image from Google

img = image.load_img("/content/daisy.gif", target_size = (64,64) )
```

```
img
```



```
x = image.img_to_array(img)
x.ndim
```

3

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

4

```
labels = ['daisy','dandelion','roses','sunflowers','tulips']
```

```
x_pred=model.predict
```

```
x_pred
```

```
<bound method Model.predict of <keras.engine.sequential.Sequential object at 0x7f3901f2d850>>
```

```
labels[np.argmax(x_pred)]
```

```
'daisy'
```

```
#Testing the model with accuracy

daisy_url = "http://m.gettywallpapers.com/wp-content/uploads/2022/07/Daisy-Wallpaper-Images.jpg"
daisy_path = tf.keras.utils.get_file('Daisy-Wallpaper-Images', origin=daisy_url)

img = tf.keras.utils.load_img(
    daisy_path, target_size=(img_height, img_width)
)
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(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(x_pred)], 100 * np.max(score))
)
```

```
Downloading data from http://m.gettywallpapers.com/wp-content/uploads/2022/07/Daisy-Wallpaper-Images.jpg
901120/897455 [==============================] - 0s 0us/step
909312/897455 [==============================] - 0s 0us/step
This image most likely belongs to daisy with a 94.37 percent confidence.
```