[1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, f1_score
import tensorflow as tf
from tensorflow.keras.layers import (
    Conv2D,
    MaxPooling2D,
    Dropout,
    Flatten,
    Dense,
    AveragePooling2D,
    LeakyReLU,
    BatchNormalization,
)
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
import time
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import random
```

[2]:

```
# Set all random seeds
np.random.seed(42)
tf.random.set_seed(42)
random.seed(42)
os.environ["PYTHONHASHSEED"] = "42"

# Your existing constant
NUM_CLASSES = 5
```

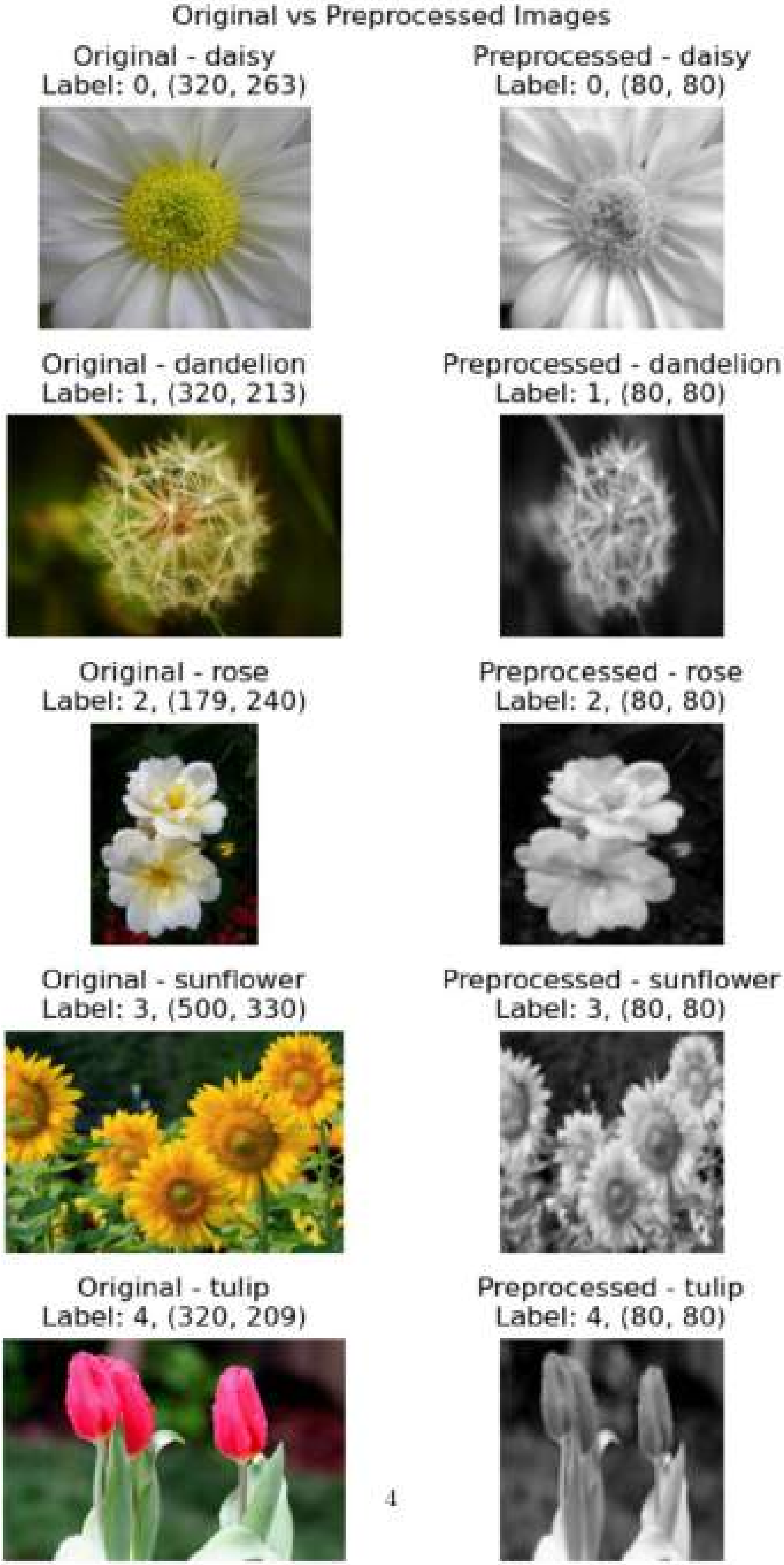
1 Resize all images to 80x80 and ensure grayscale

```
[3]: def load_and_preprocess_data(data_dir="flowers", target_size=(80, 80)):
         """Load and preprocess flower images"""
         images = []
         labels = []
         class_names = os.listdir(data_dir)

         # Set up plot for visualization
         num_classes = len(class_names)
         fig, axes = plt.subplots(num_classes, 2, figsize=(6, 2 * num_classes))
         plt.suptitle("Original vs Preprocessed Images")

         for idx, class_name in tqdm(enumerate(class_names), total=len(class_names)):
             class_dir = os.path.join(data_dir, class_name)
             first_image = True

             for img_name in os.listdir(class_dir):
                 img_path = os.path.join(class_dir, img_name)
                 try:
                     # Load and process image
                     img = Image.open(img_path)

                     # Plot first image of each class
                     if first_image:
                         # Original image
                         axes[idx, 0].imshow(img)
                         axes[idx, 0].axis("off")
                         axes[idx, 0].set_title(
                             f"Original - {class_name}\nLabel: {idx}, {img.size}"
                         )

                         # Process image
                         img_resized = img.resize(target_size)
                         img_gray = img_resized.convert("L")

                         # Preprocessed image
                         axes[idx, 1].imshow(img_gray, cmap="gray")
                         axes[idx, 1].axis("off")
                         axes[idx, 1].set_title(
                             f"Preprocessed - {class_name}\nLabel: {idx}, {img_gray.
   ↪size}"
                         )
                         first_image = False

                     # Process image for model
                     img = img.resize(target_size)
```

```
                img = img.convert("L")
                img_array = np.array(img) / 255.0
                img_array = img_array.reshape(target_size + (1,))
                images.append(img_array)
                labels.append(idx)

            except Exception as e:
                print(f"Error processing {img_path}: {str(e)}")

    plt.tight_layout()
    plt.show()

    assert len(images) == len(labels), "Number of images and labels must match"
    assert target_size and all(
        img.shape == target_size + (1,) for img in images
    ), "Each image must be of target size and be of grayscale"
    return np.array(images), np.array(labels), class_names
```

[4]:
```
X, y, class_names = load_and_preprocess_data()
```

```
 0%|          | 0/5 [00:00<?, ?it/s]
```


Original vs Preprocessed Images
Original - daisy
Label: 0, (320, 263)
Preprocessed - daisy
Label: 0, (80, 80)
Original - dandelion
Label: 1, (320, 213)
Preprocessed - dandelion
Label: 1, (80, 80)
Original - rose
Label: 2, (179, 240)
Preprocessed - rose
Label: 2, (80, 80)
Original - sunflower
Label: 3, (500, 330)
Preprocessed - sunflower
Label: 3, (80, 80)
Original - tulip
Label: 4, (320, 209)
Preprocessed - tulip
Label: 4, (80, 80)

2 Split the dataset

[5]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=42
)
```

3 Training and Plotting Functions

[6]:

```
def train_and_evaluate(
    model, X_train, y_train, X_test, y_test, epochs=30, batch_size=32
):
    """Train and evaluate the model"""
    # Compile model
    model.compile(
        optimizer="adam", loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
    )

    # Train model
    start_time = time.time()
    history = model.fit(
        X_train,
        y_train,
        validation_data=(X_test, y_test),
        epochs=epochs,
        batch_size=batch_size,
    )
    training_time = time.time() - start_time

    return history, training_time
```

[7]:

```
def plot_training_history(history, title):
    """Plot training history"""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # Plot accuracy
    ax1.plot(history.history["accuracy"], label="Training")
    ax1.plot(history.history["val_accuracy"], label="Validation")
    ax1.set_title(f"Model Accuracy - {title}")
    ax1.set_xlabel("Epoch")
    ax1.set_ylabel("Accuracy")
    ax1.legend()
```

```
    # Plot loss
    ax2.plot(history.history["loss"], label="Training")
    ax2.plot(history.history["val_loss"], label="Validation")
    ax2.set_title(f"Model Loss - {title}")
    ax2.set_xlabel("Epoch")
    ax2.set_ylabel("Loss")
    ax2.legend()

    plt.tight_layout()
    plt.show()
```

[8]:
```
def plot_confusion_matrix(y_true, y_pred, class_names, title):
    """Plot confusion matrix"""
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=class_names,
        yticklabels=class_names,
    )
    plt.title(f"Confusion Matrix - {title}")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()
```

[9]:
```
def plot_prediction_samples(
    X_test: np.ndarray,
    y_test: np.ndarray,
    y_pred: np.ndarray,
    class_names: list,
    n_samples: int = 4,
) -> None:
    """
    Plot samples of correct and incorrect predictions from a model.

    Args:
        X_test: Test image data
        y_test: True test labels
        y_pred: Predicted test labels
        class_names: List of class names
        n_samples: Number of samples to show for each case (default=4)
    """
    correct_indices = np.where(y_pred == y_test)[0]
```

```
    incorrect_indices = np.where(y_pred != y_test)[0]

    # Select random samples from both
    correct_samples = np.random.choice(correct_indices, n_samples,␣
↪replace=False)
    incorrect_samples = np.random.choice(incorrect_indices, n_samples,␣
↪replace=False)

    # Create a figure to display the results
    plt.figure(figsize=(8, 5))

    # Plot correct predictions
    for i, idx in enumerate(correct_samples):
        plt.subplot(2, n_samples, i + 1)
        plt.imshow(X_test[idx], cmap="gray")
        plt.axis("off")
        plt.title(
            f"True: {class_names[y_test[idx]]}\nPred:␣
↪{class_names[y_pred[idx]]}",
            color="green",
            fontsize=8,
        )

    # Plot incorrect predictions
    for i, idx in enumerate(incorrect_samples):
        plt.subplot(2, n_samples, n_samples + i + 1)
        plt.imshow(X_test[idx], cmap="gray")
        plt.axis("off")
        plt.title(
            f"True: {class_names[y_test[idx]]}\nPred:␣
↪{class_names[y_pred[idx]]}",
            color="red",
            fontsize=8,
        )

    plt.suptitle("Correct (top) vs Incorrect (bottom) Predictions", fontsize=10)
    plt.tight_layout()  # Add vertical and horizontal spacing between subplots
    plt.show()
```

```
[10]: results = []
```

4 Model 1

```
[11]: model1 = Sequential(name="Model1")

      model1.add(
          Conv2D(
              filters=16,
              kernel_size=(3, 3),
              padding="same",
              activation="relu",
              input_shape=(80, 80, 1),
          )
      )
      model1.add(MaxPooling2D(pool_size=(2, 2)))
      model1.add(Dropout(0.1))

      model1.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same",␣
       ↪activation="relu"))
      model1.add(MaxPooling2D(pool_size=(2, 2)))
      model1.add(Dropout(0.1))

      model1.add(Conv2D(filters=64, kernel_size=(3, 3), padding="same",␣
       ↪activation="relu"))
      model1.add(MaxPooling2D(pool_size=(2, 2)))
      model1.add(Dropout(0.1))

      model1.add(Flatten())
      model1.add(Dense(NUM_CLASSES, activation="softmax"))

      model1.compile(
          optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
       ↪metrics=["accuracy"]
      )
      model1.summary()
```

```
Model: "Model1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 80, 80, 16)        160

 max_pooling2d (MaxPooling2D  (None, 40, 40, 16)       0
 )

 dropout (Dropout)           (None, 40, 40, 16)        0

 conv2d_1 (Conv2D)           (None, 40, 40, 32)        4640
```

```
 max_pooling2d_1 (MaxPooling  (None, 20, 20, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 20, 20, 32)        0

 conv2d_2 (Conv2D)           (None, 20, 20, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 10, 10, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 10, 10, 64)        0

 flatten (Flatten)           (None, 6400)              0

 dense (Dense)               (None, 5)                 32005

=================================================================
Total params: 55,301
Trainable params: 55,301
Non-trainable params: 0
_________________________________________________________________
```

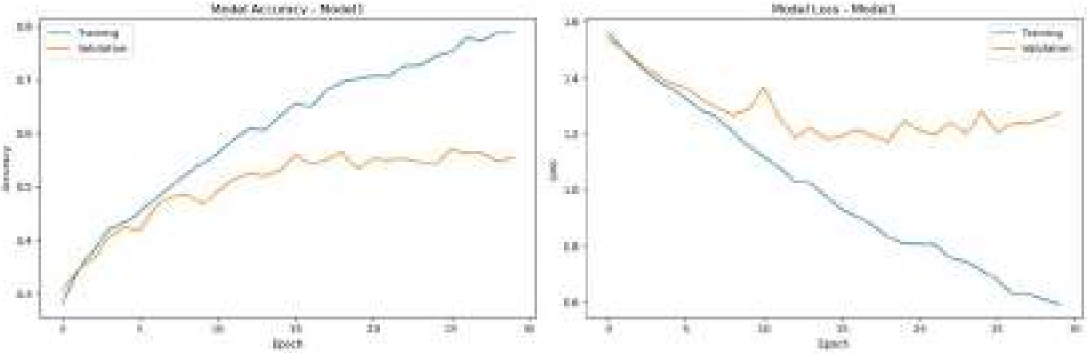
[12]:

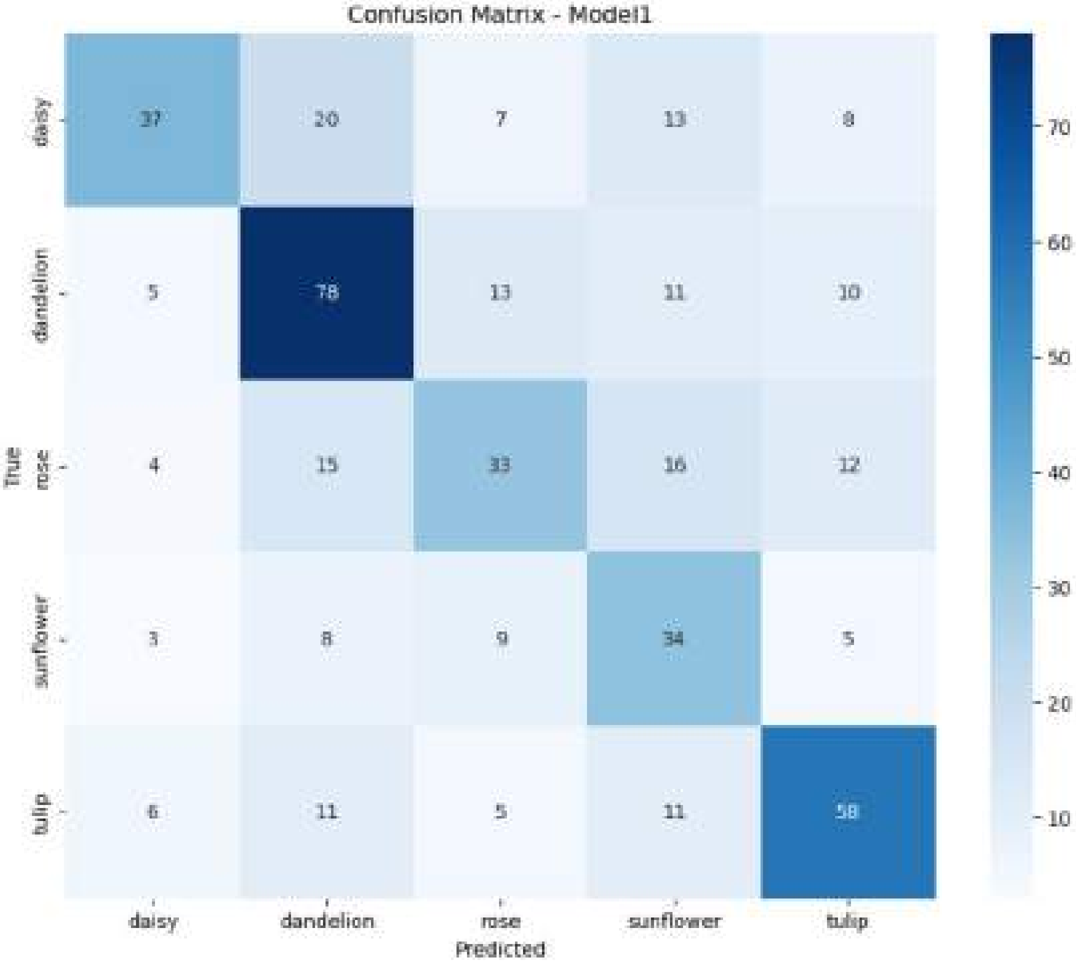
```
history, training_time = train_and_evaluate(
    model1, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model1.evaluate(X_test, y_test)
y_pred = np.argmax(model1.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model1",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model1.count_params(),
    }
)

plot_training_history(history, f"Model1")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model1")
print("\nResults for Model1:")
print(f"Configuration: {results[0]['configuration']}")
print(f"Test Accuracy: {results[0]['test_accuracy']:.4f}")
print(f"F1 Score: {results[0]['f1_score']:.4f}")
print(f"Training Time: {results[0]['training_time']:.2f} seconds")
```

```
print(f"Number of Parameters: {results[0]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 4s 35ms/step - loss: 1.5604 - accuracy:
0.2808 - val_loss: 1.5358 - val_accuracy: 0.3056
Epoch 2/30
31/31 [==============================] - 1s 19ms/step - loss: 1.4963 - accuracy:
0.3436 - val_loss: 1.5009 - val_accuracy: 0.3449
Epoch 3/30
31/31 [==============================] - 1s 19ms/step - loss: 1.4436 - accuracy:
0.3810 - val_loss: 1.4512 - val_accuracy: 0.3657
Epoch 4/30
31/31 [==============================] - 1s 19ms/step - loss: 1.3952 - accuracy:
0.4214 - val_loss: 1.4136 - val_accuracy: 0.4074
Epoch 5/30
31/31 [==============================] - 1s 19ms/step - loss: 1.3627 - accuracy:
0.4324 - val_loss: 1.3811 - val_accuracy: 0.4236
Epoch 6/30
31/31 [==============================] - 1s 19ms/step - loss: 1.3259 - accuracy:
0.4535 - val_loss: 1.3641 - val_accuracy: 0.4190
Epoch 7/30
31/31 [==============================] - 1s 19ms/step - loss: 1.2834 - accuracy:
0.4777 - val_loss: 1.3232 - val_accuracy: 0.4630
Epoch 8/30
31/31 [==============================] - 1s 21ms/step - loss: 1.2588 - accuracy:
0.5004 - val_loss: 1.2924 - val_accuracy: 0.4815
Epoch 9/30
31/31 [==============================] - 1s 20ms/step - loss: 1.2060 - accuracy:
0.5241 - val_loss: 1.2668 - val_accuracy: 0.4861
Epoch 10/30
31/31 [==============================] - 1s 20ms/step - loss: 1.1565 - accuracy:
0.5429 - val_loss: 1.2856 - val_accuracy: 0.4676
Epoch 11/30
31/31 [==============================] - 1s 19ms/step - loss: 1.1193 - accuracy:
0.5622 - val_loss: 1.3649 - val_accuracy: 0.4907
Epoch 12/30
31/31 [==============================] - 1s 19ms/step - loss: 1.0785 - accuracy:
0.5876 - val_loss: 1.2584 - val_accuracy: 0.5139
Epoch 13/30
31/31 [==============================] - 1s 20ms/step - loss: 1.0308 - accuracy:
0.6080 - val_loss: 1.1867 - val_accuracy: 0.5255
Epoch 14/30
31/31 [==============================] - 1s 19ms/step - loss: 1.0233 - accuracy:
0.6054 - val_loss: 1.2214 - val_accuracy: 0.5208
Epoch 15/30
31/31 [==============================] - 1s 19ms/step - loss: 0.9776 - accuracy:
0.6327 - val_loss: 1.1812 - val_accuracy: 0.5301
```

```
Epoch 16/30
31/31 [==============================] - 1s 19ms/step - loss: 0.9302 - accuracy:
0.6551 - val_loss: 1.1930 - val_accuracy: 0.5602
Epoch 17/30
31/31 [==============================] - 1s 19ms/step - loss: 0.9068 - accuracy:
0.6502 - val_loss: 1.2139 - val_accuracy: 0.5417
Epoch 18/30
31/31 [==============================] - 1s 19ms/step - loss: 0.8736 - accuracy:
0.6808 - val_loss: 1.1920 - val_accuracy: 0.5509
Epoch 19/30
31/31 [==============================] - 1s 19ms/step - loss: 0.8324 - accuracy:
0.6958 - val_loss: 1.1730 - val_accuracy: 0.5648
Epoch 20/30
31/31 [==============================] - 1s 21ms/step - loss: 0.8080 - accuracy:
0.7019 - val_loss: 1.2490 - val_accuracy: 0.5324
Epoch 21/30
31/31 [==============================] - 1s 20ms/step - loss: 0.8065 - accuracy:
0.7066 - val_loss: 1.2111 - val_accuracy: 0.5532
Epoch 22/30
31/31 [==============================] - 1s 20ms/step - loss: 0.8051 - accuracy:
0.7060 - val_loss: 1.1974 - val_accuracy: 0.5509
Epoch 23/30
31/31 [==============================] - 1s 19ms/step - loss: 0.7571 - accuracy:
0.7264 - val_loss: 1.2402 - val_accuracy: 0.5532
Epoch 24/30
31/31 [==============================] - 1s 22ms/step - loss: 0.7434 - accuracy:
0.7272 - val_loss: 1.2004 - val_accuracy: 0.5463
Epoch 25/30
31/31 [==============================] - 1s 19ms/step - loss: 0.7147 - accuracy:
0.7436 - val_loss: 1.2795 - val_accuracy: 0.5417
Epoch 26/30
31/31 [==============================] - 1s 19ms/step - loss: 0.6794 - accuracy:
0.7514 - val_loss: 1.2020 - val_accuracy: 0.5694
Epoch 27/30
31/31 [==============================] - 1s 20ms/step - loss: 0.6290 - accuracy:
0.7776 - val_loss: 1.2380 - val_accuracy: 0.5625
Epoch 28/30
31/31 [==============================] - 1s 19ms/step - loss: 0.6271 - accuracy:
0.7727 - val_loss: 1.2389 - val_accuracy: 0.5625
Epoch 29/30
31/31 [==============================] - 1s 19ms/step - loss: 0.6095 - accuracy:
0.7876 - val_loss: 1.2523 - val_accuracy: 0.5486
Epoch 30/30
31/31 [==============================] - 1s 19ms/step - loss: 0.5929 - accuracy:
0.7869 - val_loss: 1.2725 - val_accuracy: 0.5556
14/14 [==============================] - 0s 14ms/step - loss: 1.2725 - accuracy:
0.5556
14/14 [==============================] - 0s 4ms/step
```





	daisy	dandelion	rose	sunflower	tulip
daisy	37	20	7	13	8
dandelion	5	78	13	11	10
rose	4	15	33	16	12
sunflower	3	8	9	34	5
tulip	6	11	5	11	58

```
Results for Model1:
Configuration: Model1
```

```
Test Accuracy: 0.5556
F1 Score: 0.5541
Training Time: 22.06 seconds
Number of Parameters: 55,301
```

[13]:
```
plot_prediction_samples(X_test, y_test, y_pred, class_names)
```

5 Model 2

[14]:
```
model2 = Sequential(name="Model2")

model2.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.1))

model2.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same",␣
 ↪activation="relu"))
```

```
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.1))

model2.add(Conv2D(filters=64, kernel_size=(5, 5), padding="same",␣
 ↪activation="relu"))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.1))

model2.add(Flatten())
model2.add(Dense(NUM_CLASSES, activation="softmax"))

model2.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model2.summary()
```

```
Model: "Model2"

-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 80, 80, 16)        160

 max_pooling2d_3 (MaxPooling  (None, 40, 40, 16)       0
 2D)

 dropout_3 (Dropout)         (None, 40, 40, 16)        0

 conv2d_4 (Conv2D)           (None, 40, 40, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 20, 20, 32)       0
 2D)

 dropout_4 (Dropout)         (None, 20, 20, 32)        0

 conv2d_5 (Conv2D)           (None, 20, 20, 64)        51264

 max_pooling2d_5 (MaxPooling  (None, 10, 10, 64)       0
 2D)

 dropout_5 (Dropout)         (None, 10, 10, 64)        0

 flatten_1 (Flatten)         (None, 6400)              0

 dense_1 (Dense)             (None, 5)                 32005

=================================================================
```

```
Total params: 88,069
Trainable params: 88,069
Non-trainable params: 0
-----------------------------------------------------------------
```

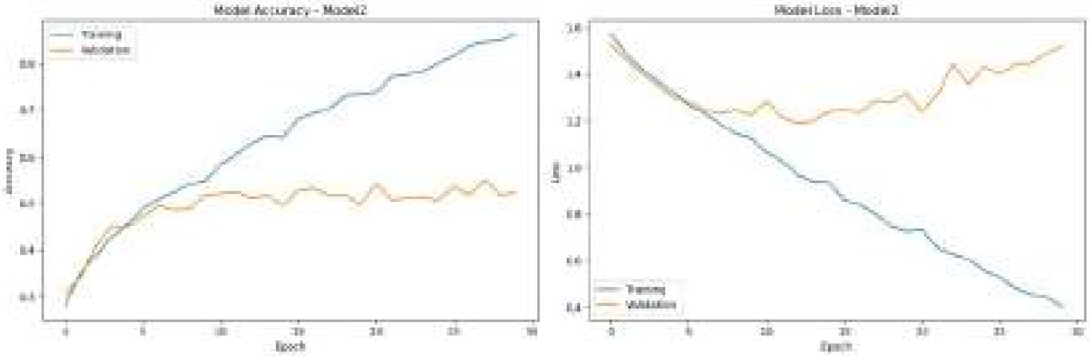
[15]:

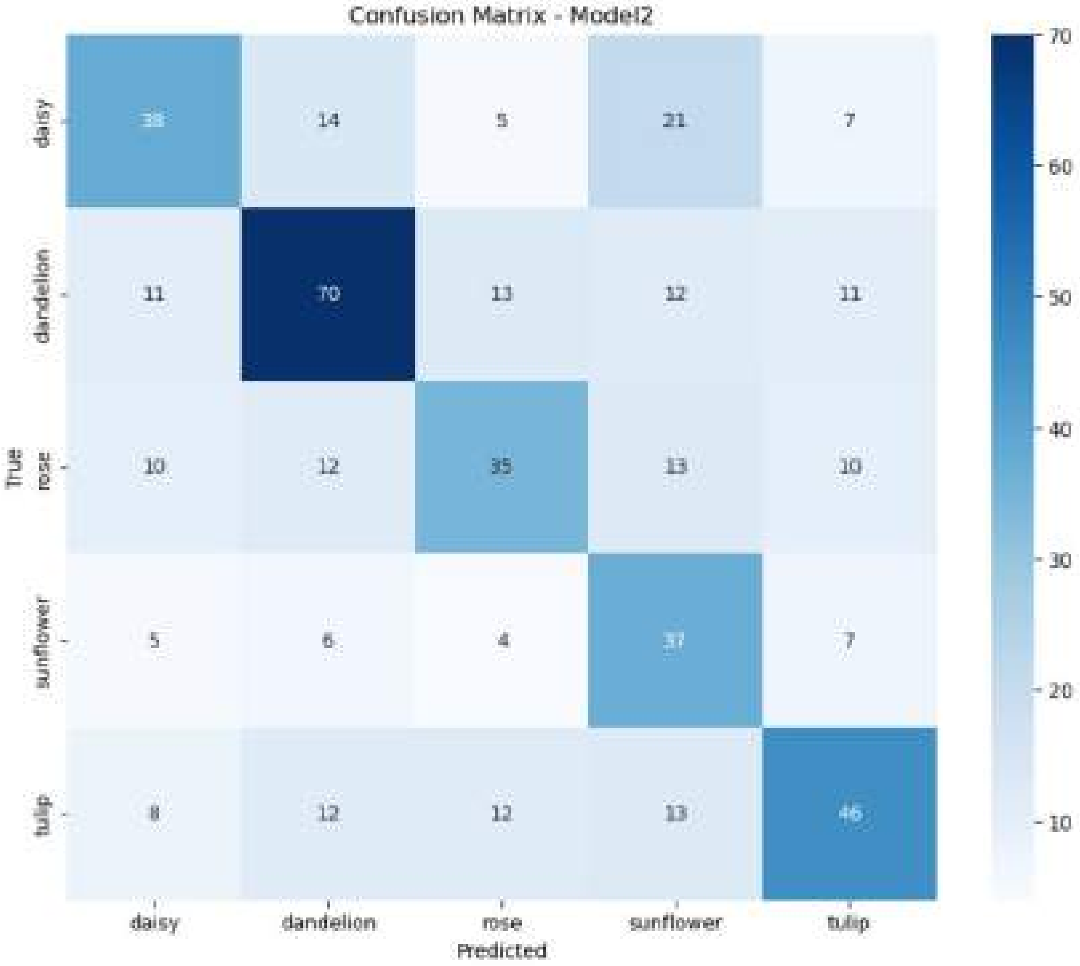
```
history, training_time = train_and_evaluate(
    model2, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model2.evaluate(X_test, y_test)
y_pred = np.argmax(model2.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model2",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model2.count_params(),
    }
)

plot_training_history(history, f"Model2")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model2")
print("\nResults for Model2:")
print(f"Configuration: {results[1]['configuration']}")
print(f"Test Accuracy: {results[1]['test_accuracy']:.4f}")
print(f"F1 Score: {results[1]['f1_score']:.4f}")
print(f"Training Time: {results[1]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[1]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 30ms/step - loss: 1.5707 - accuracy:
0.2795 - val_loss: 1.5276 - val_accuracy: 0.3079
Epoch 2/30
31/31 [==============================] - 1s 20ms/step - loss: 1.4834 - accuracy:
0.3532 - val_loss: 1.4602 - val_accuracy: 0.3426
Epoch 3/30
31/31 [==============================] - 1s 21ms/step - loss: 1.4189 - accuracy:
0.3897 - val_loss: 1.4007 - val_accuracy: 0.4097
Epoch 4/30
31/31 [==============================] - 1s 21ms/step - loss: 1.3699 - accuracy:
0.4291 - val_loss: 1.3518 - val_accuracy: 0.4514
Epoch 5/30
31/31 [==============================] - 1s 21ms/step - loss: 1.3212 - accuracy:
0.4548 - val_loss: 1.2976 - val_accuracy: 0.4491
Epoch 6/30
31/31 [==============================] - 1s 21ms/step - loss: 1.2666 - accuracy:
```

```
0.4891 - val_loss: 1.2826 - val_accuracy: 0.4722
Epoch 7/30
31/31 [==============================] - 1s 22ms/step - loss: 1.2316 - accuracy:
0.5102 - val_loss: 1.2453 - val_accuracy: 0.4954
Epoch 8/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1871 - accuracy:
0.5243 - val_loss: 1.2291 - val_accuracy: 0.4861
Epoch 9/30
31/31 [==============================] - 1s 20ms/step - loss: 1.1433 - accuracy:
0.5416 - val_loss: 1.2463 - val_accuracy: 0.4884
Epoch 10/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1261 - accuracy:
0.5480 - val_loss: 1.2244 - val_accuracy: 0.5185
Epoch 11/30
31/31 [==============================] - 1s 21ms/step - loss: 1.0644 - accuracy:
0.5840 - val_loss: 1.2841 - val_accuracy: 0.5208
Epoch 12/30
31/31 [==============================] - 1s 21ms/step - loss: 1.0291 - accuracy:
0.6049 - val_loss: 1.2139 - val_accuracy: 0.5255
Epoch 13/30
31/31 [==============================] - 1s 22ms/step - loss: 0.9679 - accuracy:
0.6299 - val_loss: 1.1900 - val_accuracy: 0.5116
Epoch 14/30
31/31 [==============================] - 1s 20ms/step - loss: 0.9392 - accuracy:
0.6450 - val_loss: 1.1977 - val_accuracy: 0.5185
Epoch 15/30
31/31 [==============================] - 1s 21ms/step - loss: 0.9363 - accuracy:
0.6404 - val_loss: 1.2387 - val_accuracy: 0.4954
Epoch 16/30
31/31 [==============================] - 1s 23ms/step - loss: 0.8617 - accuracy:
0.6824 - val_loss: 1.2435 - val_accuracy: 0.5278
Epoch 17/30
31/31 [==============================] - 1s 21ms/step - loss: 0.8381 - accuracy:
0.6927 - val_loss: 1.2369 - val_accuracy: 0.5324
Epoch 18/30
31/31 [==============================] - 1s 21ms/step - loss: 0.8016 - accuracy:
0.7024 - val_loss: 1.2864 - val_accuracy: 0.5162
Epoch 19/30
31/31 [==============================] - 1s 21ms/step - loss: 0.7466 - accuracy:
0.7308 - val_loss: 1.2776 - val_accuracy: 0.5185
Epoch 20/30
31/31 [==============================] - 1s 21ms/step - loss: 0.7249 - accuracy:
0.7331 - val_loss: 1.3179 - val_accuracy: 0.4977
Epoch 21/30
31/31 [==============================] - 1s 22ms/step - loss: 0.7328 - accuracy:
0.7375 - val_loss: 1.2383 - val_accuracy: 0.5417
Epoch 22/30
31/31 [==============================] - 1s 21ms/step - loss: 0.6526 - accuracy:
```

```
0.7714 - val_loss: 1.3144 - val_accuracy: 0.5093
Epoch 23/30
31/31 [==============================] - 1s 20ms/step - loss: 0.6269 - accuracy:
0.7755 - val_loss: 1.4440 - val_accuracy: 0.5116
Epoch 24/30
31/31 [==============================] - 1s 21ms/step - loss: 0.6049 - accuracy:
0.7822 - val_loss: 1.3550 - val_accuracy: 0.5139
Epoch 25/30
31/31 [==============================] - 1s 21ms/step - loss: 0.5581 - accuracy:
0.8005 - val_loss: 1.4335 - val_accuracy: 0.5069
Epoch 26/30
31/31 [==============================] - 1s 21ms/step - loss: 0.5290 - accuracy:
0.8157 - val_loss: 1.3976 - val_accuracy: 0.5370
Epoch 27/30
31/31 [==============================] - 1s 22ms/step - loss: 0.4810 - accuracy:
0.8386 - val_loss: 1.4451 - val_accuracy: 0.5185
Epoch 28/30
31/31 [==============================] - 1s 20ms/step - loss: 0.4547 - accuracy:
0.8458 - val_loss: 1.4457 - val_accuracy: 0.5509
Epoch 29/30
31/31 [==============================] - 1s 20ms/step - loss: 0.4432 - accuracy:
0.8492 - val_loss: 1.4893 - val_accuracy: 0.5185
Epoch 30/30
31/31 [==============================] - 1s 21ms/step - loss: 0.4031 - accuracy:
0.8625 - val_loss: 1.5188 - val_accuracy: 0.5231
14/14 [==============================] - 0s 6ms/step - loss: 1.5187 - accuracy:
0.5231
14/14 [==============================] - 0s 4ms/step
```





```
Results for Model2:
Configuration: Model2
Test Accuracy: 0.5231
F1 Score: 0.5243
Training Time: 20.96 seconds
Number of Parameters: 88,069
```

[16]:

```
plot_prediction_samples(X_test, y_test, y_pred, class_names)
```

6 Model 3

[17]:

```
model3 = Sequential(name="Model3")

model3.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.1))

model3.add(Conv2D(filters=32, kernel_size=(5, 5), padding="same",␣
 ↳activation="relu"))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.1))

model3.add(Conv2D(filters=64, kernel_size=(5, 5), padding="same",␣
 ↳activation="relu"))
model3.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model3.add(Dropout(0.1))

model3.add(Flatten())
model3.add(Dense(NUM_CLASSES, activation="softmax"))

model3.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model3.summary()
```

```
Model: "Model3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 80, 80, 16)        160

 max_pooling2d_6 (MaxPooling  (None, 40, 40, 16)       0
 2D)

 dropout_6 (Dropout)         (None, 40, 40, 16)        0

 conv2d_7 (Conv2D)           (None, 40, 40, 32)        12832

 max_pooling2d_7 (MaxPooling  (None, 20, 20, 32)       0
 2D)

 dropout_7 (Dropout)         (None, 20, 20, 32)        0

 conv2d_8 (Conv2D)           (None, 20, 20, 64)        51264

 max_pooling2d_8 (MaxPooling  (None, 10, 10, 64)       0
 2D)

 dropout_8 (Dropout)         (None, 10, 10, 64)        0

 flatten_2 (Flatten)         (None, 6400)              0

 dense_2 (Dense)             (None, 5)                 32005

=================================================================
Total params: 96,261
Trainable params: 96,261
Non-trainable params: 0
_________________________________________________________________
```

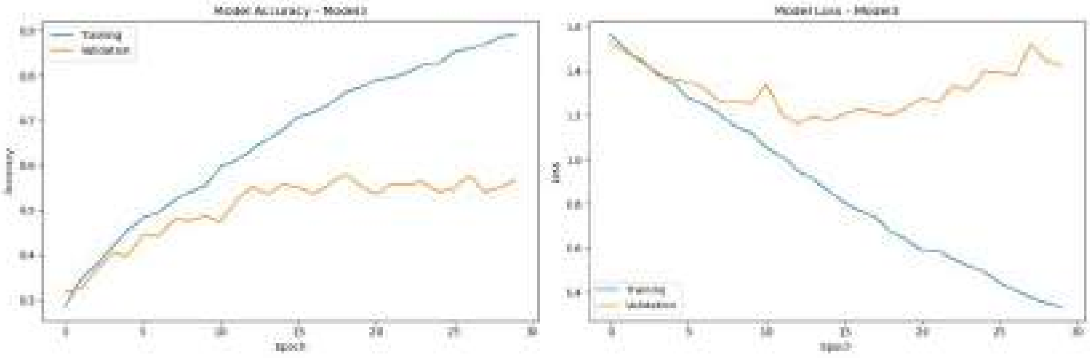
[18]:

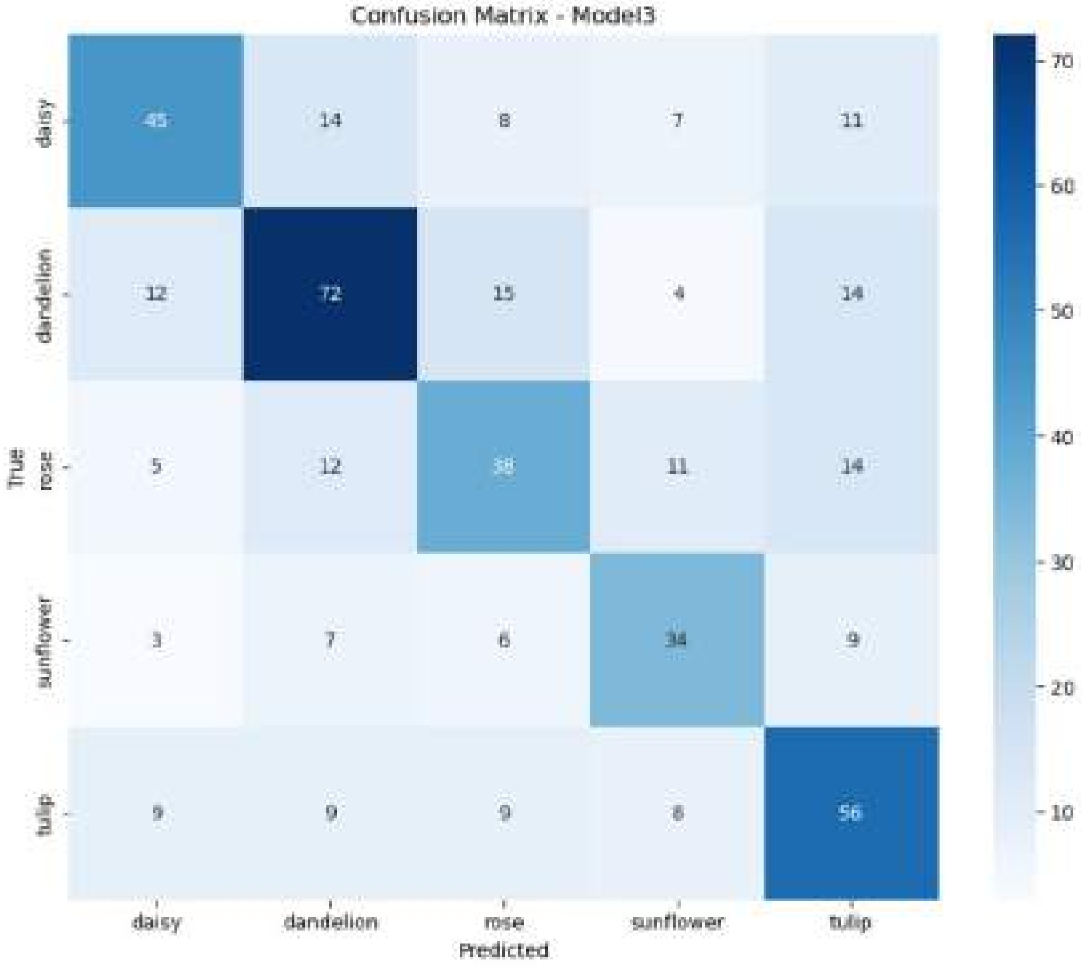
```
history, training_time = train_and_evaluate(
    model3, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model3.evaluate(X_test, y_test)
y_pred = np.argmax(model3.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3.count_params(),
    }
)

plot_training_history(history, f"Model3")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3")
print("\nResults for Model3:")
print(f"Configuration: {results[2]['configuration']}")
print(f"Test Accuracy: {results[2]['test_accuracy']:.4f}")
print(f"F1 Score: {results[2]['f1_score']:.4f}")
print(f"Training Time: {results[2]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[2]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 34ms/step - loss: 1.5638 - accuracy:
0.2847 - val_loss: 1.5247 - val_accuracy: 0.3171
Epoch 2/30
31/31 [==============================] - 1s 23ms/step - loss: 1.4922 - accuracy:
0.3457 - val_loss: 1.4837 - val_accuracy: 0.3264
Epoch 3/30
31/31 [==============================] - 1s 22ms/step - loss: 1.4400 - accuracy:
0.3781 - val_loss: 1.4513 - val_accuracy: 0.3657
Epoch 4/30
31/31 [==============================] - 1s 23ms/step - loss: 1.3898 - accuracy:
0.4172 - val_loss: 1.3803 - val_accuracy: 0.4028
Epoch 5/30
31/31 [==============================] - 1s 23ms/step - loss: 1.3444 - accuracy:
0.4551 - val_loss: 1.3659 - val_accuracy: 0.3981
Epoch 6/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2778 - accuracy:
0.4821 - val_loss: 1.3484 - val_accuracy: 0.4444
Epoch 7/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2489 - accuracy:
0.4952 - val_loss: 1.3231 - val_accuracy: 0.4421
```

```
Epoch 8/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2049 - accuracy:
0.5210 - val_loss: 1.2558 - val_accuracy: 0.4792
Epoch 9/30
31/31 [==============================] - 1s 23ms/step - loss: 1.1483 - accuracy:
0.5416 - val_loss: 1.2646 - val_accuracy: 0.4769
Epoch 10/30
31/31 [==============================] - 1s 22ms/step - loss: 1.1218 - accuracy:
0.5539 - val_loss: 1.2508 - val_accuracy: 0.4861
Epoch 11/30
31/31 [==============================] - 1s 23ms/step - loss: 1.0528 - accuracy:
0.5969 - val_loss: 1.3389 - val_accuracy: 0.4745
Epoch 12/30
31/31 [==============================] - 1s 23ms/step - loss: 1.0161 - accuracy:
0.6108 - val_loss: 1.2077 - val_accuracy: 0.5208
Epoch 13/30
31/31 [==============================] - 1s 24ms/step - loss: 0.9528 - accuracy:
0.6358 - val_loss: 1.1602 - val_accuracy: 0.5532
Epoch 14/30
31/31 [==============================] - 1s 23ms/step - loss: 0.9125 - accuracy:
0.6579 - val_loss: 1.1980 - val_accuracy: 0.5370
Epoch 15/30
31/31 [==============================] - 1s 22ms/step - loss: 0.8591 - accuracy:
0.6788 - val_loss: 1.1727 - val_accuracy: 0.5579
Epoch 16/30
31/31 [==============================] - 1s 23ms/step - loss: 0.8054 - accuracy:
0.7086 - val_loss: 1.2056 - val_accuracy: 0.5509
Epoch 17/30
31/31 [==============================] - 1s 24ms/step - loss: 0.7706 - accuracy:
0.7181 - val_loss: 1.2270 - val_accuracy: 0.5370
Epoch 18/30
31/31 [==============================] - 1s 23ms/step - loss: 0.7382 - accuracy:
0.7354 - val_loss: 1.2124 - val_accuracy: 0.5579
Epoch 19/30
31/31 [==============================] - 1s 24ms/step - loss: 0.6728 - accuracy:
0.7627 - val_loss: 1.2014 - val_accuracy: 0.5810
Epoch 20/30
31/31 [==============================] - 1s 24ms/step - loss: 0.6360 - accuracy:
0.7732 - val_loss: 1.2333 - val_accuracy: 0.5532
Epoch 21/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5905 - accuracy:
0.7907 - val_loss: 1.2787 - val_accuracy: 0.5370
Epoch 22/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5898 - accuracy:
0.7954 - val_loss: 1.2576 - val_accuracy: 0.5579
Epoch 23/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5492 - accuracy:
0.8062 - val_loss: 1.3304 - val_accuracy: 0.5579
```

```
Epoch 24/30
31/31 [==============================] - 1s 22ms/step - loss: 0.5204 - accuracy:
0.8232 - val_loss: 1.3140 - val_accuracy: 0.5625
Epoch 25/30
31/31 [==============================] - 1s 23ms/step - loss: 0.4922 - accuracy:
0.8263 - val_loss: 1.3980 - val_accuracy: 0.5417
Epoch 26/30
31/31 [==============================] - 1s 24ms/step - loss: 0.4425 - accuracy:
0.8520 - val_loss: 1.3941 - val_accuracy: 0.5486
Epoch 27/30
31/31 [==============================] - 1s 23ms/step - loss: 0.4080 - accuracy:
0.8597 - val_loss: 1.3804 - val_accuracy: 0.5764
Epoch 28/30
31/31 [==============================] - 1s 23ms/step - loss: 0.3778 - accuracy:
0.8700 - val_loss: 1.5187 - val_accuracy: 0.5417
Epoch 29/30
31/31 [==============================] - 1s 23ms/step - loss: 0.3521 - accuracy:
0.8862 - val_loss: 1.4474 - val_accuracy: 0.5509
Epoch 30/30
31/31 [==============================] - 1s 24ms/step - loss: 0.3341 - accuracy:
0.8906 - val_loss: 1.4233 - val_accuracy: 0.5671
14/14 [==============================] - 0s 7ms/step - loss: 1.4233 - accuracy:
0.5671
14/14 [==============================] - 0s 3ms/step
```

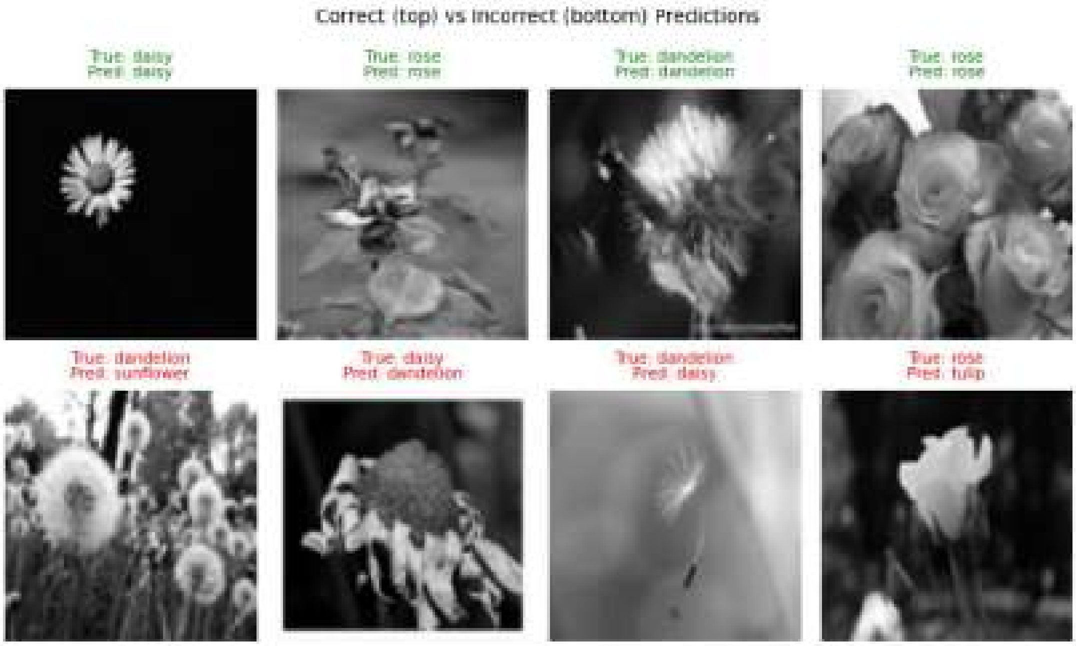




```
Results for Model3:
Configuration: Model3
Test Accuracy: 0.5671
F1 Score: 0.5669
Training Time: 23.07 seconds
Number of Parameters: 96,261
```

[19]:

```
plot_prediction_samples(X_test, y_test, y_pred, class_names)
```



7 Model 4

[20]:

```
model4 = Sequential(name="Model4")

model4.add(
    Conv2D(
        filters=16,
        kernel_size=(5, 5),
        padding="same",
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.1))

model4.add(Conv2D(filters=32, kernel_size=(5, 5), padding="same",␣
 ↪activation="relu"))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.1))

model4.add(Conv2D(filters=64, kernel_size=(5, 5), padding="same",␣
 ↪activation="relu"))
model4.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model4.add(Dropout(0.1))

model4.add(Flatten())
model4.add(Dense(NUM_CLASSES, activation="softmax"))

model4.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model4.summary()
```

```
Model: "Model4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 80, 80, 16)        416

 max_pooling2d_9 (MaxPooling  (None, 40, 40, 16)       0
 2D)

 dropout_9 (Dropout)         (None, 40, 40, 16)        0

 conv2d_10 (Conv2D)          (None, 40, 40, 32)        12832

 max_pooling2d_10 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_10 (Dropout)        (None, 20, 20, 32)        0

 conv2d_11 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_11 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_11 (Dropout)        (None, 10, 10, 64)        0

 flatten_3 (Flatten)         (None, 6400)              0

 dense_3 (Dense)             (None, 5)                 32005

=================================================================
Total params: 96,517
Trainable params: 96,517
Non-trainable params: 0
_________________________________________________________________
```

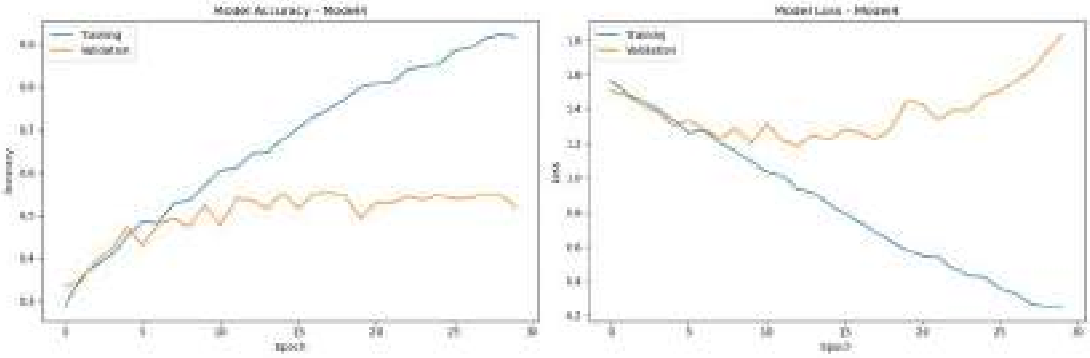
[21]:

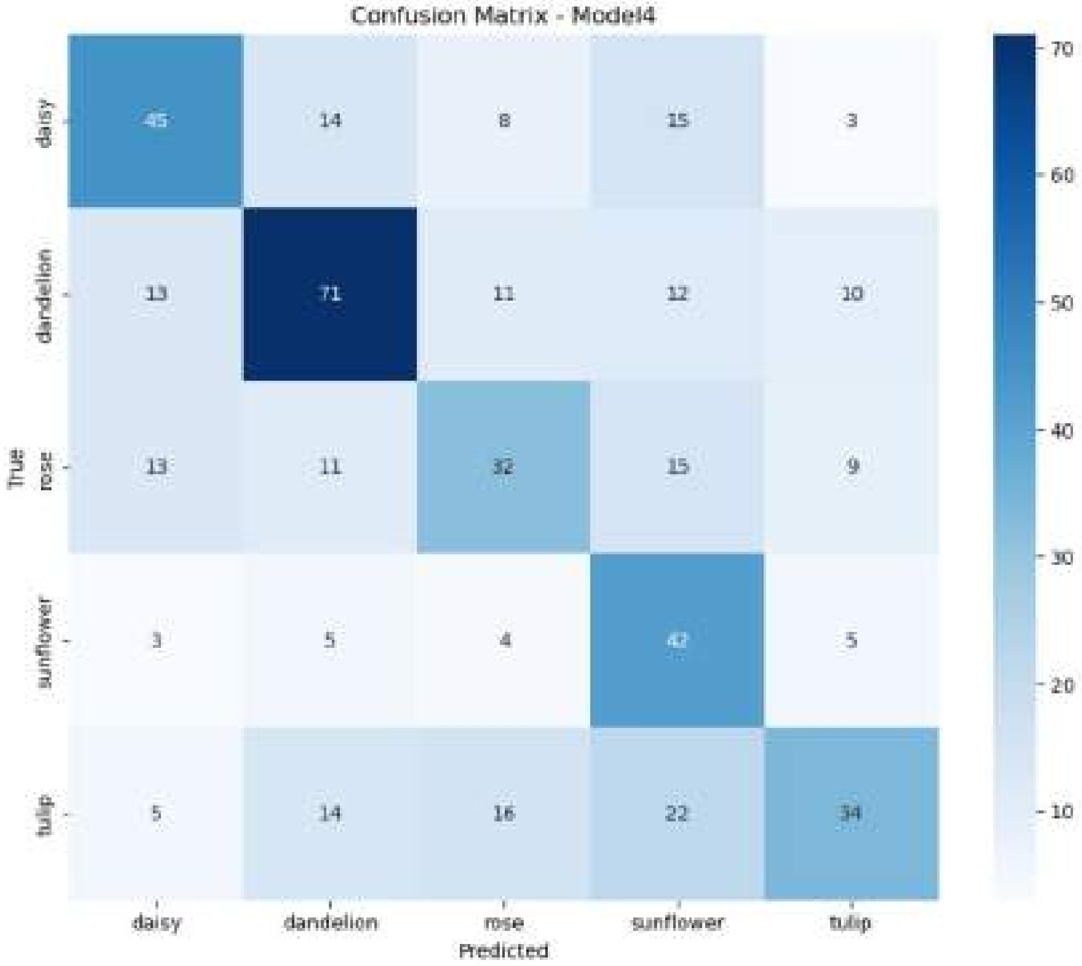
```
history, training_time = train_and_evaluate(
    model4, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model4.evaluate(X_test, y_test)
y_pred = np.argmax(model4.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model4",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model4.count_params(),
    }
)

plot_training_history(history, f"Model4")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model4")
print("\nResults for Model4:")
print(f"Configuration: {results[3]['configuration']}")
print(f"Test Accuracy: {results[3]['test_accuracy']:.4f}")
print(f"F1 Score: {results[3]['f1_score']:.4f}")
print(f"Training Time: {results[3]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[3]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 33ms/step - loss: 1.5651 - accuracy:
0.2849 - val_loss: 1.5029 - val_accuracy: 0.3356
Epoch 2/30
31/31 [==============================] - 1s 23ms/step - loss: 1.4863 - accuracy:
0.3526 - val_loss: 1.4809 - val_accuracy: 0.3426
Epoch 3/30
31/31 [==============================] - 1s 23ms/step - loss: 1.4439 - accuracy:
0.3835 - val_loss: 1.4183 - val_accuracy: 0.3935
Epoch 4/30
31/31 [==============================] - 1s 24ms/step - loss: 1.3974 - accuracy:
0.4067 - val_loss: 1.3799 - val_accuracy: 0.4213
Epoch 5/30
31/31 [==============================] - 1s 23ms/step - loss: 1.3317 - accuracy:
0.4499 - val_loss: 1.2987 - val_accuracy: 0.4722
Epoch 6/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2618 - accuracy:
0.4855 - val_loss: 1.3316 - val_accuracy: 0.4282
Epoch 7/30
31/31 [==============================] - 1s 24ms/step - loss: 1.2766 - accuracy:
0.4808 - val_loss: 1.2791 - val_accuracy: 0.4815
```

```
Epoch 8/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2012 - accuracy:
0.5274 - val_loss: 1.2285 - val_accuracy: 0.4907
Epoch 9/30
31/31 [==============================] - 1s 24ms/step - loss: 1.1554 - accuracy:
0.5344 - val_loss: 1.2857 - val_accuracy: 0.4745
Epoch 10/30
31/31 [==============================] - 1s 23ms/step - loss: 1.0940 - accuracy:
0.5689 - val_loss: 1.2016 - val_accuracy: 0.5231
Epoch 11/30
31/31 [==============================] - 1s 23ms/step - loss: 1.0353 - accuracy:
0.6051 - val_loss: 1.3083 - val_accuracy: 0.4769
Epoch 12/30
31/31 [==============================] - 1s 23ms/step - loss: 1.0126 - accuracy:
0.6095 - val_loss: 1.2165 - val_accuracy: 0.5394
Epoch 13/30
31/31 [==============================] - 1s 23ms/step - loss: 0.9349 - accuracy:
0.6445 - val_loss: 1.1831 - val_accuracy: 0.5347
Epoch 14/30
31/31 [==============================] - 1s 23ms/step - loss: 0.9144 - accuracy:
0.6479 - val_loss: 1.2516 - val_accuracy: 0.5162
Epoch 15/30
31/31 [==============================] - 1s 25ms/step - loss: 0.8465 - accuracy:
0.6782 - val_loss: 1.2181 - val_accuracy: 0.5509
Epoch 16/30
31/31 [==============================] - 1s 23ms/step - loss: 0.7945 - accuracy:
0.7027 - val_loss: 1.2806 - val_accuracy: 0.5162
Epoch 17/30
31/31 [==============================] - 1s 23ms/step - loss: 0.7397 - accuracy:
0.7308 - val_loss: 1.2618 - val_accuracy: 0.5509
Epoch 18/30
31/31 [==============================] - 1s 23ms/step - loss: 0.6821 - accuracy:
0.7480 - val_loss: 1.2266 - val_accuracy: 0.5532
Epoch 19/30
31/31 [==============================] - 1s 23ms/step - loss: 0.6302 - accuracy:
0.7722 - val_loss: 1.2845 - val_accuracy: 0.5463
Epoch 20/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5772 - accuracy:
0.7990 - val_loss: 1.4460 - val_accuracy: 0.4907
Epoch 21/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5453 - accuracy:
0.8072 - val_loss: 1.4236 - val_accuracy: 0.5301
Epoch 22/30
31/31 [==============================] - 1s 23ms/step - loss: 0.5369 - accuracy:
0.8113 - val_loss: 1.3379 - val_accuracy: 0.5278
Epoch 23/30
31/31 [==============================] - 1s 23ms/step - loss: 0.4714 - accuracy:
0.8386 - val_loss: 1.3903 - val_accuracy: 0.5440
```

```
Epoch 24/30
31/31 [==============================] - 1s 23ms/step - loss: 0.4289 - accuracy:
0.8479 - val_loss: 1.3890 - val_accuracy: 0.5347
Epoch 25/30
31/31 [==============================] - 1s 24ms/step - loss: 0.4223 - accuracy:
0.8510 - val_loss: 1.4777 - val_accuracy: 0.5486
Epoch 26/30
31/31 [==============================] - 1s 23ms/step - loss: 0.3537 - accuracy:
0.8842 - val_loss: 1.5009 - val_accuracy: 0.5370
Epoch 27/30
31/31 [==============================] - 1s 23ms/step - loss: 0.3235 - accuracy:
0.8903 - val_loss: 1.5635 - val_accuracy: 0.5394
Epoch 28/30
31/31 [==============================] - 1s 23ms/step - loss: 0.2686 - accuracy:
0.9133 - val_loss: 1.6202 - val_accuracy: 0.5509
Epoch 29/30
31/31 [==============================] - 1s 23ms/step - loss: 0.2475 - accuracy:
0.9223 - val_loss: 1.7307 - val_accuracy: 0.5463
Epoch 30/30
31/31 [==============================] - 1s 23ms/step - loss: 0.2540 - accuracy:
0.9174 - val_loss: 1.8310 - val_accuracy: 0.5185
14/14 [==============================] - 0s 10ms/step - loss: 1.8310 - accuracy:
0.5185
14/14 [==============================] - 0s 3ms/step
```





	daisy	dandelion	rose	sunflower	tulip
daisy	45	14	8	15	3
dandelion	13	71	11	12	10
rose	13	11	32	15	9
sunflower	3	5	4	47	5
tulip	5	14	16	22	34

```
Results for Model4:
Configuration: Model4
Test Accuracy: 0.5185
F1 Score: 0.5160
Training Time: 23.01 seconds
Number of Parameters: 96,517
```

[22]:
```
plot_prediction_samples(X_test, y_test, y_pred, class_names)
```

[23]:

```python
from tabulate import tabulate

# Format the data for tabulate
headers = results[0].keys()
rows = [
    [
        f"{row['configuration']}",
        f"{row['test_accuracy']:.4f}",
        f"{row['f1_score']:.4f}",
        f"{row['training_time']:.4f}",
        row["parameters"],
    ]
    for row in results
]

# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt="pretty"))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x["test_accuracy"])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
```

```
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+---------------+---------------+----------+---------------+------------+
| configuration | test_accuracy | f1_score | training_time | parameters |
+---------------+---------------+----------+---------------+------------+
|    Model1     |    0.5556     |  0.5541  |    22.0594    |   55301    |
|    Model2     |    0.5231     |  0.5243  |    20.9627    |   88069    |
|    Model3     |    0.5671     |  0.5669  |    23.0679    |   96261    |
|    Model4     |    0.5185     |  0.5160  |    23.0128    |   96517    |
+---------------+---------------+----------+---------------+------------+

Best Model Parameters:
Configuration: Model3
Test Accuracy: 0.5671
F1 Score: 0.5669
Training Time: 23.0679 seconds
Number of Parameters: 96261
```

Model 3 is the best performing model according to F1 Score and Test Accuracy

8 Using Best Model

8.0.1 (b) For the best set of parameters obtained above, using two and three fully connected layers (After Flatten)

2 layers after Flatten

[52]:

```
model3_fc_2 = Sequential(name="Model3_fc_2")

model3_fc_2.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model3_fc_2.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_2.add(Dropout(0.1))

model3_fc_2.add(
    Conv2D(filters=32, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_2.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model3_fc_2.add(Dropout(0.1))

model3_fc_2.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_2.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_2.add(Dropout(0.1))

model3_fc_2.add(Flatten())
model3_fc_2.add(Dense(128, activation="relu"))
model3_fc_2.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_2.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model3_fc_2.summary()
```

```
Model: "Model3_fc_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_36 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_36 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_36 (Dropout)        (None, 40, 40, 16)        0

 conv2d_37 (Conv2D)          (None, 40, 40, 32)        12832

 max_pooling2d_37 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_37 (Dropout)        (None, 20, 20, 32)        0

 conv2d_38 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_38 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_38 (Dropout)        (None, 10, 10, 64)        0

 flatten_12 (Flatten)        (None, 6400)              0

 dense_25 (Dense)            (None, 128)               819328
```

```
 dense_26 (Dense)            (None, 5)                 645

=================================================================
Total params: 884,229
Trainable params: 884,229
Non-trainable params: 0
_________________________________________________________________
```

[53]:

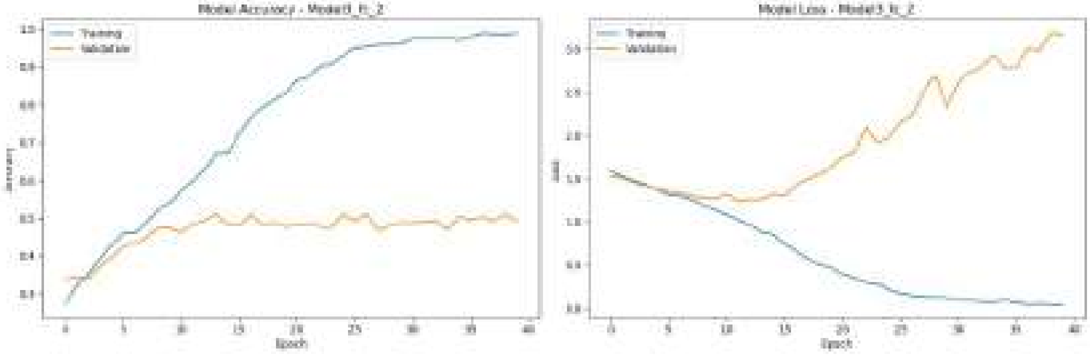
```python
history, training_time = train_and_evaluate(
    model3_fc_2, X_train, y_train, X_test, y_test, batch_size=128, epochs=40
)
test_loss, test_acc = model3_fc_2.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_2.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3_fc_2",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_2.count_params(),
    }
)

plot_training_history(history, f"Model3_fc_2")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_2")
print("\nResults for Model3_fc_2:")
print(f"Configuration: {results[4]['configuration']}")
print(f"Test Accuracy: {results[4]['test_accuracy']:.4f}")
print(f"F1 Score: {results[4]['f1_score']:.4f}")
print(f"Training Time: {results[4]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[4]['parameters']:,}")
```

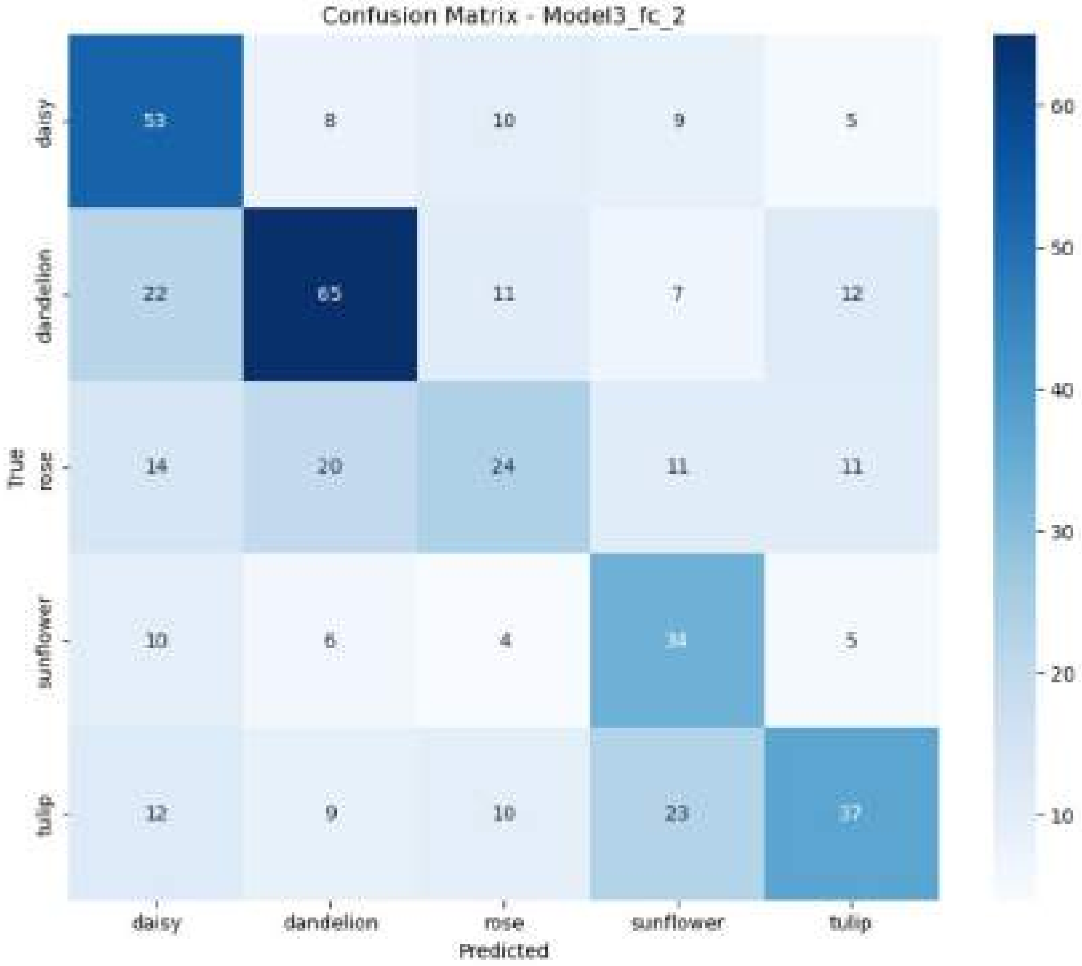
```
Epoch 1/40
31/31 [==============================] - 2s 33ms/step - loss: 1.5876 - accuracy:
0.2713 - val_loss: 1.5310 - val_accuracy: 0.3380
Epoch 2/40
31/31 [==============================] - 1s 23ms/step - loss: 1.5231 - accuracy:
0.3228 - val_loss: 1.5082 - val_accuracy: 0.3449
Epoch 3/40
31/31 [==============================] - 1s 23ms/step - loss: 1.4750 - accuracy:
0.3506 - val_loss: 1.4587 - val_accuracy: 0.3380
Epoch 4/40
31/31 [==============================] - 1s 23ms/step - loss: 1.4310 - accuracy:
0.3933 - val_loss: 1.4152 - val_accuracy: 0.3727
Epoch 5/40
31/31 [==============================] - 1s 23ms/step - loss: 1.3742 - accuracy:
```

```
0.4286 - val_loss: 1.3774 - val_accuracy: 0.3958
Epoch 6/40
31/31 [==============================] - 1s 23ms/step - loss: 1.3134 - accuracy:
0.4607 - val_loss: 1.3519 - val_accuracy: 0.4259
Epoch 7/40
31/31 [==============================] - 1s 23ms/step - loss: 1.3005 - accuracy:
0.4615 - val_loss: 1.3321 - val_accuracy: 0.4329
Epoch 8/40
31/31 [==============================] - 1s 24ms/step - loss: 1.2597 - accuracy:
0.4870 - val_loss: 1.3005 - val_accuracy: 0.4468
Epoch 9/40
31/31 [==============================] - 1s 23ms/step - loss: 1.1917 - accuracy:
0.5212 - val_loss: 1.2824 - val_accuracy: 0.4745
Epoch 10/40
31/31 [==============================] - 1s 23ms/step - loss: 1.1420 - accuracy:
0.5403 - val_loss: 1.2669 - val_accuracy: 0.4745
Epoch 11/40
31/31 [==============================] - 1s 24ms/step - loss: 1.0865 - accuracy:
0.5699 - val_loss: 1.3269 - val_accuracy: 0.4606
Epoch 12/40
31/31 [==============================] - 1s 24ms/step - loss: 1.0259 - accuracy:
0.5982 - val_loss: 1.2401 - val_accuracy: 0.4838
Epoch 13/40
31/31 [==============================] - 1s 23ms/step - loss: 0.9671 - accuracy:
0.6265 - val_loss: 1.2618 - val_accuracy: 0.4931
Epoch 14/40
31/31 [==============================] - 1s 24ms/step - loss: 0.8858 - accuracy:
0.6690 - val_loss: 1.2636 - val_accuracy: 0.5116
Epoch 15/40
31/31 [==============================] - 1s 23ms/step - loss: 0.8574 - accuracy:
0.6700 - val_loss: 1.3257 - val_accuracy: 0.4838
Epoch 16/40
31/31 [==============================] - 1s 23ms/step - loss: 0.7459 - accuracy:
0.7220 - val_loss: 1.3087 - val_accuracy: 0.4792
Epoch 17/40
31/31 [==============================] - 1s 24ms/step - loss: 0.6619 - accuracy:
0.7655 - val_loss: 1.4296 - val_accuracy: 0.5093
Epoch 18/40
31/31 [==============================] - 1s 23ms/step - loss: 0.5720 - accuracy:
0.7923 - val_loss: 1.4985 - val_accuracy: 0.4838
Epoch 19/40
31/31 [==============================] - 1s 24ms/step - loss: 0.5057 - accuracy:
0.8113 - val_loss: 1.5468 - val_accuracy: 0.4884
Epoch 20/40
31/31 [==============================] - 1s 23ms/step - loss: 0.4727 - accuracy:
0.8296 - val_loss: 1.6324 - val_accuracy: 0.4769
Epoch 21/40
31/31 [==============================] - 1s 23ms/step - loss: 0.3907 - accuracy:
```

```
0.8638 - val_loss: 1.7483 - val_accuracy: 0.4815
Epoch 22/40
31/31 [==============================] - 1s 23ms/step - loss: 0.3590 - accuracy:
0.8741 - val_loss: 1.7963 - val_accuracy: 0.4861
Epoch 23/40
31/31 [==============================] - 1s 23ms/step - loss: 0.2986 - accuracy:
0.9009 - val_loss: 2.0965 - val_accuracy: 0.4769
Epoch 24/40
31/31 [==============================] - 1s 23ms/step - loss: 0.2850 - accuracy:
0.9068 - val_loss: 1.9196 - val_accuracy: 0.4769
Epoch 25/40
31/31 [==============================] - 1s 23ms/step - loss: 0.2191 - accuracy:
0.9302 - val_loss: 1.9788 - val_accuracy: 0.5093
Epoch 26/40
31/31 [==============================] - 1s 23ms/step - loss: 0.1733 - accuracy:
0.9508 - val_loss: 2.1635 - val_accuracy: 0.4931
Epoch 27/40
31/31 [==============================] - 1s 23ms/step - loss: 0.1508 - accuracy:
0.9529 - val_loss: 2.2270 - val_accuracy: 0.5116
Epoch 28/40
31/31 [==============================] - 1s 24ms/step - loss: 0.1371 - accuracy:
0.9591 - val_loss: 2.5254 - val_accuracy: 0.4676
Epoch 29/40
31/31 [==============================] - 1s 23ms/step - loss: 0.1266 - accuracy:
0.9598 - val_loss: 2.6919 - val_accuracy: 0.4792
Epoch 30/40
31/31 [==============================] - 1s 23ms/step - loss: 0.1290 - accuracy:
0.9604 - val_loss: 2.3292 - val_accuracy: 0.4884
Epoch 31/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0916 - accuracy:
0.9758 - val_loss: 2.6233 - val_accuracy: 0.4861
Epoch 32/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0912 - accuracy:
0.9743 - val_loss: 2.7287 - val_accuracy: 0.4907
Epoch 33/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0808 - accuracy:
0.9768 - val_loss: 2.7908 - val_accuracy: 0.4931
Epoch 34/40
31/31 [==============================] - 1s 25ms/step - loss: 0.0761 - accuracy:
0.9789 - val_loss: 2.9263 - val_accuracy: 0.4699
Epoch 35/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0956 - accuracy:
0.9699 - val_loss: 2.7777 - val_accuracy: 0.5046
Epoch 36/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0679 - accuracy:
0.9794 - val_loss: 2.7894 - val_accuracy: 0.4954
Epoch 37/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0471 - accuracy:
```

```
0.9887 - val_loss: 3.0099 - val_accuracy: 0.5023
Epoch 38/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0566 - accuracy:
0.9869 - val_loss: 2.9820 - val_accuracy: 0.4931
Epoch 39/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0460 - accuracy:
0.9864 - val_loss: 3.1839 - val_accuracy: 0.5093
Epoch 40/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0468 - accuracy:
0.9889 - val_loss: 3.1707 - val_accuracy: 0.4931
14/14 [==============================] - 0s 5ms/step - loss: 3.1707 - accuracy:
0.4931
14/14 [==============================] - 0s 4ms/step
```





```
Results for Model3_fc_2:
Configuration: Model3_fc_2
Test Accuracy: 0.4931
F1 Score: 0.4686
Training Time: 30.37 seconds
Number of Parameters: 884,229
```

3 layers after Flatten

[67]:
```
model3_fc_3 = Sequential(name="Model3_fc_3")

model3_fc_3.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
```

```
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(
    Conv2D(filters=32, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(Flatten())

model3_fc_3.add(Dense(256, activation="relu"))
model3_fc_3.add(Dense(128, activation="relu"))
model3_fc_3.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
  ↪metrics=["accuracy"]
)
model3_fc_3.summary()
```

```
Model: "Model3_fc_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_45 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_45 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_45 (Dropout)        (None, 40, 40, 16)        0

 conv2d_46 (Conv2D)          (None, 40, 40, 32)        12832

 max_pooling2d_46 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)
```

```
 dropout_46 (Dropout)        (None, 20, 20, 32)        0

 conv2d_47 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_47 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_47 (Dropout)        (None, 10, 10, 64)        0

 flatten_15 (Flatten)        (None, 6400)              0

 dense_33 (Dense)            (None, 256)               1638656

 dense_34 (Dense)            (None, 128)               32896

 dense_35 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,736,453
Trainable params: 1,736,453
Non-trainable params: 0
_________________________________________________________________
```

[68]:

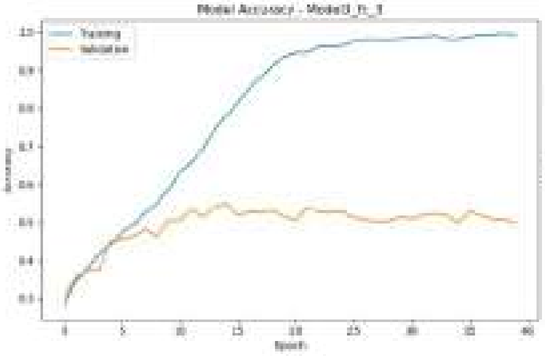
```python
history, training_time = train_and_evaluate(
    model3_fc_3, X_train, y_train, X_test, y_test, batch_size=128, epochs=40
)
test_loss, test_acc = model3_fc_3.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3_fc_3",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3.count_params(),
    }
)

plot_training_history(history, f"Model3_fc_3")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_3")

print("\nResults for Model3_fc_3:")
print(f"Configuration: {results[5]['configuration']}")
print(f"Test Accuracy: {results[5]['test_accuracy']:.4f}")
print(f"F1 Score: {results[5]['f1_score']:.4f}")
```

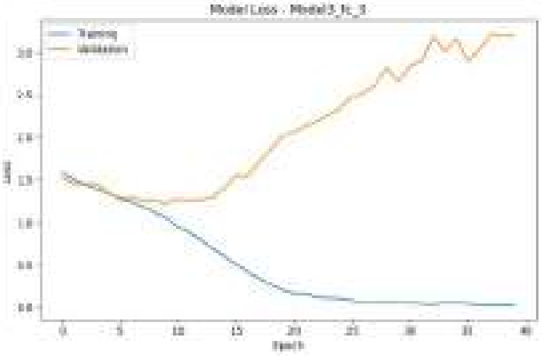
```python
print(f"Training Time: {results[5]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[5]['parameters']:,}")
```

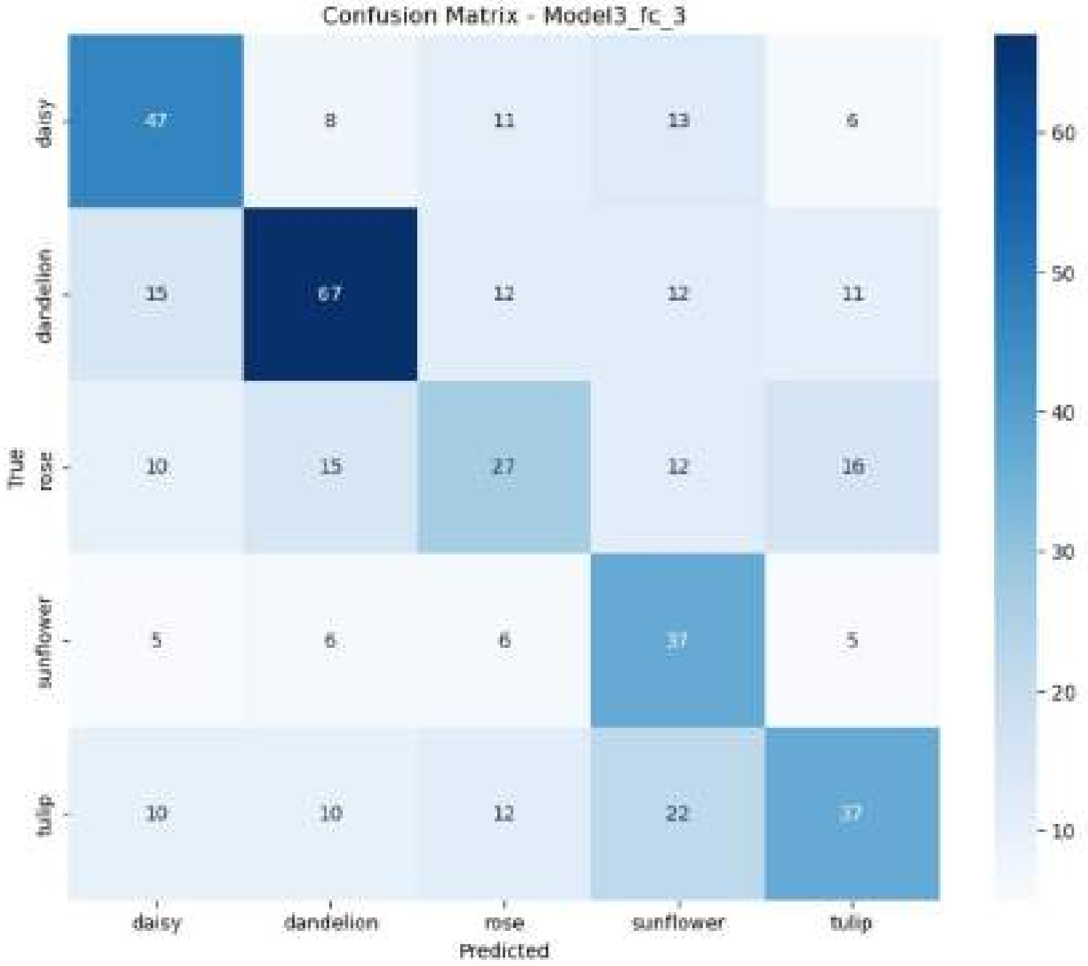
```
Epoch 1/40
31/31 [==============================] - 2s 32ms/step - loss: 1.5909 - accuracy:
0.2795 - val_loss: 1.5211 - val_accuracy: 0.3009
Epoch 2/40
31/31 [==============================] - 1s 23ms/step - loss: 1.4977 - accuracy:
0.3480 - val_loss: 1.4590 - val_accuracy: 0.3565
Epoch 3/40
31/31 [==============================] - 1s 24ms/step - loss: 1.4489 - accuracy:
0.3755 - val_loss: 1.4491 - val_accuracy: 0.3704
Epoch 4/40
31/31 [==============================] - 1s 23ms/step - loss: 1.3876 - accuracy:
0.4165 - val_loss: 1.4620 - val_accuracy: 0.3750
Epoch 5/40
31/31 [==============================] - 1s 23ms/step - loss: 1.3384 - accuracy:
0.4420 - val_loss: 1.3278 - val_accuracy: 0.4491
Epoch 6/40
31/31 [==============================] - 1s 24ms/step - loss: 1.2760 - accuracy:
0.4728 - val_loss: 1.2922 - val_accuracy: 0.4560
Epoch 7/40
31/31 [==============================] - 1s 24ms/step - loss: 1.2426 - accuracy:
0.4947 - val_loss: 1.2916 - val_accuracy: 0.4630
Epoch 8/40
31/31 [==============================] - 1s 26ms/step - loss: 1.1854 - accuracy:
0.5248 - val_loss: 1.2598 - val_accuracy: 0.4792
Epoch 9/40
31/31 [==============================] - 1s 23ms/step - loss: 1.1137 - accuracy:
0.5495 - val_loss: 1.2526 - val_accuracy: 0.4630
Epoch 10/40
31/31 [==============================] - 1s 23ms/step - loss: 1.0517 - accuracy:
0.5853 - val_loss: 1.2339 - val_accuracy: 0.5046
Epoch 11/40
31/31 [==============================] - 1s 24ms/step - loss: 0.9479 - accuracy:
0.6322 - val_loss: 1.2730 - val_accuracy: 0.5046
Epoch 12/40
31/31 [==============================] - 1s 24ms/step - loss: 0.8780 - accuracy:
0.6577 - val_loss: 1.2613 - val_accuracy: 0.5347
Epoch 13/40
31/31 [==============================] - 1s 24ms/step - loss: 0.7965 - accuracy:
0.6937 - val_loss: 1.2685 - val_accuracy: 0.5139
Epoch 14/40
31/31 [==============================] - 1s 24ms/step - loss: 0.6882 - accuracy:
0.7439 - val_loss: 1.2947 - val_accuracy: 0.5417
Epoch 15/40
31/31 [==============================] - 1s 23ms/step - loss: 0.6056 - accuracy:
```

```
0.7799 - val_loss: 1.4121 - val_accuracy: 0.5486
Epoch 16/40
31/31 [==============================] - 1s 23ms/step - loss: 0.5095 - accuracy:
0.8139 - val_loss: 1.5488 - val_accuracy: 0.5185
Epoch 17/40
31/31 [==============================] - 1s 24ms/step - loss: 0.4174 - accuracy:
0.8541 - val_loss: 1.5368 - val_accuracy: 0.5278
Epoch 18/40
31/31 [==============================] - 1s 25ms/step - loss: 0.3426 - accuracy:
0.8831 - val_loss: 1.7176 - val_accuracy: 0.5255
Epoch 19/40
31/31 [==============================] - 1s 23ms/step - loss: 0.2724 - accuracy:
0.9109 - val_loss: 1.8598 - val_accuracy: 0.5324
Epoch 20/40
31/31 [==============================] - 1s 24ms/step - loss: 0.2049 - accuracy:
0.9354 - val_loss: 2.0218 - val_accuracy: 0.5139
Epoch 21/40
31/31 [==============================] - 1s 23ms/step - loss: 0.1692 - accuracy:
0.9475 - val_loss: 2.0644 - val_accuracy: 0.5069
Epoch 22/40
31/31 [==============================] - 1s 24ms/step - loss: 0.1662 - accuracy:
0.9467 - val_loss: 2.1316 - val_accuracy: 0.5394
Epoch 23/40
31/31 [==============================] - 1s 25ms/step - loss: 0.1250 - accuracy:
0.9629 - val_loss: 2.1942 - val_accuracy: 0.5278
Epoch 24/40
31/31 [==============================] - 1s 24ms/step - loss: 0.1169 - accuracy:
0.9632 - val_loss: 2.2662 - val_accuracy: 0.5255
Epoch 25/40
31/31 [==============================] - 1s 24ms/step - loss: 0.1113 - accuracy:
0.9655 - val_loss: 2.3538 - val_accuracy: 0.5301
Epoch 26/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0855 - accuracy:
0.9753 - val_loss: 2.4732 - val_accuracy: 0.5116
Epoch 27/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0648 - accuracy:
0.9812 - val_loss: 2.5215 - val_accuracy: 0.5069
Epoch 28/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0614 - accuracy:
0.9812 - val_loss: 2.6162 - val_accuracy: 0.5023
Epoch 29/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0738 - accuracy:
0.9794 - val_loss: 2.8116 - val_accuracy: 0.5023
Epoch 30/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0592 - accuracy:
0.9830 - val_loss: 2.6508 - val_accuracy: 0.5139
Epoch 31/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0498 - accuracy:
```

```
0.9840 - val_loss: 2.8424 - val_accuracy: 0.5093
Epoch 32/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0455 - accuracy:
0.9871 - val_loss: 2.8919 - val_accuracy: 0.5185
Epoch 33/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0327 - accuracy:
0.9907 - val_loss: 3.1975 - val_accuracy: 0.5231
Epoch 34/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0540 - accuracy:
0.9822 - val_loss: 3.0109 - val_accuracy: 0.5185
Epoch 35/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0581 - accuracy:
0.9817 - val_loss: 3.1630 - val_accuracy: 0.4977
Epoch 36/40
31/31 [==============================] - 1s 24ms/step - loss: 0.0421 - accuracy:
0.9874 - val_loss: 2.8894 - val_accuracy: 0.5278
Epoch 37/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0317 - accuracy:
0.9936 - val_loss: 3.0295 - val_accuracy: 0.5185
Epoch 38/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0246 - accuracy:
0.9941 - val_loss: 3.2194 - val_accuracy: 0.5069
Epoch 39/40
31/31 [==============================] - 1s 23ms/step - loss: 0.0200 - accuracy:
0.9961 - val_loss: 3.1946 - val_accuracy: 0.5069
Epoch 40/40
31/31 [==============================] - 1s 25ms/step - loss: 0.0225 - accuracy:
0.9933 - val_loss: 3.2028 - val_accuracy: 0.4977
14/14 [==============================] - 0s 6ms/step - loss: 3.2028 - accuracy:
0.4977
14/14 [==============================] - 0s 7ms/step
```







```
Results for Model3_fc_3:
Configuration: Model3_fc_3
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.02 seconds
Number of Parameters: 1,736,453
```

[69]:

```
headers = results[0].keys()
rows = [
    [
        f"{row['configuration']}",
        f"{row['test_accuracy']:.4f}",
        f"{row['f1_score']:.4f}",
        f"{row['training_time']:.4f}",
        row["parameters"],
```

```
    ]
    for row in results
]

# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt="pretty"))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x["test_accuracy"])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+---------------+---------------+----------+---------------+------------+
| configuration | test_accuracy | f1_score | training_time | parameters |
+---------------+---------------+----------+---------------+------------+
|    Model1     |    0.5556     |  0.5541  |    22.0594    |   55301    |
|    Model2     |    0.5231     |  0.5243  |    20.9627    |   88069    |
|    Model3     |    0.5671     |  0.5669  |    23.0679    |   96261    |
|    Model4     |    0.5185     |  0.5160  |    23.0128    |   96517    |
|  Model3_fc_2  |    0.4931     |  0.4886  |    30.3716    |   884229   |
|  Model3_fc_3  |    0.5772     |  0.5796  |    31.0176    |  1736453   |
+---------------+---------------+----------+---------------+------------+

Best Model Parameters:
Configuration: Model3_fc_3
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.0176 seconds
Number of Parameters: 1736453
```

Model3_fc_3 (3 fully connected layers after Flatten is the best)

8.0.2 (c) For the best set of parameters obtained above, using average pooling instead of Max Pooling

[70]:

```
model3_fc_3_avg_pool = Sequential(name="Model3_fc_3_avg_pool")

model3_fc_3_avg_pool.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
```

```
        padding="same",
        activation="relu",
        input_shape=(80, 80, 1),
    )
)
model3_fc_3_avg_pool.add(AveragePooling2D(pool_size=(2, 2)))
model3_fc_3_avg_pool.add(Dropout(0.1))

model3_fc_3_avg_pool.add(
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu")
)
model3_fc_3_avg_pool.add(AveragePooling2D(pool_size=(2, 2)))
model3_fc_3_avg_pool.add(Dropout(0.1))

model3_fc_3_avg_pool.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_3_avg_pool.add(AveragePooling2D(pool_size=(2, 2)))
model3_fc_3_avg_pool.add(Dropout(0.1))

model3_fc_3_avg_pool.add(Flatten())

model3_fc_3_avg_pool.add(Dense(256, activation="relu"))
model3_fc_3_avg_pool.add(Dense(128, activation="relu"))
model3_fc_3_avg_pool.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3_avg_pool.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model3_fc_3_avg_pool.summary()
```

```
Model: "Model3_fc_3_avg_pool"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_48 (Conv2D)          (None, 80, 80, 16)        160

 average_pooling2d (AverageP  (None, 40, 40, 16)       0
 ooling2D)

 dropout_48 (Dropout)        (None, 40, 40, 16)        0

 conv2d_49 (Conv2D)          (None, 40, 40, 32)        4640

 average_pooling2d_1 (Averag  (None, 20, 20, 32)       0
 ePooling2D)
```

```
 dropout_49 (Dropout)        (None, 20, 20, 32)        0

 conv2d_50 (Conv2D)          (None, 20, 20, 64)        51264

 average_pooling2d_2 (Averag  (None, 10, 10, 64)       0
 ePooling2D)

 dropout_50 (Dropout)        (None, 10, 10, 64)        0

 flatten_16 (Flatten)        (None, 6400)              0

 dense_36 (Dense)            (None, 256)               1638656

 dense_37 (Dense)            (None, 128)               32896

 dense_38 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,261
Trainable params: 1,728,261
Non-trainable params: 0
_________________________________________________________________
```

[71]:

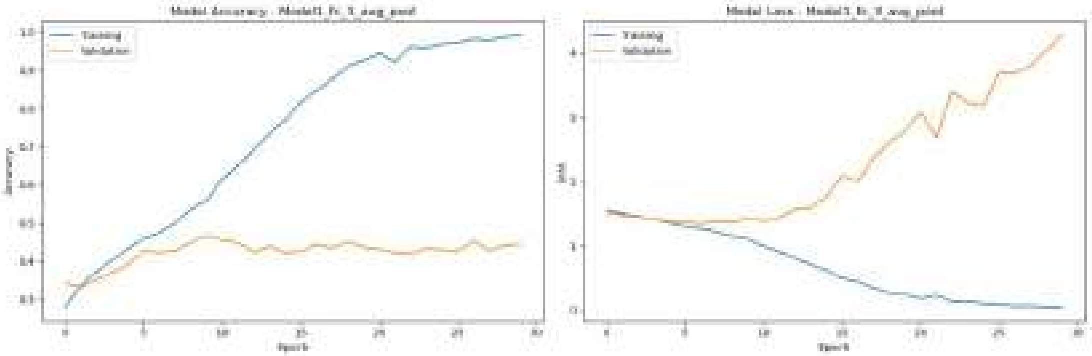
```
history, training_time = train_and_evaluate(
    model3_fc_3_avg_pool, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model3_fc_3_avg_pool.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3_avg_pool.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model1_fc_3_avg_pool",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3_avg_pool.count_params(),
    }
)

plot_training_history(history, f"Model1_fc_3_avg_pool")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model1_fc_3_avg_pool")
print("\nResults for Model1_fc_3_avg_pool:")
print(f"Configuration: {results[6]['configuration']}")
print(f"Test Accuracy: {results[6]['test_accuracy']:.4f}")
print(f"F1 Score: {results[6]['f1_score']:.4f}")
```

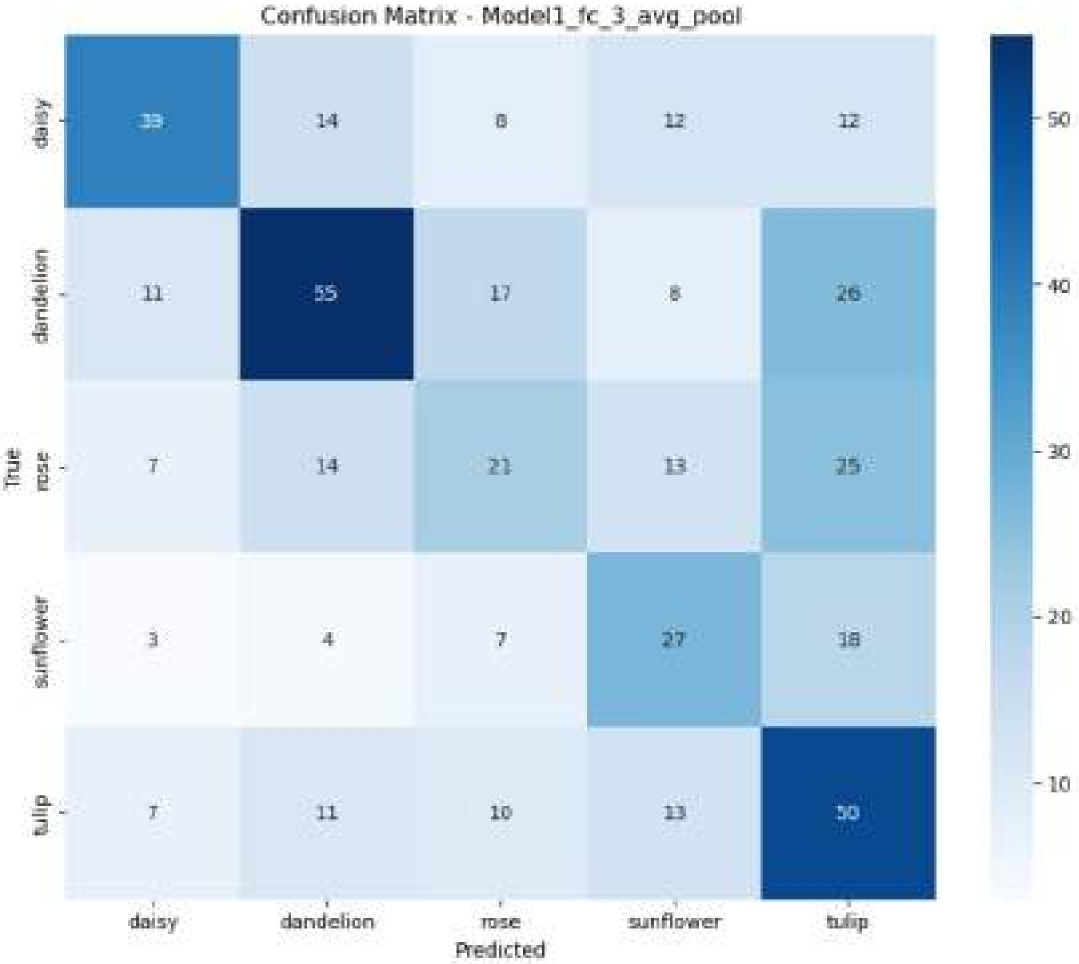
```python
print(f"Training Time: {results[6]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[6]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 32ms/step - loss: 1.5593 - accuracy:
0.2798 - val_loss: 1.5214 - val_accuracy: 0.3426
Epoch 2/30
31/31 [==============================] - 1s 21ms/step - loss: 1.5044 - accuracy:
0.3372 - val_loss: 1.4783 - val_accuracy: 0.3333
Epoch 3/30
31/31 [==============================] - 1s 21ms/step - loss: 1.4532 - accuracy:
0.3732 - val_loss: 1.4571 - val_accuracy: 0.3519
Epoch 4/30
31/31 [==============================] - 1s 20ms/step - loss: 1.4133 - accuracy:
0.4039 - val_loss: 1.4158 - val_accuracy: 0.3704
Epoch 5/30
31/31 [==============================] - 1s 21ms/step - loss: 1.3537 - accuracy:
0.4335 - val_loss: 1.3667 - val_accuracy: 0.3912
Epoch 6/30
31/31 [==============================] - 1s 21ms/step - loss: 1.3009 - accuracy:
0.4602 - val_loss: 1.3682 - val_accuracy: 0.4282
Epoch 7/30
31/31 [==============================] - 1s 21ms/step - loss: 1.2677 - accuracy:
0.4754 - val_loss: 1.3387 - val_accuracy: 0.4236
Epoch 8/30
31/31 [==============================] - 1s 21ms/step - loss: 1.2204 - accuracy:
0.4999 - val_loss: 1.3809 - val_accuracy: 0.4282
Epoch 9/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1537 - accuracy:
0.5372 - val_loss: 1.3609 - val_accuracy: 0.4491
Epoch 10/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1034 - accuracy:
0.5580 - val_loss: 1.4230 - val_accuracy: 0.4676
Epoch 11/30
31/31 [==============================] - 1s 22ms/step - loss: 0.9921 - accuracy:
0.6154 - val_loss: 1.3805 - val_accuracy: 0.4560
Epoch 12/30
31/31 [==============================] - 1s 21ms/step - loss: 0.9069 - accuracy:
0.6499 - val_loss: 1.4372 - val_accuracy: 0.4491
Epoch 13/30
31/31 [==============================] - 1s 21ms/step - loss: 0.8087 - accuracy:
0.6914 - val_loss: 1.5660 - val_accuracy: 0.4236
Epoch 14/30
31/31 [==============================] - 1s 21ms/step - loss: 0.7069 - accuracy:
0.7346 - val_loss: 1.5985 - val_accuracy: 0.4398
Epoch 15/30
31/31 [==============================] - 1s 21ms/step - loss: 0.6165 - accuracy:
```

```
0.7694 - val_loss: 1.7581 - val_accuracy: 0.4236
Epoch 16/30
31/31 [==============================] - 1s 21ms/step - loss: 0.4935 - accuracy:
0.8172 - val_loss: 2.0833 - val_accuracy: 0.4259
Epoch 17/30
31/31 [==============================] - 1s 22ms/step - loss: 0.4403 - accuracy:
0.8461 - val_loss: 2.0097 - val_accuracy: 0.4444
Epoch 18/30
31/31 [==============================] - 1s 21ms/step - loss: 0.3450 - accuracy:
0.8770 - val_loss: 2.3679 - val_accuracy: 0.4352
Epoch 19/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2652 - accuracy:
0.9107 - val_loss: 2.6097 - val_accuracy: 0.4514
Epoch 20/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2395 - accuracy:
0.9256 - val_loss: 2.7792 - val_accuracy: 0.4375
Epoch 21/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1777 - accuracy:
0.9429 - val_loss: 3.0776 - val_accuracy: 0.4306
Epoch 22/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2320 - accuracy:
0.9215 - val_loss: 2.6986 - val_accuracy: 0.4236
Epoch 23/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1270 - accuracy:
0.9622 - val_loss: 3.3911 - val_accuracy: 0.4213
Epoch 24/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1322 - accuracy:
0.9593 - val_loss: 3.2312 - val_accuracy: 0.4329
Epoch 25/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1000 - accuracy:
0.9686 - val_loss: 3.1737 - val_accuracy: 0.4306
Epoch 26/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0869 - accuracy:
0.9717 - val_loss: 3.6954 - val_accuracy: 0.4259
Epoch 27/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0656 - accuracy:
0.9817 - val_loss: 3.6802 - val_accuracy: 0.4537
Epoch 28/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0721 - accuracy:
0.9786 - val_loss: 3.7941 - val_accuracy: 0.4282
Epoch 29/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0512 - accuracy:
0.9866 - val_loss: 4.0355 - val_accuracy: 0.4421
Epoch 30/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0400 - accuracy:
0.9912 - val_loss: 4.2751 - val_accuracy: 0.4444
14/14 [==============================] - 0s 5ms/step - loss: 4.2751 - accuracy:
0.4444
```

```
14/14 [==============================] - 0s 4ms/step
```





	daisy	dandelion	rose	sunflower	tulip
daisy	39	14	8	12	12
dandelion	11	55	17	8	26
rose	7	14	21	13	25
sunflower	3	4	7	27	18
tulip	7	11	10	13	50

```
Results for Model1_fc_3_avg_pool:
Configuration: Model1_fc_3_avg_pool
Test Accuracy: 0.4444
F1 Score: 0.4447
Training Time: 21.23 seconds
Number of Parameters: 1,728,261
```

[72]:

```python
headers = results[0].keys()
rows = [
    [
        f"{row['configuration']}",
        f"{row['test_accuracy']:.4f}",
        f"{row['f1_score']:.4f}",
        f"{row['training_time']:.4f}",
        row["parameters"],
    ]
    for row in results
]

# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt="pretty"))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x["test_accuracy"])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+----------------------+---------------+----------+---------------+------------+
|    configuration     | test_accuracy | f1_score | training_time | parameters |
+----------------------+---------------+----------+---------------+------------+
|        Model1        |    0.5556     |  0.5541  |    22.0594    |   55301    |
|        Model2        |    0.5231     |  0.5243  |    20.9627    |   88069    |
|        Model3        |    0.5671     |  0.5669  |    23.0679    |   96261    |
|        Model4        |    0.5185     |  0.5160  |    23.0128    |   96517    |
|     Model3_fc_2      |    0.4931     |  0.4886  |    30.3716    |   884229   |
|     Model3_fc_3      |    0.5772     |  0.5796  |    31.0176    |  1736453   |
| Model1_fc_3_avg_pool |    0.4444     |  0.4447  |    21.2291    |  1728261   |
+----------------------+---------------+----------+---------------+------------+

Best Model Parameters:
Configuration: Model3_fc_3
```

```
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.0176 seconds
Number of Parameters: 1736453
```

Average pooling is not better. Continuing with the max pooling version and 3 fully connected layers, experimenting with different activations

8.0.3 (d) For the best set of parameters obtained above, using Sigmoid and Leaky ReLU

ReLU was already tested previously

Sigmoid

[73]:

```
model3_fc_3_sigmoid = Sequential(name="Model3_fc_3_sigmoid")

model3_fc_3_sigmoid.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        activation="sigmoid",
        input_shape=(80, 80, 1),
    )
)
model3_fc_3_sigmoid.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_sigmoid.add(Dropout(0.1))

model3_fc_3_sigmoid.add(
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="sigmoid")
)
model3_fc_3_sigmoid.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_sigmoid.add(Dropout(0.1))

model3_fc_3_sigmoid.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="sigmoid")
)
model3_fc_3_sigmoid.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_sigmoid.add(Dropout(0.1))

model3_fc_3_sigmoid.add(Flatten())

model3_fc_3_sigmoid.add(Dense(256, activation="sigmoid"))
model3_fc_3_sigmoid.add(Dense(128, activation="sigmoid"))
model3_fc_3_sigmoid.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3_sigmoid.compile(
```

```
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model3_fc_3_sigmoid.summary()
```

```
Model: "Model3_fc_3_sigmoid"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_51 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_48 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_51 (Dropout)        (None, 40, 40, 16)        0

 conv2d_52 (Conv2D)          (None, 40, 40, 32)        4640

 max_pooling2d_49 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_52 (Dropout)        (None, 20, 20, 32)        0

 conv2d_53 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_50 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_53 (Dropout)        (None, 10, 10, 64)        0

 flatten_17 (Flatten)        (None, 6400)              0

 dense_39 (Dense)            (None, 256)               1638656

 dense_40 (Dense)            (None, 128)               32896

 dense_41 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,261
Trainable params: 1,728,261
Non-trainable params: 0
_________________________________________________________________
```

[74]:

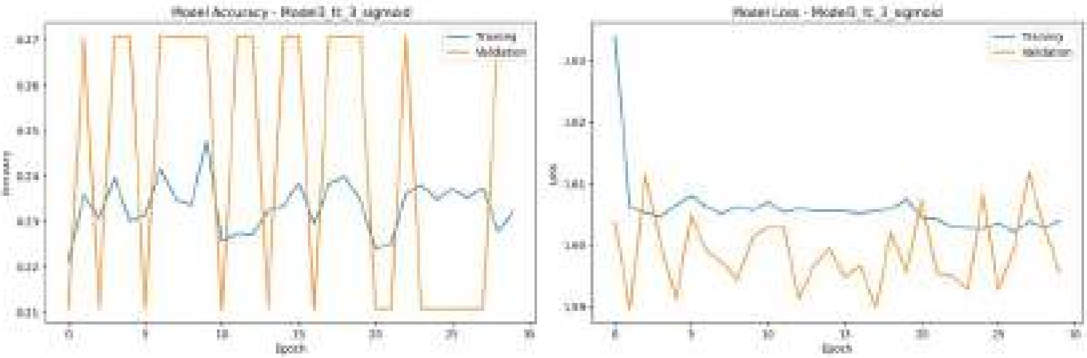
```
history, training_time = train_and_evaluate(
    model3_fc_3_sigmoid, X_train, y_train, X_test, y_test, batch_size=128
)
```

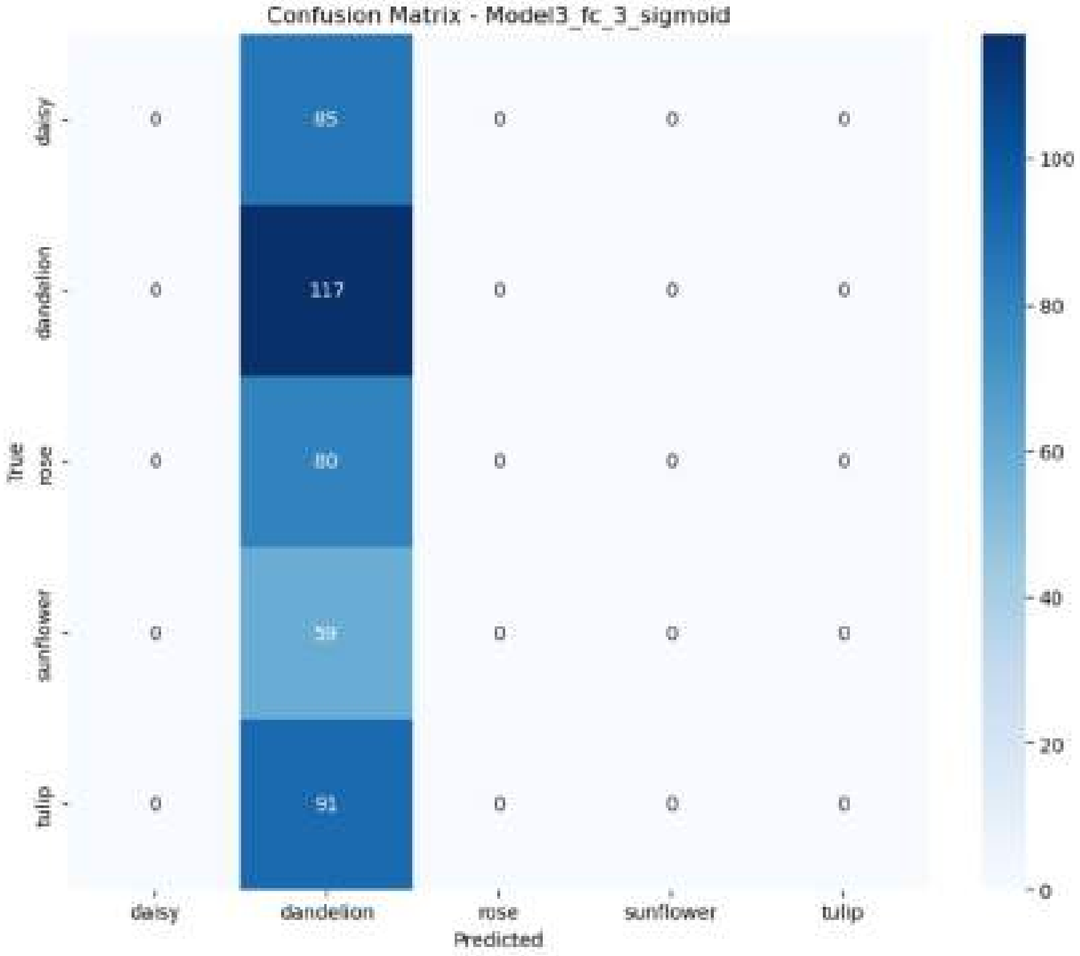
```
test_loss, test_acc = model3_fc_3_sigmoid.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3_sigmoid.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3_fc_3_sigmoid",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3_sigmoid.count_params(),
    }
)

plot_training_history(history, f"Model3_fc_3_sigmoid")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_3_sigmoid")
print("\nResults for Model3_fc_3_sigmoid:")
print(f"Configuration: {results[7]['configuration']}")
print(f"Test Accuracy: {results[7]['test_accuracy']:.4f}")
print(f"F1 Score: {results[7]['f1_score']:.4f}")
print(f"Training Time: {results[7]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[7]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 38ms/step - loss: 1.6339 - accuracy:
0.2214 - val_loss: 1.6038 - val_accuracy: 0.2106
Epoch 2/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6062 - accuracy:
0.2358 - val_loss: 1.5896 - val_accuracy: 0.2708
Epoch 3/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6053 - accuracy:
0.2306 - val_loss: 1.6113 - val_accuracy: 0.2106
Epoch 4/30
31/31 [==============================] - 1s 26ms/step - loss: 1.6045 - accuracy:
0.2396 - val_loss: 1.6003 - val_accuracy: 0.2708
Epoch 5/30
31/31 [==============================] - 1s 26ms/step - loss: 1.6066 - accuracy:
0.2301 - val_loss: 1.5913 - val_accuracy: 0.2708
Epoch 6/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6079 - accuracy:
0.2314 - val_loss: 1.6048 - val_accuracy: 0.2106
Epoch 7/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6062 - accuracy:
0.2417 - val_loss: 1.5990 - val_accuracy: 0.2708
Epoch 8/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6050 - accuracy:
0.2347 - val_loss: 1.5973 - val_accuracy: 0.2708
```

```
Epoch 9/30
31/31 [==============================] - 1s 26ms/step - loss: 1.6063 - accuracy:
0.2337 - val_loss: 1.5943 - val_accuracy: 0.2708
Epoch 10/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6056 - accuracy:
0.2476 - val_loss: 1.6014 - val_accuracy: 0.2708
Epoch 11/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6070 - accuracy:
0.2257 - val_loss: 1.6029 - val_accuracy: 0.2106
Epoch 12/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6053 - accuracy:
0.2273 - val_loss: 1.6029 - val_accuracy: 0.2708
Epoch 13/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6059 - accuracy:
0.2270 - val_loss: 1.5915 - val_accuracy: 0.2708
Epoch 14/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6056 - accuracy:
0.2324 - val_loss: 1.5967 - val_accuracy: 0.2106
Epoch 15/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6055 - accuracy:
0.2335 - val_loss: 1.5995 - val_accuracy: 0.2708
Epoch 16/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6055 - accuracy:
0.2384 - val_loss: 1.5948 - val_accuracy: 0.2708
Epoch 17/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6051 - accuracy:
0.2296 - val_loss: 1.5966 - val_accuracy: 0.2106
Epoch 18/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6056 - accuracy:
0.2384 - val_loss: 1.5899 - val_accuracy: 0.2708
Epoch 19/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6059 - accuracy:
0.2399 - val_loss: 1.6021 - val_accuracy: 0.2708
Epoch 20/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6075 - accuracy:
0.2347 - val_loss: 1.5955 - val_accuracy: 0.2708
Epoch 21/30
31/31 [==============================] - 1s 23ms/step - loss: 1.6045 - accuracy:
0.2239 - val_loss: 1.6075 - val_accuracy: 0.2106
Epoch 22/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6042 - accuracy:
0.2250 - val_loss: 1.5954 - val_accuracy: 0.2106
Epoch 23/30
31/31 [==============================] - 1s 25ms/step - loss: 1.6030 - accuracy:
0.2360 - val_loss: 1.5950 - val_accuracy: 0.2708
Epoch 24/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6028 - accuracy:
0.2381 - val_loss: 1.5928 - val_accuracy: 0.2106
```

```
Epoch 25/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6027 - accuracy:
0.2347 - val_loss: 1.6081 - val_accuracy: 0.2106
Epoch 26/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6037 - accuracy:
0.2371 - val_loss: 1.5927 - val_accuracy: 0.2106
Epoch 27/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6021 - accuracy:
0.2353 - val_loss: 1.5993 - val_accuracy: 0.2106
Epoch 28/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6039 - accuracy:
0.2373 - val_loss: 1.6117 - val_accuracy: 0.2106
Epoch 29/30
31/31 [==============================] - 1s 24ms/step - loss: 1.6029 - accuracy:
0.2278 - val_loss: 1.6027 - val_accuracy: 0.2708
Epoch 30/30
31/31 [==============================] - 1s 23ms/step - loss: 1.6040 - accuracy:
0.2322 - val_loss: 1.5956 - val_accuracy: 0.2708
14/14 [==============================] - 0s 15ms/step - loss: 1.5956 - accuracy:
0.2708
14/14 [==============================] - 0s 4ms/step
```





```
Results for Model3_fc_3_sigmoid:
Configuration: Model3_fc_3_sigmoid
Test Accuracy: 0.2708
F1 Score: 0.1154
Training Time: 24.52 seconds
Number of Parameters: 1,728,261
```

Leaky ReLU

[75]:

```
model3_fc_3_leaky_relu = Sequential(name="Model3_fc_3_leaky_relu")

model3_fc_3_leaky_relu.add(
    Conv2D(filters=16, kernel_size=(3, 3), padding="same", input_shape=(80, 80,␣
 ↪1))
)
model3_fc_3_leaky_relu.add(LeakyReLU(alpha=0.01))
```

```
model3_fc_3_leaky_relu.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_leaky_relu.add(Dropout(0.1))

model3_fc_3_leaky_relu.add(Conv2D(filters=32, kernel_size=(3, 3),␣
↪padding="same"))
model3_fc_3_leaky_relu.add(LeakyReLU(alpha=0.01))
model3_fc_3_leaky_relu.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_leaky_relu.add(Dropout(0.1))

model3_fc_3_leaky_relu.add(Conv2D(filters=64, kernel_size=(5, 5),␣
↪padding="same"))
model3_fc_3_leaky_relu.add(LeakyReLU(alpha=0.01))
model3_fc_3_leaky_relu.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_leaky_relu.add(Dropout(0.1))

model3_fc_3_leaky_relu.add(Flatten())

model3_fc_3_leaky_relu.add(Dense(256))
model3_fc_3_leaky_relu.add(LeakyReLU(alpha=0.01))
model3_fc_3_leaky_relu.add(Dense(128))
model3_fc_3_leaky_relu.add(LeakyReLU(alpha=0.01))
model3_fc_3_leaky_relu.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3_leaky_relu.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
↪metrics=["accuracy"]
)
model3_fc_3_leaky_relu.summary()
```

```
Model: "Model3_fc_3_leaky_relu"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_54 (Conv2D)          (None, 80, 80, 16)        160

 leaky_re_lu (LeakyReLU)     (None, 80, 80, 16)        0

 max_pooling2d_51 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_54 (Dropout)        (None, 40, 40, 16)        0

 conv2d_55 (Conv2D)          (None, 40, 40, 32)        4640

 leaky_re_lu_1 (LeakyReLU)   (None, 40, 40, 32)        0

 max_pooling2d_52 (MaxPoolin  (None, 20, 20, 32)       0
```

```
 g2D)

 dropout_55 (Dropout)        (None, 20, 20, 32)        0

 conv2d_56 (Conv2D)          (None, 20, 20, 64)        51264

 leaky_re_lu_2 (LeakyReLU)   (None, 20, 20, 64)        0

 max_pooling2d_53 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_56 (Dropout)        (None, 10, 10, 64)        0

 flatten_18 (Flatten)        (None, 6400)              0

 dense_42 (Dense)            (None, 256)               1638656

 leaky_re_lu_3 (LeakyReLU)   (None, 256)               0

 dense_43 (Dense)            (None, 128)               32896

 leaky_re_lu_4 (LeakyReLU)   (None, 128)               0

 dense_44 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,261
Trainable params: 1,728,261
Non-trainable params: 0
_________________________________________________________________
```

[76]:

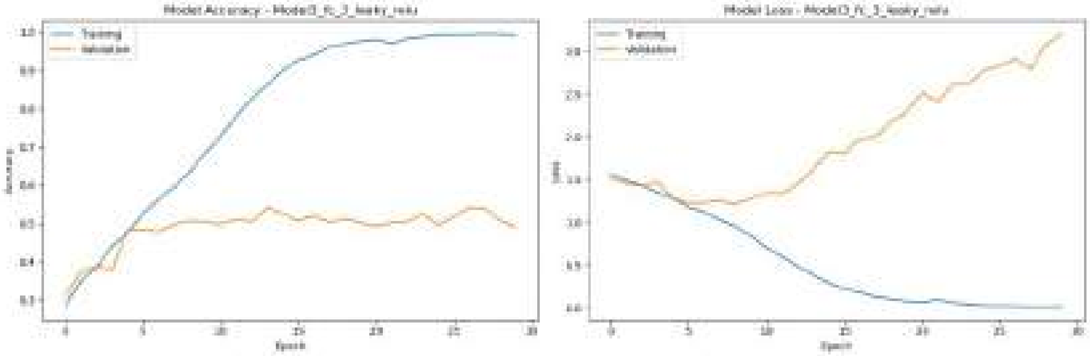
```
history, training_time = train_and_evaluate(
    model3_fc_3_leaky_relu, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model3_fc_3_leaky_relu.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3_leaky_relu.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3_fc_3_leaky_relu",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3_leaky_relu.count_params(),
    }
)
```

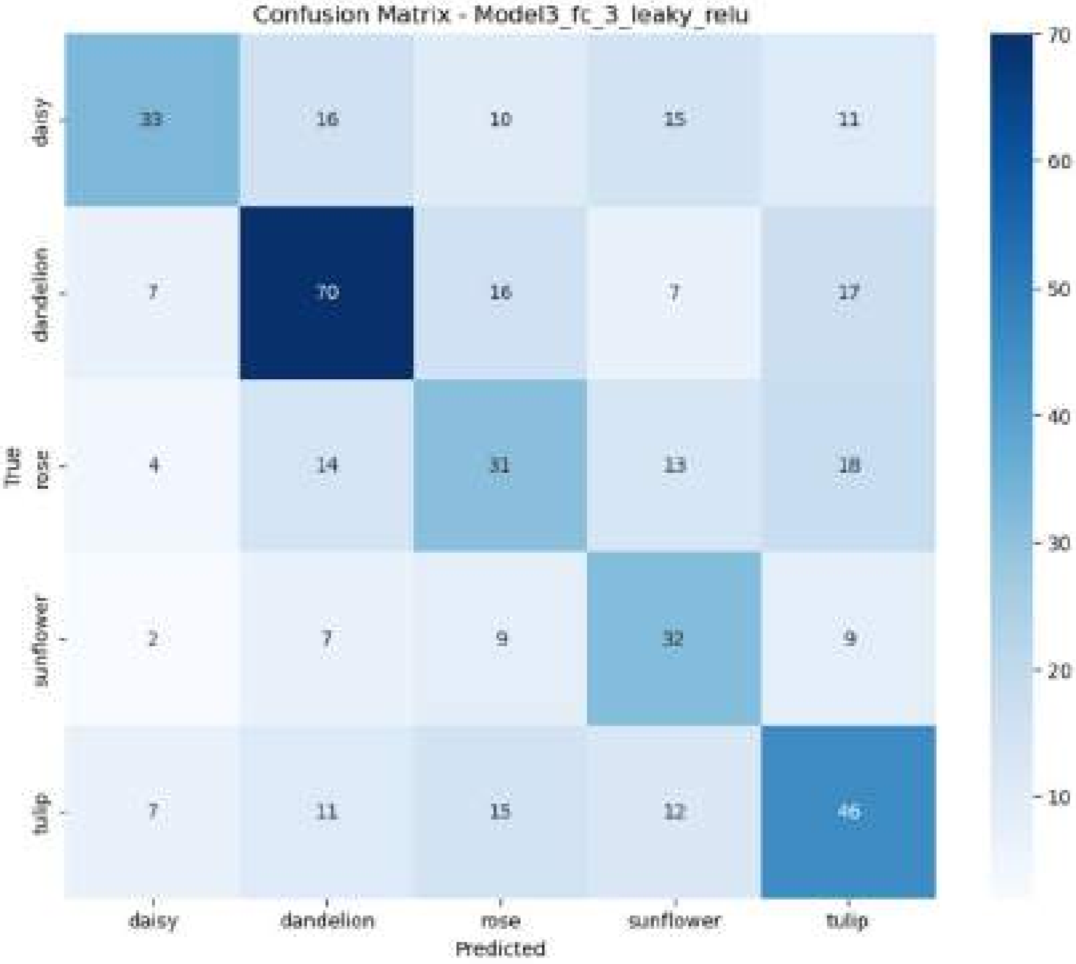
```
plot_training_history(history, f"Model3_fc_3_leaky_relu")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_3_leaky_relu")
print("\nResults for Model3_fc_3_leaky_relu:")
print(f"Configuration: {results[8]['configuration']}")
print(f"Test Accuracy: {results[8]['test_accuracy']:.4f}")
print(f"F1 Score: {results[8]['f1_score']:.4f}")
print(f"Training Time: {results[8]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[8]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 33ms/step - loss: 1.5635 - accuracy:
0.2829 - val_loss: 1.5145 - val_accuracy: 0.3079
Epoch 2/30
31/31 [==============================] - 1s 24ms/step - loss: 1.4882 - accuracy:
0.3498 - val_loss: 1.4556 - val_accuracy: 0.3727
Epoch 3/30
31/31 [==============================] - 1s 24ms/step - loss: 1.4318 - accuracy:
0.3833 - val_loss: 1.4252 - val_accuracy: 0.3889
Epoch 4/30
31/31 [==============================] - 1s 24ms/step - loss: 1.3492 - accuracy:
0.4399 - val_loss: 1.4720 - val_accuracy: 0.3773
Epoch 5/30
31/31 [==============================] - 1s 24ms/step - loss: 1.2792 - accuracy:
0.4757 - val_loss: 1.2824 - val_accuracy: 0.4815
Epoch 6/30
31/31 [==============================] - 1s 24ms/step - loss: 1.1790 - accuracy:
0.5261 - val_loss: 1.2293 - val_accuracy: 0.4838
Epoch 7/30
31/31 [==============================] - 1s 25ms/step - loss: 1.1216 - accuracy:
0.5632 - val_loss: 1.2311 - val_accuracy: 0.4792
Epoch 8/30
31/31 [==============================] - 1s 24ms/step - loss: 1.0275 - accuracy:
0.5956 - val_loss: 1.2585 - val_accuracy: 0.4977
Epoch 9/30
31/31 [==============================] - 1s 24ms/step - loss: 0.9554 - accuracy:
0.6340 - val_loss: 1.2111 - val_accuracy: 0.5069
Epoch 10/30
31/31 [==============================] - 1s 26ms/step - loss: 0.8432 - accuracy:
0.6834 - val_loss: 1.2819 - val_accuracy: 0.5046
Epoch 11/30
31/31 [==============================] - 1s 24ms/step - loss: 0.7076 - accuracy:
0.7284 - val_loss: 1.3438 - val_accuracy: 0.4977
Epoch 12/30
31/31 [==============================] - 1s 24ms/step - loss: 0.6014 - accuracy:
0.7812 - val_loss: 1.3353 - val_accuracy: 0.5116
Epoch 13/30
```

```
31/31 [==============================] - 1s 25ms/step - loss: 0.4879 - accuracy:
0.8273 - val_loss: 1.4499 - val_accuracy: 0.5046
Epoch 14/30
31/31 [==============================] - 1s 24ms/step - loss: 0.3905 - accuracy:
0.8638 - val_loss: 1.6162 - val_accuracy: 0.5417
Epoch 15/30
31/31 [==============================] - 1s 24ms/step - loss: 0.2897 - accuracy:
0.9009 - val_loss: 1.8274 - val_accuracy: 0.5255
Epoch 16/30
31/31 [==============================] - 1s 24ms/step - loss: 0.2258 - accuracy:
0.9279 - val_loss: 1.8146 - val_accuracy: 0.5069
Epoch 17/30
31/31 [==============================] - 1s 24ms/step - loss: 0.1852 - accuracy:
0.9416 - val_loss: 1.9713 - val_accuracy: 0.5208
Epoch 18/30
31/31 [==============================] - 1s 25ms/step - loss: 0.1326 - accuracy:
0.9624 - val_loss: 2.0016 - val_accuracy: 0.5023
Epoch 19/30
31/31 [==============================] - 1s 24ms/step - loss: 0.1060 - accuracy:
0.9699 - val_loss: 2.1808 - val_accuracy: 0.5139
Epoch 20/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0803 - accuracy:
0.9773 - val_loss: 2.2903 - val_accuracy: 0.5023
Epoch 21/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0646 - accuracy:
0.9817 - val_loss: 2.5198 - val_accuracy: 0.4931
Epoch 22/30
31/31 [==============================] - 1s 26ms/step - loss: 0.0975 - accuracy:
0.9709 - val_loss: 2.4102 - val_accuracy: 0.5023
Epoch 23/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0582 - accuracy:
0.9840 - val_loss: 2.6160 - val_accuracy: 0.5046
Epoch 24/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0477 - accuracy:
0.9889 - val_loss: 2.6234 - val_accuracy: 0.5255
Epoch 25/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0253 - accuracy:
0.9946 - val_loss: 2.7769 - val_accuracy: 0.4954
Epoch 26/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0256 - accuracy:
0.9943 - val_loss: 2.8342 - val_accuracy: 0.5185
Epoch 27/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0236 - accuracy:
0.9943 - val_loss: 2.9119 - val_accuracy: 0.5417
Epoch 28/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0202 - accuracy:
0.9959 - val_loss: 2.7910 - val_accuracy: 0.5394
Epoch 29/30
```

```
31/31 [==============================] - 1s 25ms/step - loss: 0.0187 - accuracy:
0.9954 - val_loss: 3.0893 - val_accuracy: 0.5069
Epoch 30/30
31/31 [==============================] - 1s 25ms/step - loss: 0.0200 - accuracy:
0.9936 - val_loss: 3.2019 - val_accuracy: 0.4907
14/14 [==============================] - 0s 6ms/step - loss: 3.2018 - accuracy:
0.4907
14/14 [==============================] - 0s 6ms/step
```





	daisy	dandelion	rose	sunflower	tulip
daisy	33	16	10	15	11
dandelion	7	70	16	7	17
rose	4	14	31	13	18
sunflower	2	7	9	32	9
tulip	7	11	15	12	46

```
Results for Model3_fc_3_leaky_relu:
Configuration: Model3_fc_3_leaky_relu
Test Accuracy: 0.4907
F1 Score: 0.4910
Training Time: 24.46 seconds
Number of Parameters: 1,728,261
```

[77]:

```
headers = results[0].keys()
rows = [
    [
        f"{row['configuration']}",
        f"{row['test_accuracy']:.4f}",
        f"{row['f1_score']:.4f}",
        f"{row['training_time']:.4f}",
        row["parameters"],
```

```python
    ]
    for row in results
]

# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt="pretty"))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x["test_accuracy"])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+------------------------+---------------+----------+---------------+-----------
-+
|     configuration      | test_accuracy | f1_score | training_time | parameters
|
+------------------------+---------------+----------+---------------+-----------
-+
|         Model1         |    0.5556     |  0.5541  |    22.0594    |   55301
|
|         Model2         |    0.5231     |  0.5243  |    20.9627    |   88069
|
|         Model3         |    0.5671     |  0.5669  |    23.0679    |   96261
|
|         Model4         |    0.5185     |  0.5160  |    23.0128    |   96517
|
|      Model3_fc_2       |    0.4931     |  0.4886  |    30.3716    |   884229
|
|      Model3_fc_3       |    0.5772     |  0.5796  |    31.0176    |  1736453
|
|  Model1_fc_3_avg_pool  |    0.4444     |  0.4447  |    21.2291    |  1728261
|
|  Model3_fc_3_sigmoid   |    0.2708     |  0.1154  |    24.5248    |  1728261
|
| Model3_fc_3_leaky_relu |    0.4907     |  0.4910  |    24.4649    |  1728261
|
+------------------------+---------------+----------+---------------+-----------
-+

Best Model Parameters:
```

```
Configuration: Model3_fc_3
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.0176 seconds
Number of Parameters: 1736453
```

ReLU seems to be the better activation function

8.0.4 (e) For the best set of parameters obtained above, varying regularization param

Dropout = 0.25

[78]:

```
model3_fc_3_dropout = Sequential(name="Model3_fc_3_dropout")

model3_fc_3_dropout.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        input_shape=(80, 80, 1),
        activation="relu",
    )
)
model3_fc_3_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_dropout.add(Dropout(0.25))

model3_fc_3_dropout.add(
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu")
)
model3_fc_3_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_dropout.add(Dropout(0.25))

model3_fc_3_dropout.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_3_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_dropout.add(Dropout(0.25))

model3_fc_3_dropout.add(Flatten())

model3_fc_3_dropout.add(Dense(256, activation="relu"))
model3_fc_3_dropout.add(Dense(128, activation="relu"))
model3_fc_3_dropout.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3_dropout.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
  ↪metrics=["accuracy"]
)
```

```
model3_fc_3_dropout.summary()
```

```
Model: "Model3_fc_3_dropout"
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_57 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_54 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_57 (Dropout)        (None, 40, 40, 16)        0

 conv2d_58 (Conv2D)          (None, 40, 40, 32)        4640

 max_pooling2d_55 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_58 (Dropout)        (None, 20, 20, 32)        0

 conv2d_59 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_56 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_59 (Dropout)        (None, 10, 10, 64)        0

 flatten_19 (Flatten)        (None, 6400)              0

 dense_45 (Dense)            (None, 256)               1638656

 dense_46 (Dense)            (None, 128)               32896

 dense_47 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,261
Trainable params: 1,728,261
Non-trainable params: 0
-----------------------------------------------------------------
```

[79]:

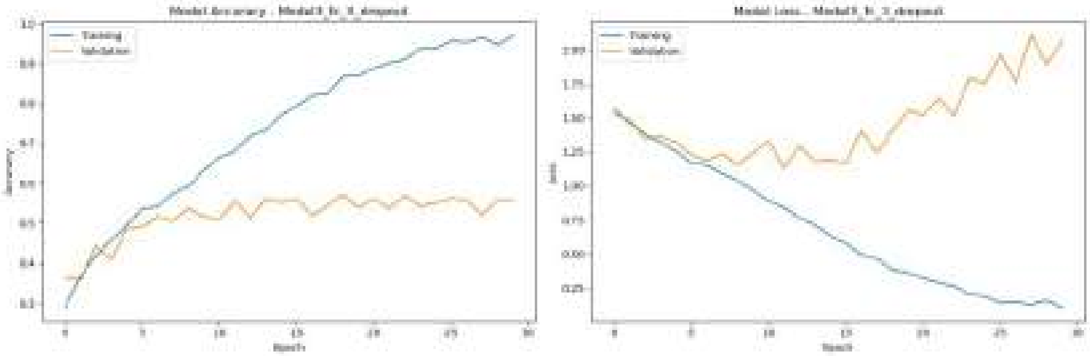
```
history, training_time = train_and_evaluate(
    model3_fc_3_dropout, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model3_fc_3_dropout.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3_dropout.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")
```

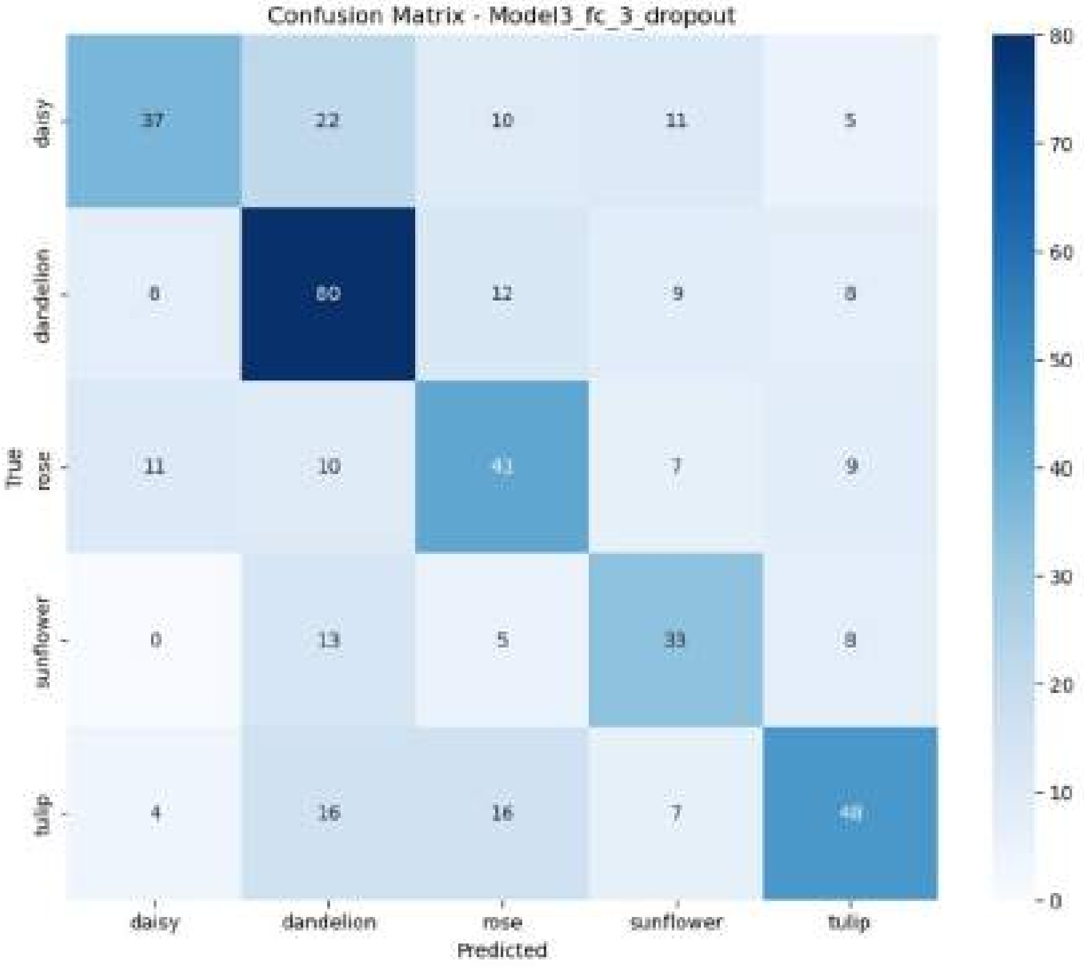
```
results.append(
    {
        "configuration": f"Model3_fc_3_dropout",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3_dropout.count_params(),
    }
)

plot_training_history(history, f"Model3_fc_3_dropout")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_3_dropout")
print("\nResults for Model3_fc_3_dropout:")
print(f"Configuration: {results[9]['configuration']}")
print(f"Test Accuracy: {results[9]['test_accuracy']:.4f}")
print(f"F1 Score: {results[9]['f1_score']:.4f}")
print(f"Training Time: {results[9]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[9]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 31ms/step - loss: 1.5639 - accuracy:
0.2885 - val_loss: 1.5248 - val_accuracy: 0.3634
Epoch 2/30
31/31 [==============================] - 1s 22ms/step - loss: 1.4605 - accuracy:
0.3681 - val_loss: 1.4909 - val_accuracy: 0.3588
Epoch 3/30
31/31 [==============================] - 1s 22ms/step - loss: 1.3804 - accuracy:
0.4185 - val_loss: 1.3475 - val_accuracy: 0.4421
Epoch 4/30
31/31 [==============================] - 1s 21ms/step - loss: 1.3200 - accuracy:
0.4577 - val_loss: 1.3629 - val_accuracy: 0.4097
Epoch 5/30
31/31 [==============================] - 1s 23ms/step - loss: 1.2529 - accuracy:
0.4940 - val_loss: 1.3189 - val_accuracy: 0.4861
Epoch 6/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1695 - accuracy:
0.5364 - val_loss: 1.2271 - val_accuracy: 0.4907
Epoch 7/30
31/31 [==============================] - 1s 21ms/step - loss: 1.1521 - accuracy:
0.5413 - val_loss: 1.1862 - val_accuracy: 0.5162
Epoch 8/30
31/31 [==============================] - 1s 21ms/step - loss: 1.0914 - accuracy:
0.5758 - val_loss: 1.2377 - val_accuracy: 0.5046
Epoch 9/30
31/31 [==============================] - 1s 21ms/step - loss: 1.0393 - accuracy:
0.5918 - val_loss: 1.1529 - val_accuracy: 0.5370
```

```
Epoch 10/30
31/31 [==============================] - 1s 21ms/step - loss: 0.9681 - accuracy:
0.6327 - val_loss: 1.2407 - val_accuracy: 0.5162
Epoch 11/30
31/31 [==============================] - 1s 22ms/step - loss: 0.8883 - accuracy:
0.6638 - val_loss: 1.3270 - val_accuracy: 0.5093
Epoch 12/30
31/31 [==============================] - 1s 22ms/step - loss: 0.8425 - accuracy:
0.6798 - val_loss: 1.1328 - val_accuracy: 0.5579
Epoch 13/30
31/31 [==============================] - 1s 21ms/step - loss: 0.7610 - accuracy:
0.7189 - val_loss: 1.2942 - val_accuracy: 0.5116
Epoch 14/30
31/31 [==============================] - 1s 23ms/step - loss: 0.7124 - accuracy:
0.7323 - val_loss: 1.1774 - val_accuracy: 0.5625
Epoch 15/30
31/31 [==============================] - 1s 22ms/step - loss: 0.6319 - accuracy:
0.7699 - val_loss: 1.1913 - val_accuracy: 0.5509
Epoch 16/30
31/31 [==============================] - 1s 21ms/step - loss: 0.5780 - accuracy:
0.7931 - val_loss: 1.1686 - val_accuracy: 0.5602
Epoch 17/30
31/31 [==============================] - 1s 22ms/step - loss: 0.4945 - accuracy:
0.8190 - val_loss: 1.4023 - val_accuracy: 0.5185
Epoch 18/30
31/31 [==============================] - 1s 22ms/step - loss: 0.4635 - accuracy:
0.8229 - val_loss: 1.2438 - val_accuracy: 0.5440
Epoch 19/30
31/31 [==============================] - 1s 21ms/step - loss: 0.3809 - accuracy:
0.8674 - val_loss: 1.4005 - val_accuracy: 0.5718
Epoch 20/30
31/31 [==============================] - 1s 21ms/step - loss: 0.3626 - accuracy:
0.8692 - val_loss: 1.5576 - val_accuracy: 0.5394
Epoch 21/30
31/31 [==============================] - 1s 22ms/step - loss: 0.3191 - accuracy:
0.8870 - val_loss: 1.5190 - val_accuracy: 0.5602
Epoch 22/30
31/31 [==============================] - 1s 22ms/step - loss: 0.2874 - accuracy:
0.8999 - val_loss: 1.6464 - val_accuracy: 0.5370
Epoch 23/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2603 - accuracy:
0.9091 - val_loss: 1.5169 - val_accuracy: 0.5694
Epoch 24/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2039 - accuracy:
0.9364 - val_loss: 1.7920 - val_accuracy: 0.5417
Epoch 25/30
31/31 [==============================] - 1s 22ms/step - loss: 0.1868 - accuracy:
0.9362 - val_loss: 1.7533 - val_accuracy: 0.5509
```

```
Epoch 26/30
31/31 [==============================] - 1s 22ms/step - loss: 0.1379 - accuracy:
0.9555 - val_loss: 1.9712 - val_accuracy: 0.5625
Epoch 27/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1470 - accuracy:
0.9521 - val_loss: 1.7662 - val_accuracy: 0.5579
Epoch 28/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1240 - accuracy:
0.9645 - val_loss: 2.1082 - val_accuracy: 0.5208
Epoch 29/30
31/31 [==============================] - 1s 23ms/step - loss: 0.1573 - accuracy:
0.9454 - val_loss: 1.9006 - val_accuracy: 0.5579
Epoch 30/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1029 - accuracy:
0.9691 - val_loss: 2.0693 - val_accuracy: 0.5579
14/14 [==============================] - 0s 4ms/step - loss: 2.0693 - accuracy:
0.5579
14/14 [==============================] - 0s 7ms/step
```





```
Results for Model3_fc_3_dropout:
Configuration: Model3_fc_3_dropout
Test Accuracy: 0.5579
F1 Score: 0.5555
Training Time: 21.69 seconds
Number of Parameters: 1,728,261
```

BatchNormalization after each layer except first

[80]:
```
model3_fc_3_batchnorm = Sequential(name="Model3_fc_3_batchnorm")

model3_fc_3_batchnorm.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
```

```
        input_shape=(80, 80, 1),
        activation="relu",
    )
)
model3_fc_3_batchnorm.add(MaxPooling2D(pool_size=(2, 2)))


model3_fc_3_batchnorm.add(
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu")
)
model3_fc_3_batchnorm.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_batchnorm.add(BatchNormalization())

model3_fc_3_batchnorm.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model3_fc_3_batchnorm.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3_batchnorm.add(BatchNormalization())

model3_fc_3_batchnorm.add(Flatten())

model3_fc_3_batchnorm.add(Dense(256, activation="relu"))
model3_fc_3_batchnorm.add(Dense(128, activation="relu"))
model3_fc_3_batchnorm.add(Dense(NUM_CLASSES, activation="softmax"))

model3_fc_3_batchnorm.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
)
model3_fc_3_batchnorm.summary()
```

```
Model: "Model3_fc_3_batchnorm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_60 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_57 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 conv2d_61 (Conv2D)          (None, 40, 40, 32)        4640

 max_pooling2d_58 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 batch_normalization (BatchN  (None, 20, 20, 32)       128
 ormalization)
```

```
 conv2d_62 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_59 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 batch_normalization_1 (Batc  (None, 10, 10, 64)       256
 hNormalization)

 flatten_20 (Flatten)        (None, 6400)              0

 dense_48 (Dense)            (None, 256)               1638656

 dense_49 (Dense)            (None, 128)               32896

 dense_50 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,645
Trainable params: 1,728,453
Non-trainable params: 192
_________________________________________________________________
```

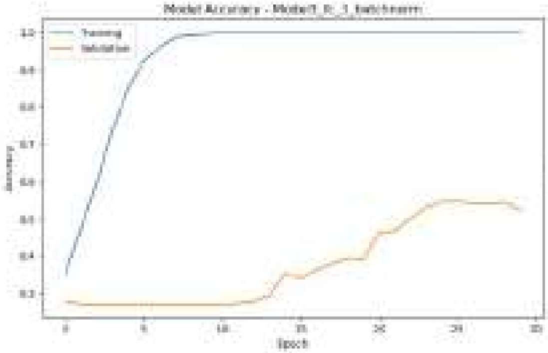
[81]:

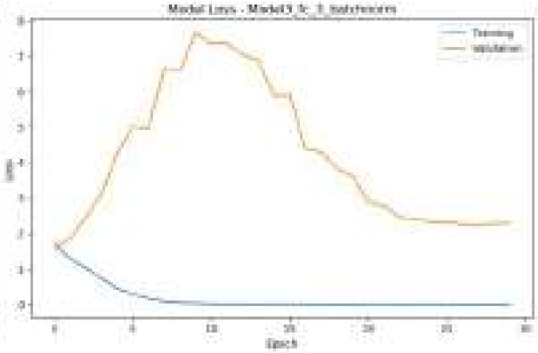
```python
history, training_time = train_and_evaluate(
    model3_fc_3_batchnorm, X_train, y_train, X_test, y_test, batch_size=128
)
test_loss, test_acc = model3_fc_3_batchnorm.evaluate(X_test, y_test)
y_pred = np.argmax(model3_fc_3_batchnorm.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model3_fc_3_batchnorm",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model3_fc_3_batchnorm.count_params(),
    }
)

plot_training_history(history, f"Model3_fc_3_batchnorm")
plot_confusion_matrix(y_test, y_pred, class_names, f"Model3_fc_3_batchnorm")
print("\nResults for Model3_fc_3_batchnorm:")
print(f"Configuration: {results[10]['configuration']}")
print(f"Test Accuracy: {results[10]['test_accuracy']:.4f}")
print(f"F1 Score: {results[10]['f1_score']:.4f}")
print(f"Training Time: {results[10]['training_time']:.2f} seconds")
```

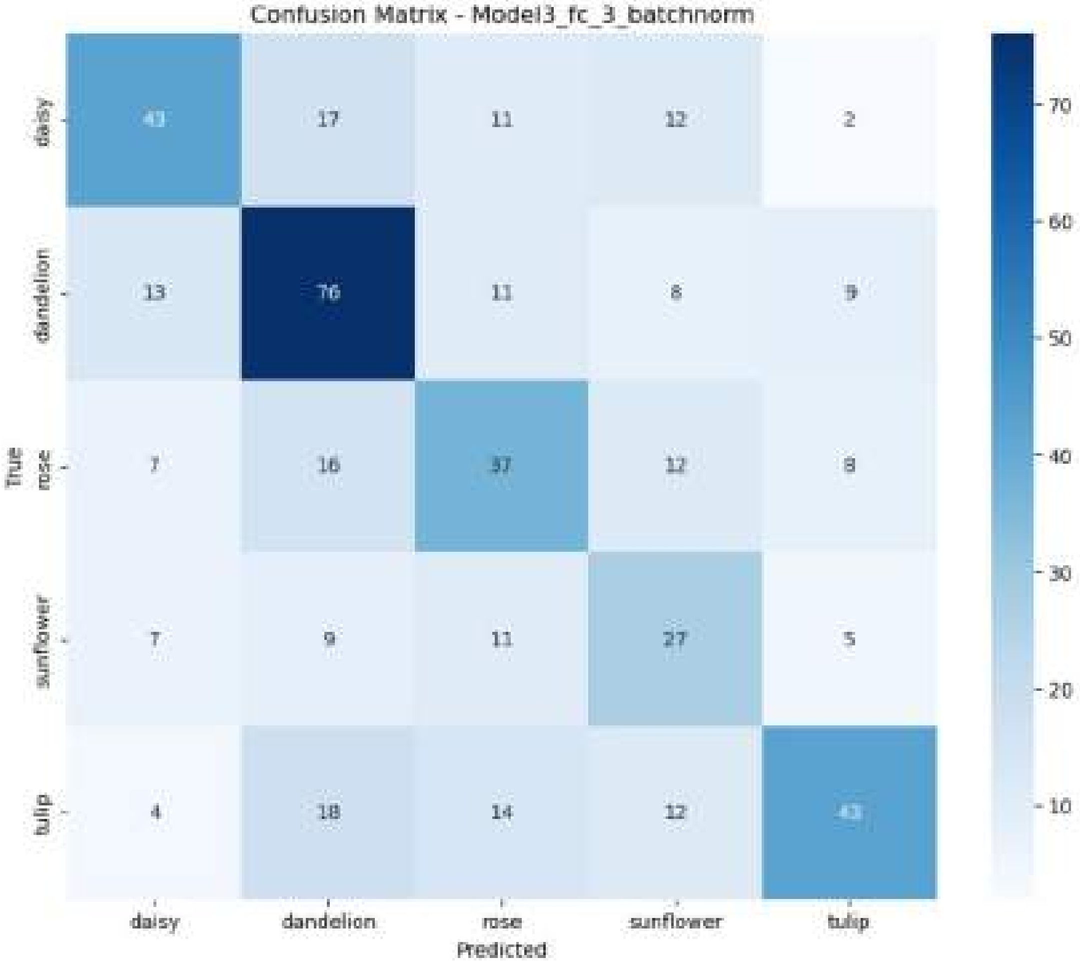
```python
print(f"Number of Parameters: {results[10]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 32ms/step - loss: 1.7471 - accuracy:
0.3524 - val_loss: 1.5727 - val_accuracy: 0.2778
Epoch 2/30
31/31 [==============================] - 1s 20ms/step - loss: 1.2834 - accuracy:
0.4770 - val_loss: 1.8517 - val_accuracy: 0.2708
Epoch 3/30
31/31 [==============================] - 1s 21ms/step - loss: 1.0311 - accuracy:
0.5967 - val_loss: 2.4507 - val_accuracy: 0.2708
Epoch 4/30
31/31 [==============================] - 1s 20ms/step - loss: 0.7270 - accuracy:
0.7382 - val_loss: 3.1063 - val_accuracy: 0.2708
Epoch 5/30
31/31 [==============================] - 1s 20ms/step - loss: 0.4557 - accuracy:
0.8505 - val_loss: 4.2454 - val_accuracy: 0.2708
Epoch 6/30
31/31 [==============================] - 1s 23ms/step - loss: 0.2819 - accuracy:
0.9230 - val_loss: 5.0217 - val_accuracy: 0.2708
Epoch 7/30
31/31 [==============================] - 1s 20ms/step - loss: 0.1621 - accuracy:
0.9598 - val_loss: 4.9282 - val_accuracy: 0.2708
Epoch 8/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0764 - accuracy:
0.9864 - val_loss: 6.6667 - val_accuracy: 0.2708
Epoch 9/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0393 - accuracy:
0.9928 - val_loss: 6.6081 - val_accuracy: 0.2708
Epoch 10/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0258 - accuracy:
0.9954 - val_loss: 7.6644 - val_accuracy: 0.2708
Epoch 11/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0126 - accuracy:
0.9987 - val_loss: 7.3615 - val_accuracy: 0.2708
Epoch 12/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0098 - accuracy:
0.9985 - val_loss: 7.3724 - val_accuracy: 0.2731
Epoch 13/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0072 - accuracy:
0.9990 - val_loss: 7.0305 - val_accuracy: 0.2778
Epoch 14/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0046 - accuracy:
0.9990 - val_loss: 6.8818 - val_accuracy: 0.2940
Epoch 15/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0078 - accuracy:
0.9985 - val_loss: 5.8775 - val_accuracy: 0.3542
```

```
Epoch 16/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0084 - accuracy:
0.9985 - val_loss: 5.9023 - val_accuracy: 0.3403
Epoch 17/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0045 - accuracy:
0.9992 - val_loss: 4.3920 - val_accuracy: 0.3634
Epoch 18/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0050 - accuracy:
0.9990 - val_loss: 4.2909 - val_accuracy: 0.3796
Epoch 19/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0042 - accuracy:
0.9990 - val_loss: 3.8198 - val_accuracy: 0.3935
Epoch 20/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0056 - accuracy:
0.9987 - val_loss: 3.6123 - val_accuracy: 0.3912
Epoch 21/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0032 - accuracy:
0.9990 - val_loss: 2.8745 - val_accuracy: 0.4630
Epoch 22/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0030 - accuracy:
0.9992 - val_loss: 2.7692 - val_accuracy: 0.4676
Epoch 23/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0031 - accuracy:
0.9987 - val_loss: 2.4613 - val_accuracy: 0.5023
Epoch 24/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0028 - accuracy:
0.9990 - val_loss: 2.3984 - val_accuracy: 0.5301
Epoch 25/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0030 - accuracy:
0.9987 - val_loss: 2.3175 - val_accuracy: 0.5463
Epoch 26/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0028 - accuracy:
0.9990 - val_loss: 2.3304 - val_accuracy: 0.5463
Epoch 27/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0027 - accuracy:
0.9987 - val_loss: 2.2504 - val_accuracy: 0.5417
Epoch 28/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0025 - accuracy:
0.9990 - val_loss: 2.2483 - val_accuracy: 0.5394
Epoch 29/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0019 - accuracy:
0.9990 - val_loss: 2.2781 - val_accuracy: 0.5440
Epoch 30/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0024 - accuracy:
0.9990 - val_loss: 2.2841 - val_accuracy: 0.5231
14/14 [==============================] - 0s 5ms/step - loss: 2.2841 - accuracy:
0.5231
14/14 [==============================] - 0s 5ms/step
```







	daisy	dandelion	rose	sunflower	tulip
daisy	43	17	11	12	2
dandelion	13	76	11	8	9
rose	7	16	37	12	8
sunflower	7	9	11	27	5
tulip	4	18	14	12	43

```
Results for Model3_fc_3_batchnorm:
Configuration: Model3_fc_3_batchnorm
```

```
Test Accuracy: 0.5231
F1 Score: 0.5241
Training Time: 20.86 seconds
Number of Parameters: 1,728,645
```

Dropout of 0.1 after each layer and BatchNormalization after each layer except first

```
[82]: model1_fc_3_batchnorm_dropout = Sequential(name="Model1_fc_3_batchnorm_dropout")

model1_fc_3_batchnorm_dropout.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        padding="same",
        input_shape=(80, 80, 1),
        activation="relu",
    )
)
model1_fc_3_batchnorm_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model1_fc_3_batchnorm_dropout.add(Dropout(0.1))

model1_fc_3_batchnorm_dropout.add(
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu")
)
model1_fc_3_batchnorm_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model1_fc_3_batchnorm_dropout.add(BatchNormalization())
model1_fc_3_batchnorm_dropout.add(Dropout(0.1))

model1_fc_3_batchnorm_dropout.add(
    Conv2D(filters=64, kernel_size=(5, 5), padding="same", activation="relu")
)
model1_fc_3_batchnorm_dropout.add(MaxPooling2D(pool_size=(2, 2)))
model1_fc_3_batchnorm_dropout.add(BatchNormalization())
model1_fc_3_batchnorm_dropout.add(Dropout(0.1))

model1_fc_3_batchnorm_dropout.add(Flatten())

model1_fc_3_batchnorm_dropout.add(Dense(256, activation="relu"))
model1_fc_3_batchnorm_dropout.add(Dense(128, activation="relu"))
model1_fc_3_batchnorm_dropout.add(Dense(NUM_CLASSES, activation="softmax"))

model1_fc_3_batchnorm_dropout.compile(
    optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
  ↪metrics=["accuracy"]
)
model1_fc_3_batchnorm_dropout.summary()
```

```
Model: "Model1_fc_3_batchnorm_dropout"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_63 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_60 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_60 (Dropout)        (None, 40, 40, 16)        0

 conv2d_64 (Conv2D)          (None, 40, 40, 32)        4640

 max_pooling2d_61 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 batch_normalization_2 (Batc  (None, 20, 20, 32)       128
 hNormalization)

 dropout_61 (Dropout)        (None, 20, 20, 32)        0

 conv2d_65 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_62 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 batch_normalization_3 (Batc  (None, 10, 10, 64)       256
 hNormalization)

 dropout_62 (Dropout)        (None, 10, 10, 64)        0

 flatten_21 (Flatten)        (None, 6400)              0

 dense_51 (Dense)            (None, 256)               1638656

 dense_52 (Dense)            (None, 128)               32896

 dense_53 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,728,645
Trainable params: 1,728,453
Non-trainable params: 192
_________________________________________________________________
```

[83]:

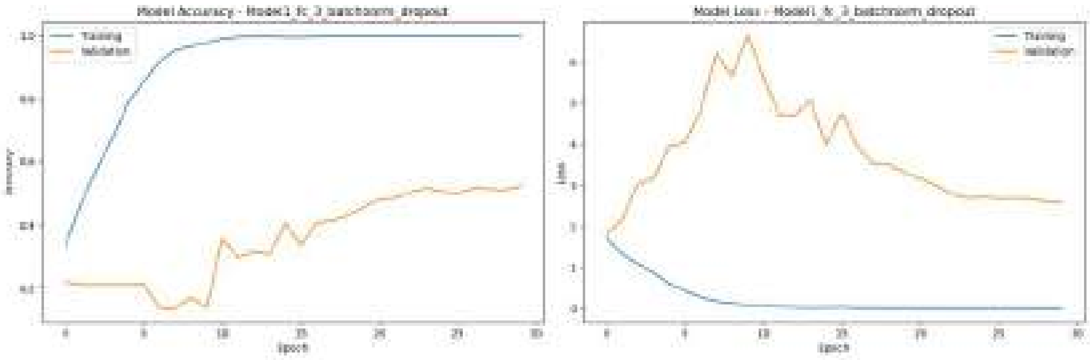
```
history, training_time = train_and_evaluate(
    model1_fc_3_batchnorm_dropout, X_train, y_train, X_test, y_test,␣
 ↪batch_size=128
```

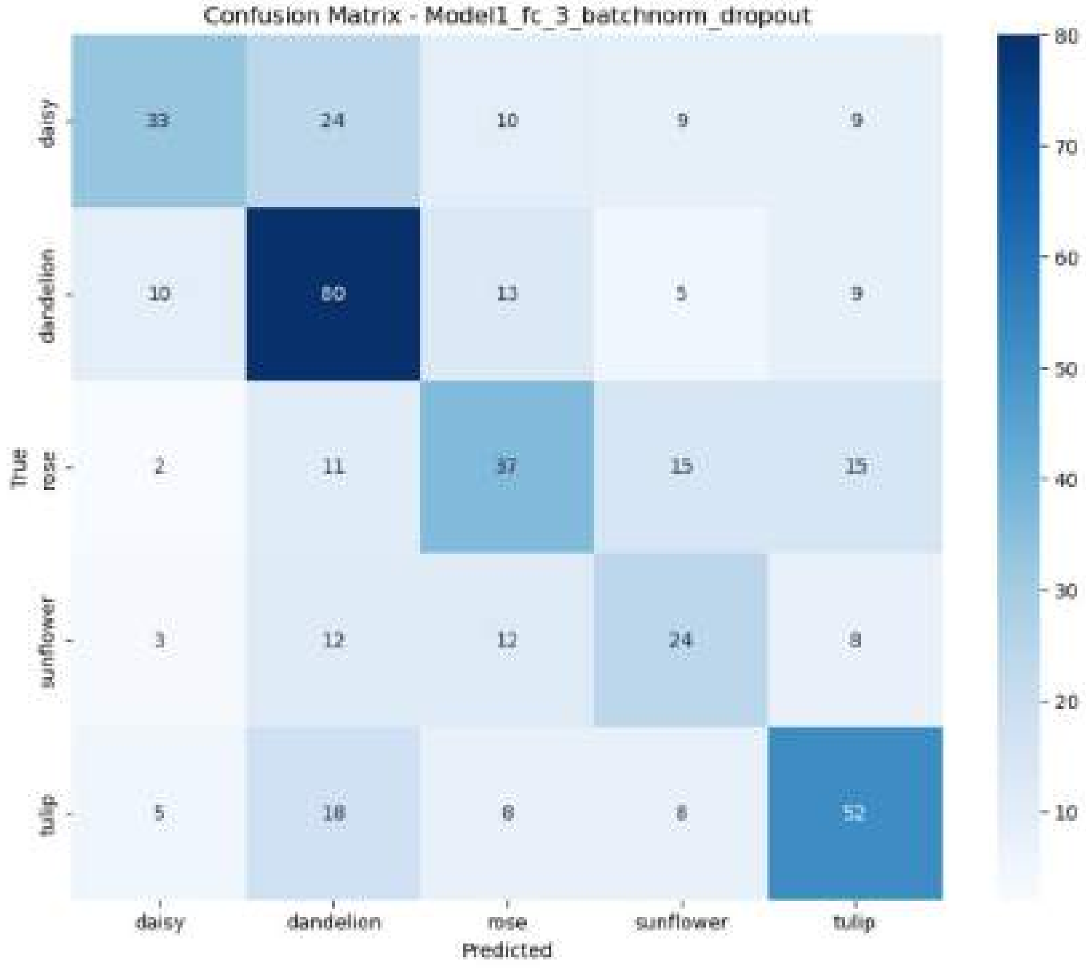
```
)
test_loss, test_acc = model1_fc_3_batchnorm_dropout.evaluate(X_test, y_test)
y_pred = np.argmax(model1_fc_3_batchnorm_dropout.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred, average="weighted")

results.append(
    {
        "configuration": f"Model1_fc_3_batchnorm_dropout",
        "test_accuracy": test_acc,
        "f1_score": f1,
        "training_time": training_time,
        "parameters": model1_fc_3_batchnorm_dropout.count_params(),
    }
)

plot_training_history(history, f"Model1_fc_3_batchnorm_dropout")
plot_confusion_matrix(y_test, y_pred, class_names,␣
 ↪f"Model1_fc_3_batchnorm_dropout")
print("\nResults for Model1_fc_3_batchnorm_dropout:")
print(f"Configuration: {results[11]['configuration']}")
print(f"Test Accuracy: {results[11]['test_accuracy']:.4f}")
print(f"F1 Score: {results[11]['f1_score']:.4f}")
print(f"Training Time: {results[11]['training_time']:.2f} seconds")
print(f"Number of Parameters: {results[11]['parameters']:,}")
```

```
Epoch 1/30
31/31 [==============================] - 2s 35ms/step - loss: 1.7373 - accuracy:
0.3346 - val_loss: 1.8075 - val_accuracy: 0.2199
Epoch 2/30
31/31 [==============================] - 1s 22ms/step - loss: 1.3233 - accuracy:
0.4690 - val_loss: 2.1702 - val_accuracy: 0.2106
Epoch 3/30
31/31 [==============================] - 1s 22ms/step - loss: 1.0926 - accuracy:
0.5773 - val_loss: 3.0314 - val_accuracy: 0.2106
Epoch 4/30
31/31 [==============================] - 1s 22ms/step - loss: 0.8671 - accuracy:
0.6749 - val_loss: 3.1754 - val_accuracy: 0.2106
Epoch 5/30
31/31 [==============================] - 1s 22ms/step - loss: 0.6027 - accuracy:
0.7882 - val_loss: 3.9454 - val_accuracy: 0.2106
Epoch 6/30
31/31 [==============================] - 1s 22ms/step - loss: 0.4327 - accuracy:
0.8543 - val_loss: 4.0347 - val_accuracy: 0.2130
Epoch 7/30
31/31 [==============================] - 1s 22ms/step - loss: 0.2764 - accuracy:
0.9179 - val_loss: 4.8202 - val_accuracy: 0.1366
Epoch 8/30
```

```
31/31 [==============================] - 1s 22ms/step - loss: 0.1796 - accuracy:
0.9532 - val_loss: 6.2193 - val_accuracy: 0.1366
Epoch 9/30
31/31 [==============================] - 1s 22ms/step - loss: 0.1238 - accuracy:
0.9647 - val_loss: 5.6638 - val_accuracy: 0.1713
Epoch 10/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0896 - accuracy:
0.9773 - val_loss: 6.6504 - val_accuracy: 0.1389
Epoch 11/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0576 - accuracy:
0.9897 - val_loss: 5.6453 - val_accuracy: 0.3565
Epoch 12/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0371 - accuracy:
0.9956 - val_loss: 4.7257 - val_accuracy: 0.2963
Epoch 13/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0254 - accuracy:
0.9972 - val_loss: 4.7069 - val_accuracy: 0.3171
Epoch 14/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0154 - accuracy:
0.9985 - val_loss: 5.0723 - val_accuracy: 0.3056
Epoch 15/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0202 - accuracy:
0.9956 - val_loss: 3.9918 - val_accuracy: 0.4051
Epoch 16/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0263 - accuracy:
0.9951 - val_loss: 4.7448 - val_accuracy: 0.3380
Epoch 17/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0143 - accuracy:
0.9972 - val_loss: 3.9449 - val_accuracy: 0.4074
Epoch 18/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0109 - accuracy:
0.9990 - val_loss: 3.5216 - val_accuracy: 0.4144
Epoch 19/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0072 - accuracy:
0.9990 - val_loss: 3.5095 - val_accuracy: 0.4306
Epoch 20/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0096 - accuracy:
0.9987 - val_loss: 3.3118 - val_accuracy: 0.4583
Epoch 21/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0060 - accuracy:
0.9990 - val_loss: 3.1616 - val_accuracy: 0.4815
Epoch 22/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0066 - accuracy:
0.9992 - val_loss: 2.9975 - val_accuracy: 0.4861
Epoch 23/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0078 - accuracy:
0.9987 - val_loss: 2.7896 - val_accuracy: 0.5046
Epoch 24/30
```

```
31/31 [==============================] - 1s 22ms/step - loss: 0.0051 - accuracy:
0.9990 - val_loss: 2.6926 - val_accuracy: 0.5162
Epoch 25/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0057 - accuracy:
0.9990 - val_loss: 2.7151 - val_accuracy: 0.5069
Epoch 26/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0054 - accuracy:
0.9990 - val_loss: 2.6837 - val_accuracy: 0.5000
Epoch 27/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0056 - accuracy:
0.9987 - val_loss: 2.6891 - val_accuracy: 0.5162
Epoch 28/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0045 - accuracy:
0.9990 - val_loss: 2.6806 - val_accuracy: 0.5139
Epoch 29/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0038 - accuracy:
0.9990 - val_loss: 2.6136 - val_accuracy: 0.5116
Epoch 30/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0090 - accuracy:
0.9982 - val_loss: 2.5911 - val_accuracy: 0.5231
14/14 [==============================] - 0s 6ms/step - loss: 2.5912 - accuracy:
0.5231
14/14 [==============================] - 0s 4ms/step
```





```
Results for Model1_fc_3_batchnorm_dropout:
Configuration: Model1_fc_3_batchnorm_dropout
Test Accuracy: 0.5231
F1 Score: 0.5188
Training Time: 22.76 seconds
Number of Parameters: 1,728,645
```

[84]:

```
headers = results[0].keys()
rows = [
    [
        f"{row['configuration']}",
        f"{row['test_accuracy']:.4f}",
        f"{row['f1_score']:.4f}",
        f"{row['training_time']:.4f}",
        row["parameters"],
```

```
    ]
    for row in results
]

# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt="pretty"))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x["test_accuracy"])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+-------------------------------+---------------+----------+---------------+----
--------+
|         configuration         | test_accuracy | f1_score | training_time |
parameters |
+-------------------------------+---------------+----------+---------------+----
--------+
|            Model1             |    0.5556     |  0.5541  |    22.0594    |
55301    |
|            Model2             |    0.5231     |  0.5243  |    20.9627    |
88069    |
|            Model3             |    0.5671     |  0.5669  |    23.0679    |
96261    |
|            Model4             |    0.5185     |  0.5160  |    23.0128    |
96517    |
|          Model3_fc_2          |    0.4931     |  0.4886  |    30.3716    |
884229   |
|          Model3_fc_3          |    0.5772     |  0.5796  |    31.0176    |
1736453   |
|     Model1_fc_3_avg_pool      |    0.4444     |  0.4447  |    21.2291    |
1728261   |
|      Model3_fc_3_sigmoid      |    0.2708     |  0.1154  |    24.5248    |
1728261   |
|    Model3_fc_3_leaky_relu     |    0.4907     |  0.4910  |    24.4649    |
1728261   |
|      Model3_fc_3_dropout      |    0.5579     |  0.5555  |    21.6860    |
1728261   |
|     Model3_fc_3_batchnorm     |    0.5231     |  0.5241  |    20.8621    |
1728645   |
```

```
| Model1_fc_3_batchnorm_dropout |    0.5231     | 0.5188 |    22.7609    |
1728645   |
+-------------------------------+---------------+--------+---------------+----
---------+

Best Model Parameters:
Configuration: Model3_fc_3
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.0176 seconds
Number of Parameters: 1736453
```

8.0.5 (f) For the best set of parameters obtained above, adding 1, 2, 3 conv layers

[85]:

```python
models = []

for num_conv_layers in range(1, 4):
    model = Sequential(name=f"Model3_fc_3_conv{num_conv_layers}")

    # First conv layer
    model.add(
        Conv2D(
            filters=16,
            kernel_size=(3, 3),
            padding="same",
            activation="relu",
            input_shape=(80, 80, 1),
        )
    )
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.1))

    # Add additional conv layers based on loop counter
    for i in range(num_conv_layers - 1):
        filters = 32 * (2**i)  # Double filters each layer: 32, 64
        model.add(
            Conv2D(
                filters=filters, kernel_size=(5, 5), padding="same",␣
↪activation="relu"
            )
        )
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.1))

    model.add(Flatten())

    # Dense layers
```

```
    model.add(Dense(256, activation="relu"))
    model.add(Dense(128, activation="relu"))
    model.add(Dense(NUM_CLASSES, activation="softmax"))

    model.compile(
        optimizer=Adam(), loss="sparse_categorical_crossentropy",␣
 ↪metrics=["accuracy"]
    )
    model.summary()

    models.append(model)

model3_fc_3_conv1, model3_fc_3_conv2, model3_fc_3_conv3 = models

for model in models:
    history, training_time = train_and_evaluate(model, X_train, y_train,␣
 ↪X_test, y_test, batch_size=128)
    test_loss, test_acc = model.evaluate(X_test, y_test)
    y_pred = np.argmax(model.predict(X_test), axis=1)
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        'configuration': model.name,
        'test_accuracy': test_acc,
        'f1_score': f1,
        'training_time': training_time,
        'parameters': model.count_params()
    })

    plot_training_history(history, model.name)
    plot_confusion_matrix(y_test, y_pred, class_names, model.name)
    print(f"\nResults for {model.name}:")
    print(f"Configuration: {results[-1]['configuration']}")
    print(f"Test Accuracy: {results[-1]['test_accuracy']:.4f}")
    print(f"F1 Score: {results[-1]['f1_score']:.4f}")
    print(f"Training Time: {results[-1]['training_time']:.2f} seconds")
    print(f"Number of Parameters: {results[-1]['parameters']:,}")
```

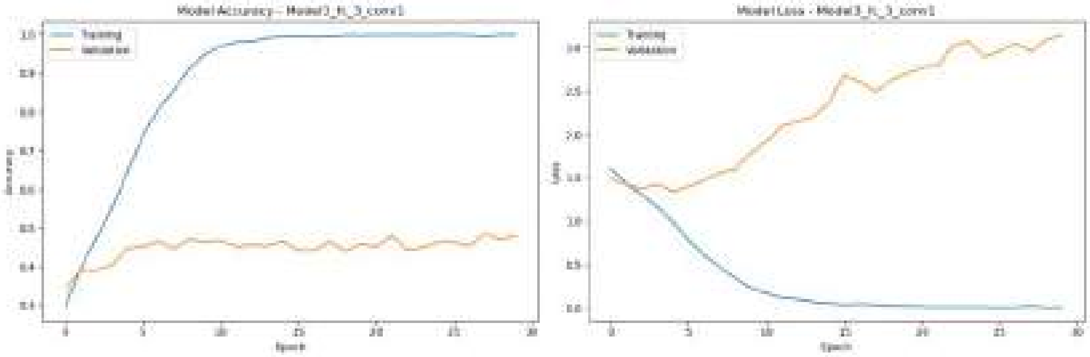
```
Model: "Model3_fc_3_conv1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_66 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_63 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)
```

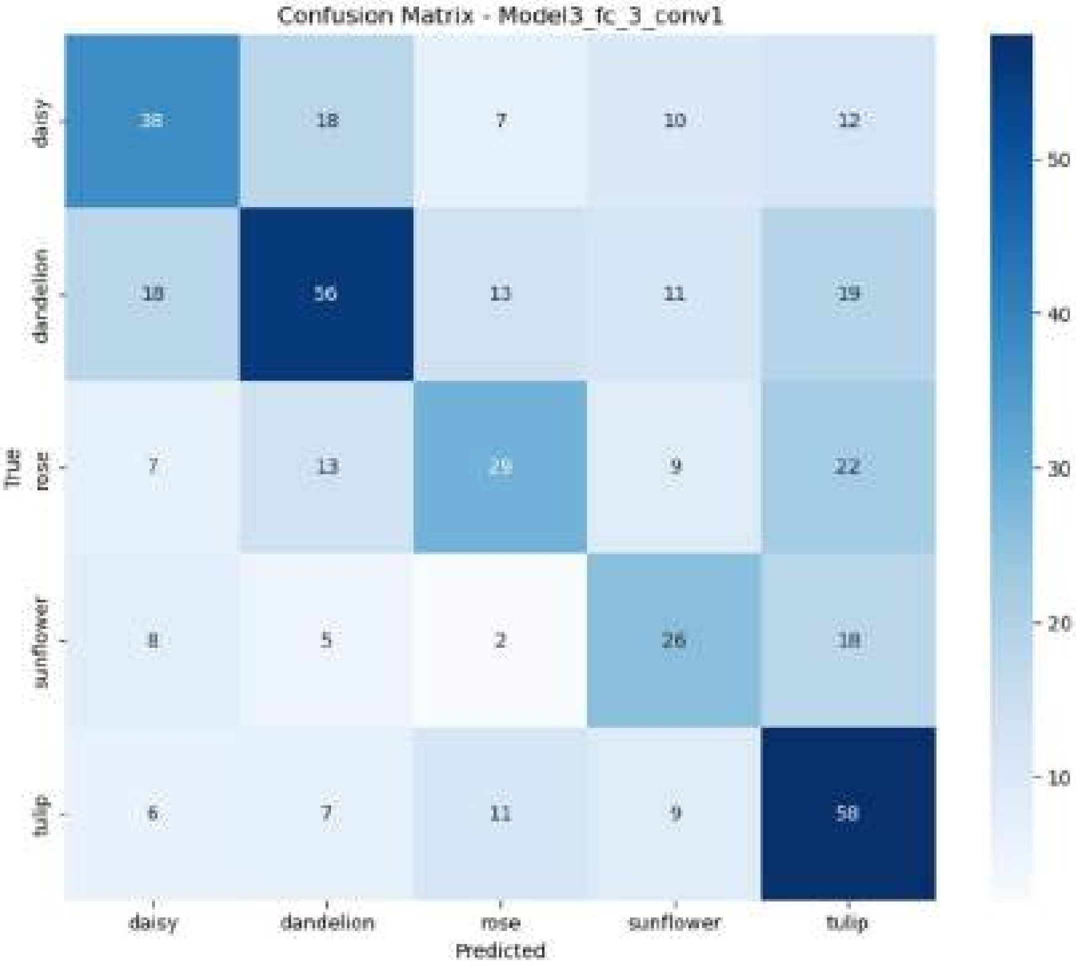
```
 dropout_63 (Dropout)        (None, 40, 40, 16)        0

 flatten_22 (Flatten)        (None, 25600)             0

 dense_54 (Dense)            (None, 256)               6553856

 dense_55 (Dense)            (None, 128)               32896

 dense_56 (Dense)            (None, 5)                 645

=================================================================
Total params: 6,587,557
Trainable params: 6,587,557
Non-trainable params: 0
_________________________________________________________________
Model: "Model3_fc_3_conv2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_67 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_64 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_64 (Dropout)        (None, 40, 40, 16)        0

 conv2d_68 (Conv2D)          (None, 40, 40, 32)        12832

 max_pooling2d_65 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_65 (Dropout)        (None, 20, 20, 32)        0

 flatten_23 (Flatten)        (None, 12800)             0

 dense_57 (Dense)            (None, 256)               3277056

 dense_58 (Dense)            (None, 128)               32896

 dense_59 (Dense)            (None, 5)                 645

=================================================================
Total params: 3,323,589
Trainable params: 3,323,589
Non-trainable params: 0
_________________________________________________________________
Model: "Model3_fc_3_conv3"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_69 (Conv2D)          (None, 80, 80, 16)        160

 max_pooling2d_66 (MaxPoolin  (None, 40, 40, 16)       0
 g2D)

 dropout_66 (Dropout)        (None, 40, 40, 16)        0

 conv2d_70 (Conv2D)          (None, 40, 40, 32)        12832

 max_pooling2d_67 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 dropout_67 (Dropout)        (None, 20, 20, 32)        0

 conv2d_71 (Conv2D)          (None, 20, 20, 64)        51264

 max_pooling2d_68 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 dropout_68 (Dropout)        (None, 10, 10, 64)        0

 flatten_24 (Flatten)        (None, 6400)              0

 dense_60 (Dense)            (None, 256)               1638656

 dense_61 (Dense)            (None, 128)               32896

 dense_62 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,736,453
Trainable params: 1,736,453
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
31/31 [==============================] - 2s 30ms/step - loss: 1.6072 - accuracy:
0.2950 - val_loss: 1.4899 - val_accuracy: 0.3403
Epoch 2/30
31/31 [==============================] - 0s 15ms/step - loss: 1.4317 - accuracy:
0.3951 - val_loss: 1.4228 - val_accuracy: 0.3912
Epoch 3/30
31/31 [==============================] - 0s 16ms/step - loss: 1.3036 - accuracy:
0.4734 - val_loss: 1.3792 - val_accuracy: 0.3912
Epoch 4/30
31/31 [==============================] - 0s 16ms/step - loss: 1.1612 - accuracy:
0.5501 - val_loss: 1.4265 - val_accuracy: 0.4028
```

```
Epoch 5/30
31/31 [==============================] - 1s 16ms/step - loss: 0.9858 - accuracy:
0.6440 - val_loss: 1.3427 - val_accuracy: 0.4468
Epoch 6/30
31/31 [==============================] - 0s 16ms/step - loss: 0.7796 - accuracy:
0.7377 - val_loss: 1.4014 - val_accuracy: 0.4514
Epoch 7/30
31/31 [==============================] - 0s 16ms/step - loss: 0.6059 - accuracy:
0.8077 - val_loss: 1.4725 - val_accuracy: 0.4653
Epoch 8/30
31/31 [==============================] - 0s 16ms/step - loss: 0.4701 - accuracy:
0.8561 - val_loss: 1.5641 - val_accuracy: 0.4468
Epoch 9/30
31/31 [==============================] - 0s 16ms/step - loss: 0.3431 - accuracy:
0.9125 - val_loss: 1.6025 - val_accuracy: 0.4699
Epoch 10/30
31/31 [==============================] - 0s 15ms/step - loss: 0.2308 - accuracy:
0.9485 - val_loss: 1.7896 - val_accuracy: 0.4630
Epoch 11/30
31/31 [==============================] - 0s 15ms/step - loss: 0.1713 - accuracy:
0.9699 - val_loss: 1.9193 - val_accuracy: 0.4676
Epoch 12/30
31/31 [==============================] - 0s 15ms/step - loss: 0.1226 - accuracy:
0.9804 - val_loss: 2.0984 - val_accuracy: 0.4514
Epoch 13/30
31/31 [==============================] - 0s 16ms/step - loss: 0.1048 - accuracy:
0.9815 - val_loss: 2.1487 - val_accuracy: 0.4560
Epoch 14/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0710 - accuracy:
0.9915 - val_loss: 2.2030 - val_accuracy: 0.4537
Epoch 15/30
31/31 [==============================] - 1s 17ms/step - loss: 0.0550 - accuracy:
0.9946 - val_loss: 2.3611 - val_accuracy: 0.4653
Epoch 16/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0408 - accuracy:
0.9961 - val_loss: 2.6803 - val_accuracy: 0.4421
Epoch 17/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0474 - accuracy:
0.9941 - val_loss: 2.6141 - val_accuracy: 0.4421
Epoch 18/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0413 - accuracy:
0.9951 - val_loss: 2.4899 - val_accuracy: 0.4653
Epoch 19/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0241 - accuracy:
0.9982 - val_loss: 2.6281 - val_accuracy: 0.4398
Epoch 20/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0238 - accuracy:
0.9979 - val_loss: 2.7010 - val_accuracy: 0.4583
```

```
Epoch 21/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0175 - accuracy:
0.9985 - val_loss: 2.7712 - val_accuracy: 0.4514
Epoch 22/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0150 - accuracy:
0.9987 - val_loss: 2.7905 - val_accuracy: 0.4792
Epoch 23/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0152 - accuracy:
0.9987 - val_loss: 3.0273 - val_accuracy: 0.4444
Epoch 24/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0163 - accuracy:
0.9985 - val_loss: 3.0711 - val_accuracy: 0.4491
Epoch 25/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0153 - accuracy:
0.9985 - val_loss: 2.8950 - val_accuracy: 0.4630
Epoch 26/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0119 - accuracy:
0.9990 - val_loss: 2.9587 - val_accuracy: 0.4630
Epoch 27/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0143 - accuracy:
0.9982 - val_loss: 3.0551 - val_accuracy: 0.4537
Epoch 28/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0226 - accuracy:
0.9959 - val_loss: 2.9552 - val_accuracy: 0.4838
Epoch 29/30
31/31 [==============================] - 0s 16ms/step - loss: 0.0102 - accuracy:
0.9990 - val_loss: 3.0943 - val_accuracy: 0.4699
Epoch 30/30
31/31 [==============================] - 0s 15ms/step - loss: 0.0119 - accuracy:
0.9987 - val_loss: 3.1488 - val_accuracy: 0.4792
14/14 [==============================] - 0s 5ms/step - loss: 3.1488 - accuracy:
0.4792
14/14 [==============================] - 0s 3ms/step
```

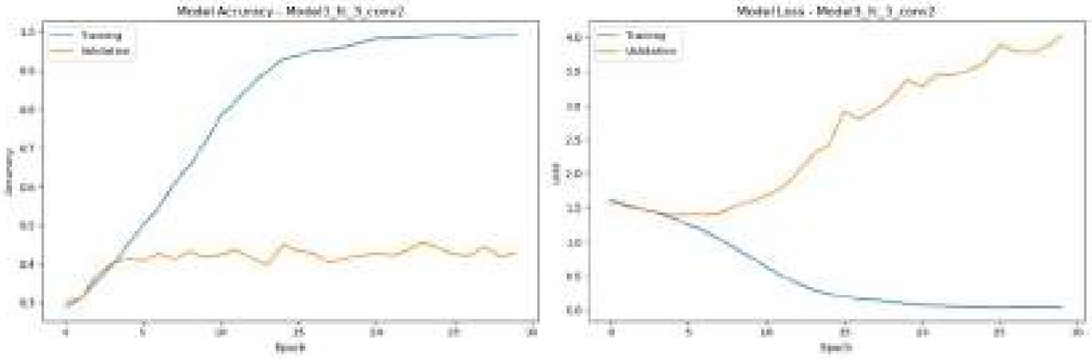


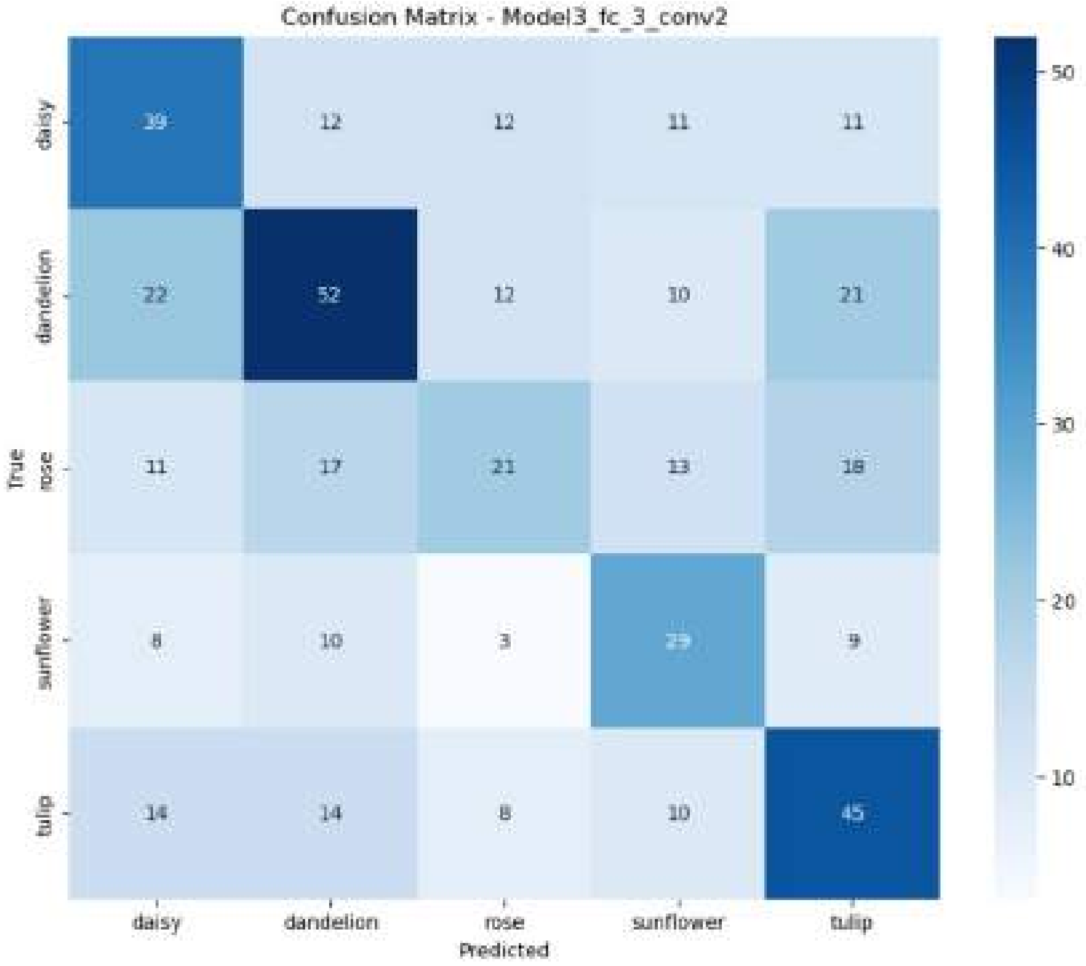


```
Results for Model3_fc_3_conv1:
Configuration: Model3_fc_3_conv1
Test Accuracy: 0.4792
F1 Score: 0.4767
Training Time: 16.17 seconds
Number of Parameters: 6,587,557
Epoch 1/30
31/31 [==============================] - 2s 32ms/step - loss: 1.6258 - accuracy:
0.2880 - val_loss: 1.5853 - val_accuracy: 0.3032
Epoch 2/30
31/31 [==============================] - 1s 21ms/step - loss: 1.5330 - accuracy:
0.3112 - val_loss: 1.5138 - val_accuracy: 0.3102
Epoch 3/30
31/31 [==============================] - 1s 20ms/step - loss: 1.4772 - accuracy:
0.3555 - val_loss: 1.4747 - val_accuracy: 0.3704
```

```
Epoch 4/30
31/31 [==============================] - 1s 21ms/step - loss: 1.4265 - accuracy:
0.3969 - val_loss: 1.4353 - val_accuracy: 0.4005
Epoch 5/30
31/31 [==============================] - 1s 20ms/step - loss: 1.3508 - accuracy:
0.4479 - val_loss: 1.3965 - val_accuracy: 0.4144
Epoch 6/30
31/31 [==============================] - 1s 21ms/step - loss: 1.2432 - accuracy:
0.5012 - val_loss: 1.4058 - val_accuracy: 0.4074
Epoch 7/30
31/31 [==============================] - 1s 22ms/step - loss: 1.1629 - accuracy:
0.5454 - val_loss: 1.4050 - val_accuracy: 0.4282
Epoch 8/30
31/31 [==============================] - 1s 20ms/step - loss: 1.0353 - accuracy:
0.6075 - val_loss: 1.4225 - val_accuracy: 0.4144
Epoch 9/30
31/31 [==============================] - 1s 21ms/step - loss: 0.9017 - accuracy:
0.6538 - val_loss: 1.5186 - val_accuracy: 0.4329
Epoch 10/30
31/31 [==============================] - 1s 21ms/step - loss: 0.7750 - accuracy:
0.7140 - val_loss: 1.5991 - val_accuracy: 0.4190
Epoch 11/30
31/31 [==============================] - 1s 21ms/step - loss: 0.6264 - accuracy:
0.7815 - val_loss: 1.6793 - val_accuracy: 0.4259
Epoch 12/30
31/31 [==============================] - 1s 20ms/step - loss: 0.5011 - accuracy:
0.8219 - val_loss: 1.7878 - val_accuracy: 0.4375
Epoch 13/30
31/31 [==============================] - 1s 21ms/step - loss: 0.3891 - accuracy:
0.8641 - val_loss: 2.0178 - val_accuracy: 0.4144
Epoch 14/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2986 - accuracy:
0.8996 - val_loss: 2.2816 - val_accuracy: 0.3981
Epoch 15/30
31/31 [==============================] - 1s 21ms/step - loss: 0.2378 - accuracy:
0.9284 - val_loss: 2.4153 - val_accuracy: 0.4491
Epoch 16/30
31/31 [==============================] - 1s 23ms/step - loss: 0.1977 - accuracy:
0.9382 - val_loss: 2.9004 - val_accuracy: 0.4352
Epoch 17/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1581 - accuracy:
0.9537 - val_loss: 2.7982 - val_accuracy: 0.4282
Epoch 18/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1446 - accuracy:
0.9542 - val_loss: 2.9403 - val_accuracy: 0.4028
Epoch 19/30
31/31 [==============================] - 1s 21ms/step - loss: 0.1161 - accuracy:
0.9622 - val_loss: 3.1075 - val_accuracy: 0.4167
```

```
Epoch 20/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0934 - accuracy:
0.9722 - val_loss: 3.3786 - val_accuracy: 0.4213
Epoch 21/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0683 - accuracy:
0.9825 - val_loss: 3.2799 - val_accuracy: 0.4306
Epoch 22/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0666 - accuracy:
0.9825 - val_loss: 3.4496 - val_accuracy: 0.4236
Epoch 23/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0571 - accuracy:
0.9846 - val_loss: 3.4393 - val_accuracy: 0.4352
Epoch 24/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0505 - accuracy:
0.9874 - val_loss: 3.5200 - val_accuracy: 0.4560
Epoch 25/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0416 - accuracy:
0.9897 - val_loss: 3.6214 - val_accuracy: 0.4421
Epoch 26/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0360 - accuracy:
0.9912 - val_loss: 3.8942 - val_accuracy: 0.4259
Epoch 27/30
31/31 [==============================] - 1s 22ms/step - loss: 0.0484 - accuracy:
0.9853 - val_loss: 3.8001 - val_accuracy: 0.4236
Epoch 28/30
31/31 [==============================] - 1s 20ms/step - loss: 0.0427 - accuracy:
0.9884 - val_loss: 3.7852 - val_accuracy: 0.4444
Epoch 29/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0419 - accuracy:
0.9900 - val_loss: 3.8563 - val_accuracy: 0.4213
Epoch 30/30
31/31 [==============================] - 1s 21ms/step - loss: 0.0359 - accuracy:
0.9907 - val_loss: 4.0303 - val_accuracy: 0.4306
14/14 [==============================] - 0s 5ms/step - loss: 4.0303 - accuracy:
0.4306
14/14 [==============================] - 0s 7ms/step
```

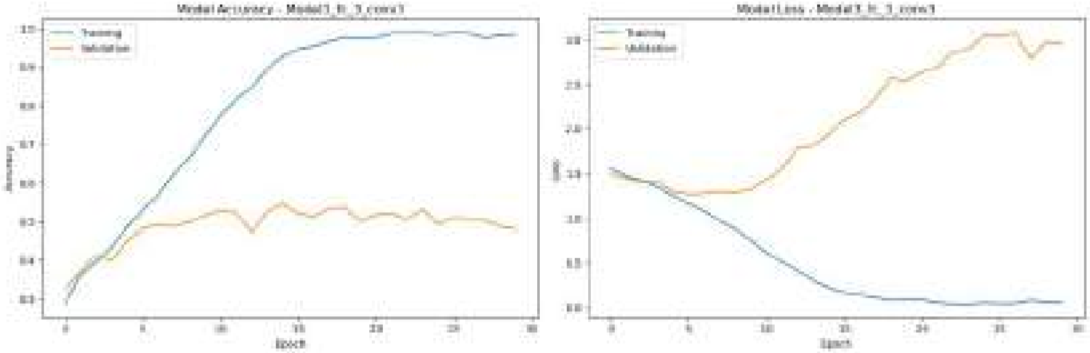


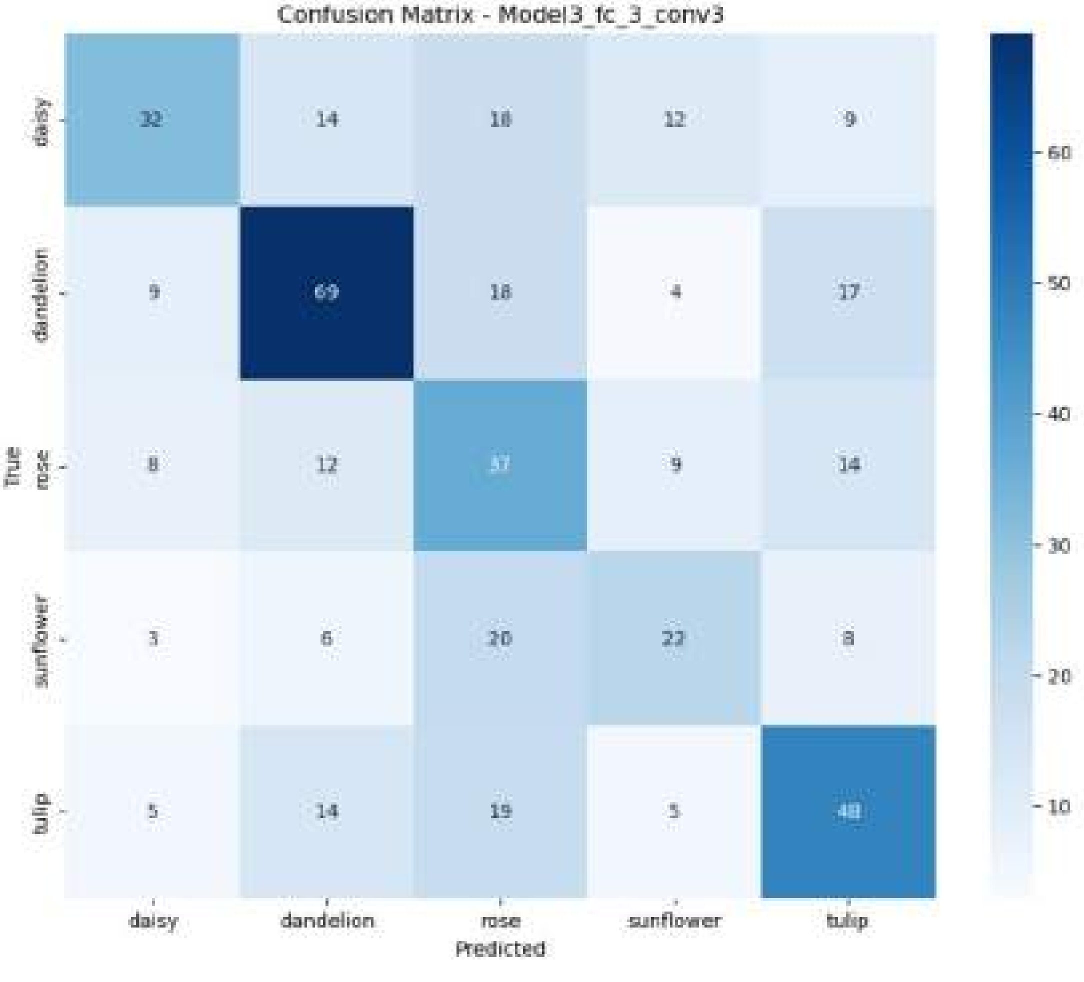


```
Results for Model3_fc_3_conv2:
Configuration: Model3_fc_3_conv2
Test Accuracy: 0.4306
F1 Score: 0.4270
Training Time: 20.68 seconds
Number of Parameters: 3,323,589
Epoch 1/30
31/31 [==============================] - 2s 33ms/step - loss: 1.5595 - accuracy:
0.2862 - val_loss: 1.4950 - val_accuracy: 0.3241
Epoch 2/30
31/31 [==============================] - 1s 23ms/step - loss: 1.4700 - accuracy:
0.3640 - val_loss: 1.4503 - val_accuracy: 0.3727
Epoch 3/30
```

```
31/31 [==============================] - 1s 24ms/step - loss: 1.4210 - accuracy:
0.3954 - val_loss: 1.4053 - val_accuracy: 0.4120
Epoch 4/30
31/31 [==============================] - 1s 23ms/step - loss: 1.3483 - accuracy:
0.4337 - val_loss: 1.4105 - val_accuracy: 0.4005
Epoch 5/30
31/31 [==============================] - 1s 24ms/step - loss: 1.2529 - accuracy:
0.4880 - val_loss: 1.3002 - val_accuracy: 0.4491
Epoch 6/30
31/31 [==============================] - 1s 23ms/step - loss: 1.1625 - accuracy:
0.5302 - val_loss: 1.2706 - val_accuracy: 0.4838
Epoch 7/30
31/31 [==============================] - 1s 24ms/step - loss: 1.0821 - accuracy:
0.5691 - val_loss: 1.2767 - val_accuracy: 0.4931
Epoch 8/30
31/31 [==============================] - 1s 24ms/step - loss: 0.9697 - accuracy:
0.6237 - val_loss: 1.3031 - val_accuracy: 0.4884
Epoch 9/30
31/31 [==============================] - 1s 23ms/step - loss: 0.8835 - accuracy:
0.6654 - val_loss: 1.2895 - val_accuracy: 0.5000
Epoch 10/30
31/31 [==============================] - 1s 24ms/step - loss: 0.7522 - accuracy:
0.7218 - val_loss: 1.3270 - val_accuracy: 0.5162
Epoch 11/30
31/31 [==============================] - 1s 24ms/step - loss: 0.6124 - accuracy:
0.7748 - val_loss: 1.4373 - val_accuracy: 0.5301
Epoch 12/30
31/31 [==============================] - 1s 24ms/step - loss: 0.5129 - accuracy:
0.8180 - val_loss: 1.5567 - val_accuracy: 0.5231
Epoch 13/30
31/31 [==============================] - 1s 25ms/step - loss: 0.4185 - accuracy:
0.8479 - val_loss: 1.7819 - val_accuracy: 0.4722
Epoch 14/30
31/31 [==============================] - 1s 24ms/step - loss: 0.3042 - accuracy:
0.8958 - val_loss: 1.8124 - val_accuracy: 0.5278
Epoch 15/30
31/31 [==============================] - 1s 23ms/step - loss: 0.2204 - accuracy:
0.9284 - val_loss: 1.9456 - val_accuracy: 0.5463
Epoch 16/30
31/31 [==============================] - 1s 24ms/step - loss: 0.1692 - accuracy:
0.9472 - val_loss: 2.1069 - val_accuracy: 0.5208
Epoch 17/30
31/31 [==============================] - 1s 23ms/step - loss: 0.1405 - accuracy:
0.9550 - val_loss: 2.1753 - val_accuracy: 0.5139
Epoch 18/30
31/31 [==============================] - 1s 24ms/step - loss: 0.1124 - accuracy:
0.9671 - val_loss: 2.3452 - val_accuracy: 0.5347
Epoch 19/30
```

```
31/31 [==============================] - 1s 25ms/step - loss: 0.0812 - accuracy:
0.9776 - val_loss: 2.5715 - val_accuracy: 0.5370
Epoch 20/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0892 - accuracy:
0.9735 - val_loss: 2.5364 - val_accuracy: 0.5000
Epoch 21/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0874 - accuracy:
0.9771 - val_loss: 2.6398 - val_accuracy: 0.5185
Epoch 22/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0519 - accuracy:
0.9864 - val_loss: 2.6965 - val_accuracy: 0.5208
Epoch 23/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0410 - accuracy:
0.9892 - val_loss: 2.8650 - val_accuracy: 0.5069
Epoch 24/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0377 - accuracy:
0.9887 - val_loss: 2.9027 - val_accuracy: 0.5324
Epoch 25/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0528 - accuracy:
0.9838 - val_loss: 3.0484 - val_accuracy: 0.4931
Epoch 26/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0438 - accuracy:
0.9876 - val_loss: 3.0349 - val_accuracy: 0.5116
Epoch 27/30
31/31 [==============================] - 1s 23ms/step - loss: 0.0462 - accuracy:
0.9866 - val_loss: 3.0793 - val_accuracy: 0.5069
Epoch 28/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0796 - accuracy:
0.9763 - val_loss: 2.7917 - val_accuracy: 0.5023
Epoch 29/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0595 - accuracy:
0.9822 - val_loss: 2.9805 - val_accuracy: 0.4884
Epoch 30/30
31/31 [==============================] - 1s 24ms/step - loss: 0.0538 - accuracy:
0.9848 - val_loss: 2.9558 - val_accuracy: 0.4815
14/14 [==============================] - 0s 5ms/step - loss: 2.9558 - accuracy:
0.4815
14/14 [==============================] - 0s 6ms/step
```



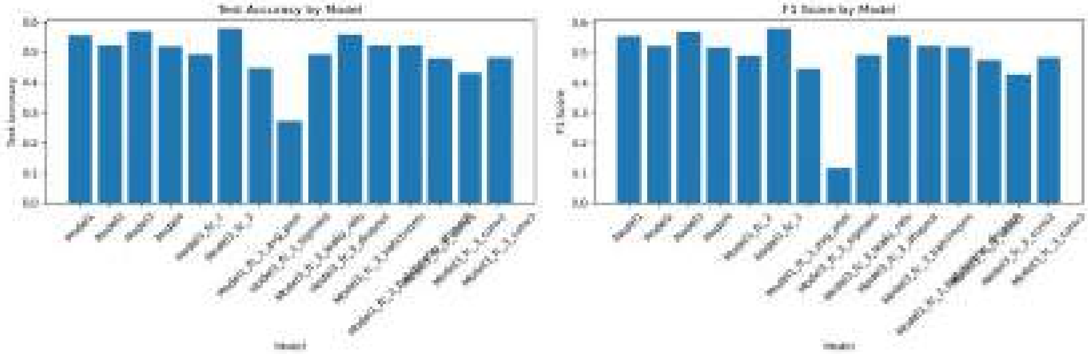


```
Results for Model3_fc_3_conv3:
Configuration: Model3_fc_3_conv3
```

```
Test Accuracy: 0.4815
F1 Score: 0.4834
Training Time: 23.43 seconds
Number of Parameters: 1,736,453
```

[86]:

```python
# Create figure with 2 subplots side by side
plt.figure(figsize=(15, 5))

# Plot test accuracy
plt.subplot(1, 2, 1)
plt.bar([r['configuration'] for r in results], [r['test_accuracy'] for r in␣
↪results])
plt.title('Test Accuracy by Model')
plt.xlabel('Model')
plt.ylabel('Test Accuracy')
plt.xticks(rotation=45)

# Plot F1 scores
plt.subplot(1, 2, 2)
plt.bar([r['configuration'] for r in results], [r['f1_score'] for r in results])
plt.title('F1 Score by Model')
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



[87]:

```python
headers = results[0].keys()
rows = [[f"{row['configuration']}",
         f"{row['test_accuracy']:.4f}",
         f"{row['f1_score']:.4f}",
         f"{row['training_time']:.4f}",
         row['parameters']] for row in results]
```

```python
# Print formatted table
print("\nModel Comparison Results:")
print(tabulate(rows, headers=headers, tablefmt='pretty'))

# Find best model based on test accuracy
best_model = max(results, key=lambda x: x['test_accuracy'])
print("\nBest Model Parameters:")
print(f"Configuration: {best_model['configuration']}")
print(f"Test Accuracy: {best_model['test_accuracy']:.4f}")
print(f"F1 Score: {best_model['f1_score']:.4f}")
print(f"Training Time: {best_model['training_time']:.4f} seconds")
print(f"Number of Parameters: {best_model['parameters']}")
```

```
Model Comparison Results:
+-------------------------------+---------------+----------+---------------+----
--------+
|         configuration         | test_accuracy | f1_score | training_time |
parameters |
+-------------------------------+---------------+----------+---------------+----
--------+
|            Model1             |    0.5556     |  0.5541  |    22.0594    |
55301    |
|            Model2             |    0.5231     |  0.5243  |    20.9627    |
88069    |
|            Model3             |    0.5671     |  0.5669  |    23.0679    |
96261    |
|            Model4             |    0.5185     |  0.5160  |    23.0128    |
96517    |
|          Model3_fc_2          |    0.4931     |  0.4886  |    30.3716    |
884229    |
|          Model3_fc_3          |    0.5772     |  0.5796  |    31.0176    |
1736453   |
|     Model1_fc_3_avg_pool      |    0.4444     |  0.4447  |    21.2291    |
1728261   |
|      Model3_fc_3_sigmoid      |    0.2708     |  0.1154  |    24.5248    |
1728261   |
|    Model3_fc_3_leaky_relu     |    0.4907     |  0.4910  |    24.4649    |
1728261   |
|      Model3_fc_3_dropout      |    0.5579     |  0.5555  |    21.6860    |
1728261   |
|     Model3_fc_3_batchnorm     |    0.5231     |  0.5241  |    20.8621    |
1728645   |
| Model1_fc_3_batchnorm_dropout |    0.5231     |  0.5188  |    22.7609    |
1728645   |
|       Model3_fc_3_conv1       |    0.4792     |  0.4767  |    16.1726    |
```

```
6587557    |
|       Model3_fc_3_conv2        |     0.4306     | 0.4270  |     20.6756     |
3323589    |
|       Model3_fc_3_conv3        |     0.4815     | 0.4834  |     23.4291     |
1736453    |
+--------------------------------+----------------+---------+----------------+----
--------+

Best Model Parameters:
Configuration: Model3_fc_3
Test Accuracy: 0.5772
F1 Score: 0.5796
Training Time: 31.0176 seconds
Number of Parameters: 1736453
```

8.1 For the best model on the MNIST dataset in Assignment 4, train a model with MNIST data using the best set of parameters obtained in Question . Compare the test accuracy and the self-created images.

[88]:

```
from keras.datasets import mnist
```

[90]:

```
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess data
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

# Split training data into train and validation (80-20)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.
 ↪2, random_state=42)

# Create model with same architecture
model3_fc_3 = Sequential(name='Model3_fc_3')

model3_fc_3.add(Conv2D(filters = 16, kernel_size = (3, 3), padding = 'same',␣
 ↪activation ='relu', input_shape = (28, 28, 1)))
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'same',␣
 ↪activation ='relu'))
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(Conv2D(filters = 64, kernel_size = (5, 5), padding = 'same',␣
 ↪activation ='relu'))
```

```
model3_fc_3.add(MaxPooling2D(pool_size=(2, 2)))
model3_fc_3.add(Dropout(0.1))

model3_fc_3.add(Flatten())

model3_fc_3.add(Dense(256, activation = 'relu'))
model3_fc_3.add(Dense(128, activation = 'relu'))
model3_fc_3.add(Dense(10, activation = 'softmax'))  # 10 classes for MNIST

model3_fc_3.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
history = model3_fc_3.fit(x_train, y_train,
                          batch_size=256,
                          epochs=10,
                          validation_data=(x_val, y_val))

# Evaluate on test set
test_loss, test_accuracy = model3_fc_3.evaluate(x_test, y_test)
print(f"\nTest accuracy: {test_accuracy:.4f}")
```

```
Epoch 1/10
188/188 [==============================] - 4s 13ms/step - loss: 0.3997 -
accuracy: 0.8741 - val_loss: 0.0936 - val_accuracy: 0.9718
Epoch 2/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0929 -
accuracy: 0.9706 - val_loss: 0.0545 - val_accuracy: 0.9833
Epoch 3/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0636 -
accuracy: 0.9794 - val_loss: 0.0502 - val_accuracy: 0.9851
Epoch 4/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0480 -
accuracy: 0.9854 - val_loss: 0.0418 - val_accuracy: 0.9871
Epoch 5/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0395 -
accuracy: 0.9874 - val_loss: 0.0414 - val_accuracy: 0.9876
Epoch 6/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0326 -
accuracy: 0.9897 - val_loss: 0.0329 - val_accuracy: 0.9899
Epoch 7/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0282 -
accuracy: 0.9908 - val_loss: 0.0334 - val_accuracy: 0.9903
Epoch 8/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0246 -
accuracy: 0.9920 - val_loss: 0.0311 - val_accuracy: 0.9912
Epoch 9/10
```

```
188/188 [==============================] - 2s 11ms/step - loss: 0.0244 -
accuracy: 0.9925 - val_loss: 0.0362 - val_accuracy: 0.9893
Epoch 10/10
188/188 [==============================] - 2s 11ms/step - loss: 0.0207 -
accuracy: 0.9931 - val_loss: 0.0318 - val_accuracy: 0.9912
313/313 [==============================] - 1s 5ms/step - loss: 0.0272 -
accuracy: 0.9920

Test accuracy: 0.9920
```

[94]:

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageOps

def preprocess_image(image_path: str) -> np.ndarray:
    """
    Preprocess an image to make it similar to MNIST dataset images by detecting␣
 ↪the digit,
    thresholding, cropping, centering, padding, and resizing to 28x28 pixels.

    Args:
        image_path (str): Path to the image file.

    Returns:
        np.ndarray: Preprocessed image as a numpy array.
    """
    # Load and convert to grayscale
    img = Image.open(image_path).convert('L')

    # Invert the image: MNIST has white digits on a black background
    img = ImageOps.invert(img)

    # Apply binary threshold
    threshold = img.point(lambda p: 255 if p > 128 else 0)
    threshold = threshold.convert('1')  # Convert to binary image

    # Convert to numpy array for processing
    np_threshold = np.array(threshold)

    # Find bounding box of the digit
    bbox = threshold.getbbox()
    if bbox:
        cropped_img = threshold.crop(bbox)
    else:
        # If no content is found, return a blank image
        cropped_img = Image.new('1', (28, 28), 0)
        return np.array(cropped_img).astype(np.float32) / 255.0
```

```
    # Get dimensions of the cropped image
    cropped_width, cropped_height = cropped_img.size

    # Determine the size of the new square image to add padding
    max_side = max(cropped_width, cropped_height)
    square_img = Image.new('1', (max_side, max_side), 0)  # Black background
    paste_position = (
        (max_side - cropped_width) // 2,
        (max_side - cropped_height) // 2
    )
    square_img.paste(cropped_img, paste_position)

    # Add padding to reach the desired size before resizing
    # MNIST digits are typically centered with some padding
    padding = 10  # Total padding to add on each side
    padded_size = max_side + 2 * padding
    padded_img = Image.new('1', (padded_size, padded_size), 0)
    padded_paste_position = (
        (padded_size - max_side) // 2,
        (padded_size - max_side) // 2
    )
    padded_img.paste(square_img, padded_paste_position)

    # Resize to 28x28 pixels using the appropriate resampling filter
    try:
        # For Pillow >= 10.0.0
        resized_img = padded_img.resize((28, 28), Image.Resampling.LANCZOS)
    except AttributeError:
        # For Pillow < 10.0.0
        resized_img = padded_img.resize((28, 28), Image.LANCZOS)

    # Convert to numpy array and normalize to [0, 1]
    final_img = np.array(resized_img).astype(np.float32)
    print(f"np.min(resized_img): {np.min(final_img)}, np.max(resized_img): {np.
 ↪max(final_img)}")
    # final_img = final_img.flatten()

    print(final_img.shape)
    return final_img

# Image filenames (ensure these images are in your working directory)
image_filenames = [
    '1.png', '3.png',
    '5.png', '7.png', '8.png'
]
```

```
# Process each image
import os

# Create the 'preprocessed' directory if it doesn't exist
preprocessed_dir = 'preprocessed'
os.makedirs(preprocessed_dir, exist_ok=True)

preprocessed_images = np.array([preprocess_image(filename) for filename in␣
  ↪image_filenames])

# Save the preprocessed images in the 'preprocessed' directory
for img, filename in zip(preprocessed_images, image_filenames):
    img_reshaped = img.reshape(28, 28)
    img_pil = Image.fromarray((img_reshaped * 255).astype(np.uint8), mode='L')
    img_pil.save(os.path.join(preprocessed_dir, filename))

# Plot the preprocessed images
fig, axes = plt.subplots(1, len(preprocessed_images), figsize=(15, 3))
for ax, img, filename in zip(axes, preprocessed_images, image_filenames):
    ax.imshow(img.reshape(28, 28), cmap='gray')
    ax.set_title(filename)
    ax.axis('off')
plt.tight_layout()
plt.show()
```

```
np.min(resized_img): 0.0, np.max(resized_img): 1.0
(28, 28)
np.min(resized_img): 0.0, np.max(resized_img): 1.0
(28, 28)
np.min(resized_img): 0.0, np.max(resized_img): 1.0
(28, 28)
np.min(resized_img): 0.0, np.max(resized_img): 1.0
(28, 28)
np.min(resized_img): 0.0, np.max(resized_img): 1.0
(28, 28)
```

[95]:

```python
predictions = model3_fc_3.predict(preprocessed_images)

# Plot the grid of images along with their corresponding predictions
fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for ax, img, pred in zip(axes, preprocessed_images, predictions):
    ax.imshow(img.reshape(28, 28), cmap='gray')
    ax.axis('off')
    ax.set_title(f"Pred: {np.argmax(pred)}")

plt.suptitle('Testing with Handwritten Digits')
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 486ms/step
```

CNNs work better on patterns.