Introduction to Computer Vision: Plant Seedlings Classification

Problem Statement

Context

In recent times, the field of agriculture has been in urgent need of modernizing, since the amount of manual work people need to put in to check if plants are growing correctly is still highly extensive. Despite several advances in agricultural technology, people working in the agricultural industry still need to have the ability to sort and recognize different plants and weeds, which takes a lot of time and effort in the long term. The potential is ripe for this trillion-dollar industry to be greatly impacted by technological innovations that cut down on the requirement for manual labor, and this is where Artificial Intelligence can actually benefit the workers in this field, as **the time and energy required to identify plant seedlings will be greatly shortened by the use of AI and Deep Learning.** The ability to do so far more efficiently and even more effectively than experienced manual labor, could lead to better crop yields, the freeing up of human inolvement for higher-order agricultural decision making, and in the long term will result in more sustainable environmental practices in agriculture as well.

Objective

The aim of this project is to Build a Convolutional Neural Netowrk to classify plant seedlings into their respective categories.

Data Dictionary

The Aarhus University Signal Processing group, in collaboration with the University of Southern Denmark, has recently released a dataset containing **images of unique plants belonging to 12 different species.**

- The dataset can be download from Olympus.
- The data file names are:
  - images.npy
  - Labels.csv
- Due to the large volume of data, the images were converted to the images.npy file and the labels are also put into Labels.csv, so that you can work on the data/project seamlessly without having to worry about the high data volume.
- The goal of the project is to create a classifier capable of determining a plant's species from an image.

List of Species

- Black-grass
- Charlock
- Cleavers
- Common Chickweed
- Common Wheat
- Fat Hen
- Loose Silky-bent
- Maize
- Scentless Mayweed
- Shepherds Purse
- Small-flowered Cranesbill
- Sugar beet

Note: Please use GPU runtime on Google Colab to execute the code faster.

Importing necessary libraries

In [ ]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import cv2
import seaborn as sns

# Tensorflow, Keras & Sklearn modules
import tensorflow as tf
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, Sequential, clone_model
from tensorflow.keras.layers import Dense,Dropout, Flatten, Conv2D, MaxPooling2D,BatchNormalization, GlobalAvera
from tensorflow.keras.optimizers import Adam,SGD
from tensorflow.keras.applications import VGG16
from tensorflow.keras import backend
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.preprocessing import LabelBinarizer
from sklearn.utils.class_weight import compute_class_weight
from sklearn.utils.validation import check_is_fitted
from sklearn.exceptions import NotFittedError
from sklearn.model_selection import train_test_split
# Display images using OpenCV
from google.colab.patches import cv2_imshow
import random
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [ ]:
```
# Mounting google drive and initilizing path variable
from google.colab import drive
drive.mount("/content/drive")
path = '/content/drive/MyDrive/PGPAIML/Project-5/'
```

```
Mounted at /content/drive
```

In [ ]:
```
# Load the image file of dataset
images = np.load(path+'images.npy')

# Load the labels file of dataset
labels = pd.read_csv(path+'labels.csv')
```

Data Overview

Understand the shape of the dataset

In [ ]:
```
print(images.shape)
print(labels.shape)
```

```
(4750, 128, 128, 3)
(4750, 1)
```

Observations

- We have 4750 labeled images as observations.
- The image size is 128 X 128 pixels and 3 channels.

Exploratory Data Analysis

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.

Utility functions

In [ ]:
```
# Function to display images in the provided range.
def show_images_from_data(images,labels, rows, cols, ranges):
  num_classes=10
  categories=np.unique(labels)
  keys=dict(labels['Label'])
  index = 0
  fig = plt.figure(figsize=((3 * cols), (3.2 * rows)))
  for i in range(cols):
      for j in range(rows):
          display_index = ranges[index]
          index += 1
          ax = fig.add_subplot(rows, cols, i * rows + j + 1)
          ax.imshow(images[display_index])
          ax.set_title(keys[display_index])
  plt.show()
```

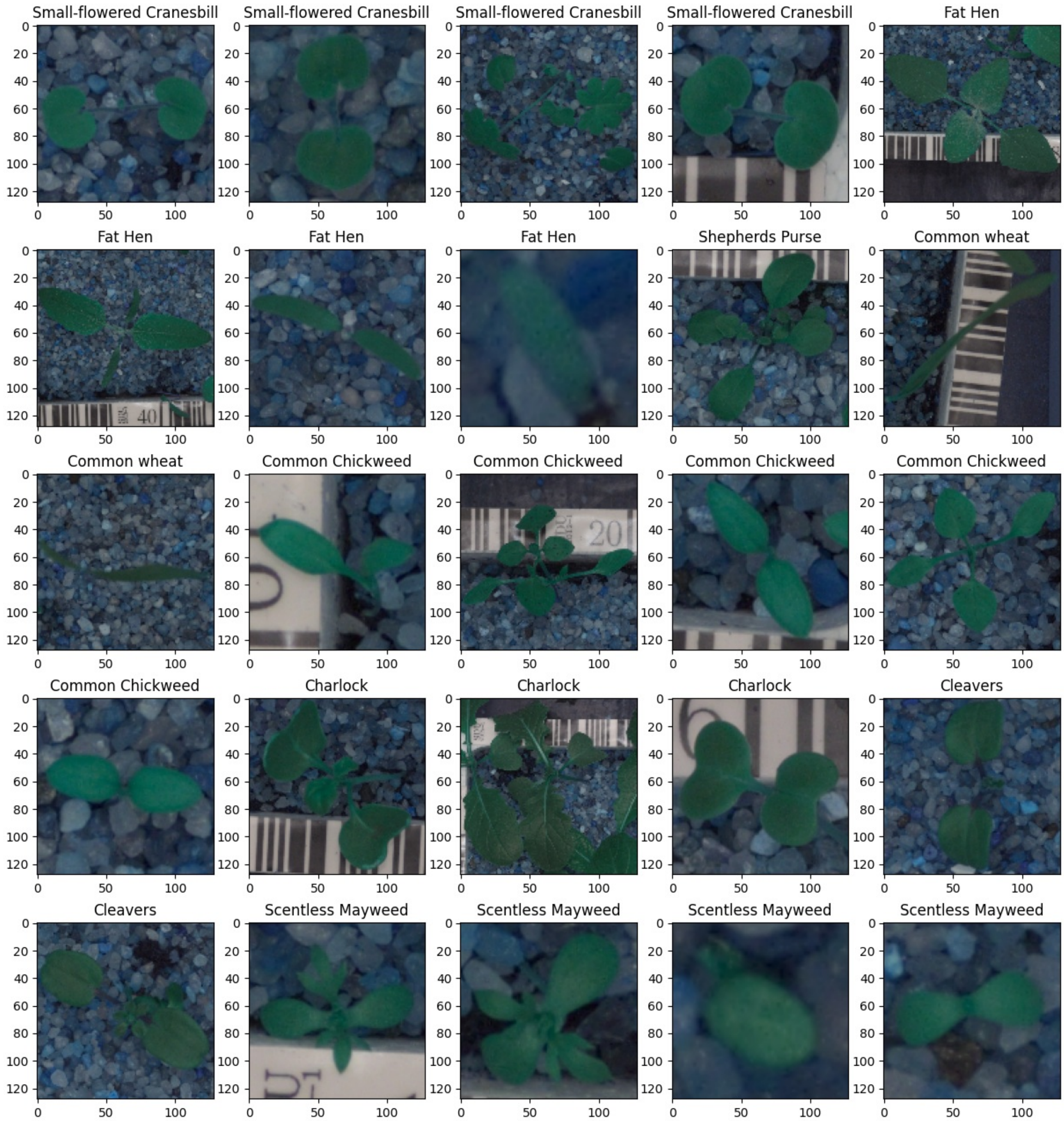
In [ ]:

```python
# Function to display labes count in a bar plot.
def display_label_count(labeles):

  # Calculating label counts and percentages
  label_counts = labels['Label'].value_counts()
  label_percentages = (label_counts / len(labels) * 100).round(2)

  label_summary = pd.DataFrame({'Count': label_counts, 'Percentage': label_percentages})

  # Ploting the count plot with percentages
  plt.figure(figsize=(12, 6))
  sns.barplot(x=label_summary.index, y=label_summary['Count'], palette='viridis')
  plt.xticks(rotation=90)
  plt.title('Label Distribution with Percentages')
  plt.xlabel('Labels')
  plt.ylabel('Count')

  # Annotate percentages on the bars
  for i, p in enumerate(label_percentages):
      plt.text(i, label_counts.iloc[i] + 5, f'{p}%', ha='center')

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

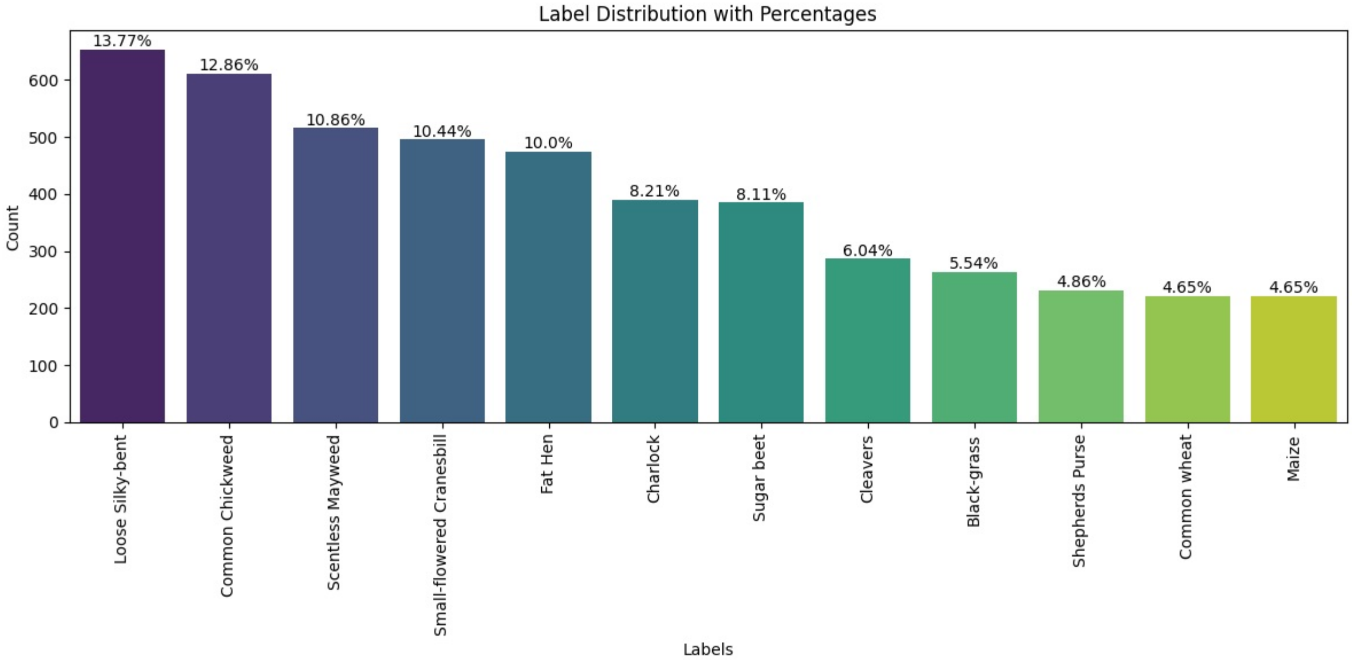
Visualizing Images & Labels

In [ ]:

```python
show_images_from_data(images, labels, 5, 5, range(0, 4749, 134))
```



```
In [ ]: display_label_count(labels)
```



Observations

- Looking at the images, domain knowledge and/or expertise are needed to classify them into proper categories, even for humans.
- The dataset apparently has an imbalanced distribution of labels.
- The most common label is `Loose Silky-bent` (13.77%), followed closely by `Common Chickweed` (12.86%).
- The least represented labels are `Maize` (4.65%), `Common Wheat` (4.65%), and `Shepherd's Purse` (4.86%).
- The class imbalance could bias the model toward the majority classes.
- Applying techniques like using class_weight or oversampling, undersampling, or data augmentation should be a good idea.

Decision

- I decided to Use the **class weights** approach during training, considering the below points
- No Data Duplication is needed
- Simpler, It integrates seamlessly into the `fit()` method
- Class weights dynamically adjust the importance of each class during training, which helps the model learn balanced features without manipulating the data.

Data Pre-Processing

Convert the BGR images to RGB images.

- As OpenCV Reads Images in BGR Format but TensorFlow/Keras expects images in RGB format, we need to convert BGR to RGB for consistency.

```
In [ ]: # Image format conversion
for i in range(len(images)):
  image_bgr = images[i];
  # Converting images using cv2
  image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
  images[i] = image_rgb;
```

Resize the images

As the size of the images is large, it may be computationally expensive to train on these larger images; therefore, it is preferable to reduce the image size from 128 to 64.

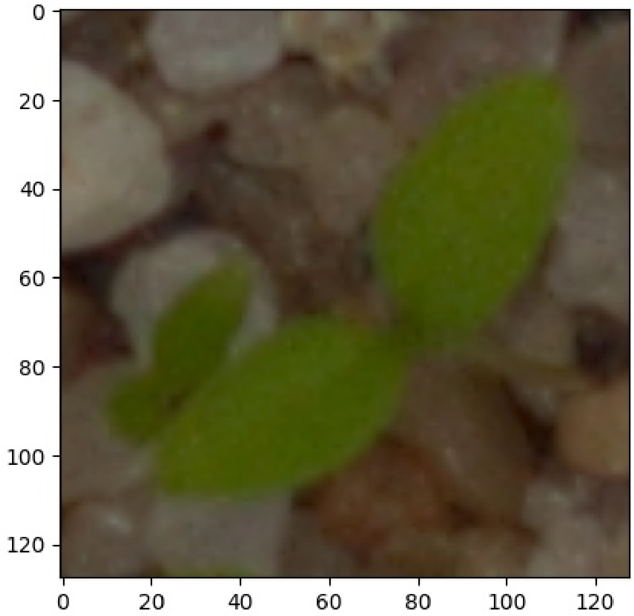
- Advantages of Reducing the Image Size:
  - Smaller image sizes mean fewer input features, which significantly reduces the number of computations in convolutional and fully connected layers.
  - This is especially useful when training from scratch on limited hardware or for faster experimentation.
- Quicker Training:
  - With smaller input dimensions, the training process is faster while still capturing essential patterns (as long as the reduction

doesn't lose critical information).

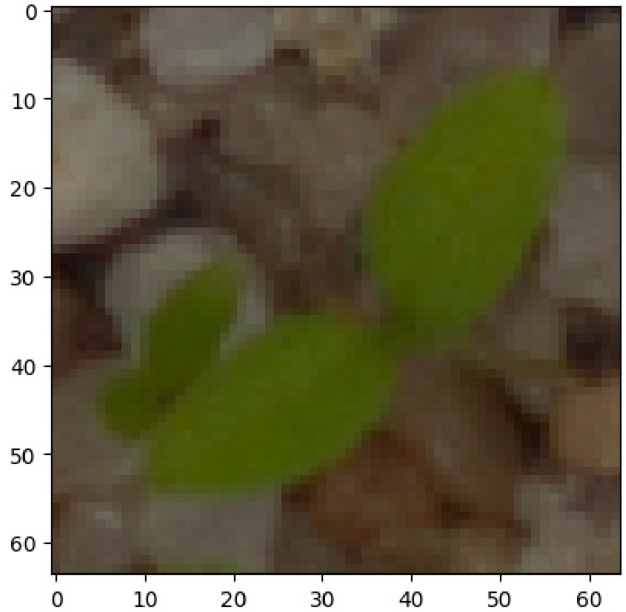
- Avoid Overfitting:
  - Smaller images can help prevent overfitting by limiting the model's ability to memorize unnecessary details.

In [ ]:
```
images_resized = np.array([cv2.resize(img, (64, 64)) for img in images])
```

In [ ]:
```
random_number = random.randint(1, 4700)
plt.imshow(images[random_number]);
```



In [ ]:
```
plt.imshow(images_resized[random_number]);
```



Observations

To ensure that resizing to 64x64 does not result in a significant loss of information in the images, I displayed the same image above in both sizes. The observations show no significant distortions, though the resolution is affected. It is a good sign to proceed with the reduced image sizes.

Data Preparation for Modeling

Split the dataset

In [ ]:
```
# Splitting data into Train (70%) and Temp (30%)
x_train, x_temp, y_train, y_temp = train_test_split(
    images_resized, labels, test_size=0.3, random_state=1, stratify=labels
)

# Splitting Temp into Validation (50% of Temp -> 15% overall) and Test (50% of Temp -> 15% overall)
x_val, x_test, y_val, y_test = train_test_split(
```

```
    x_temp, y_temp, test_size=0.5, random_state=1, stratify=y_temp
)

# Printing the shapes
print(f"Training Set: X={x_train.shape}, Y={y_train.shape}")
print(f"Validation Set: X={x_val.shape}, Y={y_val.shape}")
print(f"Test Set: X={x_test.shape}, Y={y_test.shape}")
```

```
Training Set: X=(3325, 64, 64, 3), Y=(3325, 1)
Validation Set: X=(712, 64, 64, 3), Y=(712, 1)
Test Set: X=(713, 64, 64, 3), Y=(713, 1)
```

Encode the target labels

In [ ]:
```
# Initializing
label_binarizer = LabelBinarizer()

# Fit and transform
y_encoded = label_binarizer.fit_transform(labels['Label'])

# Verifying the shape
print("Encoded Labels Shape:", y_encoded.shape)
print("Sample Encoded Labels:\n", y_encoded[:5])

# Splitting data again after encoding if needed
y_train_encoded = label_binarizer.transform(y_train)
y_val_encoded = label_binarizer.transform(y_val)
y_test_encoded = label_binarizer.transform(y_test)
```

```
Encoded Labels Shape: (4750, 12)
Sample Encoded Labels:
 [[0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0]]
```

Data Normalization

In [ ]:
```
# Converting the image data to float and normalize pixel values
x_train_normalized = x_train.astype('float32') / 255.0
x_val_normalized = x_val.astype('float32') / 255.0
x_test_normalized = x_test.astype('float32') / 255.0

# Verifying normalization by checking min and max values
print(f"Training data range: {x_train_normalized.min()} to {x_train_normalized.max()}")
print(f"Validation data range: {x_val_normalized.min()} to {x_val_normalized.max()}")
print(f"Test data range: {x_test_normalized.min()} to {x_test_normalized.max()}")
```

```
Training data range: 0.0 to 1.0
Validation data range: 0.0 to 1.0
Test data range: 0.0 to 1.0
```

Model Building

Utility functions - Common Codes and Model Evaluation functions

In [ ]:
```
# COOMON-CODE --> For Calculating Class Weights
y_train_labels = np.argmax(y_train_encoded, axis=1)  # y_train_encoded is one-hot encoding
# Extract unique class labels
class_labels = np.unique(y_train_labels)
class_weights = compute_class_weight('balanced', classes=class_labels, y=y_train_labels)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}
print("Computed Class Weights:", class_weights_dict)
```

```
Computed Class Weights: {0: 1.5058876811594204, 1: 1.014957264957265, 2: 1.378524046434494, 3: 0.647390965732087
2, 4: 1.7876344086021505, 5: 0.8345883534136547, 6: 0.6049854439592431, 7: 1.7876344086021505, 8: 0.767543859649
1229, 9: 1.7103909465020577, 10: 0.7985110470701249, 11: 1.0300495662949194}
```

In [ ]:
```
# Function for setting seed for reproducibility & clearing session
def reset_environment(seed=1):
  backend.clear_session()
  np.random.seed(seed)
  random.seed(seed)
  tf.random.set_seed(seed)
  tf.random.set_seed(seed)
  tf.keras.utils.set_random_seed(seed)
  # tf.config.experimental.enable_op_determinism() <-- this does not work when using GPU in Google CoLab
```

In [ ]:
```
# Display loss plot for provided training history
```

```
def plot_loss(his):
  plt.plot(his.history['loss'])
  plt.plot(his.history['val_loss'])
  plt.title('Model Loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Validation'], loc='upper left')
  plt.show()

# Display accuracy plot for provided training history
def plot_accuracy(his):
  plt.plot(his.history['accuracy'])
  plt.plot(his.history['val_accuracy'])
  plt.title('Model Accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Validation'], loc='upper left')
  plt.show()
```

In [ ]:
```
# Function to display multi-class classification confusion matrix
def plot_confusion_matrix_and_report(model, x, y):
  y_pred = model.predict(x)
  # Calculate confusion matrix
  # Obtaining the categorical values
  y_pred_arg = np.argmax(y_pred, axis=1)
  y_arg = np.argmax(y, axis=1)

  # Plotting the Confusion Matrix
  cm = tf.math.confusion_matrix(y_arg, y_pred_arg)
  f, ax = plt.subplots(figsize=(12, 12))
  sns.heatmap(
      cm,
      annot=True,
      linewidths=.4,
      fmt="d",
      square=True,
      ax=ax
  )
  # Setting the labels
  ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
  ax.set_title('Confusion Matrix');
  ax.xaxis.set_ticklabels(list(label_binarizer.classes_),rotation=40)
  ax.yaxis.set_ticklabels(list(label_binarizer.classes_),rotation=20)
  plt.show()
  cr = classification_report(y_arg, y_pred_arg)
  print(cr)
```

Initial Model - first iteration

In [ ]:
```
# Building the model
model1 = Sequential([
    # First Conv2D layer
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(64, 64, 3)),

    # Second Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # First MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # Third Conv2D layer
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),

    # Second MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # Flatten the output
    Flatten(),

    # First Dense layer with 16 neurons
    Dense(16, activation='relu'),

    # Output layer with 12 neurons for 12 classes
    Dense(12, activation='softmax')
])

# Compile the model
model1.compile(optimizer=Adam(),
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Print the model summary
```

```
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
conv2d_1 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 16)	524,304
dense_1 (Dense)	(None, 12)	204

Total params: 617,756 (2.36 MB)

Trainable params: 617,756 (2.36 MB)

Non-trainable params: 0 (0.00 B)

Fitting the model on the train data

In [ ]:
```
# Resetting the environment default seed=1
reset_environment()

# Fit the model
history1 = model1.fit(
            x_train_normalized,
            y_train_encoded,
            epochs=50,
            validation_data=(x_val_normalized,y_val_encoded),
            batch_size=32,
            class_weight=class_weights_dict,
            verbose=2
)
```

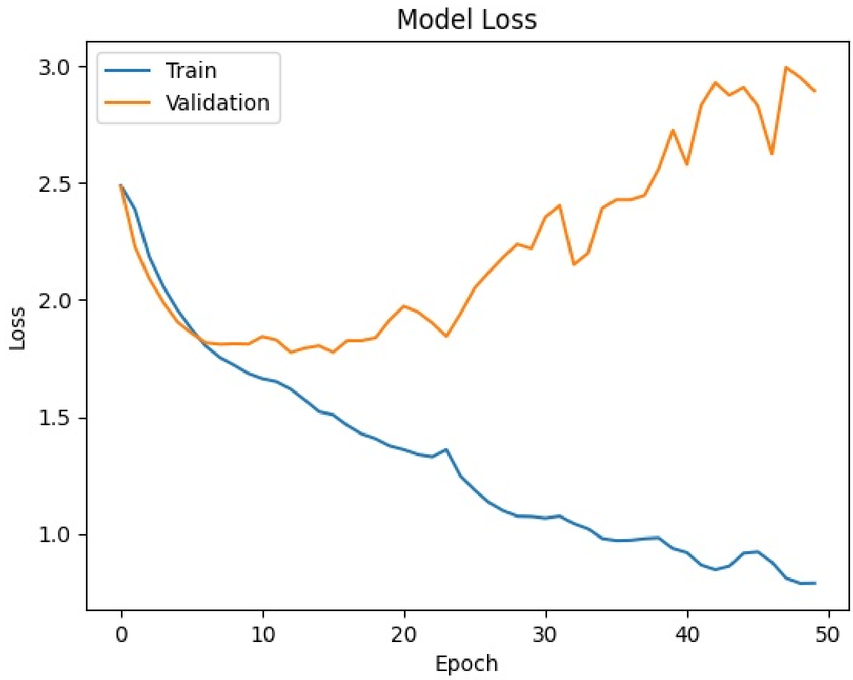
```
Epoch 1/50
104/104 - 9s - 89ms/step - accuracy: 0.0586 - loss: 2.4879 - val_accuracy: 0.1081 - val_loss: 2.4846
Epoch 2/50
104/104 - 4s - 35ms/step - accuracy: 0.1684 - loss: 2.3858 - val_accuracy: 0.2191 - val_loss: 2.2268
Epoch 3/50
104/104 - 2s - 24ms/step - accuracy: 0.1808 - loss: 2.1848 - val_accuracy: 0.1910 - val_loss: 2.0911
Epoch 4/50
104/104 - 1s - 12ms/step - accuracy: 0.2033 - loss: 2.0571 - val_accuracy: 0.2331 - val_loss: 1.9894
Epoch 5/50
104/104 - 1s - 12ms/step - accuracy: 0.2376 - loss: 1.9568 - val_accuracy: 0.2669 - val_loss: 1.9061
Epoch 6/50
104/104 - 1s - 13ms/step - accuracy: 0.2514 - loss: 1.8740 - val_accuracy: 0.2992 - val_loss: 1.8571
Epoch 7/50
104/104 - 2s - 23ms/step - accuracy: 0.2863 - loss: 1.8027 - val_accuracy: 0.3343 - val_loss: 1.8168
Epoch 8/50
104/104 - 1s - 12ms/step - accuracy: 0.3173 - loss: 1.7535 - val_accuracy: 0.3427 - val_loss: 1.8103
Epoch 9/50
104/104 - 1s - 12ms/step - accuracy: 0.3245 - loss: 1.7212 - val_accuracy: 0.3539 - val_loss: 1.8127
Epoch 10/50
104/104 - 1s - 12ms/step - accuracy: 0.3432 - loss: 1.6857 - val_accuracy: 0.3413 - val_loss: 1.8111
Epoch 11/50
104/104 - 1s - 12ms/step - accuracy: 0.3573 - loss: 1.6636 - val_accuracy: 0.3539 - val_loss: 1.8432
Epoch 12/50
104/104 - 1s - 12ms/step - accuracy: 0.3711 - loss: 1.6506 - val_accuracy: 0.3315 - val_loss: 1.8283
Epoch 13/50
104/104 - 1s - 12ms/step - accuracy: 0.3696 - loss: 1.6189 - val_accuracy: 0.3441 - val_loss: 1.7755
Epoch 14/50
104/104 - 1s - 12ms/step - accuracy: 0.3865 - loss: 1.5723 - val_accuracy: 0.3469 - val_loss: 1.7939
Epoch 15/50
104/104 - 1s - 13ms/step - accuracy: 0.4024 - loss: 1.5228 - val_accuracy: 0.3553 - val_loss: 1.8043
Epoch 16/50
104/104 - 1s - 13ms/step - accuracy: 0.4084 - loss: 1.5085 - val_accuracy: 0.3497 - val_loss: 1.7758
Epoch 17/50
104/104 - 1s - 13ms/step - accuracy: 0.4102 - loss: 1.4640 - val_accuracy: 0.3301 - val_loss: 1.8260
Epoch 18/50
104/104 - 2s - 24ms/step - accuracy: 0.4274 - loss: 1.4278 - val_accuracy: 0.3343 - val_loss: 1.8257
Epoch 19/50
104/104 - 1s - 12ms/step - accuracy: 0.4241 - loss: 1.4066 - val_accuracy: 0.3469 - val_loss: 1.8382
Epoch 20/50
104/104 - 1s - 13ms/step - accuracy: 0.4370 - loss: 1.3758 - val_accuracy: 0.3525 - val_loss: 1.9152
```

```
Epoch 21/50
104/104 - 2s - 24ms/step - accuracy: 0.4349 - loss: 1.3601 - val_accuracy: 0.3301 - val_loss: 1.9733
Epoch 22/50
104/104 - 1s - 13ms/step - accuracy: 0.4358 - loss: 1.3393 - val_accuracy: 0.3315 - val_loss: 1.9477
Epoch 23/50
104/104 - 3s - 25ms/step - accuracy: 0.4361 - loss: 1.3295 - val_accuracy: 0.3399 - val_loss: 1.9014
Epoch 24/50
104/104 - 1s - 13ms/step - accuracy: 0.4355 - loss: 1.3604 - val_accuracy: 0.3497 - val_loss: 1.8440
Epoch 25/50
104/104 - 1s - 13ms/step - accuracy: 0.4740 - loss: 1.2472 - val_accuracy: 0.3553 - val_loss: 1.9416
Epoch 26/50
104/104 - 1s - 12ms/step - accuracy: 0.4953 - loss: 1.1875 - val_accuracy: 0.3539 - val_loss: 2.0510
Epoch 27/50
104/104 - 1s - 13ms/step - accuracy: 0.5089 - loss: 1.1347 - val_accuracy: 0.3441 - val_loss: 2.1178
Epoch 28/50
104/104 - 2s - 24ms/step - accuracy: 0.5191 - loss: 1.0999 - val_accuracy: 0.3469 - val_loss: 2.1812
Epoch 29/50
104/104 - 1s - 12ms/step - accuracy: 0.5338 - loss: 1.0761 - val_accuracy: 0.3525 - val_loss: 2.2380
Epoch 30/50
104/104 - 1s - 12ms/step - accuracy: 0.5374 - loss: 1.0744 - val_accuracy: 0.3553 - val_loss: 2.2195
Epoch 31/50
104/104 - 3s - 24ms/step - accuracy: 0.5230 - loss: 1.0672 - val_accuracy: 0.3680 - val_loss: 2.3535
Epoch 32/50
104/104 - 3s - 25ms/step - accuracy: 0.5293 - loss: 1.0764 - val_accuracy: 0.3399 - val_loss: 2.4034
Epoch 33/50
104/104 - 2s - 24ms/step - accuracy: 0.5377 - loss: 1.0444 - val_accuracy: 0.3652 - val_loss: 2.1508
Epoch 34/50
104/104 - 1s - 12ms/step - accuracy: 0.5429 - loss: 1.0224 - val_accuracy: 0.3455 - val_loss: 2.1989
Epoch 35/50
104/104 - 2s - 24ms/step - accuracy: 0.5765 - loss: 0.9808 - val_accuracy: 0.3483 - val_loss: 2.3913
Epoch 36/50
104/104 - 1s - 13ms/step - accuracy: 0.5774 - loss: 0.9716 - val_accuracy: 0.3652 - val_loss: 2.4277
Epoch 37/50
104/104 - 1s - 12ms/step - accuracy: 0.5829 - loss: 0.9731 - val_accuracy: 0.3722 - val_loss: 2.4276
Epoch 38/50
104/104 - 1s - 12ms/step - accuracy: 0.5699 - loss: 0.9800 - val_accuracy: 0.3581 - val_loss: 2.4464
Epoch 39/50
104/104 - 1s - 13ms/step - accuracy: 0.5771 - loss: 0.9834 - val_accuracy: 0.3343 - val_loss: 2.5602
Epoch 40/50
104/104 - 2s - 24ms/step - accuracy: 0.5994 - loss: 0.9377 - val_accuracy: 0.3427 - val_loss: 2.7252
Epoch 41/50
104/104 - 1s - 12ms/step - accuracy: 0.5997 - loss: 0.9203 - val_accuracy: 0.3497 - val_loss: 2.5780
Epoch 42/50
104/104 - 1s - 12ms/step - accuracy: 0.6298 - loss: 0.8677 - val_accuracy: 0.3455 - val_loss: 2.8309
Epoch 43/50
104/104 - 1s - 12ms/step - accuracy: 0.6523 - loss: 0.8475 - val_accuracy: 0.3553 - val_loss: 2.9278
Epoch 44/50
104/104 - 1s - 12ms/step - accuracy: 0.6415 - loss: 0.8627 - val_accuracy: 0.3722 - val_loss: 2.8745
Epoch 45/50
104/104 - 1s - 12ms/step - accuracy: 0.6235 - loss: 0.9178 - val_accuracy: 0.3764 - val_loss: 2.9087
Epoch 46/50
104/104 - 1s - 12ms/step - accuracy: 0.6042 - loss: 0.9236 - val_accuracy: 0.3553 - val_loss: 2.8286
Epoch 47/50
104/104 - 1s - 12ms/step - accuracy: 0.6325 - loss: 0.8793 - val_accuracy: 0.3848 - val_loss: 2.6239
Epoch 48/50
104/104 - 1s - 12ms/step - accuracy: 0.6493 - loss: 0.8115 - val_accuracy: 0.3497 - val_loss: 2.9943
Epoch 49/50
104/104 - 3s - 25ms/step - accuracy: 0.6725 - loss: 0.7891 - val_accuracy: 0.3750 - val_loss: 2.9498
Epoch 50/50
104/104 - 2s - 23ms/step - accuracy: 0.6761 - loss: 0.7900 - val_accuracy: 0.3778 - val_loss: 2.8940
```

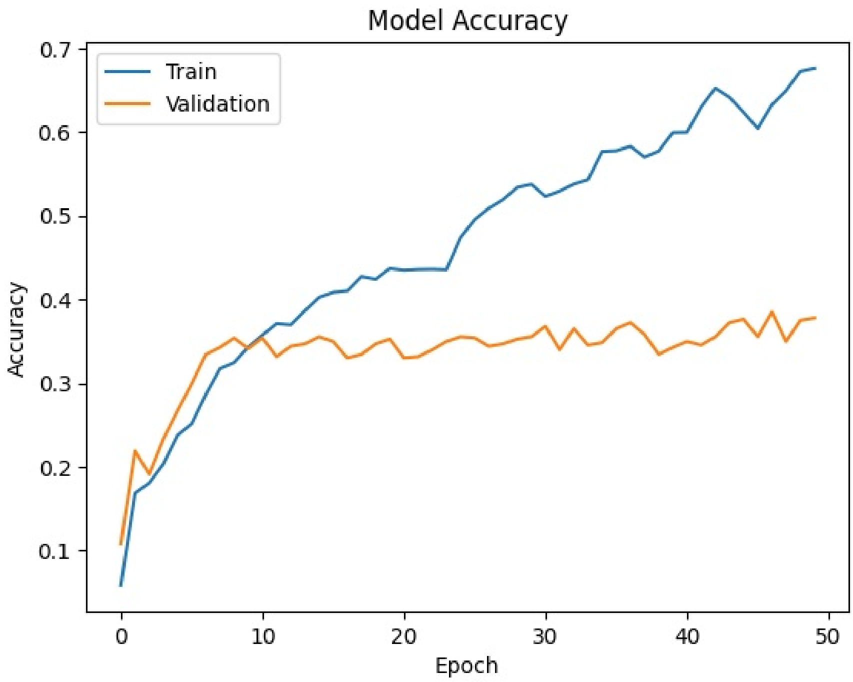
Model Evaluation

In [ ]:

```
plot_loss(history1)
```

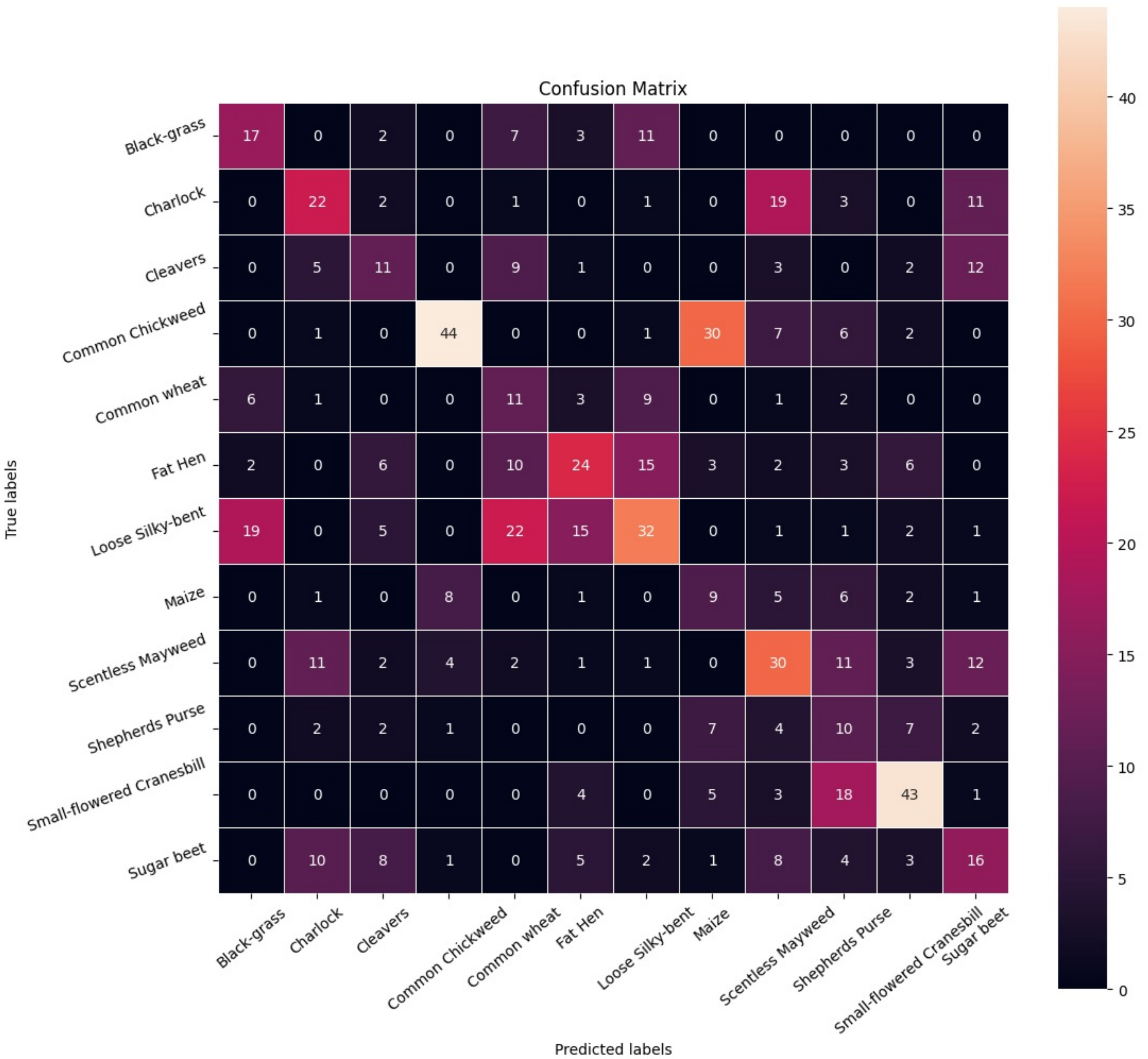



```
In [ ]: plot_accuracy(history1)
```




```
In [ ]: plot_confusion_matrix_and_report(model1, x_val_normalized, y_val_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```


Confusion Matrix
True labels
Predicted labels
Black-grass
Charlock
Cleavers
Common Chickweed
Common wheat
Fat Hen
Loose Silky-bent
Maize
Scentless Mayweed
Shepherds Purse
Small-flowered Cranesbill
Sugar beet
17 0 2 0 7 3 11 0 0 0 0 0
0 22 2 0 1 0 1 0 19 3 0 11
0 5 11 0 9 1 0 0 3 0 2 12
0 1 0 44 0 0 1 30 7 6 2 0
6 1 0 0 11 3 9 0 1 2 0 0
2 0 6 0 10 24 15 3 2 3 6 0
19 0 5 0 22 15 32 0 1 1 2 1
0 1 0 8 0 1 0 9 5 6 2 1
0 11 2 4 2 1 1 0 30 11 3 12
0 2 2 1 0 0 0 7 4 10 7 2
0 0 0 0 0 4 0 5 3 18 43 1
0 10 8 1 0 5 2 1 8 4 3 16
0
5
10
15
20
25
30
35
40

```
              precision    recall  f1-score   support

           0       0.39      0.42      0.40        40
           1       0.42      0.37      0.39        59
           2       0.29      0.26      0.27        43
           3       0.76      0.48      0.59        91
           4       0.18      0.33      0.23        33
           5       0.42      0.34      0.38        71
           6       0.44      0.33      0.38        98
           7       0.16      0.27      0.20        33
           8       0.36      0.39      0.38        77
           9       0.16      0.29      0.20        35
          10       0.61      0.58      0.60        74
          11       0.29      0.28      0.28        58

    accuracy                           0.38       712
   macro avg       0.37      0.36      0.36       712
weighted avg       0.42      0.38      0.39       712
```

Observations

Loss and Accuracy Plots:

- The training loss consistently decreases, and training accuracy steadily increases, indicating the model is learning the training data well. Training accuracy reaches **~70%** by the end of 50 epochs.
- Validation loss decreases initially but starts to oscillate and increase after ~25 epochs, indicating overfitting. Validation accuracy plateaus at **~38%**, showing poor generalization to the validation set.

Confusion Matrix:

- The model shows significant confusion among classes, especially for classes like:
  - `Loose Silky-bent` is predicted as `Fat Hen` or `Common Chickweed` frequently.
  - `Scentless Mayweed` is confused with `Shepherds Purse` and `Small-flowered Cranesbill`.
- Some classes (e.g., `Common Chickweed` and `Loose Silky-bent`) are predicted relatively better, but others have poor recall.

Classification Report:

- Average precision: **0.42** (low, indicating many false positives).
- Average recall: **0.36** (low, indicating many false negatives).
- Weighted F1-score: **0.39**, showing imbalanced performance across classes.
- Certain classes like `Cleavers` and `Small-flowered Cranesbill` perform relatively better but are still far from optimal.
- Some classes (e.g., `Scentless Mayweed`, `Loose Silky-bent`) have very low recall and F1-scores.

Issues with this Model:

1. **Overfitting**: Training accuracy is significantly higher than validation accuracy, and validation loss increases over epochs. The model is overfitting to the training data.

2. **Class Imbalance**: The model struggles to handle class imbalance effectively, despite using `class_weight`. This is evident in the poor recall and precision for minority classes.

3. **Model Complexity**: The current architecture may not have sufficient capacity to capture complex features in the dataset. A deeper model with more layers or filters may help.

4. **No Regularization**: No techniques like Batch Normalization or Dropout are applied, leading to overfitting.

Model Performance Improvement

Second Iteration

Improvement Opinions

- Add BatchNormalization after convolutional layers to stabilize learning.
- Add more convolutional layers or increase the number of filters in existing layers to allow the model to capture more complex patterns
- Use a learning rate of 1e-4 instead of the default 1e-3 to allow finer adjustments to weights.

In [ ]:
```
# Building the second model
model2 = Sequential([
    # First Conv2D layer
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(64, 64, 3)),
```

```
    # Second Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # First MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # Third Conv2D layer <-- Added New
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # Fourth Conv2D layer
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),

    # Second MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # BatchNormalization <-- Added New
    BatchNormalization(),

    # Flatten the output
    Flatten(),

    # First Dense layer with 16 neurons
    Dense(16, activation='relu'),

    # Output layer with 12 neurons for 12 classes
    Dense(12, activation='softmax')
])

# Compile the model
model2.compile(optimizer=Adam(learning_rate=1e-4),# Reduced learning_rate=0.0005 <-- Added new
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Print the model summary
model2.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 64, 64, 32)	896
conv2d_5 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36,928
conv2d_7 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
flatten_1 (Flatten)	(None, 32768)	0
dense_2 (Dense)	(None, 16)	524,304
dense_3 (Dense)	(None, 12)	204

Total params: 655,196 (2.50 MB)

Trainable params: 654,940 (2.50 MB)

Non-trainable params: 256 (1.00 KB)

In [ ]:

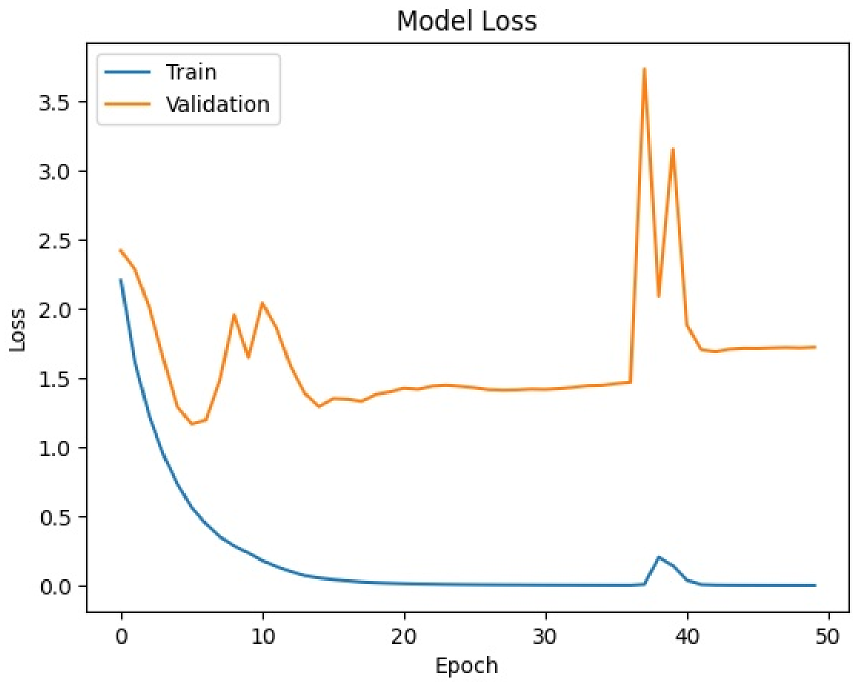
```
# Resetting the environment default seed=1
reset_environment()

# Fit the model
history2 = model2.fit(
            x_train_normalized,
            y_train_encoded,
            epochs=50,
            validation_data=(x_val_normalized,y_val_encoded),
            batch_size=32,
            class_weight=class_weights_dict,
            verbose=2
)
```

```
Epoch 1/50
104/104 - 10s - 97ms/step - accuracy: 0.2162 - loss: 2.2047 - val_accuracy: 0.1854 - val_loss: 2.4180
```

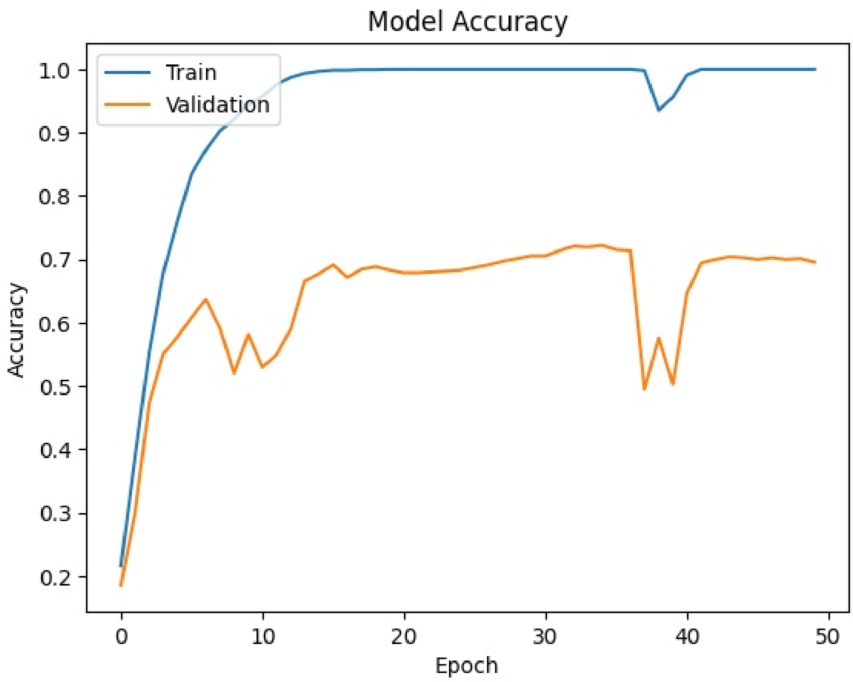
```
Epoch 2/50
104/104 - 2s - 15ms/step - accuracy: 0.3874 - loss: 1.6098 - val_accuracy: 0.2992 - val_loss: 2.2801
Epoch 3/50
104/104 - 2s - 15ms/step - accuracy: 0.5531 - loss: 1.2259 - val_accuracy: 0.4733 - val_loss: 2.0092
Epoch 4/50
104/104 - 2s - 15ms/step - accuracy: 0.6788 - loss: 0.9448 - val_accuracy: 0.5506 - val_loss: 1.6380
Epoch 5/50
104/104 - 2s - 24ms/step - accuracy: 0.7603 - loss: 0.7325 - val_accuracy: 0.5772 - val_loss: 1.2894
Epoch 6/50
104/104 - 3s - 24ms/step - accuracy: 0.8352 - loss: 0.5630 - val_accuracy: 0.6081 - val_loss: 1.1665
Epoch 7/50
104/104 - 2s - 24ms/step - accuracy: 0.8722 - loss: 0.4467 - val_accuracy: 0.6362 - val_loss: 1.1952
Epoch 8/50
104/104 - 1s - 14ms/step - accuracy: 0.9023 - loss: 0.3515 - val_accuracy: 0.5913 - val_loss: 1.4839
Epoch 9/50
104/104 - 2s - 15ms/step - accuracy: 0.9209 - loss: 0.2861 - val_accuracy: 0.5197 - val_loss: 1.9568
Epoch 10/50
104/104 - 3s - 25ms/step - accuracy: 0.9426 - loss: 0.2348 - val_accuracy: 0.5815 - val_loss: 1.6466
Epoch 11/50
104/104 - 2s - 15ms/step - accuracy: 0.9570 - loss: 0.1802 - val_accuracy: 0.5295 - val_loss: 2.0402
Epoch 12/50
104/104 - 2s - 24ms/step - accuracy: 0.9753 - loss: 0.1371 - val_accuracy: 0.5492 - val_loss: 1.8564
Epoch 13/50
104/104 - 3s - 25ms/step - accuracy: 0.9868 - loss: 0.1007 - val_accuracy: 0.5899 - val_loss: 1.5816
Epoch 14/50
104/104 - 3s - 24ms/step - accuracy: 0.9931 - loss: 0.0711 - val_accuracy: 0.6657 - val_loss: 1.3870
Epoch 15/50
104/104 - 2s - 15ms/step - accuracy: 0.9964 - loss: 0.0544 - val_accuracy: 0.6770 - val_loss: 1.2908
Epoch 16/50
104/104 - 3s - 25ms/step - accuracy: 0.9982 - loss: 0.0427 - val_accuracy: 0.6910 - val_loss: 1.3511
Epoch 17/50
104/104 - 2s - 24ms/step - accuracy: 0.9982 - loss: 0.0342 - val_accuracy: 0.6713 - val_loss: 1.3461
Epoch 18/50
104/104 - 1s - 14ms/step - accuracy: 0.9991 - loss: 0.0261 - val_accuracy: 0.6840 - val_loss: 1.3300
Epoch 19/50
104/104 - 3s - 25ms/step - accuracy: 0.9991 - loss: 0.0202 - val_accuracy: 0.6882 - val_loss: 1.3796
Epoch 20/50
104/104 - 2s - 24ms/step - accuracy: 0.9997 - loss: 0.0167 - val_accuracy: 0.6826 - val_loss: 1.3975
Epoch 21/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0140 - val_accuracy: 0.6784 - val_loss: 1.4246
Epoch 22/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0119 - val_accuracy: 0.6784 - val_loss: 1.4164
Epoch 23/50
104/104 - 2s - 24ms/step - accuracy: 0.9997 - loss: 0.0107 - val_accuracy: 0.6798 - val_loss: 1.4395
Epoch 24/50
104/104 - 3s - 25ms/step - accuracy: 0.9997 - loss: 0.0092 - val_accuracy: 0.6812 - val_loss: 1.4460
Epoch 25/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0081 - val_accuracy: 0.6826 - val_loss: 1.4377
Epoch 26/50
104/104 - 3s - 24ms/step - accuracy: 0.9997 - loss: 0.0072 - val_accuracy: 0.6868 - val_loss: 1.4278
Epoch 27/50
104/104 - 3s - 25ms/step - accuracy: 0.9997 - loss: 0.0065 - val_accuracy: 0.6910 - val_loss: 1.4128
Epoch 28/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0059 - val_accuracy: 0.6966 - val_loss: 1.4098
Epoch 29/50
104/104 - 2s - 16ms/step - accuracy: 0.9997 - loss: 0.0056 - val_accuracy: 0.7008 - val_loss: 1.4114
Epoch 30/50
104/104 - 2s - 23ms/step - accuracy: 0.9997 - loss: 0.0049 - val_accuracy: 0.7051 - val_loss: 1.4172
Epoch 31/50
104/104 - 3s - 24ms/step - accuracy: 0.9997 - loss: 0.0044 - val_accuracy: 0.7051 - val_loss: 1.4150
Epoch 32/50
104/104 - 3s - 24ms/step - accuracy: 0.9997 - loss: 0.0040 - val_accuracy: 0.7135 - val_loss: 1.4217
Epoch 33/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0037 - val_accuracy: 0.7205 - val_loss: 1.4312
Epoch 34/50
104/104 - 3s - 25ms/step - accuracy: 0.9997 - loss: 0.0034 - val_accuracy: 0.7191 - val_loss: 1.4418
Epoch 35/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0032 - val_accuracy: 0.7219 - val_loss: 1.4453
Epoch 36/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0029 - val_accuracy: 0.7149 - val_loss: 1.4582
Epoch 37/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0027 - val_accuracy: 0.7135 - val_loss: 1.4672
Epoch 38/50
104/104 - 3s - 24ms/step - accuracy: 0.9976 - loss: 0.0091 - val_accuracy: 0.4944 - val_loss: 3.7317
Epoch 39/50
104/104 - 2s - 15ms/step - accuracy: 0.9347 - loss: 0.2042 - val_accuracy: 0.5758 - val_loss: 2.0891
Epoch 40/50
104/104 - 2s - 24ms/step - accuracy: 0.9555 - loss: 0.1440 - val_accuracy: 0.5028 - val_loss: 3.1505
Epoch 41/50
104/104 - 2s - 15ms/step - accuracy: 0.9904 - loss: 0.0384 - val_accuracy: 0.6461 - val_loss: 1.8803
Epoch 42/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0077 - val_accuracy: 0.6938 - val_loss: 1.7021
Epoch 43/50
```

```
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0042 - val_accuracy: 0.6994 - val_loss: 1.6878
Epoch 44/50
104/104 - 1s - 14ms/step - accuracy: 0.9997 - loss: 0.0033 - val_accuracy: 0.7037 - val_loss: 1.7050
Epoch 45/50
104/104 - 1s - 14ms/step - accuracy: 0.9997 - loss: 0.0028 - val_accuracy: 0.7022 - val_loss: 1.7121
Epoch 46/50
104/104 - 1s - 14ms/step - accuracy: 0.9997 - loss: 0.0026 - val_accuracy: 0.6994 - val_loss: 1.7112
Epoch 47/50
104/104 - 3s - 25ms/step - accuracy: 0.9997 - loss: 0.0023 - val_accuracy: 0.7022 - val_loss: 1.7150
Epoch 48/50
104/104 - 2s - 24ms/step - accuracy: 0.9997 - loss: 0.0021 - val_accuracy: 0.6994 - val_loss: 1.7178
Epoch 49/50
104/104 - 3s - 25ms/step - accuracy: 0.9997 - loss: 0.0020 - val_accuracy: 0.7008 - val_loss: 1.7151
Epoch 50/50
104/104 - 2s - 15ms/step - accuracy: 0.9997 - loss: 0.0019 - val_accuracy: 0.6952 - val_loss: 1.7200
```

In [ ]:
```
plot_loss(history2)
```

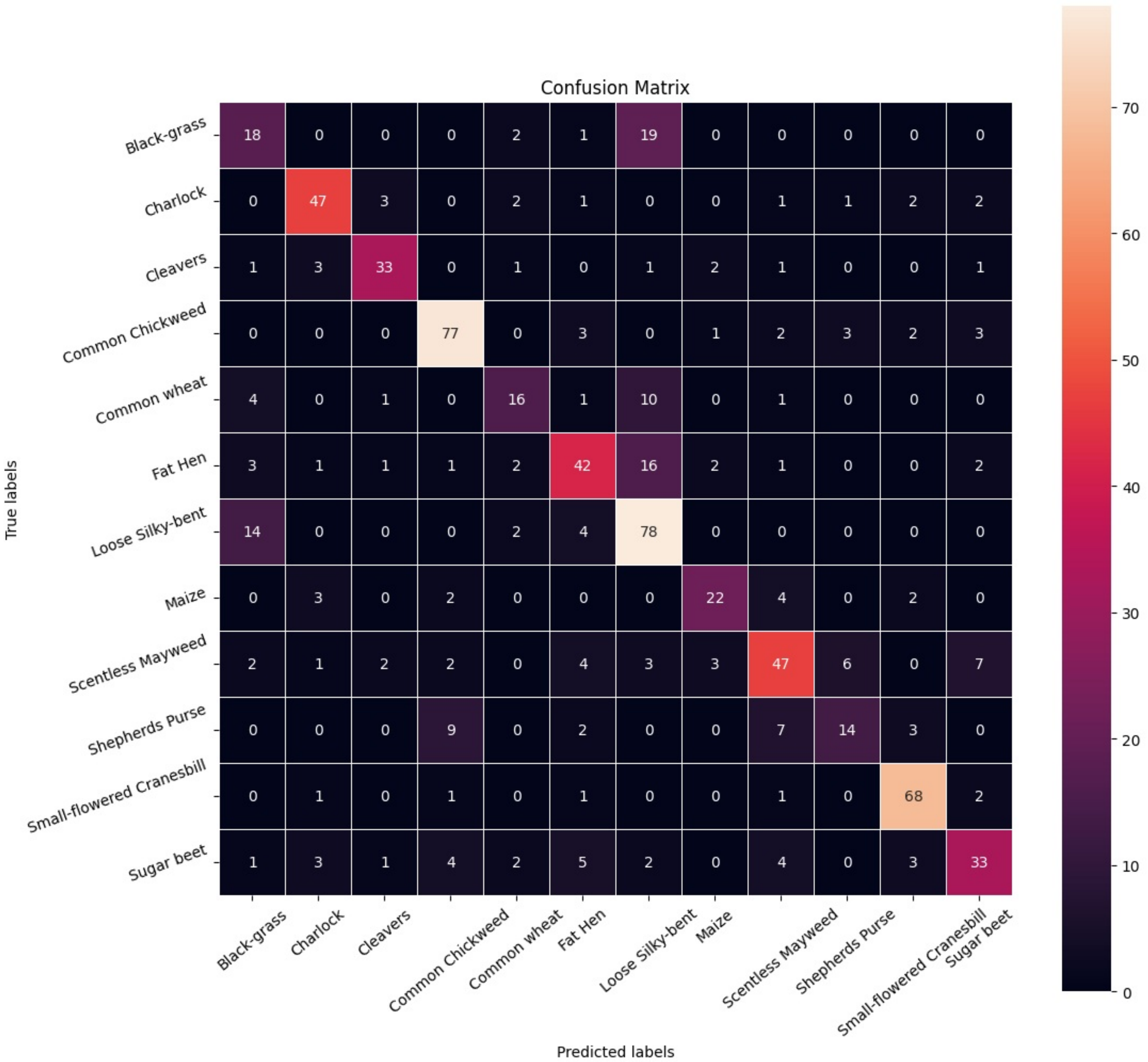


In [ ]:
```
plot_accuracy(history2)
```



In [ ]:
```
plot_confusion_matrix_and_report(model2, x_val_normalized, y_val_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step
```



	precision	recall	f1-score	support
0	0.42	0.45	0.43	40
1	0.80	0.80	0.80	59
2	0.80	0.77	0.79	43
3	0.80	0.85	0.82	91
4	0.59	0.48	0.53	33
5	0.66	0.59	0.62	71
6	0.60	0.80	0.69	98
7	0.73	0.67	0.70	33
8	0.68	0.61	0.64	77
9	0.58	0.40	0.47	35
10	0.85	0.92	0.88	74
11	0.66	0.57	0.61	58
accuracy			0.70	712
macro avg	0.68	0.66	0.67	712
weighted avg	0.70	0.70	0.69	712

Observations

Loss and Accuracy Plots:

- Training loss decreases consistently to near **0.0**, and training accuracy reaches **~1.0**. This shows that the model is fitting the training data very well, indicating a potential risk of overfitting.
- Validation loss decreases initially but fluctuates and rises after around 25 epochs. Validation accuracy stabilizes at **~70%**, which is a

significant improvement compared to the first model.

Confusion Matrix:

- The confusion matrix shows a **better spread of predictions across classes** compared to the first model.
- Classes such as `Loose Silky-bent`, `Common Chickweed`, and `Small-flowered Cranesbill` are predicted more accurately. Misclassifications are still present but significantly reduced.
- Some confusions persist, for example:
- `Loose Silky-bent` is occasionally misclassified as `Black-grass`.
- `Scentless Mayweed` is confused with `Shepherds Purse`.

Classification Report:

- Precision: **0.68** (macro avg), up from **0.42** in the first model.
- Recall: **0.66** (macro avg), up from **0.36** in the first model.
- Weighted F1-score: **0.69**, showing much better balance across classes.
- Most classes show significant improvement in precision and recall.
- `Cleavers`, `Loose Silky-bent`, and `Small-flowered Cranesbill` have F1-scores over **0.70**, while underrepresented classes like `Maize` and `Shepherds Purse` still need improvement.

Improvements Made:

1. **Model Complexity**: Shifting to fourth `Conv2D` layer by adding extra `Conv2D` with **64 filters** improved the model's feature extraction capability.

2. **Batch Normalization**: The addition of `BatchNormalization` stabilized training and helped generalization, contributing to better validation performance.

3. **Learning Rate Adjustment**: Lowering the learning rate to `1e-4` allowed the model to converge more effectively, resulting in improved validation accuracy.

Still exists issues:

1. **Overfitting**: Training accuracy is significantly higher than validation accuracy, and validation loss fluctuates and increases after ~25 epochs. This indicates overfitting, despite the improvements.

2. **Misclassifications**: Certain confusions persist, especially for underrepresented or visually similar classes.

3. **Class Imbalance**: Class imbalance is still a challenge, as seen in the relatively lower performance for minority classes like `Maize` and `Shepherds Purse`.

Third Iteration

Improvement Options

- Data Augmentation - Apply ImageDataGenerator to artificially increase the variability of training data and improve generalization.
- Add another dense layer before the output layer or increase neurons in the existing dense layer.
- Add Dropout layers after dense layers and convolutional layers to reduce overfitting.

In [ ]:

```
# Creating & Fitting DataGenerator <-- Added new

augmented_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
augmented_datagen.fit(x_train_normalized)




# Building the third model
model3 = Sequential([
    # First Conv2D layer
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(64, 64, 3)),

    # Second Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # First MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),
```

```
    # Third Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # Fourth Conv2D layer
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),

    # Second MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # First Dropout Layer<-- Added New
    Dropout(0.25),


    # BatchNormalization
    BatchNormalization(),

    # Flatten the output
    Flatten(),

    # First Dense layer with 16 neurons
    Dense(16, activation='relu'),

    # Second Dense layer with 32 neurons <-- Added New
    Dense(32, activation='relu'),

    # Second Dropout Layer<-- Added New
    Dropout(0.25),

    # Output layer with 12 neurons for 12 classes
    Dense(12, activation='softmax')
])

# Compile the model
model3.compile(optimizer=Adam(learning_rate=1e-4),# Reduced learning_rate=0.0005
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Print the model summary
model3.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 64, 64, 32)	896
conv2d_5 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36,928
conv2d_7 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_2 (Dropout)	(None, 16, 16, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
flatten_1 (Flatten)	(None, 32768)	0
dense_3 (Dense)	(None, 16)	524,304
dense_4 (Dense)	(None, 32)	544
dropout_3 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 12)	396

Total params: 655,932 (2.50 MB)

Trainable params: 655,676 (2.50 MB)

Non-trainable params: 256 (1.00 KB)

In [ ]:

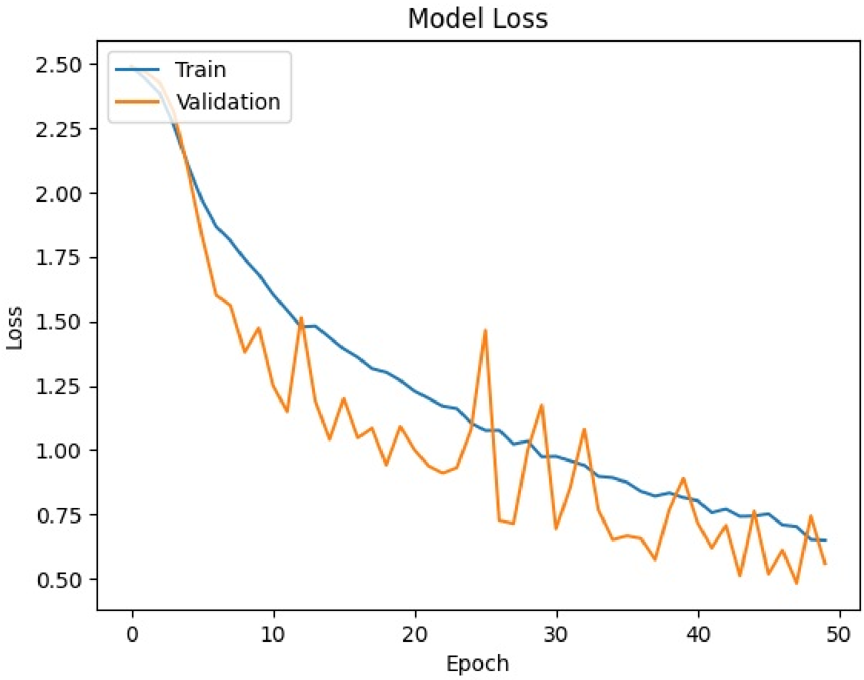
```
# Resetting the environment default seed=1
reset_environment()

# Fit the model
history3 = model3.fit(
          augmented_datagen.flow( # Data Augmentation <-- Added New
```

```
            x_train_normalized,
            y_train_encoded,
            batch_size=32),
        epochs=50,
        validation_data=(x_val_normalized,
                         y_val_encoded),
        class_weight=class_weights_dict,
        verbose=2)
```

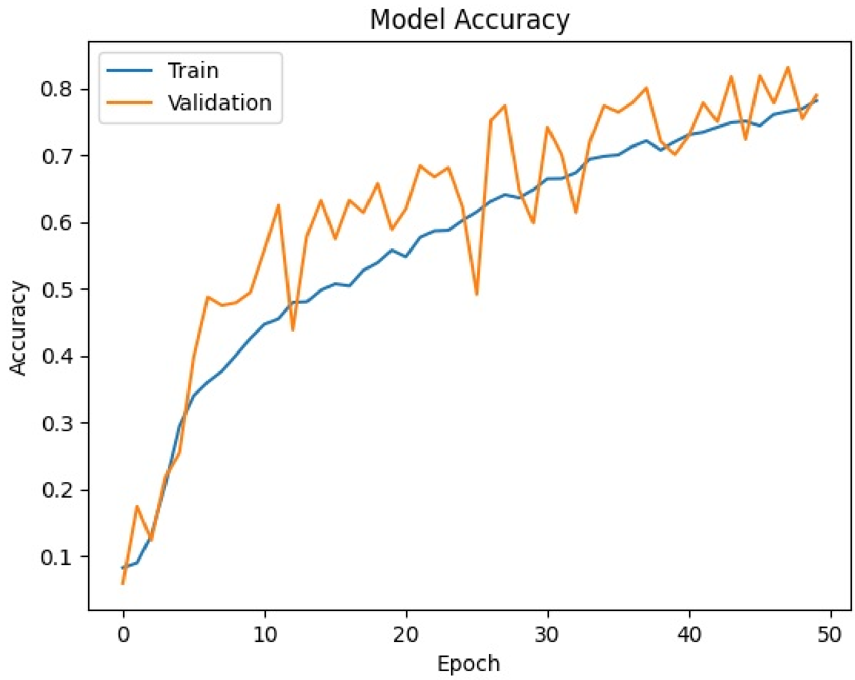
```
Epoch 1/50
104/104 - 25s - 238ms/step - accuracy: 0.0824 - loss: 2.4888 - val_accuracy: 0.0590 - val_loss: 2.4854
Epoch 2/50
104/104 - 24s - 233ms/step - accuracy: 0.0896 - loss: 2.4392 - val_accuracy: 0.1742 - val_loss: 2.4669
Epoch 3/50
104/104 - 4s - 38ms/step - accuracy: 0.1308 - loss: 2.3840 - val_accuracy: 0.1236 - val_loss: 2.4258
Epoch 4/50
104/104 - 6s - 59ms/step - accuracy: 0.2066 - loss: 2.2591 - val_accuracy: 0.2177 - val_loss: 2.3125
Epoch 5/50
104/104 - 5s - 45ms/step - accuracy: 0.2941 - loss: 2.1051 - val_accuracy: 0.2542 - val_loss: 2.0872
Epoch 6/50
104/104 - 4s - 38ms/step - accuracy: 0.3398 - loss: 1.9675 - val_accuracy: 0.3975 - val_loss: 1.8269
Epoch 7/50
104/104 - 4s - 42ms/step - accuracy: 0.3603 - loss: 1.8671 - val_accuracy: 0.4874 - val_loss: 1.6008
Epoch 8/50
104/104 - 5s - 52ms/step - accuracy: 0.3765 - loss: 1.8125 - val_accuracy: 0.4747 - val_loss: 1.5606
Epoch 9/50
104/104 - 10s - 91ms/step - accuracy: 0.4006 - loss: 1.7406 - val_accuracy: 0.4789 - val_loss: 1.3795
Epoch 10/50
104/104 - 5s - 49ms/step - accuracy: 0.4256 - loss: 1.6812 - val_accuracy: 0.4944 - val_loss: 1.4736
Epoch 11/50
104/104 - 10s - 98ms/step - accuracy: 0.4466 - loss: 1.6049 - val_accuracy: 0.5576 - val_loss: 1.2525
Epoch 12/50
104/104 - 5s - 47ms/step - accuracy: 0.4550 - loss: 1.5417 - val_accuracy: 0.6250 - val_loss: 1.1483
Epoch 13/50
104/104 - 4s - 38ms/step - accuracy: 0.4794 - loss: 1.4777 - val_accuracy: 0.4382 - val_loss: 1.5128
Epoch 14/50
104/104 - 6s - 61ms/step - accuracy: 0.4803 - loss: 1.4814 - val_accuracy: 0.5787 - val_loss: 1.1904
Epoch 15/50
104/104 - 4s - 41ms/step - accuracy: 0.4977 - loss: 1.4364 - val_accuracy: 0.6320 - val_loss: 1.0410
Epoch 16/50
104/104 - 4s - 37ms/step - accuracy: 0.5071 - loss: 1.3926 - val_accuracy: 0.5744 - val_loss: 1.2002
Epoch 17/50
104/104 - 5s - 45ms/step - accuracy: 0.5041 - loss: 1.3596 - val_accuracy: 0.6320 - val_loss: 1.0475
Epoch 18/50
104/104 - 5s - 47ms/step - accuracy: 0.5275 - loss: 1.3164 - val_accuracy: 0.6138 - val_loss: 1.0857
Epoch 19/50
104/104 - 5s - 45ms/step - accuracy: 0.5389 - loss: 1.3017 - val_accuracy: 0.6573 - val_loss: 0.9416
Epoch 20/50
104/104 - 5s - 48ms/step - accuracy: 0.5576 - loss: 1.2704 - val_accuracy: 0.5885 - val_loss: 1.0916
Epoch 21/50
104/104 - 4s - 43ms/step - accuracy: 0.5477 - loss: 1.2291 - val_accuracy: 0.6194 - val_loss: 1.0011
Epoch 22/50
104/104 - 4s - 37ms/step - accuracy: 0.5768 - loss: 1.2007 - val_accuracy: 0.6840 - val_loss: 0.9379
Epoch 23/50
104/104 - 8s - 80ms/step - accuracy: 0.5865 - loss: 1.1704 - val_accuracy: 0.6671 - val_loss: 0.9104
Epoch 24/50
104/104 - 4s - 38ms/step - accuracy: 0.5874 - loss: 1.1610 - val_accuracy: 0.6812 - val_loss: 0.9318
Epoch 25/50
104/104 - 4s - 39ms/step - accuracy: 0.6021 - loss: 1.1050 - val_accuracy: 0.6222 - val_loss: 1.0808
Epoch 26/50
104/104 - 6s - 56ms/step - accuracy: 0.6150 - loss: 1.0763 - val_accuracy: 0.4916 - val_loss: 1.4650
Epoch 27/50
104/104 - 4s - 37ms/step - accuracy: 0.6307 - loss: 1.0768 - val_accuracy: 0.7514 - val_loss: 0.7269
Epoch 28/50
104/104 - 5s - 49ms/step - accuracy: 0.6406 - loss: 1.0228 - val_accuracy: 0.7739 - val_loss: 0.7146
Epoch 29/50
104/104 - 7s - 63ms/step - accuracy: 0.6358 - loss: 1.0350 - val_accuracy: 0.6475 - val_loss: 1.0014
Epoch 30/50
104/104 - 10s - 93ms/step - accuracy: 0.6481 - loss: 0.9739 - val_accuracy: 0.5983 - val_loss: 1.1749
Epoch 31/50
104/104 - 5s - 47ms/step - accuracy: 0.6644 - loss: 0.9754 - val_accuracy: 0.7416 - val_loss: 0.6945
Epoch 32/50
104/104 - 4s - 37ms/step - accuracy: 0.6647 - loss: 0.9579 - val_accuracy: 0.7008 - val_loss: 0.8546
Epoch 33/50
104/104 - 4s - 40ms/step - accuracy: 0.6737 - loss: 0.9408 - val_accuracy: 0.6138 - val_loss: 1.0805
Epoch 34/50
104/104 - 6s - 56ms/step - accuracy: 0.6938 - loss: 0.8978 - val_accuracy: 0.7205 - val_loss: 0.7687
Epoch 35/50
104/104 - 4s - 38ms/step - accuracy: 0.6980 - loss: 0.8925 - val_accuracy: 0.7739 - val_loss: 0.6521
Epoch 36/50
104/104 - 4s - 42ms/step - accuracy: 0.7002 - loss: 0.8748 - val_accuracy: 0.7640 - val_loss: 0.6673
Epoch 37/50
104/104 - 6s - 54ms/step - accuracy: 0.7128 - loss: 0.8401 - val_accuracy: 0.7781 - val_loss: 0.6574
Epoch 38/50
```

```
104/104 - 4s - 40ms/step - accuracy: 0.7212 - loss: 0.8209 - val_accuracy: 0.8006 - val_loss: 0.5743
Epoch 39/50
104/104 - 7s - 65ms/step - accuracy: 0.7074 - loss: 0.8336 - val_accuracy: 0.7205 - val_loss: 0.7702
Epoch 40/50
104/104 - 4s - 40ms/step - accuracy: 0.7200 - loss: 0.8153 - val_accuracy: 0.7008 - val_loss: 0.8896
Epoch 41/50
104/104 - 4s - 38ms/step - accuracy: 0.7305 - loss: 0.8041 - val_accuracy: 0.7289 - val_loss: 0.7188
Epoch 42/50
104/104 - 5s - 49ms/step - accuracy: 0.7341 - loss: 0.7577 - val_accuracy: 0.7781 - val_loss: 0.6198
Epoch 43/50
104/104 - 5s - 46ms/step - accuracy: 0.7417 - loss: 0.7715 - val_accuracy: 0.7500 - val_loss: 0.7079
Epoch 44/50
104/104 - 4s - 40ms/step - accuracy: 0.7486 - loss: 0.7428 - val_accuracy: 0.8174 - val_loss: 0.5114
Epoch 45/50
104/104 - 5s - 45ms/step - accuracy: 0.7507 - loss: 0.7442 - val_accuracy: 0.7233 - val_loss: 0.7640
Epoch 46/50
104/104 - 5s - 49ms/step - accuracy: 0.7438 - loss: 0.7523 - val_accuracy: 0.8188 - val_loss: 0.5182
Epoch 47/50
104/104 - 4s - 37ms/step - accuracy: 0.7609 - loss: 0.7099 - val_accuracy: 0.7781 - val_loss: 0.6114
Epoch 48/50
104/104 - 7s - 65ms/step - accuracy: 0.7654 - loss: 0.7031 - val_accuracy: 0.8315 - val_loss: 0.4837
Epoch 49/50
104/104 - 9s - 84ms/step - accuracy: 0.7693 - loss: 0.6529 - val_accuracy: 0.7542 - val_loss: 0.7448
Epoch 50/50
104/104 - 7s - 66ms/step - accuracy: 0.7814 - loss: 0.6502 - val_accuracy: 0.7893 - val_loss: 0.5601
```

In [ ]:
```
plot_loss(history3)
```

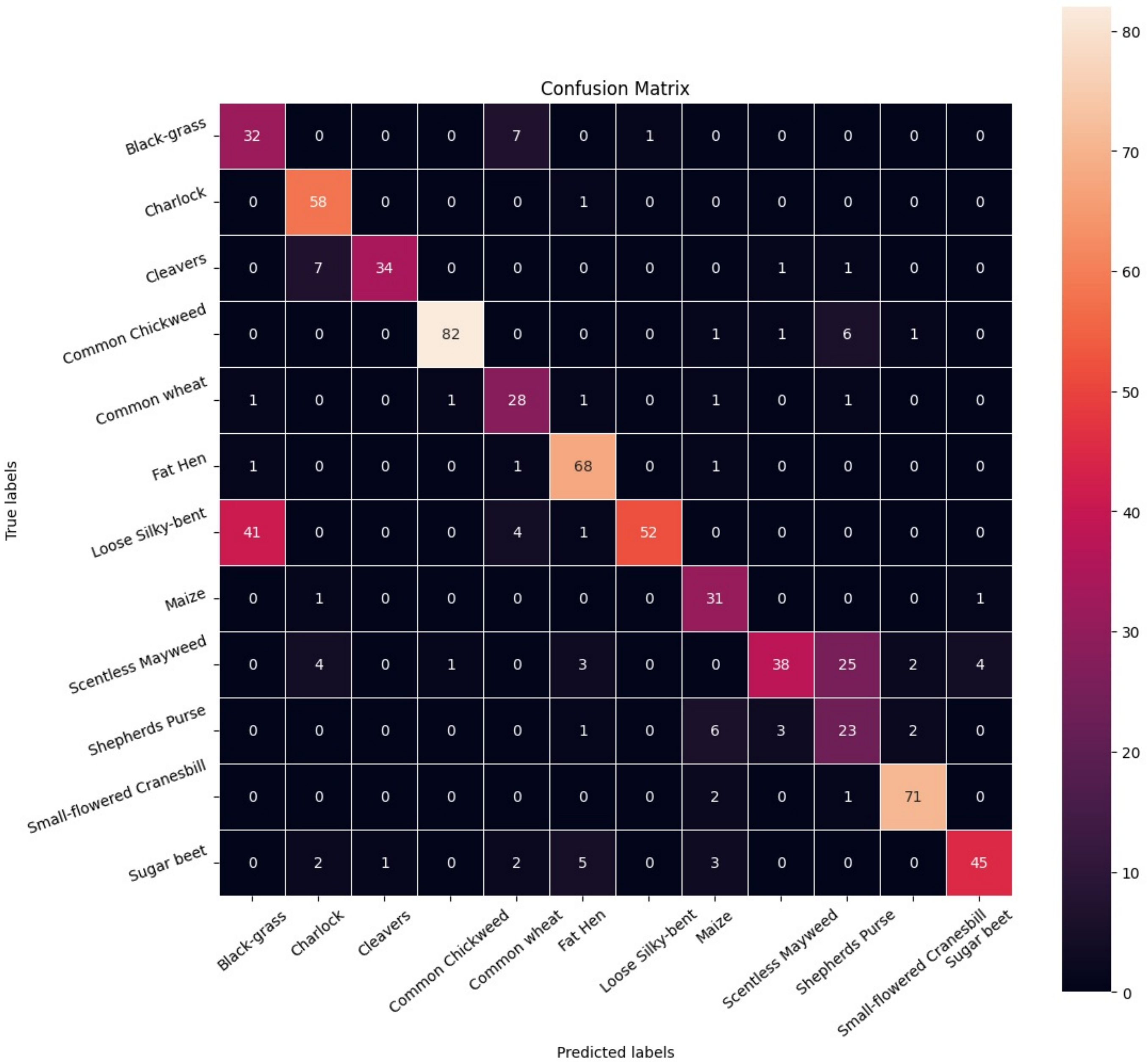


In [ ]:
```
plot_accuracy(history3)
```




```
In [ ]: plot_confusion_matrix_and_report(model3, x_val_normalized, y_val_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step
```



```
              precision    recall  f1-score   support

           0       0.43      0.80      0.56        40
           1       0.81      0.98      0.89        59
           2       0.97      0.79      0.87        43
           3       0.98      0.90      0.94        91
           4       0.67      0.85      0.75        33
           5       0.85      0.96      0.90        71
           6       0.98      0.53      0.69        98
           7       0.69      0.94      0.79        33
           8       0.88      0.49      0.63        77
           9       0.40      0.66      0.50        35
          10       0.93      0.96      0.95        74
          11       0.90      0.78      0.83        58

    accuracy                           0.79       712
   macro avg       0.79      0.80      0.77       712
weighted avg       0.84      0.79      0.79       712
```

Observations

Loss and Accuracy Plots:

- Training loss steadily decreases, indicating the model is learning effectively from the augmented data. Training accuracy gradually increases, stabilizing around **79%**, which shows good learning without overfitting.
- Validation loss follows a similar decreasing trend, with occasional spikes due to the augmented data variability. Validation accuracy stabilizes around **~79%**, indicating significant improvement over previous models and good generalization.

Confusion Matrix:

- The predictions are better distributed across all 12 classes. Classes such as `Common Chickweed`, `Loose Silky-bent`, and `Small-flowered Cranesbill` show significant improvement in correct classifications.
- Some classes, such as `Loose Silky-bent`, are still misclassified as `Black-grass` or `Common Chickweed`.
- Underrepresented classes like `Shepherds Purse` and `Scentless Mayweed` show moderate confusion with possibly closely classes.

Classification Report:

- **Precision**: 0.79 (macro average), up from 0.68 in Model2.
- **Recall**: 0.80 (macro average), up from 0.66 in Model2.
- **F1-score**: 0.79 (macro average), showing balanced performance across classes.
- Minority classes like `Cleavers` (F1-score 0.87), `Small-flowered Cranesbill` (F1-score 0.95), and `Sugar beet` (F1-score 0.83) show improvement. Most classes have F1-scores above **0.70**, indicating consistent and balanced performance.

Improvements Made:

1. **Data Augmentation**: The use of `ImageDataGenerator` introduced variability in the training data, leading to better generalization and improved validation accuracy.
2. **Regularization**: Adding **Dropout layers** reduced overfitting and helped improve validation performance.
3. **Complexity**: Adding an extra dense layer and increasing the number of neurons allowed the model to capture more complex patterns.

Strengths:

- **Generalization**: Validation accuracy improved significantly to match training accuracy.
- **Balanced Performance**: The F1-scores across classes indicate that the model performs well for both majority and minority classes.
- **Reduced Overfitting**: The training and validation curves are more aligned, showing reduced overfitting compared to previous models.

Fourth Iteration

Improvement Options

- Increasing number of epoch to 75 to check if the model trains more on the training dataset.
- Use of `ReduceLROnPlateau` with a factor=0.1, patience=5 and min learning rate up to **1e-6** (0.000001)
- Also intoducing early stopping if loss does not decrease over 10 epochs.

In [ ]:
```
# Creating & Fitting DataGenerator

augmented_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
augmented_datagen.fit(x_train_normalized)

# Define ReduceLROnPlateau callback <-- Added New
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',        # Monitor validation loss
    factor=0.1,                # Reduce learning rate by a factor of 0.1
    patience=5,                # Wait 5 epochs for improvement
    min_lr=1e-6,               # Set a minimum learning rate
    verbose=1                  # Print learning rate changes
)

# Optionally, add EarlyStopping for better monitoring <-- Added New
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,               # Stop training if no improvement for 10 epochs
    restore_best_weights=True  # Restore the best weights from training
)



# Building the fourth model
model4 = Sequential([
    # First Conv2D layer
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(64, 64, 3)),

    # Second Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # First MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # Third Conv2D layer
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),

    # Fourth Conv2D layer
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),

    # Second MaxPooling layer
    MaxPooling2D(pool_size=(2, 2)),

    # First Dropout Layer
    Dropout(0.25),


    # BatchNormalization
    BatchNormalization(),

    # Flatten the output
    Flatten(),

    # First Dense layer with 16 neurons
    Dense(16, activation='relu'),

    # Second Dense layer with 32 neurons
    Dense(32, activation='relu'),

    # Second Dropout Layer
    Dropout(0.25),

    # Output layer with 12 neurons for 12 classes
    Dense(12, activation='softmax')
])

# Compile the model
model4.compile(optimizer=Adam(learning_rate=1e-4),# Reduced learning_rate=0.0005
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Print the model summary
model4.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 64, 64, 32)	896
conv2d_5 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36,928
conv2d_7 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_2 (Dropout)	(None, 16, 16, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
flatten_1 (Flatten)	(None, 32768)	0
dense_3 (Dense)	(None, 16)	524,304
dense_4 (Dense)	(None, 32)	544
dropout_3 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 12)	396

Total params: 655,932 (2.50 MB)

Trainable params: 655,676 (2.50 MB)

Non-trainable params: 256 (1.00 KB)

In [ ]:

```
# Resetting the environment default seed=1
reset_environment()

# Fit the model
history4 = model4.fit(
          augmented_datagen.flow( # Data Augmentation
              x_train_normalized,
              y_train_encoded,
              batch_size=32),
          epochs=75, # <-- Added New
          validation_data=(x_val_normalized,
                           y_val_encoded),
          class_weight=class_weights_dict,
          callbacks=[reduce_lr, early_stopping], # <-- Added New
          verbose=2)
```

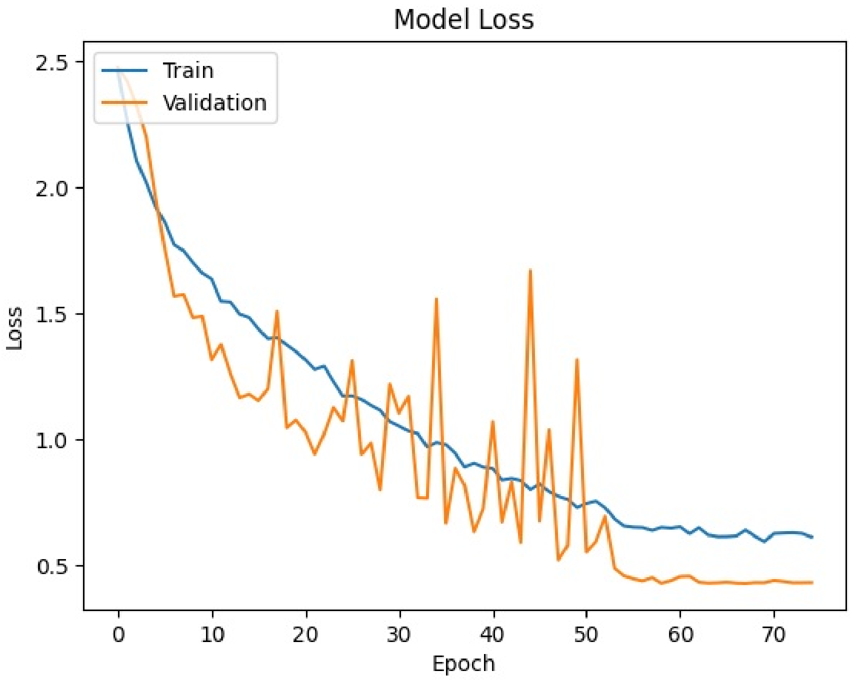
```
Epoch 1/75
104/104 - 14s - 131ms/step - accuracy: 0.0917 - loss: 2.4606 - val_accuracy: 0.1419 - val_loss: 2.4766 - learnin
g_rate: 1.0000e-04
Epoch 2/75
104/104 - 13s - 127ms/step - accuracy: 0.1678 - loss: 2.2593 - val_accuracy: 0.2037 - val_loss: 2.4191 - learnin
g_rate: 1.0000e-04
Epoch 3/75
104/104 - 5s - 49ms/step - accuracy: 0.2087 - loss: 2.1042 - val_accuracy: 0.2584 - val_loss: 2.3243 - learning_
rate: 1.0000e-04
Epoch 4/75
104/104 - 7s - 63ms/step - accuracy: 0.2424 - loss: 2.0196 - val_accuracy: 0.2584 - val_loss: 2.2018 - learning_
rate: 1.0000e-04
Epoch 5/75
104/104 - 9s - 89ms/step - accuracy: 0.2824 - loss: 1.9240 - val_accuracy: 0.3455 - val_loss: 1.9642 - learning_
rate: 1.0000e-04
Epoch 6/75
104/104 - 5s - 50ms/step - accuracy: 0.3068 - loss: 1.8605 - val_accuracy: 0.3961 - val_loss: 1.7538 - learning_
rate: 1.0000e-04
Epoch 7/75
104/104 - 4s - 37ms/step - accuracy: 0.3528 - loss: 1.7715 - val_accuracy: 0.4537 - val_loss: 1.5667 - learning_
rate: 1.0000e-04
Epoch 8/75
104/104 - 6s - 58ms/step - accuracy: 0.3456 - loss: 1.7474 - val_accuracy: 0.4270 - val_loss: 1.5738 - learning_
rate: 1.0000e-04
Epoch 9/75
104/104 - 5s - 46ms/step - accuracy: 0.3660 - loss: 1.7005 - val_accuracy: 0.4831 - val_loss: 1.4819 - learning_
rate: 1.0000e-04
Epoch 10/75
104/104 - 4s - 41ms/step - accuracy: 0.4012 - loss: 1.6588 - val_accuracy: 0.4621 - val_loss: 1.4873 - learning_
rate: 1.0000e-04
Epoch 11/75
104/104 - 7s - 64ms/step - accuracy: 0.3979 - loss: 1.6353 - val_accuracy: 0.5211 - val_loss: 1.3143 - learning_
```

```
rate: 1.0000e-04
Epoch 12/75
104/104 - 9s - 84ms/step - accuracy: 0.4373 - loss: 1.5464 - val_accuracy: 0.5169 - val_loss: 1.3746 - learning_
rate: 1.0000e-04
Epoch 13/75
104/104 - 7s - 66ms/step - accuracy: 0.4382 - loss: 1.5432 - val_accuracy: 0.5084 - val_loss: 1.2595 - learning_
rate: 1.0000e-04
Epoch 14/75
104/104 - 4s - 38ms/step - accuracy: 0.4547 - loss: 1.4959 - val_accuracy: 0.5941 - val_loss: 1.1640 - learning_
rate: 1.0000e-04
Epoch 15/75
104/104 - 4s - 38ms/step - accuracy: 0.4683 - loss: 1.4821 - val_accuracy: 0.5801 - val_loss: 1.1780 - learning_
rate: 1.0000e-04
Epoch 16/75
104/104 - 6s - 61ms/step - accuracy: 0.4776 - loss: 1.4364 - val_accuracy: 0.6110 - val_loss: 1.1521 - learning_
rate: 1.0000e-04
Epoch 17/75
104/104 - 4s - 38ms/step - accuracy: 0.4974 - loss: 1.3982 - val_accuracy: 0.5520 - val_loss: 1.1993 - learning_
rate: 1.0000e-04
Epoch 18/75
104/104 - 6s - 58ms/step - accuracy: 0.4947 - loss: 1.4028 - val_accuracy: 0.4284 - val_loss: 1.5066 - learning_
rate: 1.0000e-04
Epoch 19/75
104/104 - 5s - 46ms/step - accuracy: 0.5068 - loss: 1.3732 - val_accuracy: 0.6110 - val_loss: 1.0438 - learning_
rate: 1.0000e-04
Epoch 20/75
104/104 - 4s - 41ms/step - accuracy: 0.5038 - loss: 1.3469 - val_accuracy: 0.5857 - val_loss: 1.0751 - learning_
rate: 1.0000e-04
Epoch 21/75
104/104 - 5s - 45ms/step - accuracy: 0.5248 - loss: 1.3147 - val_accuracy: 0.6461 - val_loss: 1.0268 - learning_
rate: 1.0000e-04
Epoch 22/75
104/104 - 5s - 47ms/step - accuracy: 0.5453 - loss: 1.2753 - val_accuracy: 0.6826 - val_loss: 0.9388 - learning_
rate: 1.0000e-04
Epoch 23/75
104/104 - 4s - 37ms/step - accuracy: 0.5420 - loss: 1.2885 - val_accuracy: 0.6194 - val_loss: 1.0192 - learning_
rate: 1.0000e-04
Epoch 24/75
104/104 - 4s - 43ms/step - accuracy: 0.5555 - loss: 1.2275 - val_accuracy: 0.5885 - val_loss: 1.1245 - learning_
rate: 1.0000e-04
Epoch 25/75
104/104 - 5s - 47ms/step - accuracy: 0.5762 - loss: 1.1709 - val_accuracy: 0.6208 - val_loss: 1.0716 - learning_
rate: 1.0000e-04
Epoch 26/75
104/104 - 4s - 39ms/step - accuracy: 0.5886 - loss: 1.1713 - val_accuracy: 0.5323 - val_loss: 1.3115 - learning_
rate: 1.0000e-04
Epoch 27/75
104/104 - 7s - 63ms/step - accuracy: 0.5835 - loss: 1.1570 - val_accuracy: 0.6545 - val_loss: 0.9381 - learning_
rate: 1.0000e-04
Epoch 28/75
104/104 - 4s - 41ms/step - accuracy: 0.5937 - loss: 1.1330 - val_accuracy: 0.6531 - val_loss: 0.9837 - learning_
rate: 1.0000e-04
Epoch 29/75
104/104 - 4s - 38ms/step - accuracy: 0.6018 - loss: 1.1134 - val_accuracy: 0.6784 - val_loss: 0.7970 - learning_
rate: 1.0000e-04
Epoch 30/75
104/104 - 5s - 45ms/step - accuracy: 0.6174 - loss: 1.0690 - val_accuracy: 0.5702 - val_loss: 1.2178 - learning_
rate: 1.0000e-04
Epoch 31/75
104/104 - 5s - 46ms/step - accuracy: 0.6223 - loss: 1.0510 - val_accuracy: 0.5843 - val_loss: 1.1018 - learning_
rate: 1.0000e-04
Epoch 32/75
104/104 - 4s - 37ms/step - accuracy: 0.6268 - loss: 1.0309 - val_accuracy: 0.5871 - val_loss: 1.1697 - learning_
rate: 1.0000e-04
Epoch 33/75
104/104 - 7s - 65ms/step - accuracy: 0.6346 - loss: 1.0215 - val_accuracy: 0.7261 - val_loss: 0.7669 - learning_
rate: 1.0000e-04
Epoch 34/75
104/104 - 9s - 83ms/step - accuracy: 0.6508 - loss: 0.9679 - val_accuracy: 0.6882 - val_loss: 0.7650 - learning_
rate: 1.0000e-04
Epoch 35/75
104/104 - 7s - 64ms/step - accuracy: 0.6547 - loss: 0.9858 - val_accuracy: 0.4803 - val_loss: 1.5556 - learning_
rate: 1.0000e-04
Epoch 36/75
104/104 - 4s - 38ms/step - accuracy: 0.6466 - loss: 0.9770 - val_accuracy: 0.7654 - val_loss: 0.6657 - learning_
rate: 1.0000e-04
Epoch 37/75
104/104 - 5s - 52ms/step - accuracy: 0.6547 - loss: 0.9424 - val_accuracy: 0.6671 - val_loss: 0.8831 - learning_
rate: 1.0000e-04
Epoch 38/75
104/104 - 5s - 50ms/step - accuracy: 0.6776 - loss: 0.8883 - val_accuracy: 0.6685 - val_loss: 0.8156 - learning_
rate: 1.0000e-04
Epoch 39/75
```

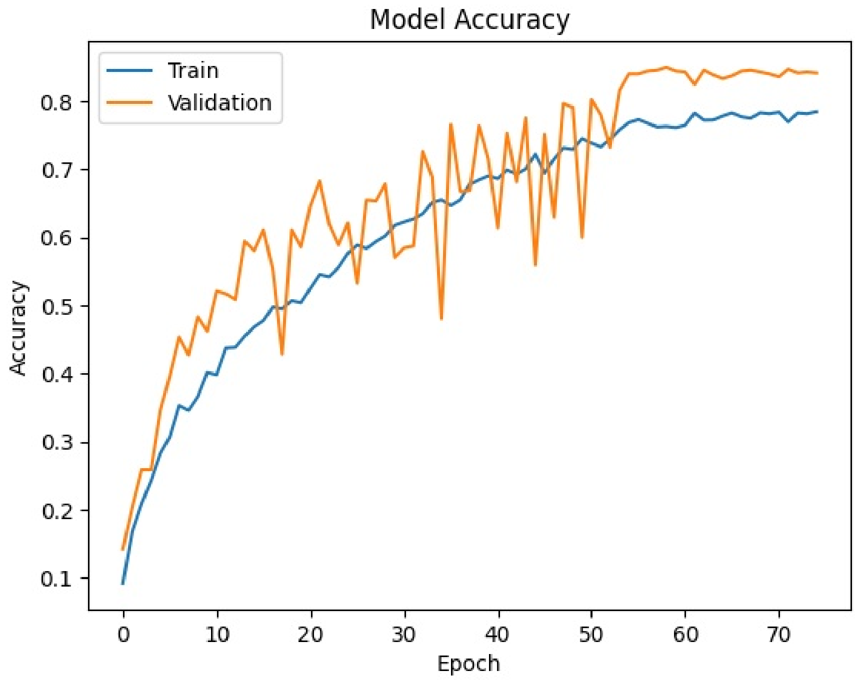
```
104/104 - 10s - 95ms/step - accuracy: 0.6842 - loss: 0.9033 - val_accuracy: 0.7640 - val_loss: 0.6301 - learning
_rate: 1.0000e-04
Epoch 40/75
104/104 - 5s - 47ms/step - accuracy: 0.6899 - loss: 0.8876 - val_accuracy: 0.7149 - val_loss: 0.7230 - learning_
rate: 1.0000e-04
Epoch 41/75
104/104 - 4s - 39ms/step - accuracy: 0.6860 - loss: 0.8815 - val_accuracy: 0.6138 - val_loss: 1.0689 - learning_
rate: 1.0000e-04
Epoch 42/75
104/104 - 4s - 42ms/step - accuracy: 0.6989 - loss: 0.8371 - val_accuracy: 0.7528 - val_loss: 0.6694 - learning_
rate: 1.0000e-04
Epoch 43/75
104/104 - 5s - 52ms/step - accuracy: 0.6929 - loss: 0.8434 - val_accuracy: 0.6812 - val_loss: 0.8303 - learning_
rate: 1.0000e-04
Epoch 44/75
104/104 - 4s - 41ms/step - accuracy: 0.7005 - loss: 0.8352 - val_accuracy: 0.7753 - val_loss: 0.5881 - learning_
rate: 1.0000e-04
Epoch 45/75
104/104 - 7s - 64ms/step - accuracy: 0.7221 - loss: 0.7982 - val_accuracy: 0.5590 - val_loss: 1.6689 - learning_
rate: 1.0000e-04
Epoch 46/75
104/104 - 9s - 84ms/step - accuracy: 0.6941 - loss: 0.8216 - val_accuracy: 0.7514 - val_loss: 0.6738 - learning_
rate: 1.0000e-04
Epoch 47/75
104/104 - 7s - 67ms/step - accuracy: 0.7149 - loss: 0.7897 - val_accuracy: 0.6292 - val_loss: 1.0360 - learning_
rate: 1.0000e-04
Epoch 48/75
104/104 - 8s - 80ms/step - accuracy: 0.7311 - loss: 0.7722 - val_accuracy: 0.7963 - val_loss: 0.5191 - learning_
rate: 1.0000e-04
Epoch 49/75
104/104 - 6s - 60ms/step - accuracy: 0.7290 - loss: 0.7600 - val_accuracy: 0.7907 - val_loss: 0.5765 - learning_
rate: 1.0000e-04
Epoch 50/75
104/104 - 10s - 95ms/step - accuracy: 0.7447 - loss: 0.7273 - val_accuracy: 0.5997 - val_loss: 1.3149 - learning
_rate: 1.0000e-04
Epoch 51/75
104/104 - 5s - 46ms/step - accuracy: 0.7380 - loss: 0.7436 - val_accuracy: 0.8020 - val_loss: 0.5507 - learning_
rate: 1.0000e-04
Epoch 52/75
104/104 - 4s - 38ms/step - accuracy: 0.7326 - loss: 0.7532 - val_accuracy: 0.7795 - val_loss: 0.5919 - learning_
rate: 1.0000e-04
Epoch 53/75

Epoch 53: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
104/104 - 5s - 43ms/step - accuracy: 0.7438 - loss: 0.7261 - val_accuracy: 0.7317 - val_loss: 0.6922 - learning_
rate: 1.0000e-04
Epoch 54/75
104/104 - 5s - 48ms/step - accuracy: 0.7579 - loss: 0.6826 - val_accuracy: 0.8160 - val_loss: 0.4857 - learning_
rate: 1.0000e-05
Epoch 55/75
104/104 - 4s - 38ms/step - accuracy: 0.7684 - loss: 0.6545 - val_accuracy: 0.8399 - val_loss: 0.4561 - learning_
rate: 1.0000e-05
Epoch 56/75
104/104 - 4s - 40ms/step - accuracy: 0.7729 - loss: 0.6492 - val_accuracy: 0.8399 - val_loss: 0.4441 - learning_
rate: 1.0000e-05
Epoch 57/75
104/104 - 6s - 53ms/step - accuracy: 0.7675 - loss: 0.6478 - val_accuracy: 0.8441 - val_loss: 0.4358 - learning_
rate: 1.0000e-05
Epoch 58/75
104/104 - 4s - 37ms/step - accuracy: 0.7618 - loss: 0.6363 - val_accuracy: 0.8455 - val_loss: 0.4489 - learning_
rate: 1.0000e-05
Epoch 59/75
104/104 - 4s - 37ms/step - accuracy: 0.7627 - loss: 0.6485 - val_accuracy: 0.8497 - val_loss: 0.4266 - learning_
rate: 1.0000e-05
Epoch 60/75
104/104 - 6s - 61ms/step - accuracy: 0.7609 - loss: 0.6454 - val_accuracy: 0.8441 - val_loss: 0.4372 - learning_
rate: 1.0000e-05
Epoch 61/75
104/104 - 4s - 37ms/step - accuracy: 0.7645 - loss: 0.6509 - val_accuracy: 0.8427 - val_loss: 0.4529 - learning_
rate: 1.0000e-05
Epoch 62/75
104/104 - 4s - 39ms/step - accuracy: 0.7823 - loss: 0.6236 - val_accuracy: 0.8244 - val_loss: 0.4541 - learning_
rate: 1.0000e-05
Epoch 63/75
104/104 - 6s - 55ms/step - accuracy: 0.7720 - loss: 0.6472 - val_accuracy: 0.8455 - val_loss: 0.4310 - learning_
rate: 1.0000e-05
Epoch 64/75

Epoch 64: ReduceLROnPlateau reducing learning rate to 1e-06.
104/104 - 4s - 42ms/step - accuracy: 0.7723 - loss: 0.6182 - val_accuracy: 0.8385 - val_loss: 0.4274 - learning_
rate: 1.0000e-05
Epoch 65/75
104/104 - 5s - 46ms/step - accuracy: 0.7780 - loss: 0.6113 - val_accuracy: 0.8329 - val_loss: 0.4286 - learning_
```

```
rate: 1.0000e-06
Epoch 66/75
104/104 - 5s - 46ms/step - accuracy: 0.7826 - loss: 0.6115 - val_accuracy: 0.8371 - val_loss: 0.4310 - learning_
rate: 1.0000e-06
Epoch 67/75
104/104 - 4s - 41ms/step - accuracy: 0.7768 - loss: 0.6144 - val_accuracy: 0.8441 - val_loss: 0.4272 - learning_
rate: 1.0000e-06
Epoch 68/75
104/104 - 7s - 67ms/step - accuracy: 0.7747 - loss: 0.6379 - val_accuracy: 0.8455 - val_loss: 0.4261 - learning_
rate: 1.0000e-06
Epoch 69/75
104/104 - 4s - 38ms/step - accuracy: 0.7829 - loss: 0.6120 - val_accuracy: 0.8427 - val_loss: 0.4294 - learning_
rate: 1.0000e-06
Epoch 70/75
104/104 - 4s - 38ms/step - accuracy: 0.7814 - loss: 0.5919 - val_accuracy: 0.8399 - val_loss: 0.4290 - learning_
rate: 1.0000e-06
Epoch 71/75
104/104 - 7s - 64ms/step - accuracy: 0.7838 - loss: 0.6242 - val_accuracy: 0.8357 - val_loss: 0.4380 - learning_
rate: 1.0000e-06
Epoch 72/75
104/104 - 4s - 37ms/step - accuracy: 0.7693 - loss: 0.6260 - val_accuracy: 0.8469 - val_loss: 0.4340 - learning_
rate: 1.0000e-06
Epoch 73/75
104/104 - 5s - 49ms/step - accuracy: 0.7826 - loss: 0.6275 - val_accuracy: 0.8413 - val_loss: 0.4287 - learning_
rate: 1.0000e-06
Epoch 74/75
104/104 - 6s - 56ms/step - accuracy: 0.7814 - loss: 0.6244 - val_accuracy: 0.8427 - val_loss: 0.4289 - learning_
rate: 1.0000e-06
Epoch 75/75
104/104 - 4s - 37ms/step - accuracy: 0.7844 - loss: 0.6103 - val_accuracy: 0.8413 - val_loss: 0.4293 - learning_
rate: 1.0000e-06
```

In [ ]:
```
plot_loss(history4)
```

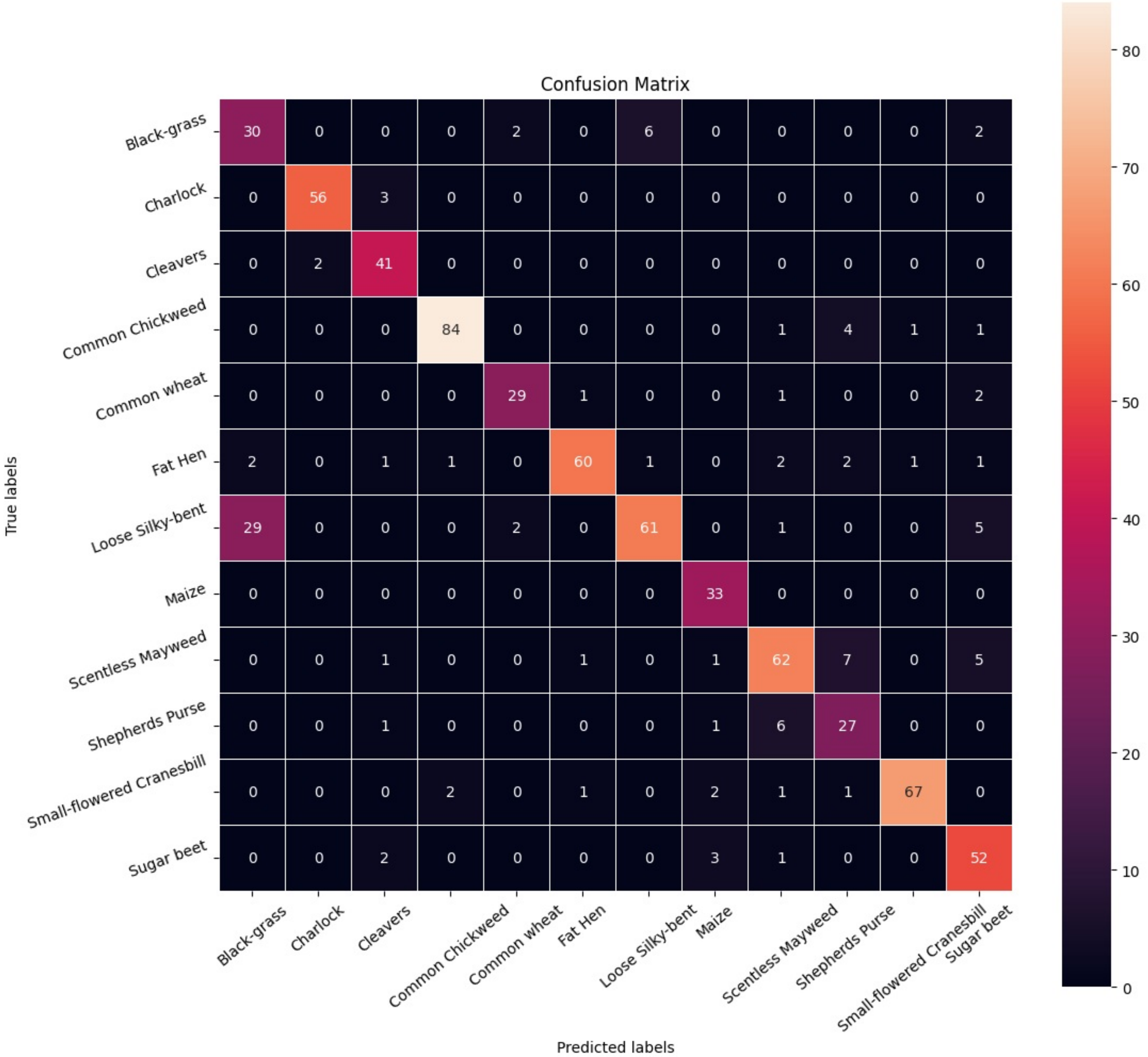


In [ ]:
```
plot_accuracy(history4)
```



In [ ]:
```
plot_confusion_matrix_and_report(model4, x_val_normalized, y_val_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```



```
              precision    recall  f1-score   support

           0       0.49      0.75      0.59        40
           1       0.97      0.95      0.96        59
           2       0.84      0.95      0.89        43
           3       0.97      0.92      0.94        91
           4       0.88      0.88      0.88        33
           5       0.95      0.85      0.90        71
           6       0.90      0.62      0.73        98
           7       0.82      1.00      0.90        33
           8       0.83      0.81      0.82        77
           9       0.66      0.77      0.71        35
          10       0.97      0.91      0.94        74
          11       0.76      0.90      0.83        58

    accuracy                           0.85       712
   macro avg       0.84      0.86      0.84       712
weighted avg       0.86      0.85      0.85       712
```

Observations

Loss and Accuracy Plots

- The loss plot shows consistent reduction in training.
- Validation losses over the epochs. There are fluctuations in validation loss due to data augmentation but a general downward trend.
- The accuracy plot indicates that training accuracy has plateaued around 78%. with training accuracy lagging behind validation accuracy.
- Validation accuracy has plateaued around 85%, with training accuracy lagging slightly, suggesting a well-balanced model.

Confusion Matrix

- Misclassifications still occur, particularly among visually similar classes such as Loose `Silky-bent` and other similar grasses.
- Some minority classes like `Shepherds Purse` still show limited improvement. Class imbalance a possible reason.

Classification Report

- Macro average precision, recall, and F1 score have improved significantly to 84-86%, with the weighted average closely matching.
- Precision and recall are high for some dominant classes (e.g., `Charlock`, `Common Chickweed`, `Cleavers`), but lower for a few minority classes.

Improvements Made

1. **Use of `ReduceLROnPlateau`**: Dynamically adjusting the learning rate helped stabilize training, leading to smoother convergence for both training and validation loss function.
2. **Increased Epochs**: Increasing epochs to **75** allowed the model to learn and refine patterns more, resulting improved generalization.
3. **Regularization**: Adding dropout layers reduced overfitting, making validation metrics more similar with training metrics.
4. **Better Data Augmentation**: Augmented data helped robustness and variability, resulting in better generalization to unseen data.

Strengths

- **Generalization**: Training and validation loss/accuracy are well-aligned, indicating less overfitting.
- **Class Performance**: Improved precision, recall, and F1-scores across most classes.
- **Stability**: The use of `ReduceLROnPlateau` ensured stable convergence, with the model responding well to learning rate adjustments.

Fifth Iteration - Using Transfer Learning

Improvement Criteria

- Using pre-trained model - `tensorflow.keras.applications.VGG16`
- This model can accept our reduced image size (64 X 64 X 3).
- Early stopping paitence increased to 20 from 10.

In [ ]:
```
# The VGG16 model with 64 X 64 X 3 input size
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))
base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 64, 64, 3)	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1,792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36,928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73,856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147,584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295,168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590,080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590,080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0

Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:

```
# Getting only the conv layers for transfer learning.
transfer_layer = base_model.get_layer('block5_pool')
vgg_base_model = Model(inputs=base_model.input, outputs=transfer_layer.output)


# Freeze the base model layers to prevent training
for layer in vgg_base_model.layers:
    layer.trainable = False

for layer in vgg_base_model.layers:
    print(layer.name, layer.trainable)

# Define ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',        # Monitor validation loss
    factor=0.1,                # Reduce learning rate by a factor of 0.1
    patience=5,                # Wait 5 epochs for improvement
    min_lr=1e-6,               # Set a minimum learning rate
    verbose=1                  # Print learning rate changes
)

# Optionally, add EarlyStopping for better monitoring
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=20,               # Stop training if no improvement for 10 epochs
    restore_best_weights=True  # Restore the best weights from training
)

# Add custom layers on top
transfer_model = Sequential([
    vgg_base_model,                             # Pre-trained VGG base model
    Flatten(),
    Dense(128, activation='relu'),              # Fully connected layer
    Dropout(0.25),                              # Dropout for regularization
    Dense(64, activation='relu'),               # Another dense layer
    Dropout(0.25),                              # Another dropout
```

```
    Dense(12, activation='softmax')          # Output layer for 12 classes
])

# Compile the model
transfer_model.compile(
    optimizer=Adam(learning_rate=1e-4),      # Learning rate for transfer learning
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

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

```
input_layer_7 False
block1_conv1 False
block1_conv2 False
block1_pool False
block2_conv1 False
block2_conv2 False
block2_pool False
block3_conv1 False
block3_conv2 False
block3_conv3 False
block3_pool False
block4_conv1 False
block4_conv2 False
block4_conv3 False
block4_pool False
block5_conv1 False
block5_conv2 False
block5_conv3 False
block5_pool False
```

Model: "sequential_3"

Layer (type)	**Output Shape**	**Param #**
functional_6 (Functional)	(None, 2, 2, 512)	14,714,688
flatten_3 (Flatten)	(None, 2048)	0
dense_9 (Dense)	(None, 128)	262,272
dropout_6 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 64)	8,256
dropout_7 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 12)	780

Total params: 14,985,996 (57.17 MB)

Trainable params: 271,308 (1.03 MB)

Non-trainable params: 14,714,688 (56.13 MB)

In [ ]:
```
# Resetting the environment default seed=1
reset_environment()

# Train the transfer learning model
history_transfer = transfer_model.fit(
    augmented_datagen.flow(
        x_train_normalized,
        y_train_encoded,
        batch_size=32
    ),
    validation_data=(x_val_normalized, y_val_encoded),
    epochs=75,
    callbacks=[reduce_lr, early_stopping],
    class_weight=class_weights_dict,
    verbose=2
)
```

```
Epoch 1/75
104/104 - 12s - 117ms/step - accuracy: 0.1158 - loss: 2.4987 - val_accuracy: 0.1798 - val_loss: 2.3882 - learnin
g_rate: 1.0000e-04
Epoch 2/75
104/104 - 5s - 49ms/step - accuracy: 0.1555 - loss: 2.3911 - val_accuracy: 0.2542 - val_loss: 2.2776 - learning_
rate: 1.0000e-04
Epoch 3/75
104/104 - 9s - 90ms/step - accuracy: 0.1973 - loss: 2.3091 - val_accuracy: 0.2669 - val_loss: 2.1859 - learning_
rate: 1.0000e-04
Epoch 4/75
104/104 - 6s - 56ms/step - accuracy: 0.2162 - loss: 2.2269 - val_accuracy: 0.3118 - val_loss: 2.1070 - learning_
```

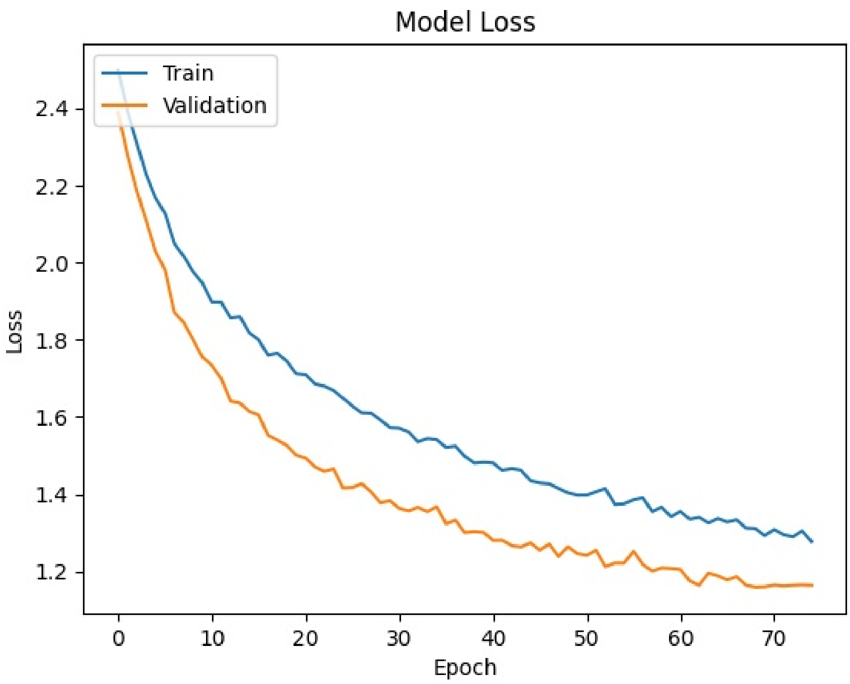
```
rate: 1.0000e-04
Epoch 5/75
104/104 - 4s - 40ms/step - accuracy: 0.2478 - loss: 2.1669 - val_accuracy: 0.3413 - val_loss: 2.0284 - learning_
rate: 1.0000e-04
Epoch 6/75
104/104 - 5s - 49ms/step - accuracy: 0.2406 - loss: 2.1274 - val_accuracy: 0.3399 - val_loss: 1.9810 - learning_
rate: 1.0000e-04
Epoch 7/75
104/104 - 6s - 60ms/step - accuracy: 0.2638 - loss: 2.0496 - val_accuracy: 0.3975 - val_loss: 1.8712 - learning_
rate: 1.0000e-04
Epoch 8/75
104/104 - 5s - 44ms/step - accuracy: 0.2950 - loss: 2.0157 - val_accuracy: 0.3666 - val_loss: 1.8441 - learning_
rate: 1.0000e-04
Epoch 9/75
104/104 - 7s - 63ms/step - accuracy: 0.2878 - loss: 1.9764 - val_accuracy: 0.3975 - val_loss: 1.8001 - learning_
rate: 1.0000e-04
Epoch 10/75
104/104 - 4s - 40ms/step - accuracy: 0.3095 - loss: 1.9463 - val_accuracy: 0.4157 - val_loss: 1.7545 - learning_
rate: 1.0000e-04
Epoch 11/75
104/104 - 4s - 39ms/step - accuracy: 0.3284 - loss: 1.8969 - val_accuracy: 0.4031 - val_loss: 1.7337 - learning_
rate: 1.0000e-04
Epoch 12/75
104/104 - 7s - 67ms/step - accuracy: 0.3206 - loss: 1.8967 - val_accuracy: 0.4452 - val_loss: 1.6995 - learning_
rate: 1.0000e-04
Epoch 13/75
104/104 - 8s - 82ms/step - accuracy: 0.3269 - loss: 1.8562 - val_accuracy: 0.4551 - val_loss: 1.6407 - learning_
rate: 1.0000e-04
Epoch 14/75
104/104 - 6s - 56ms/step - accuracy: 0.3408 - loss: 1.8594 - val_accuracy: 0.4494 - val_loss: 1.6358 - learning_
rate: 1.0000e-04
Epoch 15/75
104/104 - 4s - 40ms/step - accuracy: 0.3498 - loss: 1.8173 - val_accuracy: 0.4649 - val_loss: 1.6140 - learning_
rate: 1.0000e-04
Epoch 16/75
104/104 - 5s - 51ms/step - accuracy: 0.3573 - loss: 1.7997 - val_accuracy: 0.4635 - val_loss: 1.6053 - learning_
rate: 1.0000e-04
Epoch 17/75
104/104 - 6s - 54ms/step - accuracy: 0.3609 - loss: 1.7597 - val_accuracy: 0.4775 - val_loss: 1.5519 - learning_
rate: 1.0000e-04
Epoch 18/75
104/104 - 4s - 43ms/step - accuracy: 0.3627 - loss: 1.7649 - val_accuracy: 0.4831 - val_loss: 1.5399 - learning_
rate: 1.0000e-04
Epoch 19/75
104/104 - 6s - 56ms/step - accuracy: 0.3693 - loss: 1.7438 - val_accuracy: 0.4958 - val_loss: 1.5258 - learning_
rate: 1.0000e-04
Epoch 20/75
104/104 - 5s - 46ms/step - accuracy: 0.3844 - loss: 1.7119 - val_accuracy: 0.5014 - val_loss: 1.5010 - learning_
rate: 1.0000e-04
Epoch 21/75
104/104 - 4s - 40ms/step - accuracy: 0.3826 - loss: 1.7092 - val_accuracy: 0.5056 - val_loss: 1.4931 - learning_
rate: 1.0000e-04
Epoch 22/75
104/104 - 7s - 68ms/step - accuracy: 0.3838 - loss: 1.6856 - val_accuracy: 0.4888 - val_loss: 1.4709 - learning_
rate: 1.0000e-04
Epoch 23/75
104/104 - 4s - 40ms/step - accuracy: 0.3877 - loss: 1.6796 - val_accuracy: 0.5169 - val_loss: 1.4594 - learning_
rate: 1.0000e-04
Epoch 24/75
104/104 - 5s - 49ms/step - accuracy: 0.3931 - loss: 1.6685 - val_accuracy: 0.5070 - val_loss: 1.4655 - learning_
rate: 1.0000e-04
Epoch 25/75
104/104 - 7s - 63ms/step - accuracy: 0.4048 - loss: 1.6482 - val_accuracy: 0.5478 - val_loss: 1.4161 - learning_
rate: 1.0000e-04
Epoch 26/75
104/104 - 9s - 91ms/step - accuracy: 0.4183 - loss: 1.6279 - val_accuracy: 0.5197 - val_loss: 1.4174 - learning_
rate: 1.0000e-04
Epoch 27/75
104/104 - 5s - 50ms/step - accuracy: 0.4208 - loss: 1.6107 - val_accuracy: 0.5056 - val_loss: 1.4268 - learning_
rate: 1.0000e-04
Epoch 28/75
104/104 - 11s - 105ms/step - accuracy: 0.4144 - loss: 1.6096 - val_accuracy: 0.5197 - val_loss: 1.4056 - learnin
g_rate: 1.0000e-04
Epoch 29/75
104/104 - 9s - 83ms/step - accuracy: 0.4226 - loss: 1.5920 - val_accuracy: 0.5309 - val_loss: 1.3774 - learning_
rate: 1.0000e-04
Epoch 30/75
104/104 - 7s - 69ms/step - accuracy: 0.4250 - loss: 1.5726 - val_accuracy: 0.5421 - val_loss: 1.3825 - learning_
rate: 1.0000e-04
Epoch 31/75
104/104 - 8s - 78ms/step - accuracy: 0.4147 - loss: 1.5713 - val_accuracy: 0.5112 - val_loss: 1.3635 - learning_
rate: 1.0000e-04
Epoch 32/75
```

```
104/104 - 6s - 56ms/step - accuracy: 0.4352 - loss: 1.5608 - val_accuracy: 0.5309 - val_loss: 1.3568 - learning_
rate: 1.0000e-04
Epoch 33/75
104/104 - 9s - 83ms/step - accuracy: 0.4403 - loss: 1.5352 - val_accuracy: 0.5211 - val_loss: 1.3660 - learning_
rate: 1.0000e-04
Epoch 34/75
104/104 - 6s - 56ms/step - accuracy: 0.4316 - loss: 1.5437 - val_accuracy: 0.5435 - val_loss: 1.3548 - learning_
rate: 1.0000e-04
Epoch 35/75
104/104 - 9s - 83ms/step - accuracy: 0.4337 - loss: 1.5409 - val_accuracy: 0.5225 - val_loss: 1.3677 - learning_
rate: 1.0000e-04
Epoch 36/75
104/104 - 6s - 56ms/step - accuracy: 0.4454 - loss: 1.5211 - val_accuracy: 0.5309 - val_loss: 1.3231 - learning_
rate: 1.0000e-04
Epoch 37/75
104/104 - 4s - 40ms/step - accuracy: 0.4418 - loss: 1.5244 - val_accuracy: 0.5365 - val_loss: 1.3325 - learning_
rate: 1.0000e-04
Epoch 38/75
104/104 - 5s - 46ms/step - accuracy: 0.4490 - loss: 1.4975 - val_accuracy: 0.5604 - val_loss: 1.3006 - learning_
rate: 1.0000e-04
Epoch 39/75
104/104 - 5s - 51ms/step - accuracy: 0.4611 - loss: 1.4805 - val_accuracy: 0.5365 - val_loss: 1.3030 - learning_
rate: 1.0000e-04
Epoch 40/75
104/104 - 4s - 41ms/step - accuracy: 0.4671 - loss: 1.4825 - val_accuracy: 0.5351 - val_loss: 1.3010 - learning_
rate: 1.0000e-04
Epoch 41/75
104/104 - 5s - 48ms/step - accuracy: 0.4544 - loss: 1.4808 - val_accuracy: 0.5478 - val_loss: 1.2800 - learning_
rate: 1.0000e-04
Epoch 42/75
104/104 - 5s - 49ms/step - accuracy: 0.4701 - loss: 1.4619 - val_accuracy: 0.5534 - val_loss: 1.2805 - learning_
rate: 1.0000e-04
Epoch 43/75
104/104 - 4s - 41ms/step - accuracy: 0.4614 - loss: 1.4665 - val_accuracy: 0.5548 - val_loss: 1.2666 - learning_
rate: 1.0000e-04
Epoch 44/75
104/104 - 5s - 50ms/step - accuracy: 0.4668 - loss: 1.4623 - val_accuracy: 0.5506 - val_loss: 1.2632 - learning_
rate: 1.0000e-04
Epoch 45/75
104/104 - 5s - 46ms/step - accuracy: 0.4743 - loss: 1.4342 - val_accuracy: 0.5492 - val_loss: 1.2737 - learning_
rate: 1.0000e-04
Epoch 46/75
104/104 - 4s - 39ms/step - accuracy: 0.4728 - loss: 1.4289 - val_accuracy: 0.5632 - val_loss: 1.2551 - learning_
rate: 1.0000e-04
Epoch 47/75
104/104 - 5s - 51ms/step - accuracy: 0.4767 - loss: 1.4264 - val_accuracy: 0.5365 - val_loss: 1.2706 - learning_
rate: 1.0000e-04
Epoch 48/75
104/104 - 9s - 88ms/step - accuracy: 0.4890 - loss: 1.4150 - val_accuracy: 0.5534 - val_loss: 1.2386 - learning_
rate: 1.0000e-04
Epoch 49/75
104/104 - 7s - 65ms/step - accuracy: 0.4941 - loss: 1.4036 - val_accuracy: 0.5435 - val_loss: 1.2637 - learning_
rate: 1.0000e-04
Epoch 50/75
104/104 - 9s - 82ms/step - accuracy: 0.4842 - loss: 1.3975 - val_accuracy: 0.5548 - val_loss: 1.2461 - learning_
rate: 1.0000e-04
Epoch 51/75
104/104 - 6s - 57ms/step - accuracy: 0.4968 - loss: 1.3976 - val_accuracy: 0.5604 - val_loss: 1.2417 - learning_
rate: 1.0000e-04
Epoch 52/75
104/104 - 8s - 81ms/step - accuracy: 0.4800 - loss: 1.4062 - val_accuracy: 0.5449 - val_loss: 1.2548 - learning_
rate: 1.0000e-04
Epoch 53/75
104/104 - 6s - 60ms/step - accuracy: 0.4659 - loss: 1.4143 - val_accuracy: 0.5871 - val_loss: 1.2123 - learning_
rate: 1.0000e-04
Epoch 54/75
104/104 - 10s - 100ms/step - accuracy: 0.4845 - loss: 1.3733 - val_accuracy: 0.5590 - val_loss: 1.2214 - learnin
g_rate: 1.0000e-04
Epoch 55/75
104/104 - 4s - 42ms/step - accuracy: 0.4959 - loss: 1.3747 - val_accuracy: 0.5702 - val_loss: 1.2212 - learning_
rate: 1.0000e-04
Epoch 56/75
104/104 - 4s - 39ms/step - accuracy: 0.4887 - loss: 1.3853 - val_accuracy: 0.5435 - val_loss: 1.2523 - learning_
rate: 1.0000e-04
Epoch 57/75
104/104 - 7s - 66ms/step - accuracy: 0.4866 - loss: 1.3902 - val_accuracy: 0.5716 - val_loss: 1.2178 - learning_
rate: 1.0000e-04
Epoch 58/75
104/104 - 4s - 39ms/step - accuracy: 0.5098 - loss: 1.3549 - val_accuracy: 0.5772 - val_loss: 1.2007 - learning_
rate: 1.0000e-04
Epoch 59/75
104/104 - 5s - 49ms/step - accuracy: 0.5050 - loss: 1.3666 - val_accuracy: 0.5702 - val_loss: 1.2090 - learning_
rate: 1.0000e-04
```

```
Epoch 60/75
104/104 - 6s - 57ms/step - accuracy: 0.5056 - loss: 1.3416 - val_accuracy: 0.5801 - val_loss: 1.2075 - learning_
rate: 1.0000e-04
Epoch 61/75
104/104 - 8s - 81ms/step - accuracy: 0.4956 - loss: 1.3554 - val_accuracy: 0.5843 - val_loss: 1.2052 - learning_
rate: 1.0000e-04
Epoch 62/75
104/104 - 6s - 57ms/step - accuracy: 0.5149 - loss: 1.3350 - val_accuracy: 0.5758 - val_loss: 1.1751 - learning_
rate: 1.0000e-04
Epoch 63/75
104/104 - 9s - 83ms/step - accuracy: 0.5041 - loss: 1.3400 - val_accuracy: 0.5997 - val_loss: 1.1639 - learning_
rate: 1.0000e-04
Epoch 64/75
104/104 - 6s - 56ms/step - accuracy: 0.5101 - loss: 1.3255 - val_accuracy: 0.5730 - val_loss: 1.1950 - learning_
rate: 1.0000e-04
Epoch 65/75
104/104 - 9s - 84ms/step - accuracy: 0.5083 - loss: 1.3367 - val_accuracy: 0.5660 - val_loss: 1.1882 - learning_
rate: 1.0000e-04
Epoch 66/75
104/104 - 6s - 55ms/step - accuracy: 0.5035 - loss: 1.3278 - val_accuracy: 0.5772 - val_loss: 1.1780 - learning_
rate: 1.0000e-04
Epoch 67/75
104/104 - 9s - 88ms/step - accuracy: 0.5098 - loss: 1.3332 - val_accuracy: 0.5772 - val_loss: 1.1861 - learning_
rate: 1.0000e-04
Epoch 68/75

Epoch 68: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
104/104 - 6s - 54ms/step - accuracy: 0.5140 - loss: 1.3124 - val_accuracy: 0.5829 - val_loss: 1.1645 - learning_
rate: 1.0000e-04
Epoch 69/75
104/104 - 4s - 40ms/step - accuracy: 0.5278 - loss: 1.3108 - val_accuracy: 0.5871 - val_loss: 1.1592 - learning_
rate: 1.0000e-05
Epoch 70/75
104/104 - 7s - 63ms/step - accuracy: 0.5269 - loss: 1.2925 - val_accuracy: 0.5927 - val_loss: 1.1600 - learning_
rate: 1.0000e-05
Epoch 71/75
104/104 - 9s - 85ms/step - accuracy: 0.5236 - loss: 1.3082 - val_accuracy: 0.5885 - val_loss: 1.1646 - learning_
rate: 1.0000e-05
Epoch 72/75
104/104 - 7s - 66ms/step - accuracy: 0.5137 - loss: 1.2952 - val_accuracy: 0.5871 - val_loss: 1.1623 - learning_
rate: 1.0000e-05
Epoch 73/75
104/104 - 8s - 81ms/step - accuracy: 0.5266 - loss: 1.2897 - val_accuracy: 0.5885 - val_loss: 1.1640 - learning_
rate: 1.0000e-05
Epoch 74/75

Epoch 74: ReduceLROnPlateau reducing learning rate to 1e-06.
104/104 - 6s - 58ms/step - accuracy: 0.5215 - loss: 1.3048 - val_accuracy: 0.5885 - val_loss: 1.1649 - learning_
rate: 1.0000e-05
Epoch 75/75
104/104 - 4s - 39ms/step - accuracy: 0.5272 - loss: 1.2770 - val_accuracy: 0.5899 - val_loss: 1.1640 - learning_
rate: 1.0000e-06
```

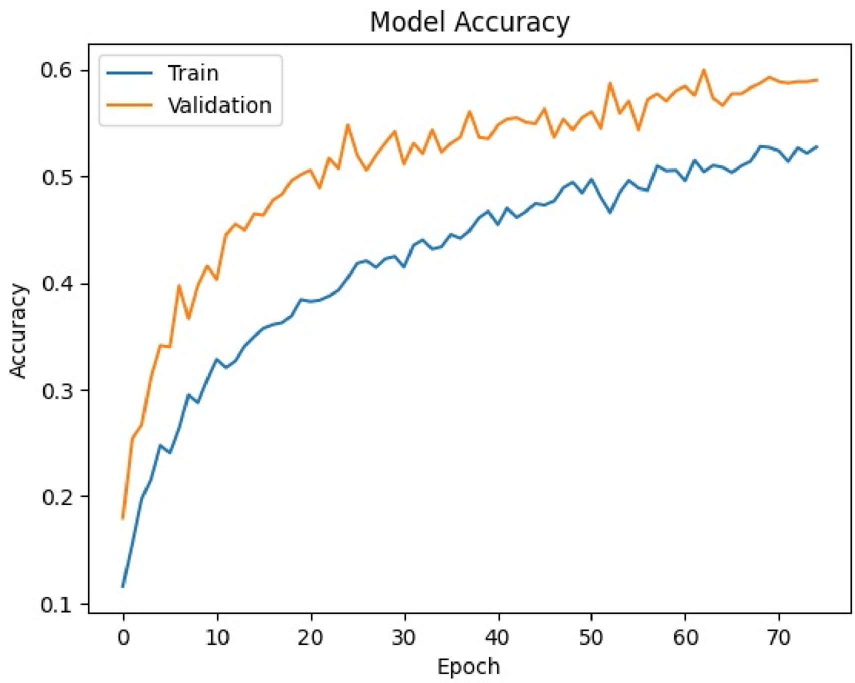
In [ ]:

```
plot_loss(history_transfer)
```



In [ ]:

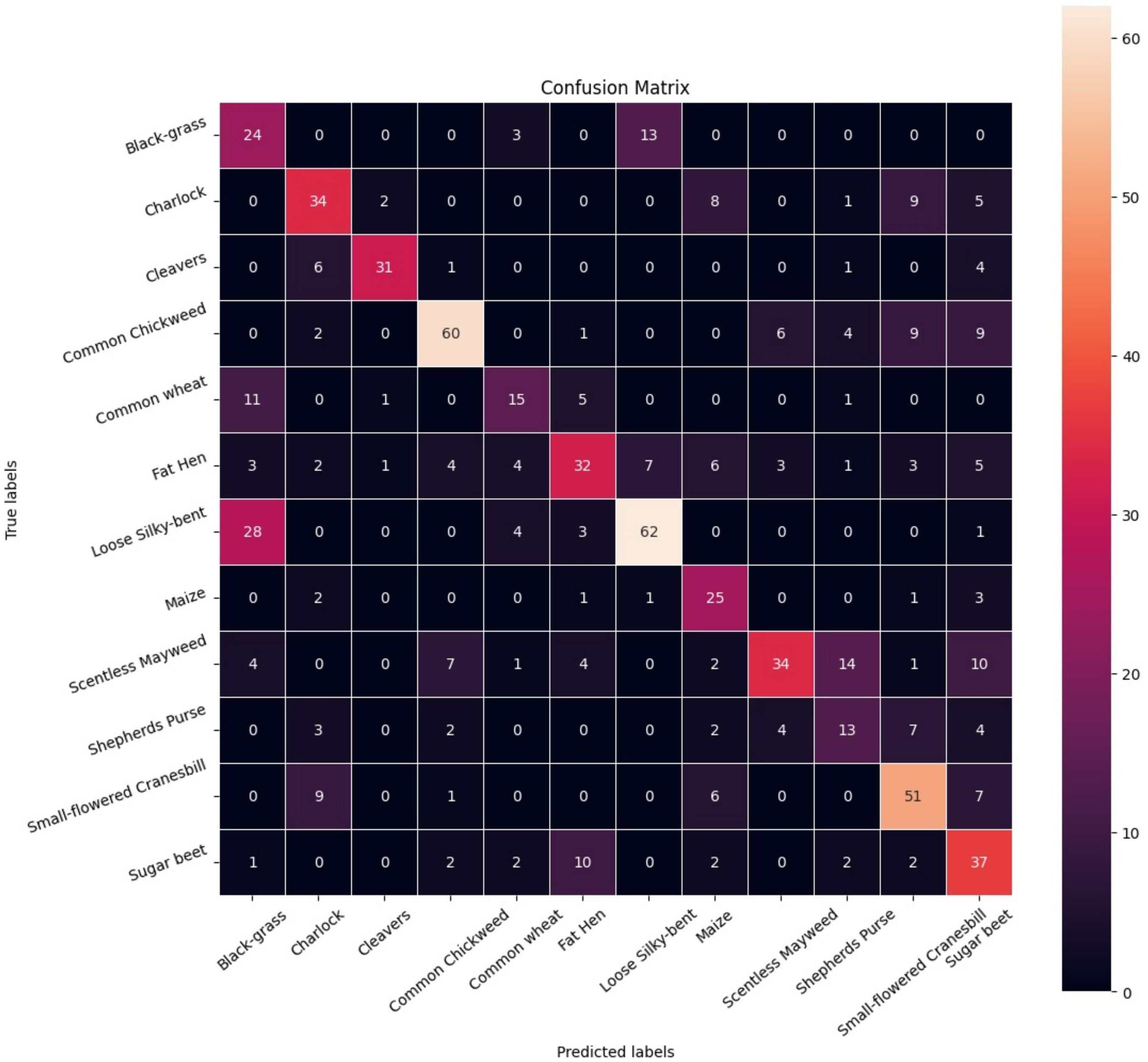
```
plot_accuracy(history_transfer)
```



Observations & Improvements

```
In [ ]: plot_confusion_matrix_and_report(transfer_model, x_val_normalized, y_val_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step
```



```
              precision    recall  f1-score   support

           0       0.34      0.60      0.43        40
           1       0.59      0.58      0.58        59
           2       0.89      0.72      0.79        43
           3       0.78      0.66      0.71        91
           4       0.52      0.45      0.48        33
           5       0.57      0.45      0.50        71
           6       0.75      0.63      0.69        98
           7       0.49      0.76      0.60        33
           8       0.72      0.44      0.55        77
           9       0.35      0.37      0.36        35
          10       0.61      0.69      0.65        74
          11       0.44      0.64      0.52        58

    accuracy                           0.59       712
   macro avg       0.59      0.58      0.57       712
weighted avg       0.62      0.59      0.59       712
```

Observations

Loss and Accuracy Plots

- The model shows gradual improvement in training accuracy.
- The validation accuracy converges around 59%.
- Both training and validation loss are decreasing, which indicates proper learning without significant overfitting.

Confusion Matrix:

- The model struggles in certain classes (e.g., `Loose Silky-bent` and `Shepherd's Purse`), with low recall and precision in some underrepresented categories.
- Overall Accuracy 59% indicates the model is not fully leveraging the power of the VGG16 base model.

Classification Report

- Macro average precision, recall, and F1 score is not close to previous two models.
- Precision and recall are also not as high as previous model but moderately OK.

All this suggests that the model might not be adequately trained or lacks capacity for this dataset.

Comparison Between **Model4** (Custom CNN) and **Transfer_Model** (VGG16 Transfer Learning)

- **Model4**:
  - Custom-built CNN with relatively fewer convolutional and dense layers. Includes Dropout, BatchNormalization, and dense layers with smaller capacities.
  - **Training Accuracy**: 81%
  - **Validation Accuracy**: 85%
  - Custom model outperforms transfer learning in this specific task, especially in validation accuracy.
  - Shows better generalization as validation accuracy is higher than training accuracy.
  - Slightly Computationally efficient
- **Transfer_Model**:
  - Utilizes the pre-trained VGG16 architecture, having its deep and hierarchical feature extraction. Pre-trained on ImageNet.
  - **Training Accuracy**: 59%
  - **Validation Accuracy**: 59%
  - Limited performance possibly due to frozen VGG16 layers and smaller input size (64x64x3).
  - Issues with generalization on this dataset, likely due to not being fine-tuned for the specific task.
  - Took almost similar training time, even with the frozen layers.

Key Takeaway

- **Model4** is better suited for this specific dataset, offering higher accuracy and better generalization with lower computational cost.
- **Transfer_Model** underperforms in its current setup, but fine-tuning, higher input resolution, or using a different pre-trained model may improve its performance.

Final Model

Brief Comparison of All 5 Models:

1. **Model1 (Base CNN)**:
   - **Performance**: Poor generalization with accuracy around 38%; struggled to extract meaningful features.
2. **Model2 (Enhanced CNN)**:
   - **Improvement**: Added BatchNormalization and additional Conv2D layers; achieved better accuracy (~70%) but still limited in learning complexity.
3. **Model3 (Data Augmentation)**:
   - **Impact**: Data augmentation and Dropout improved generalization; reached ~79% accuracy, indicating robustness to overfitting.
4. **Model4 (Optimized CNN)**:
   - **Best Performer**: Further tuning with ReduceLROnPlateau and EarlyStopping resulted in ~85% accuracy, balancing generalization and learning.
5. **Transfer_Model (VGG16)**:
   - **Underperformed**: Achieved ~59% accuracy; limited by frozen layers and smaller input size, highlighting the need for fine-tuning for better results.

Final Model Selection

I would choose **Model4 (Optimized Custom CNN)** as the final model because:

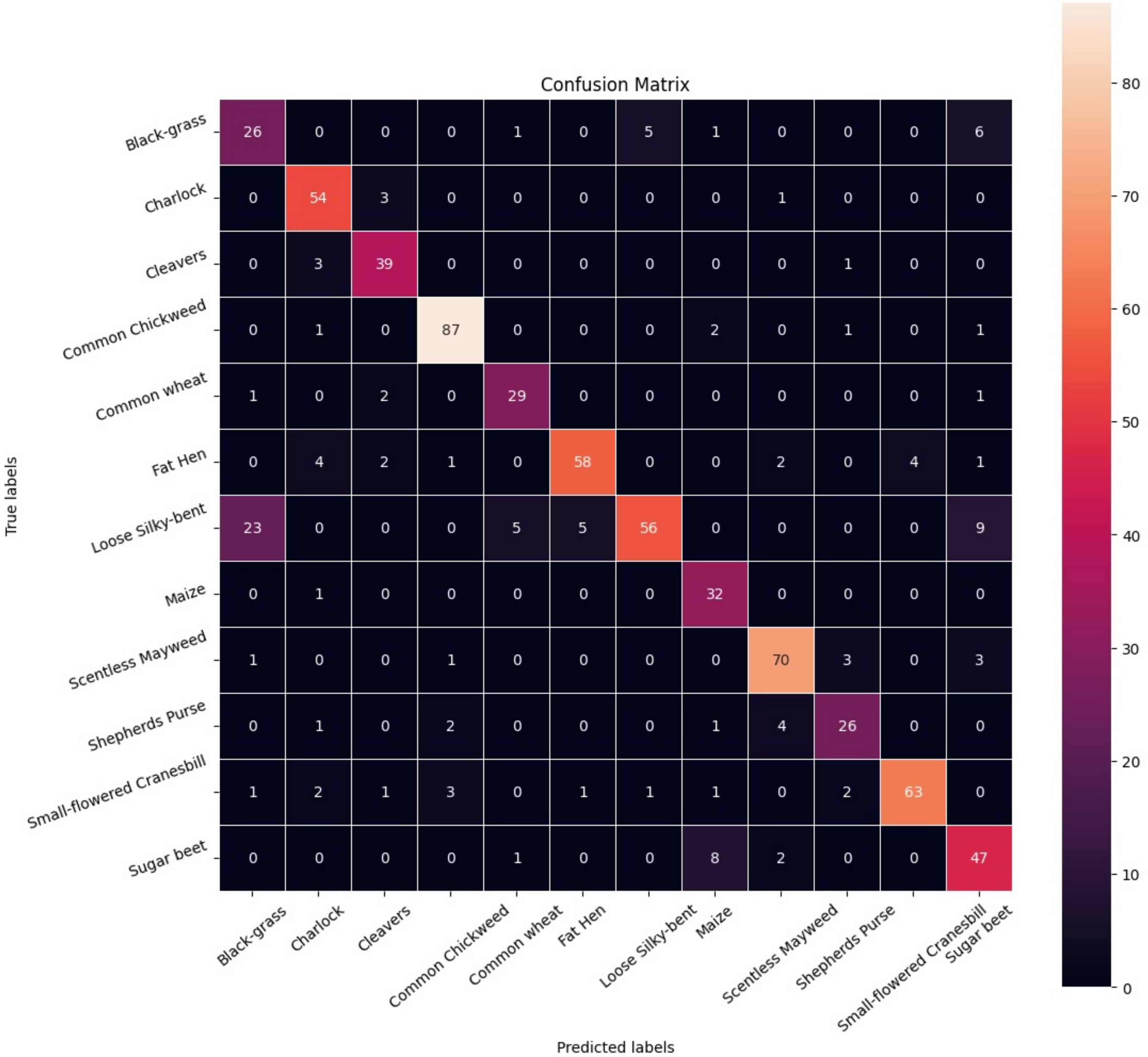
- **Highest Accuracy**: It achieved an overall accuracy of **85%**, outperforming all other models in both generalization and learning ability.
- **Balanced Performance**: The combination of BatchNormalization, Dropout, ReduceLROnPlateau, and EarlyStopping effectively prevented overfitting while optimizing training.
- **Scalability**: The architecture can be scaled or further tuned (e.g., by increasing input size or adding more layers) if needed, providing flexibility for future improvements.

Thus, **Model4** provides the best trade-off between accuracy, computational efficiency, and adaptability for this specific task.

Performance of Final model **(Model-4)** on test data

In [ ]:
```
plot_confusion_matrix_and_report(model4, x_test_normalized, y_test_encoded)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step
```



	precision	recall	f1-score	support
0	0.50	0.67	0.57	39
1	0.82	0.93	0.87	58
2	0.83	0.91	0.87	43
3	0.93	0.95	0.94	92
4	0.81	0.88	0.84	33
5	0.91	0.81	0.85	72
6	0.90	0.57	0.70	98
7	0.71	0.97	0.82	33
8	0.89	0.90	0.89	78
9	0.79	0.76	0.78	34
10	0.94	0.84	0.89	75
11	0.69	0.81	0.75	58
accuracy			0.82	713
macro avg	0.81	0.83	0.81	713
weighted avg	0.84	0.82	0.82	713

Observations:

Multiclass Classification Confusion Matrix

- The matrix shows relatively fewer misclassifications, with most predictions concentrated along the diagonal.
- Most classes, such as `Cleavers`, `Common Chickweed`, and `Fat Hen`, exhibit strong performance with F1-scores above 80%.
- Some classes, such as `Loose Silky-bent` and `Shepherd's Purse`, have misclassifications which indicates room for improvement in handling these categories.

PerformanceMetrics

- The model achieves an accuracy of **82%**, indicating it can correctly classify the majority of test samples.
- **Macro Precision**: **81%**, reflecting balanced precision across all classes.

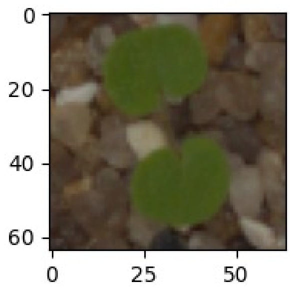
- **Macro Recall**: **83%**, showing good recall for minority classes as well.
- **Macro F1-Score**: **81%**, highlighting consistent performance across all metrics.

Conclusion: The final model demonstrates strong generalization and consistent performance on the test dataset, validating its suitability for practical use.

Visualizing the prediction

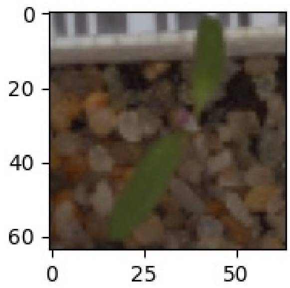
In [ ]:
```
# Visualizing the predicted and correct label of images from test data
def classify_image(index):
  plt.figure(figsize=(2,2))
  plt.imshow(x_test[index])
  plt.show()
  ## Complete the code to predict the test data using the final model selected
  print('Predicted Label', label_binarizer.inverse_transform(model4.predict((x_test_normalized[index].reshape(1
  print('True Label', label_binarizer.inverse_transform(y_test_encoded)[index])
```

In [ ]:
```
classify_image(2)
```



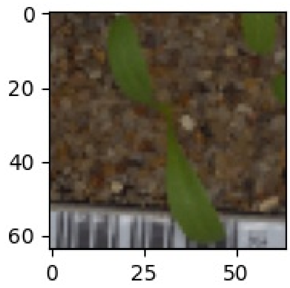
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Predicted Label ['Small-flowered Cranesbill']
True Label Small-flowered Cranesbill
```

In [ ]:
```
classify_image(4)
```



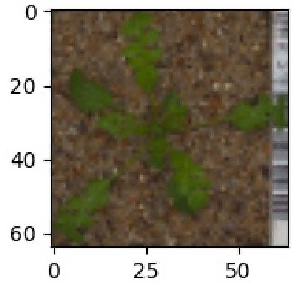
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Predicted Label ['Fat Hen']
True Label Fat Hen
```

In [ ]:
```
classify_image(10)
```



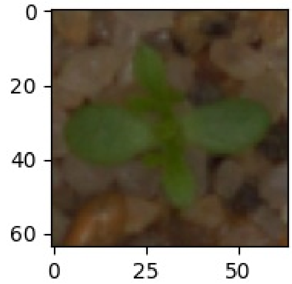
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Predicted Label ['Sugar beet']
True Label Sugar beet
```

In [ ]:
```
classify_image(15)
```



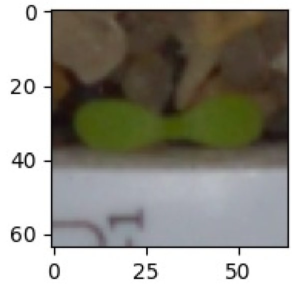

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Predicted Label ['Maize']
True Label Shepherds Purse
```

In [ ]:
```
classify_image(23)
```



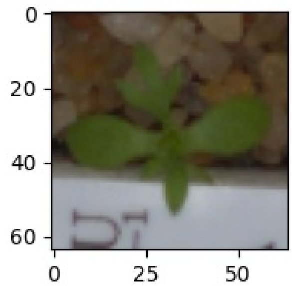

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Predicted Label ['Scentless Mayweed']
True Label Scentless Mayweed
```

In [ ]:
```
classify_image(1)
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Predicted Label ['Common Chickweed']
True Label Scentless Mayweed
```

In [ ]:
```
classify_image(61)
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Predicted Label ['Scentless Mayweed']
True Label Scentless Mayweed
```

Actionable Insights and Business Recommendations

Actionable Insights:

1. **High-performing Classes**: The model performs well on certain categories like `Common Chickweed`, `Cleavers`, and `Fat Hen`, with F1-scores above 80%. These can be trusted for reliable automation in classifying these plant species.

2. **Underperforming Classes**: Classes such as `Loose Silky-bent` and `Shepherd's Purse` show lower recall and precision, potential overlap in visual features or lack of sufficient training observations. Collecting more diverse samples for these classes may improve performance.

3. **Generalization and Robustness on Test Data**: With an overall accuracy of **82%** and balanced class-wise metrics, the model demonstrates good generalization.

Business Recommendations:

1. **Deploy the Model for Automation**: Leverage the model to automate the classification of plant seedlings, reducing manual labor costs and improving efficiency in agricultural operations. Focus on high-performing categories to build initial confidence in the system.

2. **Improve Data for Specific Classes**: Collect more high-quality labeled data for underperforming classes like `Loose Silky-bent` and `Shepherd's Purse`.

3. **Monitor and Enhance Over Time**: A feedback loop where user corrections on misclassifications are incorporated will be benificial. It will ensures continuous improvement and adaptability to new data.

4. **Explore Scalability Using Higher Resolution Images**: Evaluate the model's scalability to larger image sizes if necessary for even finer-grained classification.
