D604 Task 1

March 24, 2025

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf

import matplotlib.pyplot as plt
import seaborn as sns

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.models import Sequential

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
 ↪classification_report
from sklearn.utils.class_weight import compute_class_weight
```

[2]:
```
images = np.load("images.npy")
labels = pd.read_csv("labels.csv")

print(images.shape)

print(labels.shape)
```

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

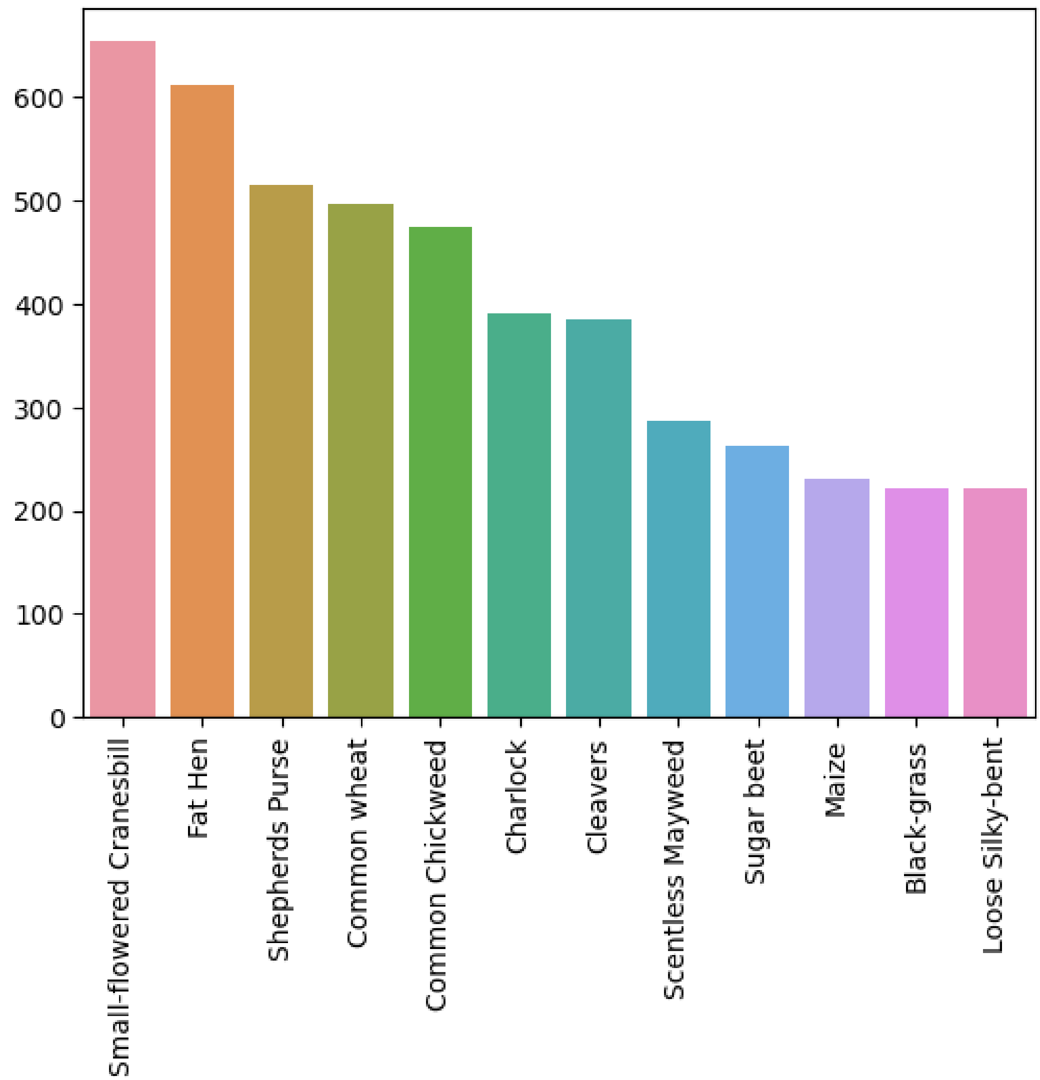
[3]:
```
class_names = labels['Label'].unique().tolist()
class_names
```

[3]:
```
['Small-flowered Cranesbill',
 'Fat Hen',
 'Shepherds Purse',
 'Common wheat',
```

```
 'Common Chickweed',
 'Charlock',
 'Cleavers',
 'Scentless Mayweed',
 'Sugar beet',
 'Maize',
 'Black-grass',
 'Loose Silky-bent']
```

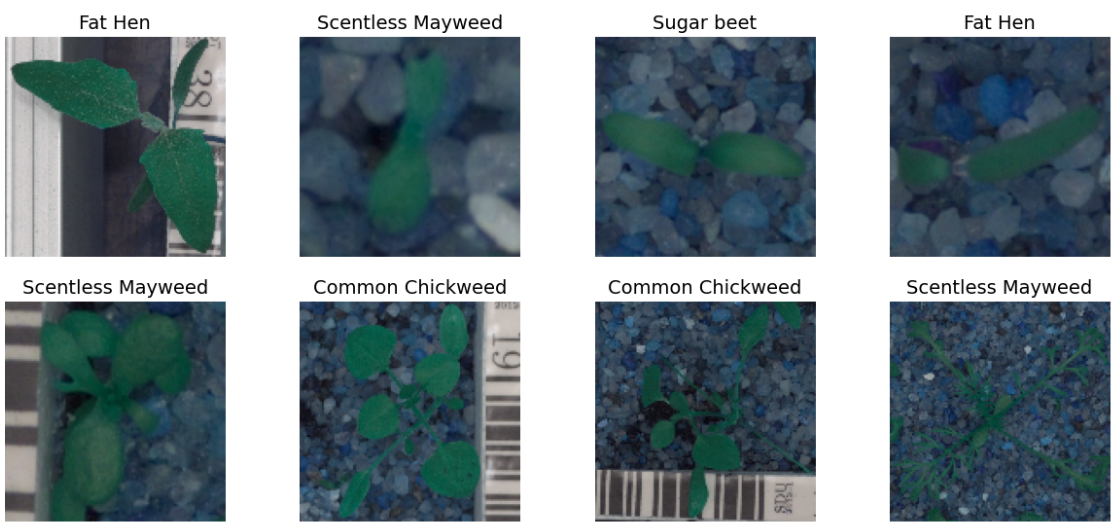
[4]:
```
values = []
for i in range(0, len(class_names)):
    values.append(labels['Label'].value_counts()[i])
```

[5]:
```
sns.barplot(x=class_names, y = values)
plt.xticks(rotation = 90)
plt.show()
```



[6]:
```
#Displaying 8 random images with labels
samps = 8
random_picks = np.random.choice(range(images.shape[0]), samps, replace=False)
random_images = images[random_picks]
samps_labels = labels['Label'].iloc[random_picks]

#Plotting the samples
plt.figure(figsize=(15, 10))
for i, (image, lable) in enumerate(zip(random_images, samps_labels)):
    plt.subplot(3, 4, i + 1)
    plt.imshow(image.astype('uint8'))
    plt.title(lable, fontsize=14)
    plt.axis('off')
plt.show()
```



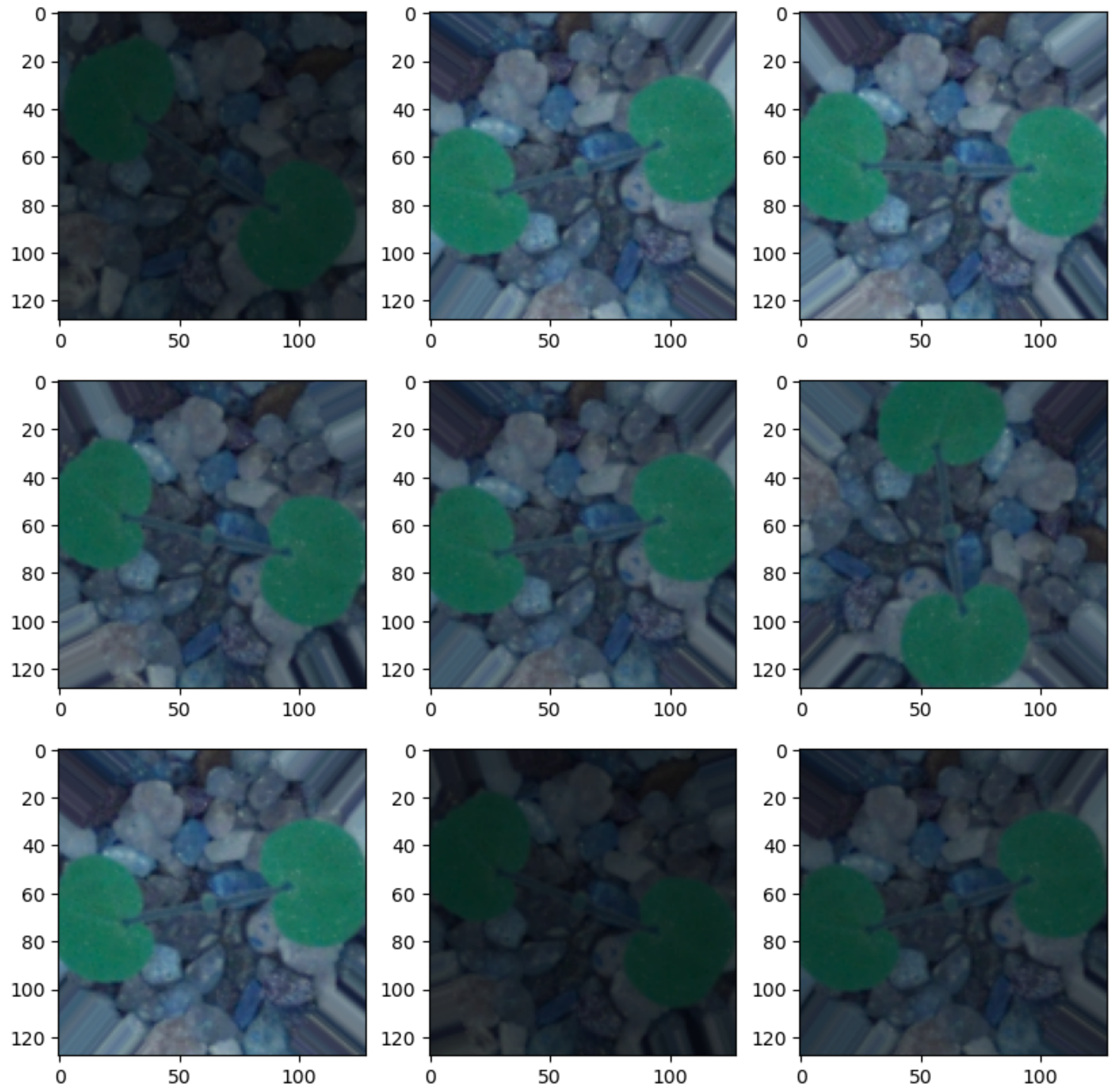
1 B2

[7]:
```
## Augmenting Data

datagen = ImageDataGenerator(brightness_range=[0.5,1.5], rotation_range=60,␣
↪fill_mode='nearest')
```

[8]:
```
#Displaying 8 random images with labels
random_images = images[15].astype('uint8')
random_images = np.expand_dims(random_images, axis=0)
```

```
#Plotting the samples
plt.figure(figsize=(10, 10))
for i, datagen in enumerate(datagen.flow(random_images, batch_size=1)):
    if i == 9:
        break
    plt.subplot(3, 3, i + 1)
    plt.imshow(datagen[0].astype('uint8'))
plt.show()
```



2 B3

[9]:
```
## Normalizing

images = images / 255
```

3 B4

[10]:
```
## Encoding

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels['Label'])
```

[11]:
```
print(label_encoder.classes_)
```

```
['Black-grass' 'Charlock' 'Cleavers' 'Common Chickweed' 'Common wheat'
 'Fat Hen' 'Loose Silky-bent' 'Maize' 'Scentless Mayweed'
 'Shepherds Purse' 'Small-flowered Cranesbill' 'Sugar beet']
```

[12]:
```
## Creating the training, validation, and test sets

X_train, X_test, y_train, y_test = train_test_split(images, encoded_labels,␣
 ↪test_size=0.3, random_state=42, stratify=encoded_labels)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=42, stratify=y_test)
```

4 B5

[13]:
```
## Encoding all datasets to categorical as required by TensorFlow

y_train_enc = to_categorical(y_train)
y_test_enc = to_categorical(y_test)
y_val_enc = to_categorical(y_val)
```

5 B6

[14]:
```
## Providing a copy of all the data sets

np.save('X_train.npy', X_train)
np.save('X_test.npy', X_test)
np.save('X_val.npy', X_val)
np.save('y_train_enc.npy', y_train_enc)
np.save('y_test_enc.npy', y_test_enc)
np.save('y_val_enc.npy', y_val_enc)
```

6 E1

[15]:
```
## From the cats/dogs youtube video recommendation

model = Sequential()

# 1st layer CNN
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(128, 128,␣
↪3)))
model.add(MaxPooling2D(2))

# 2nd layer CNN
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(2))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(len(class_names), activation='softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
model.summary()
```

```
C:\Users\cfman\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential"


 Layer (type)                    Output Shape                  ␣
↪Param #

 conv2d (Conv2D)                 (None, 126, 126, 32)              ␣
↪896

 max_pooling2d (MaxPooling2D)    (None, 63, 63, 32)                ␣
↪  0

 conv2d_1 (Conv2D)               (None, 61, 61, 64)             ␣
↪18,496

 max_pooling2d_1 (MaxPooling2D)  (None, 30, 30, 64)                ␣
↪  0
```

```
 flatten (Flatten)                      (None, 57600)                           ␣
↪   0

 dropout (Dropout)                      (None, 57600)                           ␣
↪   0

 dense (Dense)                          (None, 128)                       ␣
↪7,372,928

 dense_1 (Dense)                        (None, 12)                              ␣
↪1,548
```

Total params: 7,393,868 (28.21 MB)

Trainable params: 7,393,868 (28.21 MB)

Non-trainable params: 0 (0.00 B)

[16]:
```
test_loss, test_acc = model.evaluate(X_test, y_test_enc)
print(f"Test Accuracy: {test_acc}")
```

```
23/23                 1s 20ms/step -
accuracy: 0.0518 - loss: 0.7082
Test Accuracy: 0.06451612710952759
```

[17]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)
```

[18]:
```
history = model.fit(X_train, y_train_enc, validation_data=(X_val, y_val_enc),
                    epochs=20, batch_size=32, callbacks=[early_stopping])
```

```
Epoch 1/20
104/104               9s 81ms/step -
accuracy: 0.1645 - loss: 0.3192 - val_accuracy: 0.3933 - val_loss: 0.2236
Epoch 2/20
104/104               8s 78ms/step -
accuracy: 0.4196 - loss: 0.2168 - val_accuracy: 0.5056 - val_loss: 0.1954
Epoch 3/20
104/104               8s 77ms/step -
accuracy: 0.5045 - loss: 0.1849 - val_accuracy: 0.6053 - val_loss: 0.1682
Epoch 4/20
104/104               8s 77ms/step -
accuracy: 0.6075 - loss: 0.1552 - val_accuracy: 0.5604 - val_loss: 0.1744
Epoch 5/20
```

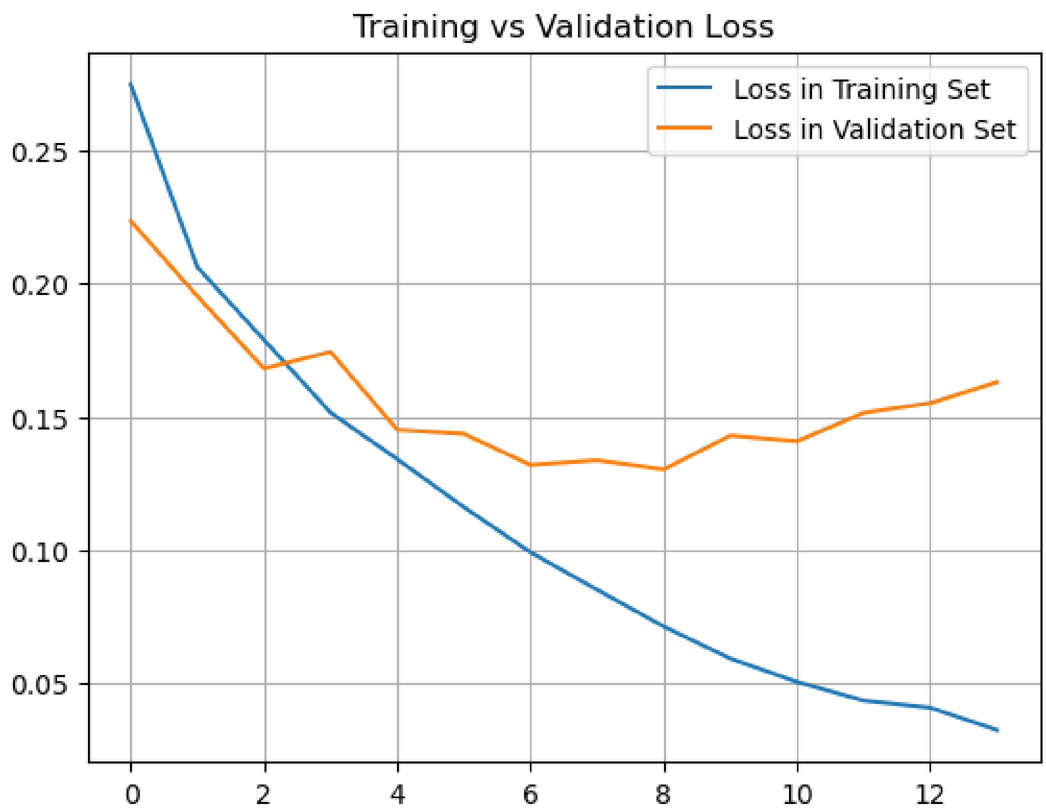
```
104/104              8s 77ms/step -
accuracy: 0.6669 - loss: 0.1391 - val_accuracy: 0.6545 - val_loss: 0.1452
Epoch 6/20
104/104              8s 77ms/step -
accuracy: 0.7270 - loss: 0.1179 - val_accuracy: 0.6587 - val_loss: 0.1436
Epoch 7/20
104/104              8s 79ms/step -
accuracy: 0.8000 - loss: 0.0979 - val_accuracy: 0.7008 - val_loss: 0.1320
Epoch 8/20
104/104              8s 79ms/step -
accuracy: 0.8430 - loss: 0.0838 - val_accuracy: 0.7093 - val_loss: 0.1337
Epoch 9/20
104/104              8s 81ms/step -
accuracy: 0.8854 - loss: 0.0666 - val_accuracy: 0.7233 - val_loss: 0.1303
Epoch 10/20
104/104              9s 83ms/step -
accuracy: 0.8897 - loss: 0.0611 - val_accuracy: 0.7219 - val_loss: 0.1429
Epoch 11/20
104/104              8s 81ms/step -
accuracy: 0.9155 - loss: 0.0503 - val_accuracy: 0.7163 - val_loss: 0.1408
Epoch 12/20
104/104              8s 79ms/step -
accuracy: 0.9294 - loss: 0.0422 - val_accuracy: 0.7065 - val_loss: 0.1515
Epoch 13/20
104/104              9s 87ms/step -
accuracy: 0.9347 - loss: 0.0420 - val_accuracy: 0.7303 - val_loss: 0.1552
Epoch 14/20
104/104              9s 86ms/step -
accuracy: 0.9588 - loss: 0.0320 - val_accuracy: 0.7022 - val_loss: 0.1630
```

[19]:
```
test_loss, test_acc = model.evaluate(X_test, y_test_enc)
print(f"Test Accuracy: {test_acc}")
```

```
23/23              0s 19ms/step -
accuracy: 0.7086 - loss: 0.1301
Test Accuracy: 0.7068723440170288
```

[20]:
```
## Plotting the loss in the training set compared to the loss in the validation␣
↪set

plt.plot(history.history['loss'], label='Loss in Training Set')
plt.plot(history.history['val_loss'], label='Loss in Validation Set')
plt.legend()
plt.title("Training vs Validation Loss")
plt.grid()
plt.show()
```

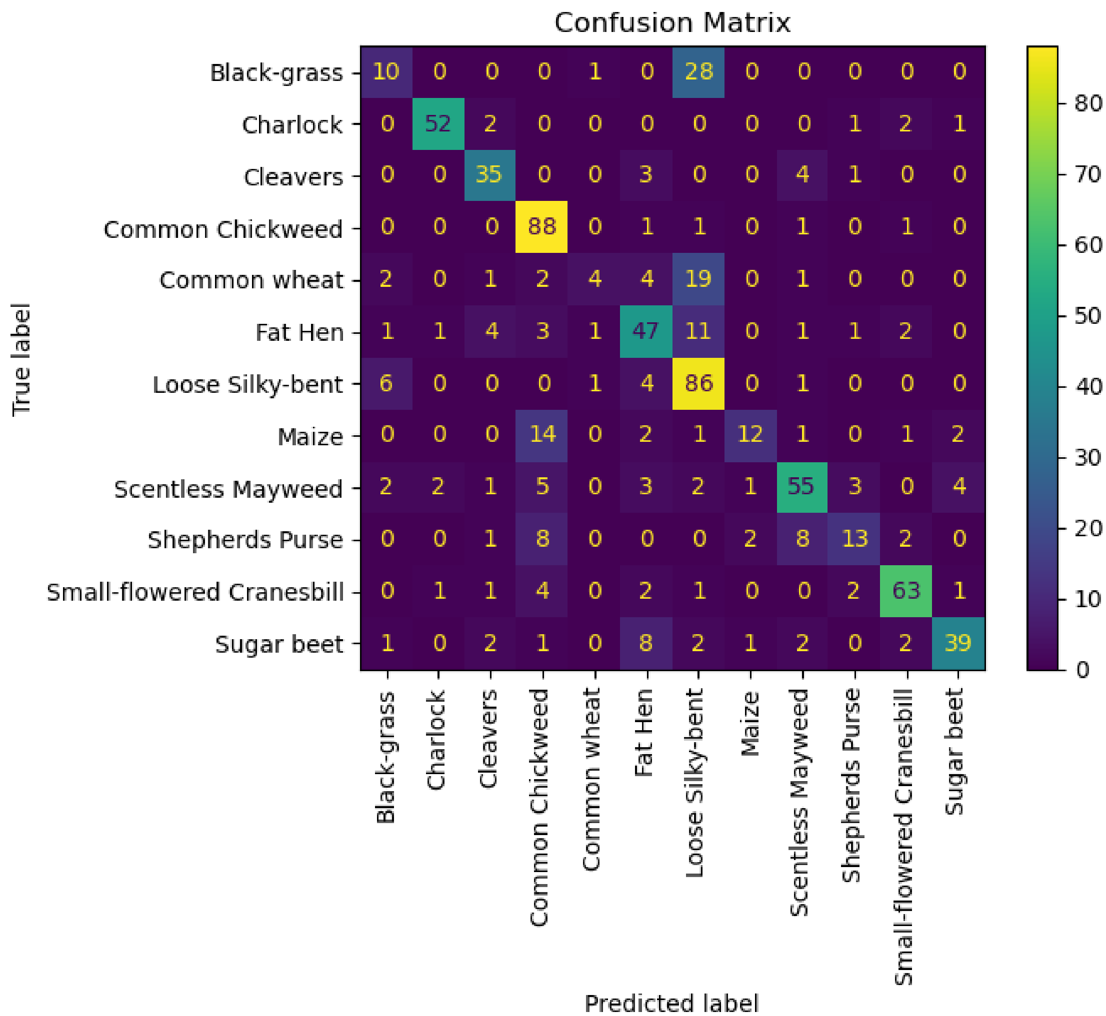


[26]:
```
## Accuracy tests
y_pred = np.argmax(model.predict(X_test), axis=1)


## For help understanding what each number means
#for i in range(0, len(label_encoder.classes_)):
            #print(label_encoder.classes_[i], ':', i)

## Building the confusion matrix
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cnf_matrix, display_labels = label_encoder.
 ↪classes_)
disp.plot()
plt.title("Confusion Matrix")
plt.xticks(rotation = 90)
plt.show()
print(classification_report(y_test, y_pred, target_names = label_encoder.
 ↪classes_))
```

```
23/23                 0s 17ms/step
```



	precision	recall	f1-score	support
Black-grass	0.45	0.26	0.33	39
Charlock	0.93	0.90	0.91	58
Cleavers	0.74	0.81	0.78	43
Common Chickweed	0.70	0.96	0.81	92
Common wheat	0.57	0.12	0.20	33
Fat Hen	0.64	0.65	0.64	72
Loose Silky-bent	0.57	0.88	0.69	98
Maize	0.75	0.36	0.49	33
Scentless Mayweed	0.74	0.71	0.72	78
Shepherds Purse	0.62	0.38	0.47	34
Small-flowered Cranesbill	0.86	0.84	0.85	75
Sugar beet	0.83	0.67	0.74	58
accuracy			0.71	713

```
   macro avg       0.70      0.63      0.64       713
weighted avg       0.71      0.71      0.69       713
```

[27]:
```
#Saving the
model.save("final_plant_model.keras")
```