Homework3

April 19, 2024

0.0.1 Name: Hari Vengadesh

Homework 3

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

[2]:
```
import numpy as np
import pandas as pd
from datetime import datetime

import PIL
import PIL.Image
import pathlib

from matplotlib import pyplot as plt
import matplotlib as mpl
import tensorflow as tf

from sklearn.metrics import confusion_matrix
import seaborn as sns

print(tf.__version__)
```

```
2.10.0
```

[3]:
```
print(tf.config.list_physical_devices('GPU'))
print(tf.test.is_built_with_cuda)
print(tf.test.gpu_device_name())
print(tf.config.get_visible_devices())
```

```
[]
<function is_built_with_cuda at 0x0000018F572F4E50>

[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

Parameters

[4]:
```
AUTOTUNE = tf.data.AUTOTUNE
```

[5]:
```
batch_size = 32
img_height = 101
img_width = 101
```

Data

[6]:
```
data_dirpathname = r'E:\Data Science\ECE565 - Machine␣
 ↪Learning\datasets\flower_photos'
data_dir = pathlib.Path(data_dirpathname)

class_names = os.listdir(data_dir)
num_classes = len(class_names)
```

0.1 1. 2.a) Code that will count number of classes

0.2 1. 2.b) Number of images in each class

[7]:
```
images_per_class = {}
for class_name in class_names:
    images_per_class[class_name] = len(os.listdir(os.path.join(data_dir,␣
 ↪class_name)))

print("Number of classes:", num_classes)
print("Number of images in each class:", images_per_class)
```

```
Number of classes: 5
Number of images in each class: {'daisy': 633, 'dandelion': 898, 'roses': 641,
'sunflowers': 699, 'tulips': 799}
```

[8]:
```
image_count =  len(list(data_dir.glob('*/*.jpg')))
print('Image count:', image_count)
```

```
Image count: 3670
```

[9]:
```
one = list(data_dir.glob('daisy/*'))
PIL.Image.open(str(one[0]))
```

[9]:

[10]:
```
sample_image = PIL.Image.open(str(one[1]))
img = np.asarray(sample_image)
img.shape
```

[10]:
```
(313, 500, 3)
```

Setup Dataset Pipeline

[11]:
```
list_ds = tf.data.Dataset.list_files(str(data_dir/'*/*'), shuffle=False)
list_ds = list_ds.shuffle(image_count, reshuffle_each_iteration=False)
list_ds
```

[11]:
```
<ShuffleDataset element_spec=TensorSpec(shape=(), dtype=tf.string, name=None)>
```

[12]:
```
for f in list_ds.take(5):
    print(f.numpy())
```

```
b'E:\\Data Science\\ECE565 - Machine
Learning\\datasets\\flower_photos\\daisy\\14399435971_ea5868c792.jpg'
b'E:\\Data Science\\ECE565 - Machine
Learning\\datasets\\flower_photos\\sunflowers\\20183028616_beb937e75c_m.jpg'
b'E:\\Data Science\\ECE565 - Machine
Learning\\datasets\\flower_photos\\sunflowers\\13095941995_9a66faa713_n.jpg'
b'E:\\Data Science\\ECE565 - Machine
Learning\\datasets\\flower_photos\\sunflowers\\7791014076_07a897cb85_n.jpg'
b'E:\\Data Science\\ECE565 - Machine
```

```
Learning\\datasets\\flower_photos\\sunflowers\\9558632814_e78a780f4f.jpg'
```

0.3 1. 1. Data must be split in train/test and validation set

[13]:
```
test_size = int(image_count*0.2)
train_ds = list_ds.skip(test_size)
val_ds = list_ds.take(test_size)
```

[14]:
```
print(tf.data.experimental.cardinality(train_ds).numpy())
print(tf.data.experimental.cardinality(val_ds).numpy())
```

```
2936
734
```

Helper Functions

[15]:
```
def get_label(file_path):
    parts = tf.strings.split(file_path, os.path.sep)
    one_hot = parts[-2] == class_names
    return tf.argmax(one_hot)
```

[16]:
```
def decode_img(img):
    img = tf.io.decode_jpeg(img, channels=3)
    return tf.image.resize(img, [img_height, img_width])
```

[17]:
```
def process_path(file_path):
    label = get_label(file_path)
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label
```

[18]:
```
# Set 'num_parallel_calls' so multiple images are loaded/processed in parallel.

train_ds = train_ds.map(process_path, num_parallel_calls=AUTOTUNE)
val_ds = val_ds.map(process_path, num_parallel_calls=AUTOTUNE)
```

0.4 1. 2.c) Image resized to 101x101xNo_Of_channels.

0.5 1. 2.d) Automatic data label extraction based on sub-directory name.

0.6 1. 2.e) Display one batch of image/label using the dataset api.

[19]:
```
for image, label in train_ds.take(1):
    print("Image shape:", image.numpy().shape)
    print("Label:", class_names[label.numpy()])
```

```
Image shape: (101, 101, 3)
Label: tulips
```

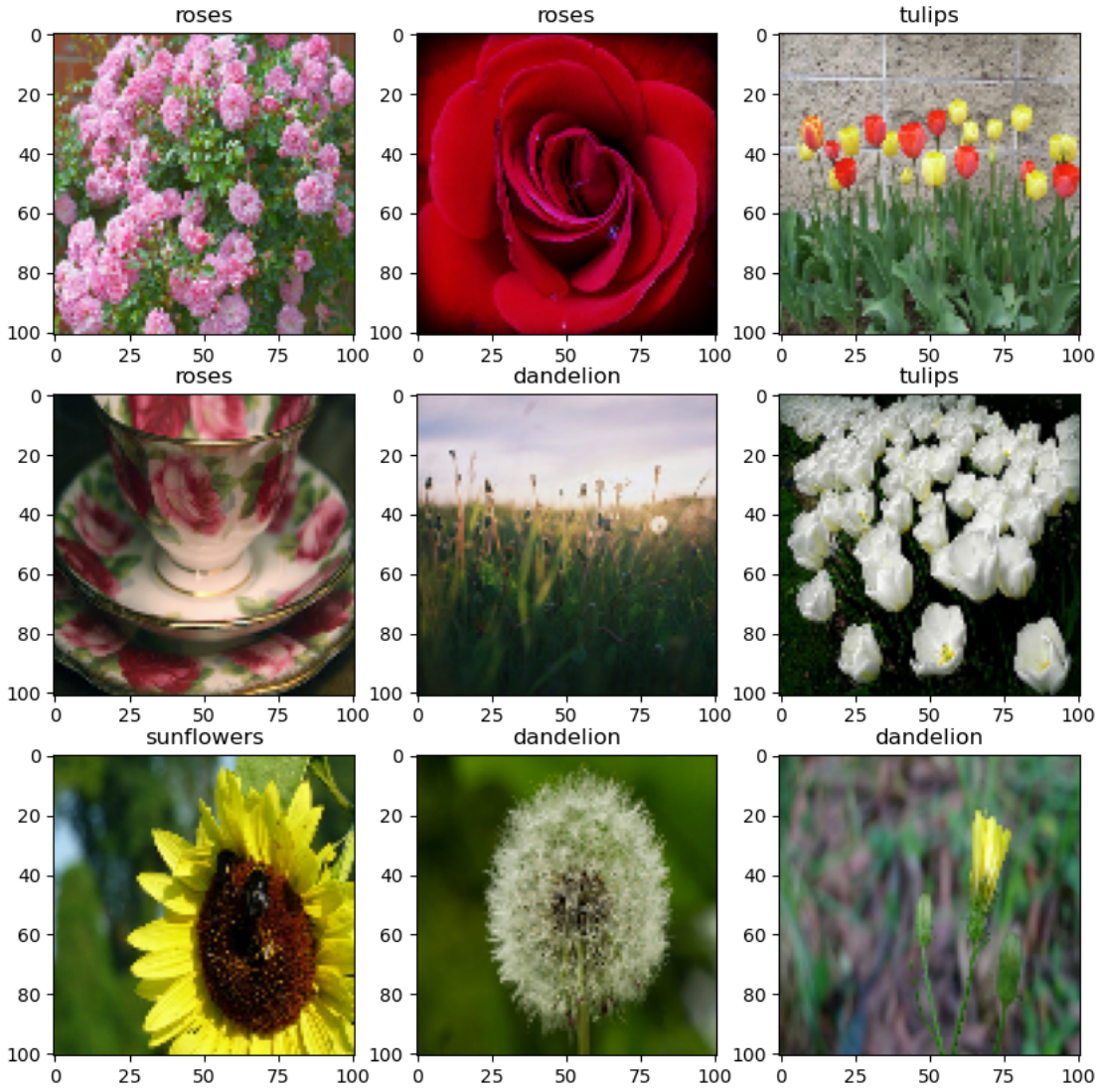
[20]:
```
def configure_for_performance(ds):
    ds = ds.cache()
    ds = ds.shuffle(buffer_size=1000)
    ds = ds.batch(batch_size=64)
    return ds
```

[21]:
```
train_ds = configure_for_performance(train_ds)
val_ds = configure_for_performance(val_ds)
```

Test Dataset Pipeline

[22]:
```
image_batch, label_batch = next(iter(train_ds))
```

[23]:
```
plt.figure(figsize=(10,10))
for i in range(9):
    ax = plt.subplot(3, 3, i+1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    label = label_batch[i]
    plt.title(class_names[label])
```



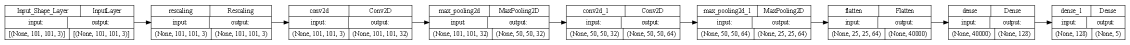
0.7 2. Model

[24]:
```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(img_height, img_width, 3),␣
 ↪name='Input_Shape_Layer'),
    tf.keras.layers.Rescaling(1./255),  # Normalize pixel values
    tf.keras.layers.Conv2D(32, 3, activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
```

```
])
```

[25]:
```
tf.keras.utils.plot_model(model=model, rankdir="LR", dpi=72, show_shapes=True)
```

[25]:



[26]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

0.7.1 Training / Validation Cycle

[27]:
```
logdir = "logs/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

0.8 3. Train/Validation

0.9 3. 1. Early stopping

[28]:
```
early_stopping_callback = tf.keras.callbacks.
↪EarlyStopping(monitor='val_accuracy', mode='max', patience=5)
```

[29]:
```
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=101,
    callbacks=[tensorboard_callback, early_stopping_callback],
    verbose=1
)
```

```
Epoch 1/101
46/46 [==============================] - 9s 187ms/step - loss: 1.6211 -
accuracy: 0.3995 - val_loss: 1.2117 - val_accuracy: 0.5068
Epoch 2/101
46/46 [==============================] - 6s 130ms/step - loss: 1.0837 -
accuracy: 0.5620 - val_loss: 1.1077 - val_accuracy: 0.5559
Epoch 3/101
46/46 [==============================] - 6s 135ms/step - loss: 0.8752 -
accuracy: 0.6662 - val_loss: 0.9830 - val_accuracy: 0.6226
Epoch 4/101
46/46 [==============================] - 6s 131ms/step - loss: 0.6994 -
accuracy: 0.7442 - val_loss: 0.9742 - val_accuracy: 0.6213
Epoch 5/101
46/46 [==============================] - 6s 132ms/step - loss: 0.5106 -
```

```
accuracy: 0.8164 - val_loss: 0.9849 - val_accuracy: 0.6444
Epoch 6/101
46/46 [==============================] - 6s 137ms/step - loss: 0.3447 -
accuracy: 0.8873 - val_loss: 1.0385 - val_accuracy: 0.6512
Epoch 7/101
46/46 [==============================] - 6s 126ms/step - loss: 0.2132 -
accuracy: 0.9373 - val_loss: 1.1377 - val_accuracy: 0.6376
Epoch 8/101
46/46 [==============================] - 6s 133ms/step - loss: 0.1276 -
accuracy: 0.9676 - val_loss: 1.2887 - val_accuracy: 0.6362
Epoch 9/101
46/46 [==============================] - 6s 139ms/step - loss: 0.0948 -
accuracy: 0.9785 - val_loss: 1.4528 - val_accuracy: 0.6471
Epoch 10/101
46/46 [==============================] - 6s 134ms/step - loss: 0.0446 -
accuracy: 0.9925 - val_loss: 1.5364 - val_accuracy: 0.6294
Epoch 11/101
46/46 [==============================] - 7s 146ms/step - loss: 0.0276 -
accuracy: 0.9966 - val_loss: 1.6931 - val_accuracy: 0.6349
```

[29]:
```
<keras.callbacks.History at 0x18f6d04dee0>
```

Evaluate Trained Model

0.10 4. Model Evaluation

0.11 4. 2. Accuracy Metrics

[30]:
```
test_loss, test_acc = model.evaluate(val_ds, verbose=2)
print('\nTest Accuracy:', test_acc)
```

```
12/12 - 1s - loss: 1.6931 - accuracy: 0.6349 - 561ms/epoch - 47ms/step

Test Accuracy: 0.6348773837089539
```

[31]:
```
all_predictions = []
all_labels = []
```

0.12 4. 1. Confusion Matrix

[32]:
```
for images, labels in val_ds:
    predictions = model.predict(images)
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = labels.numpy()

    all_predictions.extend(predicted_classes)
    all_labels.extend(true_classes)
```

```
2/2 [==============================] - 0s 52ms/step
2/2 [==============================] - 0s 23ms/step
2/2 [==============================] - 0s 42ms/step
2/2 [==============================] - 0s 39ms/step
2/2 [==============================] - 0s 27ms/step
2/2 [==============================] - 0s 24ms/step
2/2 [==============================] - 0s 33ms/step
2/2 [==============================] - 0s 27ms/step
2/2 [==============================] - 0s 24ms/step
2/2 [==============================] - 0s 66ms/step
2/2 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 117ms/step
```

[33]:
```
cm = confusion_matrix(all_labels, all_predictions)
class_names_array = np.array(class_names)
```

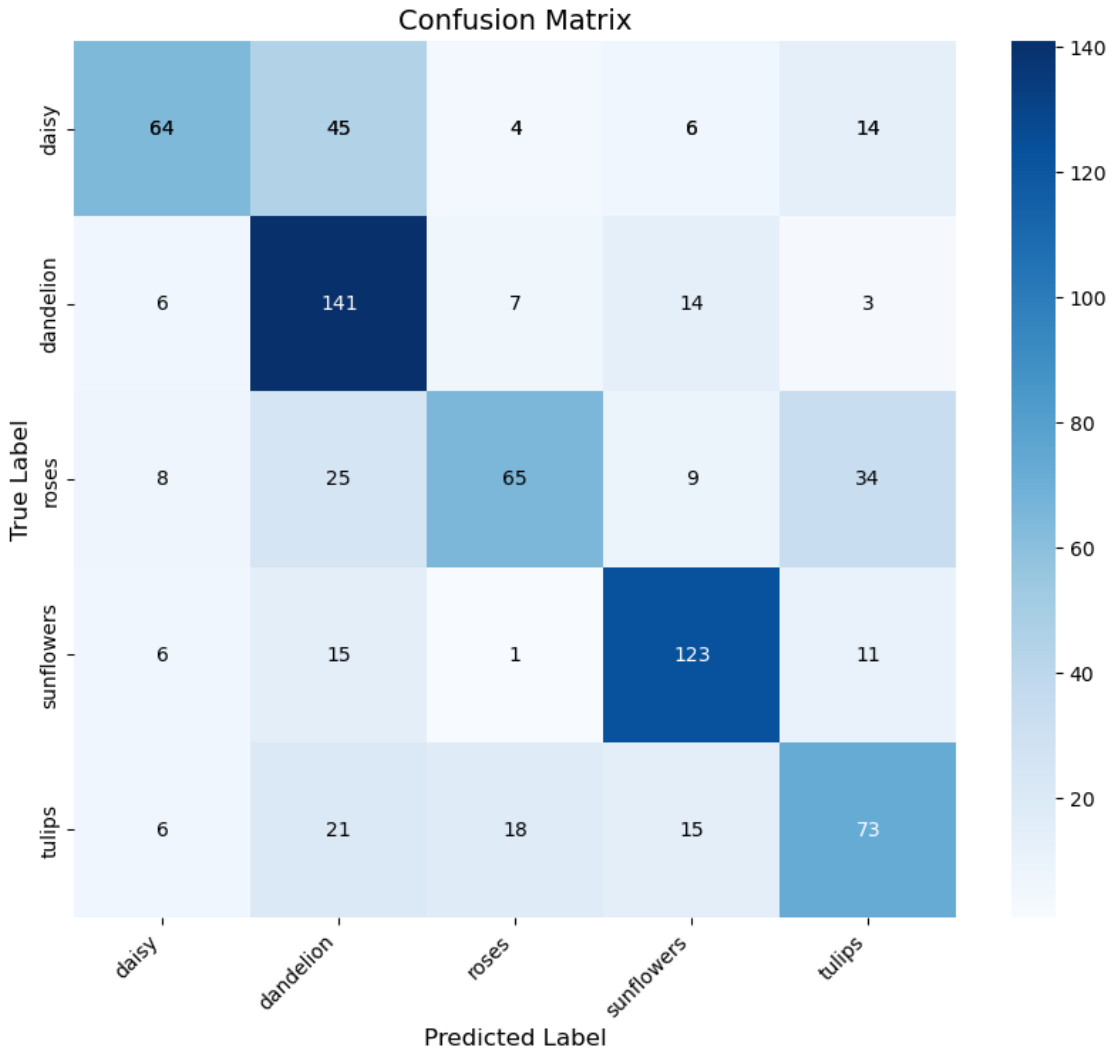
[34]:
```
plt.figure(figsize=(10, 8))  # Increase figure size to make room for labels
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                 xticklabels=class_names, yticklabels=class_names)

# Set tick labels appearance
plt.xticks(rotation=45, ha='right', fontsize=10)  # Adjust font size or␣
 ↪rotation as needed
plt.yticks(fontsize=10)

# Colorbar label and title
plt.title('Confusion Matrix', fontsize=14)  # Increase title font size
plt.ylabel('True Label', fontsize=12)
plt.xlabel('Predicted Label', fontsize=12)

# Loop over data dimensions and create text annotations with contrasting colors
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j+0.5, i+0.5, format(cm[i, j], 'd'),
                ha='center', va='center',
                color='white' if cm[i, j] > thresh else 'black')

plt.show()
```



Make Predictions

[35]:
```
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
```

[36]:
```
predictions = probability_model.predict(val_ds)
```

```
12/12 [==============================] - 0s 35ms/step
```

[37]:
```
predictions[0]
```

[37]:
```
array([0.16143993, 0.3843893 , 0.15127808, 0.15127553, 0.15161708],
      dtype=float32)
```

[38]:
```
class_names[np.argmax(predictions[0])]
```

[38]:
```
'dandelion'
```

[39]:
```
image_batch, label_batch = next(iter(val_ds))
```

[40]:
```
label_batch
```

[40]:
```
<tf.Tensor: shape=(64,), dtype=int64, numpy=
array([2, 1, 0, 3, 0, 3, 0, 0, 3, 4, 4, 1, 4, 2, 4, 2, 3, 3, 4, 4, 4, 2,
       2, 3, 3, 3, 4, 2, 4, 3, 4, 2, 1, 0, 1, 1, 0, 2, 3, 0, 2, 2, 0, 4,
       2, 4, 0, 1, 2, 1, 1, 0, 3, 2, 0, 3, 3, 1, 4, 4, 3, 1, 0, 0],
      dtype=int64)>
```

[41]:
```
predictions = probability_model.predict(image_batch)
```

```
2/2 [==============================] - 0s 22ms/step
```

[42]:
```
np.argmax(predictions, axis=1)
```

[42]:
```
array([4, 1, 0, 3, 0, 1, 0, 1, 3, 1, 4, 1, 4, 1, 1, 2, 3, 3, 4, 4, 1, 1,
       4, 1, 3, 3, 4, 3, 4, 1, 3, 2, 1, 1, 1, 1, 1, 1, 3, 0, 2, 2, 0, 4,
       3, 4, 4, 1, 2, 1, 1, 3, 4, 2, 0, 3, 1, 1, 3, 4, 3, 1, 0, 0],
      dtype=int64)
```

[43]:
```
image_batch[0].shape
```

[43]:
```
TensorShape([101, 101, 3])
```

[44]:
```
predictions_prob = probability_model.predict(image_batch)
predictions = np.argmax(predictions_prob, axis=1)
```

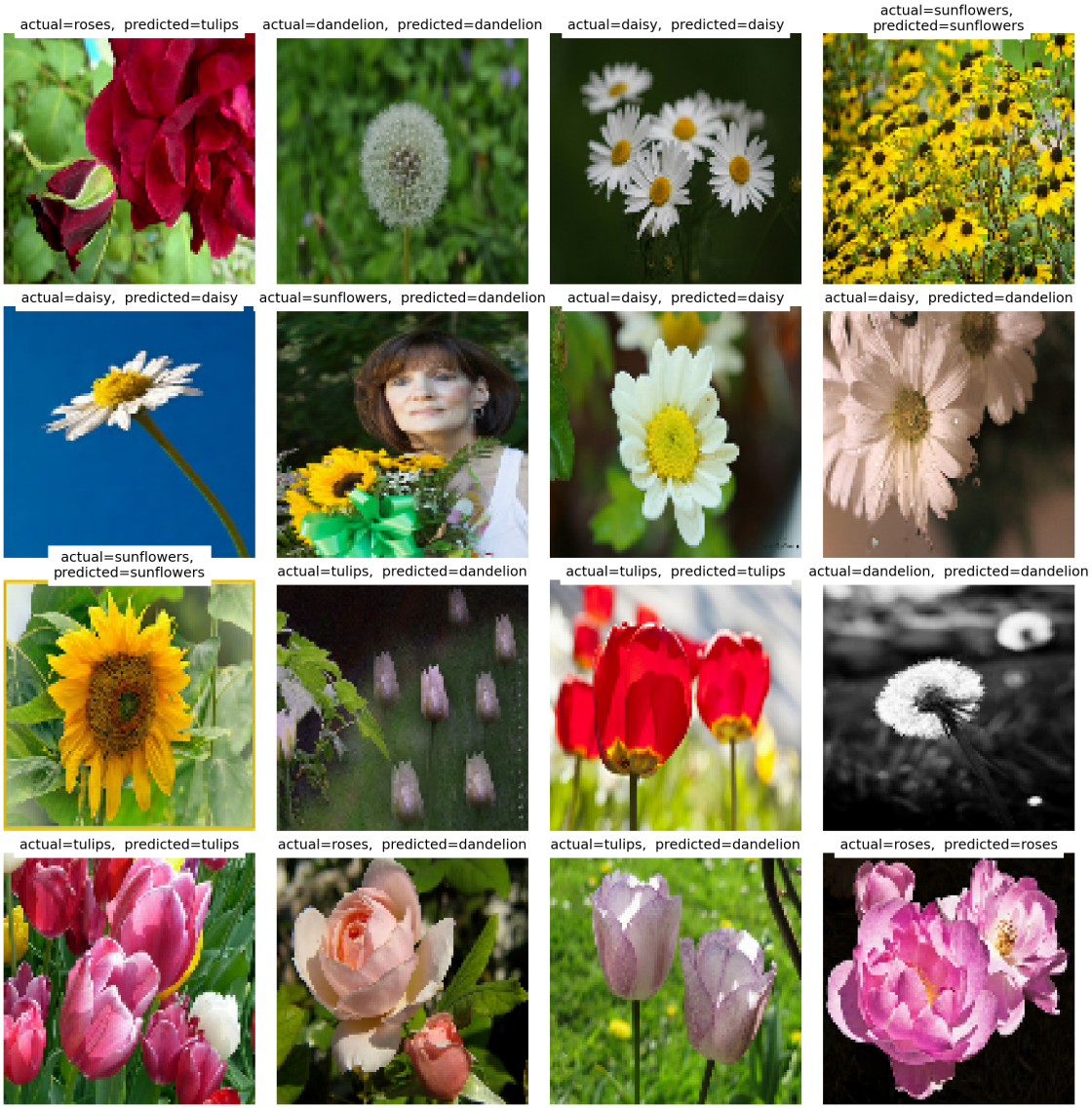
```
2/2 [==============================] - 0s 51ms/step
```

0.13 5. Model Predictions

[45]:
```
plt.figure(figsize=(20, 20))
mpl.rcParams['axes.titlesize'] = 10
mpl.rcParams['axes.titlepad'] = 5

for i in range(16):
    ax = plt.subplot(4, 4, i+1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    true_label = class_names[label_batch[i]]
    predicted_label = class_names[predictions[i]]
    title_text = f'actual={true_label},  predicted={predicted_label}'
    plt.title(title_text, wrap=True, backgroundcolor='white', fontsize=18)
    plt.axis('off')

plt.tight_layout(pad=1.0)
plt.show()
```



0.14 3. 2. Tensorboard

[46]:
```
%load_ext tensorboard
%tensorboard --logdir {logdir}
```

```
<IPython.core.display.HTML object>
```

[ ]:
```

```