FLOWER IMAGE CLASSIFICATION AND REGRESSION

PURPOSE:

Flower image analysis can be tackled with two main machine learning approaches: classification and regression. Classification aims to categorize flower images based on species. This is useful for tasks like automatically identifying flowers in photos or creating apps that help users learn about the flowers they encounter. Regression, on the other hand, focuses on predicting a continuous value from the image. This could be something like the flower's petal count, its diameter, or even its bloom time based on bud development.

PROCEDURE:

➢ CLASSIFICATION

1. Data Collection: Gather a dataset of flower images labeled with their specific species.
2. Preprocessing: Clean and prepare the images. This might involve resizing, cropping, or color correction for consistency.
3. Feature Extraction : In some approaches, extract features like color histograms or shapes to train the model.
4. Model Training: Choose a classification model, like Convolutional Neural Networks (CNNs) which excel at image recognition. Train the model on your labeled dataset.
5. Evaluation: Test the model's accuracy on unseen flower images and refine it if needed.
6. Prediction: Once satisfied, use the trained model to predict the flower species in new images.

➢ REGRESSION

1. Data Collection: Gather flower images with data on a measurable aspect like petal length or bud diameter.
2. Preprocessing: Prepare the images as before and format the measurement data.
3. Model Training: Choose a regression model, like linear regression, and train it on the image-measurement pairs.
4. Evaluation: Assess the model's accuracy in predicting measurements from new flower images.
5. Prediction: Use the model to estimate the target measurement (petal length etc.) for new flower images.

PROGRAM AND ANALYSIS:

1. **Import TensorFlow and other necessary libraries:**

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib

dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos.tar', origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')
```

OUTPUT:

Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz

228813984/228813984 [==============================] - 3s 0us/step

2. **After downloading, you should now have a copy of the dataset available**

```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

OUTPUT:

```
3670
```

```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
```

OUTPUT:

```
PIL.Image.open(str(roses[1]))
```

OUTPUT:

```
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
```

OUTPUT:

```
PIL.Image.open(str(tulips[1]))
```

OUTPUT:

3. Load and preprocess images

```
batch_size = 32
img_height = 180
img_width = 180
```

```
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)
```

OUTPUT:

Found 3670 files belonging to 5 classes.
Using 2936 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)
```

OUTPUT:

Found 3670 files belonging to 5 classes.
Using 734 files for validation.

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

OUTPUT:

['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

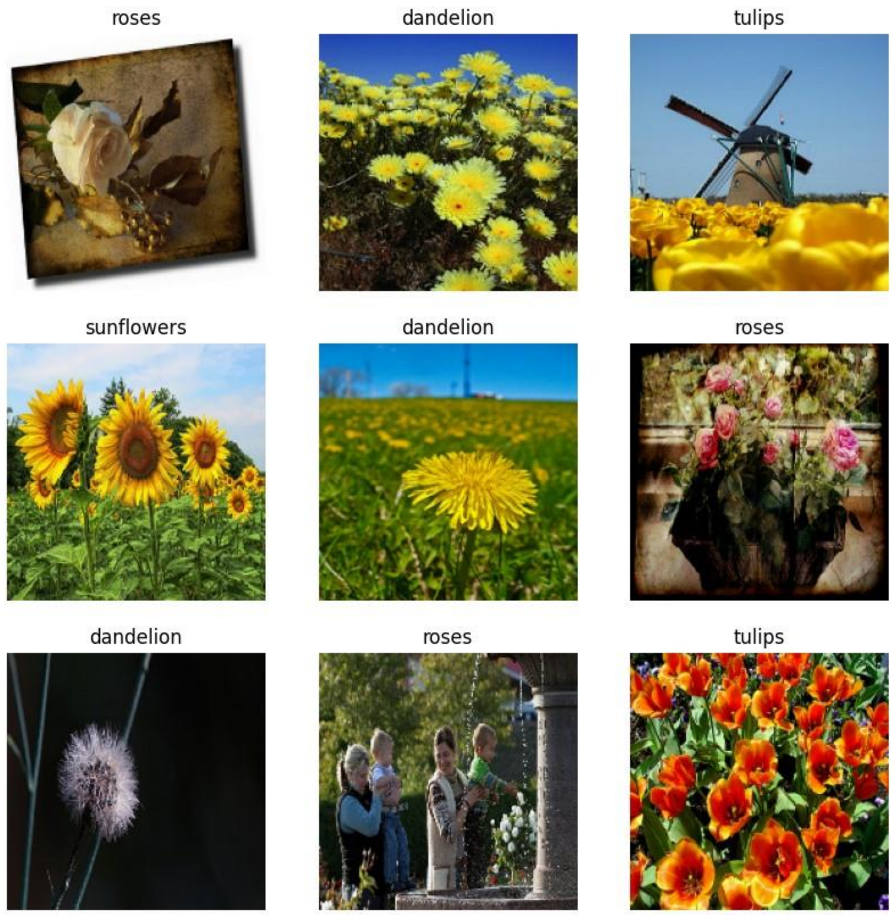
---

4. Visualize the data

```
import matplotlib.pyplot as plt

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")
```

OUTPUT:




```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

OUPUT:

(32, 180, 180, 3)
(32,)

5. **Configure the dataset for performance and Standardize the data**

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

OUTPUT:

0.0 1.0

6. Create the model , compile it and summary the model.

```
num_classes = len(class_names)

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

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3989285 (15.22 MB)
Trainable params: 3989285 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

7. Train the model

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

OUTPUT:

Epoch 1/10
92/92 [==============================] - 115s 1s/step - loss: 1.2806 - accuracy: 0.4567 - val_loss: 1.0623 - val_accuracy: 0.5804
Epoch 2/10
92/92 [==============================] - 141s 2s/step - loss: 0.9498 - accuracy: 0.6304 - val_loss: 0.9958 - val_accuracy: 0.6104
Epoch 3/10
92/92 [==============================] - 107s 1s/step - loss: 0.7479 - accuracy: 0.7176 - val_loss: 1.0373 - val_accuracy: 0.6090
Epoch 4/10
92/92 [==============================] - 101s 1s/step - loss: 0.5549 - accuracy: 0.7990 - val_loss: 0.9708 - val_accuracy: 0.6294
Epoch 5/10
92/92 [==============================] - 101s 1s/step - loss: 0.3622 - accuracy: 0.8665 - val_loss: 1.0698 - val_accuracy: 0.6553
Epoch 6/10
92/92 [==============================] - 95s 1s/step - loss: 0.1890 - accuracy: 0.9404 - val_loss: 1.4052 - val_accuracy: 0.6199
Epoch 7/10
92/92 [==============================] - 110s 1s/step - loss: 0.1015 - accuracy: 0.9714 - val_loss: 1.4746 - val_accuracy: 0.6512
Epoch 8/10
92/92 [==============================] - 98s 1s/step - loss: 0.0737 - accuracy: 0.9751 - val_loss: 1.4481 - val_accuracy: 0.6689
Epoch 9/10
92/92 [==============================] - 93s 1s/step - loss: 0.0346 - accuracy: 0.9908 - val_loss: 1.7552 - val_accuracy: 0.6540
Epoch 10/10
92/92 [==============================] - 89s 963ms/step - loss: 0.0291 - accuracy: 0.9908 - val_loss: 1.9723 - val_accuracy: 0.6540

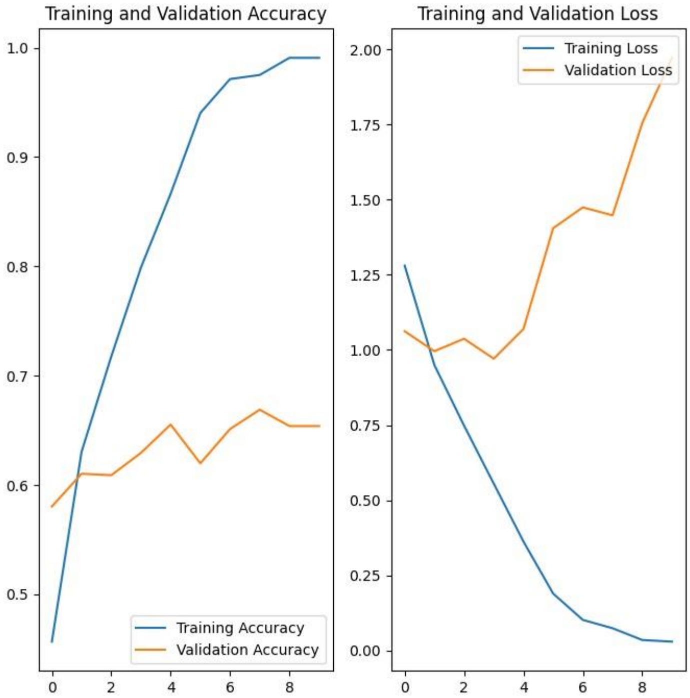
8. Visualize training results

```
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()
```

OUTPUT:



9. Overfitting

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

OUTPUT:

10. Dropout

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  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.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

OUTPUT:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPoolin  (None, 90, 90, 16)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten_1 (Flatten)         (None, 30976)             0

 dense_2 (Dense)             (None, 128)               3965056

 outputs (Dense)             (None, 5)                 645

=================================================================
Total params: 3989285 (15.22 MB)
Trainable params: 3989285 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

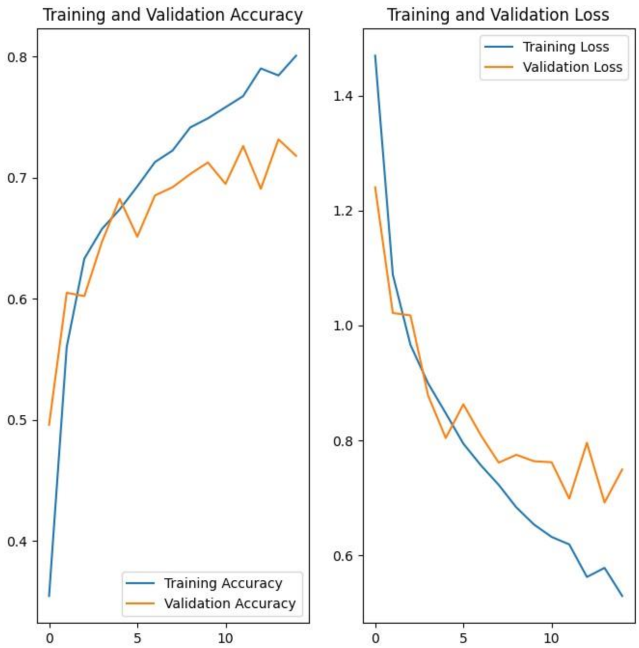
OUTPUT:

```
Epoch 1/15
92/92 [==============================] - 111s 1s/step - loss: 1.4693 - accuracy: 0.3546 - val_loss: 1.2404 - val_accuracy: 0.4959
Epoch 2/15
92/92 [==============================] - 107s 1s/step - loss: 1.0885 - accuracy: 0.5603 - val_loss: 1.0217 - val_accuracy: 0.6049
Epoch 3/15
```

92/92 [==============================] - 113s 1s/step - loss: 0.9662 - accuracy: 0.6332 - val_loss: 1.0175 - val_accuracy: 0.6022
Epoch 4/15
92/92 [==============================] - 109s 1s/step - loss: 0.8994 - accuracy: 0.6577 - val_loss: 0.8782 - val_accuracy: 0.6471
Epoch 5/15
92/92 [==============================] - 109s 1s/step - loss: 0.8473 - accuracy: 0.6737 - val_loss: 0.8042 - val_accuracy: 0.6826
Epoch 6/15
92/92 [==============================] - 115s 1s/step - loss: 0.7943 - accuracy: 0.6928 - val_loss: 0.8629 - val_accuracy: 0.6512
Epoch 7/15
92/92 [==============================] - 110s 1s/step - loss: 0.7564 - accuracy: 0.7129 - val_loss: 0.8085 - val_accuracy: 0.6853
Epoch 8/15
92/92 [==============================] - 114s 1s/step - loss: 0.7228 - accuracy: 0.7224 - val_loss: 0.7614 - val_accuracy: 0.6921
Epoch 9/15
92/92 [==============================] - 110s 1s/step - loss: 0.6835 - accuracy: 0.7415 - val_loss: 0.7750 - val_accuracy: 0.7030
Epoch 10/15
92/92 [==============================] - 106s 1s/step - loss: 0.6538 - accuracy: 0.7490 - val_loss: 0.7639 - val_accuracy: 0.7125
Epoch 11/15
92/92 [==============================] - 109s 1s/step - loss: 0.6321 - accuracy: 0.7582 - val_loss: 0.7620 - val_accuracy: 0.6948
Epoch 12/15
92/92 [==============================] - 107s 1s/step - loss: 0.6192 - accuracy: 0.7674 - val_loss: 0.6988 - val_accuracy: 0.7262
Epoch 13/15
92/92 [==============================] - 112s 1s/step - loss: 0.5625 - accuracy: 0.7902 - val_loss: 0.7959 - val_accuracy: 0.6907
Epoch 14/15
92/92 [==============================] - 113s 1s/step - loss: 0.5784 - accuracy: 0.7844 - val_loss: 0.6920 - val_accuracy: 0.7316
Epoch 15/15
92/92 [==============================] - 114s 1s/step - loss: 0.5295 - accuracy: 0.8007 - val_loss: 0.7494 - val_accuracy: 0.7180

```
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()
```
OUTPUT:



11. Prediction on new data

```
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)

img = tf.keras.utils.load_img(
    sunflower_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(score)], 100 * np.max(score))
)
```

OUTPUT:

Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg
117948/117948 [==============================] - 0s 0us/step
1/1 [==============================] - 0s 176ms/step
This image most likely belongs to sunflowers with a 98.61 percent confidence.

```
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
# Save the model.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
TF_MODEL_FILE_PATH = 'model.tflite' # The default path to the saved TensorFlow Lite model
interpreter = tf.lite.Interpreter(model_path=TF_MODEL_FILE_PATH)
interpreter.get_signature_list()
```

OUTPUT:

{'serving_default': {'inputs': ['sequential_1_input'], 'outputs': ['outputs']}}

```
classify_lite = interpreter.get_signature_runner('serving_default')
classify_lite
```

OUTPUT:

<tensorflow.lite.python.interpreter.SignatureRunner at 0x78550042ec80>

```
predictions_lite = classify_lite(sequential_1_input=img_array)['outputs']
score_lite = tf.nn.softmax(predictions_lite)
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score_lite)], 100 * np.max(score_lite))
)
```

OUTPUT:

This image most likely belongs to sunflowers with a 98.61 percent confidence.

```
print(np.max(np.abs(predictions - predictions_lite)))
```

OUTPUT:

2.3841858e-06

CONCLUSION:

In flower image analysis, both classification and regression techniques play important roles. Classification excels at identifying the specific flower species pictured. This allows for applications like automated flower identification in gardens or ecological surveys. Regression, on the other hand, can be used for tasks like predicting the bloom time or petal size based on image data. By choosing the right technique, researchers can unlock valuable insights from flower imagery.