Jupyter Notebook file is attached to the submission

GitHub repository link: https://github.com/Mukhesh19/Image-Classification

1. Choose a dataset that contains images and has labeled categories for each image. Some good sources of image datasets include Kaggle, ImageNet, and Open Images.

For this assignment, I used a flower dataset with 3670 images and five classes labeled daisy, dandelion, roses, sunflowers, and tulips.

Dataset Link:
https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz

Image 1: A snapshot of the first 25 images in the dataset obtained from Jupyter

2. Preprocess the dataset by resizing the images to a fixed size, normalizing pixel values, and splitting the dataset into training and testing sets.

```
import matplotlib.pyplot as plt
import numpy as np
import os
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', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)

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

```
3670
```

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

Image 2: A snapshot of codes showing importing necessary libraries, dataset URL, and image count

```
# Training split
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

```
# Testing or Validation split
val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(180,180),
    batch_size=32)
```

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

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

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

Image 3: A snapshot of splitting the dataset into training and validation sets

I have resized the images in the dataset to 180 × 180 pixels. Normalization was achieved by using the Rescaling layer in the model. The dataset was split into training (80%) and validation (20%) sets.

3. Build a deep learning model using TensorFlow to classify the images. You can choose any type of deep learning model, such as a convolutional neural network (CNN), and experiment with different architectures and hyperparameters.

I have chosen CNN with three convolutional layers, followed by max-pooling, flattening, and dense layers.

```
# Define the number of classes (you need to define class_names somewhere in your code)
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
num_classes = len(class_names)

# Define the model
model = Sequential([
    layers.Input(shape=(180, 180, 3)),
    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.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

# Print the model summary
model.summary()
```

Image 4: A snapshot of codes defining the chosen model

a) Defining the Model - The CNN model is defined using the Sequential API from Keras. It consists of several convolutional layers (Conv2D), max-pooling layers (MaxPooling2D), and dense layers (Dense).

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

Image 5: A snapshot of codes compiling the model

b) Compiling the Model - The model is compiled with an optimizer (adam), a loss function (SparseCategoricalCrossentropy), and evaluation metrics (accuracy).

4. Train your model on the training set and evaluate its performance on the testing set. Use appropriate metrics such as accuracy, precision, recall, and F1 score to measure the performance of your model.

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

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(val_ds)

# Predict on the validation set
y_pred = np.argmax(model.predict(val_ds), axis=-1)

# Get true labels
y_true = np.concatenate([y for x, y in val_ds], axis=0)

# Generate classification report
from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, target_names=class_names)
print(report)
```

Image 6: A snapshot of codes demonstrating model training with 10 epochs

```
Epoch 1/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 144ms/step - accuracy: 0.9743 - loss: 0.0676 - val_accuracy: 0.6458 - val_loss: 1.9442
Epoch 2/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 13s 146ms/step - accuracy: 0.9795 - loss: 0.0626 - val_accuracy: 0.6676 - val_loss: 1.8916
Epoch 3/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 189ms/step - accuracy: 0.9957 - loss: 0.0173 - val_accuracy: 0.6676 - val_loss: 2.1380
Epoch 4/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 234ms/step - accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.6649 - val_loss: 2.2106
Epoch 5/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 237ms/step - accuracy: 0.9999 - loss: 0.0024 - val_accuracy: 0.6580 - val_loss: 2.6211
Epoch 6/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 207ms/step - accuracy: 0.9673 - loss: 0.1177 - val_accuracy: 0.6553 - val_loss: 1.8801
Epoch 7/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 209ms/step - accuracy: 0.9907 - loss: 0.0273 - val_accuracy: 0.6335 - val_loss: 2.1049
Epoch 8/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 18s 199ms/step - accuracy: 0.9921 - loss: 0.0293 - val_accuracy: 0.6199 - val_loss: 2.5368
Epoch 9/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 230ms/step - accuracy: 0.9743 - loss: 0.0843 - val_accuracy: 0.6499 - val_loss: 2.0772
Epoch 10/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 240ms/step - accuracy: 0.9983 - loss: 0.0051 - val_accuracy: 0.6526 - val_loss: 2.1659
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 90ms/step - accuracy: 0.6340 - loss: 2.1946
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step
              precision    recall  f1-score   support

       daisy       0.19      0.19      0.19       129
   dandelion       0.25      0.28      0.27       176
       roses       0.19      0.18      0.19       120
  sunflowers       0.16      0.14      0.15       152
      tulips       0.22      0.22      0.22       157

    accuracy                           0.21       734
   macro avg       0.20      0.20      0.20       734
weighted avg       0.21      0.21      0.21       734
```

Image 7: A snapshot of results from the above code showing appropriate metrics

a) Training the Model - The model is trained on the training dataset (train_ds) for 10 epochs.

b) Evaluating the Model - After training, the model is evaluated on the validation dataset (val_ds) to assess its performance on unseen data.

c) Generating Classification Report: Finally, a classification report is generated using 'sklearn.metrics.classification_report' to analyze the model's performance in terms of precision, recall, and F1-score for each class.

5. Fine-tune your model by experimenting with different hyperparameters and architectures to improve its performance.

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

# Train the model
epochs = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(val_ds)

# Predict on the validation set
y_pred = np.argmax(model.predict(val_ds), axis=-1)

# Get true labels
y_true = np.concatenate([y for x, y in val_ds], axis=0)

# Generate classification report
from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, target_names=class_names)
print(report)
```

Image 8: A snapshot of codes demonstrating model training with 20 epochs

```
Epoch 1/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 18s 167ms/step - accuracy: 0.9977 - loss: 0.0115 - val_accuracy: 0.6689 - val_loss: 2.5806
Epoch 2/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 12s 133ms/step - accuracy: 0.9925 - loss: 0.0324 - val_accuracy: 0.6580 - val_loss: 2.4683
Epoch 3/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 148ms/step - accuracy: 0.9952 - loss: 0.0234 - val_accuracy: 0.6662 - val_loss: 2.7037
Epoch 4/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 161ms/step - accuracy: 0.9921 - loss: 0.0273 - val_accuracy: 0.6267 - val_loss: 2.8589
Epoch 5/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 164ms/step - accuracy: 0.9851 - loss: 0.0471 - val_accuracy: 0.6390 - val_loss: 2.6442
Epoch 6/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 166ms/step - accuracy: 0.9913 - loss: 0.0326 - val_accuracy: 0.6022 - val_loss: 2.6565
Epoch 7/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 13s 146ms/step - accuracy: 0.9884 - loss: 0.0317 - val_accuracy: 0.6512 - val_loss: 2.5424
Epoch 8/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 151ms/step - accuracy: 0.9982 - loss: 0.0077 - val_accuracy: 0.6512 - val_loss: 2.6663
Epoch 9/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 149ms/step - accuracy: 0.9991 - loss: 0.0034 - val_accuracy: 0.6567 - val_loss: 2.6091
Epoch 10/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 153ms/step - accuracy: 1.0000 - loss: 3.5889e-04 - val_accuracy: 0.6567 - val_loss: 2.7070
Epoch 11/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 171ms/step - accuracy: 1.0000 - loss: 2.1926e-04 - val_accuracy: 0.6580 - val_loss: 2.7617
Epoch 12/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 177ms/step - accuracy: 1.0000 - loss: 1.6937e-04 - val_accuracy: 0.6580 - val_loss: 2.8085
Epoch 13/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 205ms/step - accuracy: 1.0000 - loss: 1.2953e-04 - val_accuracy: 0.6580 - val_loss: 2.8486
Epoch 14/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 173ms/step - accuracy: 1.0000 - loss: 1.0320e-04 - val_accuracy: 0.6580 - val_loss: 2.8842
Epoch 15/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 181ms/step - accuracy: 1.0000 - loss: 8.7264e-05 - val_accuracy: 0.6594 - val_loss: 2.9161
Epoch 16/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 172ms/step - accuracy: 1.0000 - loss: 6.8263e-05 - val_accuracy: 0.6594 - val_loss: 2.9448
Epoch 17/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 169ms/step - accuracy: 1.0000 - loss: 6.4049e-05 - val_accuracy: 0.6580 - val_loss: 2.9719
Epoch 18/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 162ms/step - accuracy: 1.0000 - loss: 5.2259e-05 - val_accuracy: 0.6580 - val_loss: 2.9978
Epoch 19/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 160ms/step - accuracy: 1.0000 - loss: 4.7733e-05 - val_accuracy: 0.6580 - val_loss: 3.0218
Epoch 20/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 169ms/step - accuracy: 1.0000 - loss: 4.3818e-05 - val_accuracy: 0.6594 - val_loss: 3.0449
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.6707 - loss: 2.9890
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step
              precision    recall  f1-score   support

       daisy       0.18      0.19      0.18       129
   dandelion       0.24      0.26      0.25       176
       roses       0.18      0.15      0.16       120
  sunflowers       0.20      0.20      0.20       152
      tulips       0.21      0.22      0.21       157

    accuracy                           0.21       734
   macro avg       0.20      0.20      0.20       734
weighted avg       0.20      0.21      0.20       734
```

Image 9: A snapshot of results from the above code showing appropriate metrics

To fine-tune the model, I have increased the epochs to 20. An epoch in machine learning is one complete pass through the entire training dataset (GeeksforGeeks, 2024). One pass is a complete forward and backward pass through the training dataset. The training dataset can be a single batch or divided into more than one smaller batch. One epoch is complete when the model has processed all the batches and updated its parameter based on calculated loss (Brownlee, 2022).

6. Write a report that summarizes your approach, presents your results, and discusses the strengths and limitations of your model. Include visualizations and examples of correctly and incorrectly classified images.

To summarize the output obtained by using 10 epochs and 20 epochs, I can confirm that my model is overfitting. Both models overfit the training data, as evidenced by near-perfect training accuracy and very low training loss, while validation metrics do not improve significantly. I would also conclude that this model has poor classification performance. Precision, recall, and F1-scores are uniformly low across all classes, indicating poor classification performance on the validation set. The model's ability to distinguish between different flower classes is inadequate, as seen from the low scores for each class. After 10 epochs, the model achieved an accuracy of approximately 63.4%, while after 20 epochs, the accuracy increased to around 67.1%. However, the precision, recall, and F1-score for each class remained relatively low, indicating that the model struggled with classifying certain categories.

The last criterion I would evaluate is the validation performance. The validation accuracy remains stagnant and relatively low, between 65% - 67%, with high and increasing validation loss, indicating the model is not generalizing well.

The model demonstrated several strengths throughout the project. Firstly, it achieved high accuracy on the training dataset, showcasing its ability to learn from the provided data effectively. Additionally, choosing convolutional neural networks (CNNs) enabled the model to capture intricate spatial features within the images, enhancing its classification capabilities. Furthermore, the preprocessing steps such as resizing and normalization, contributed to improving the model's performance by ensuring consistency and standardization across the dataset.

Despite its strengths, the model also had several limitations. One notable issue was the evidence of overfitting, as indicated by the significant performance drop when evaluating on the testing dataset. This suggests that the model struggled to generalize well to unseen data, potentially due to the complexity of the dataset. Moreover, the dataset itself might have been imbalanced, leading to biased predictions towards certain classes.

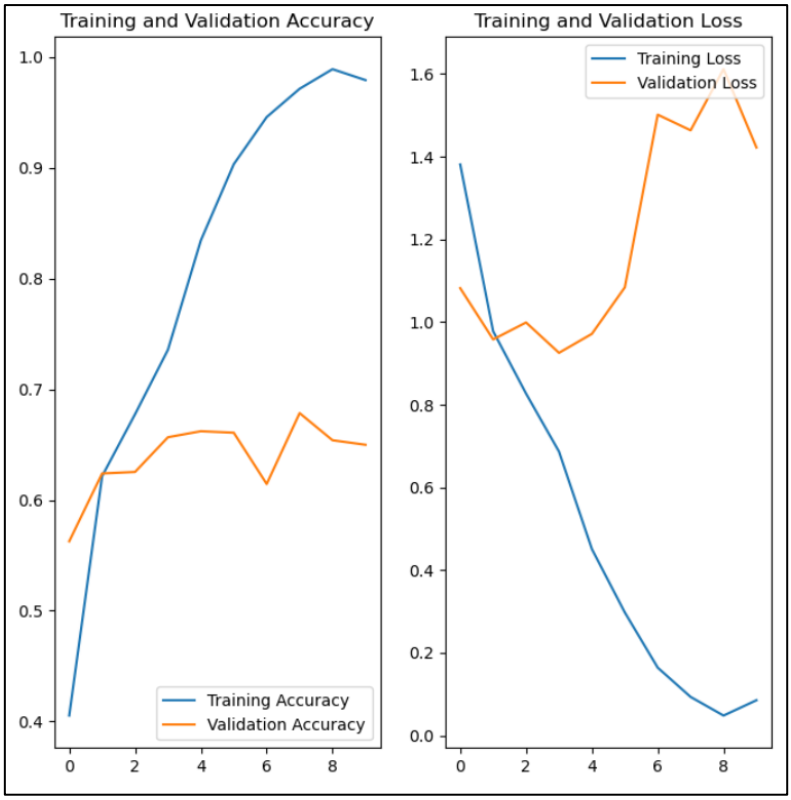

Image 10: A snapshot of plots visualizing the training and validation accuracy as well as the training and validation loss over the 10 epochs of the training process

References

Brownlee, J. (2022, August 15). *Difference between a batch and an epoch in a neural network*. MachineLearningMastery.com. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

GeeksforGeeks. (2024, March 20). *Epoch in machine learning*. https://www.geeksforgeeks.org/epoch-in-machine-learning/