Jacob Stonelake

Final Project

Computer Vision Image Analysis

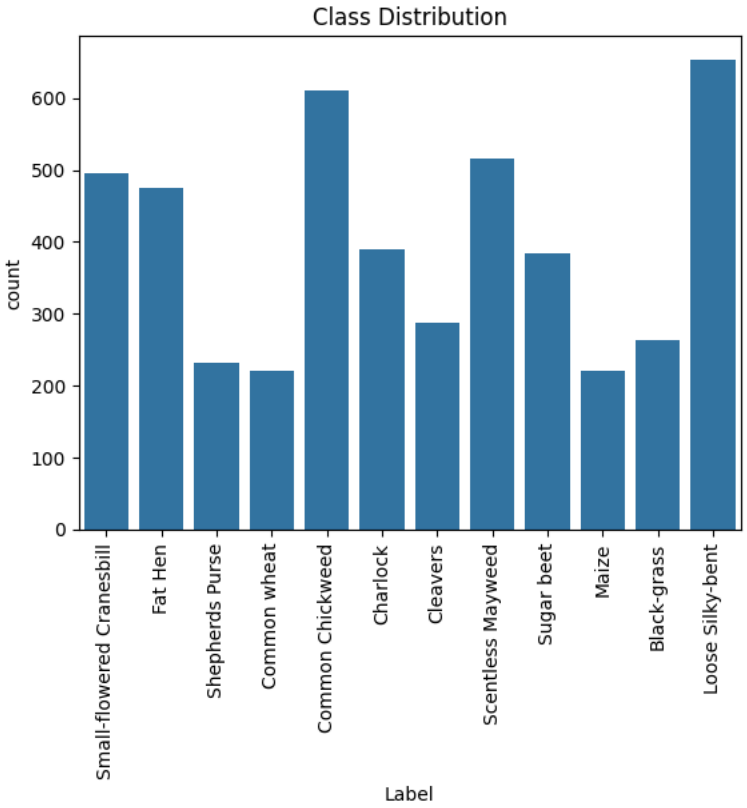
Step 2: Load Dataset and Summary

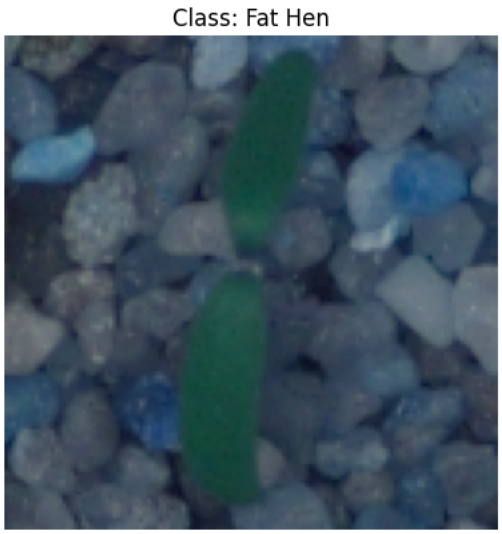
```
#Import libraries and keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import warnings

from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report, confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical


warnings.filterwarnings("ignore")

#Load dataset
images = np.load('images.npy')
labels = pd.read_csv('Labels.csv')
```

Step 3: Perform EDA on the Images

```
#Show class distribution in bar chart
sns.countplot(x='Label', data=labels)
plt.title("Class Distribution")
plt.xticks(rotation=90)
plt.show()

#View some images in dataset
unique_classes = labels['Label'].unique()
for label in unique_classes:
    index = labels[labels['Label'] == label].index[0]
    plt.imshow(images[index])
    plt.title(f"Class: {label}")
    plt.axis('off')
    plt.show()
```

[+]





Class: Shepherds Purse

Class: Common wheat

Class: Common Chickweed

Class: Charlock

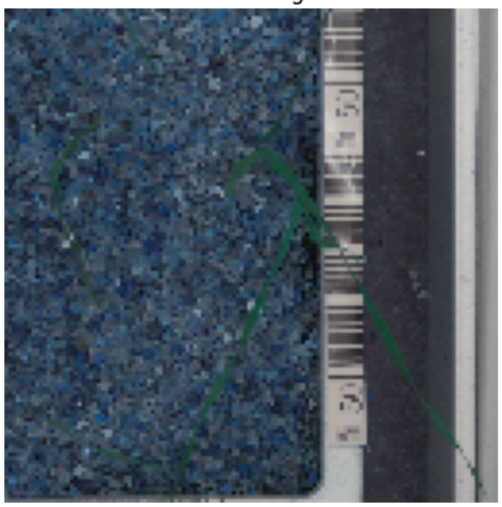
Class: Cleavers

Class: Scentless Mayweed

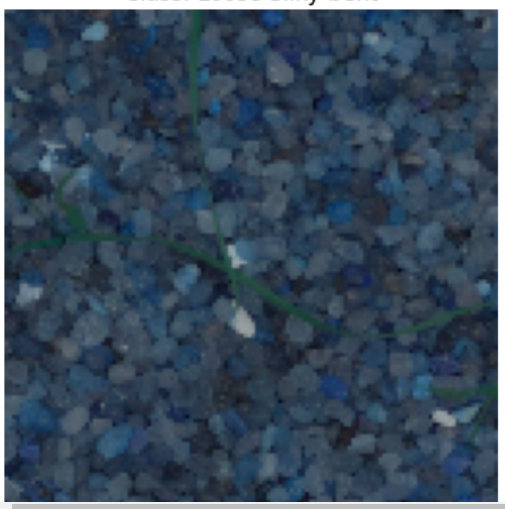
Class: Sugar beet

Class: Maize

Class: Black-grass



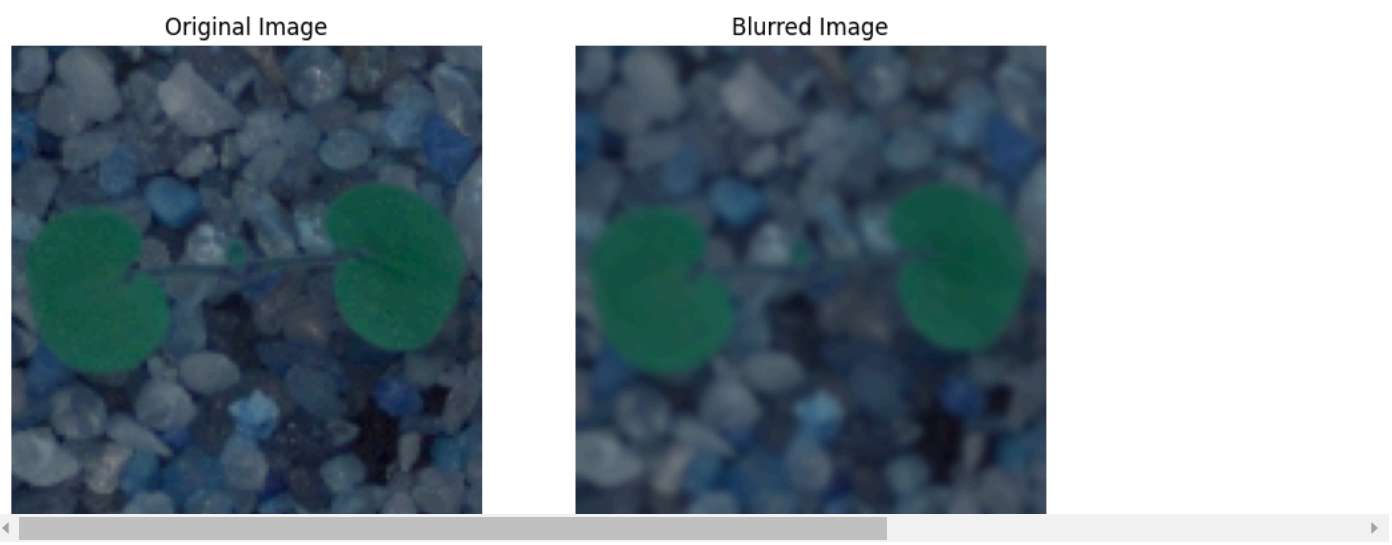
Class: Loose Silky-bent



Step 4: Illustrate Insights from EDA The dataset is fairly balanced. The species have distinct characteristics.

Step 5: Data Pre-Processing

```
#Apply blurring
def apply_gaussian_blur(images, kernel_size=(5, 5)):
    blurred_images = np.array([cv2.GaussianBlur(img, kernel_size, 0) for img in images])
    return blurred_images

#Normalize images
images_normalized = images / 255.0

blurred_images = apply_gaussian_blur(images_normalized)

#Show the effects applied
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(images_normalized[0])
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(blurred_images[0])
plt.title("Blurred Image")
plt.axis("off")

plt.show()


# Split Data
labels['Label_Index'] = labels['Label'].factorize()[0]
y = labels['Label_Index'].values

X_train, X_test, y_train, y_test = train_test_split(
    images_normalized, y, test_size=0.2, random_state=42, stratify=y
)
```



Step 6: Make Data Compatible

```
#One-hot encode, print shape to ensure compatiblity
y_train_one_hot = to_categorical(y_train, num_classes=len(labels['Label'].unique()))
y_test_one_hot = to_categorical(y_test, num_classes=len(labels['Label'].unique()))

print(f"Training Data Shape: {X_train.shape}")
print(f"Testing Data Shape: {X_test.shape}")

#Augment images
train_augmentation = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.3,
```

```
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True
)
```

```
Training Data Shape: (3800, 128, 128, 3)
Testing Data Shape: (950, 128, 128, 3)
```

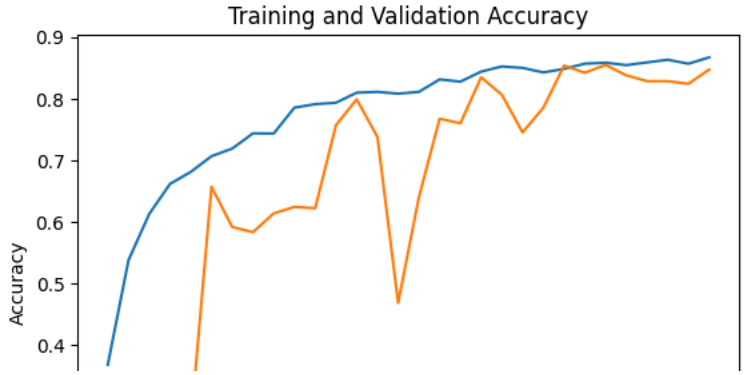
Step 7: Model Building

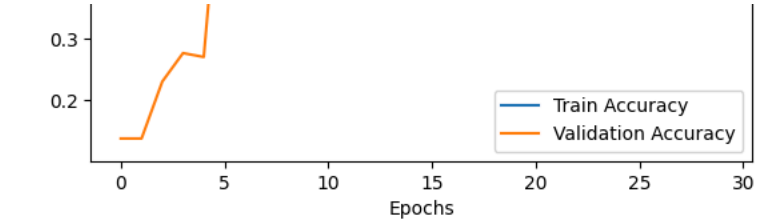
```
#Build a CNN with layers
def build_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(128, 128, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(len(labels['Label'].unique()), activation='softmax'))
    return model

model = build_cnn_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Step 8: Model Training

```
#Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1)

#Train model
history = model.fit(
    X_train, y_train_one_hot,
    validation_data=(X_test, y_test_one_hot),
    epochs=30,
    callbacks=[early_stopping, lr_scheduler],
    batch_size=32
)
#View our model performance
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
Epoch 1/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 17s 100ms/step - accuracy: 0.2769 - loss: 2.0931 - val_accuracy: 0.1379 - val_loss: 2.9068 - learning_rate:
Epoch 2/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.5179 - loss: 1.3599 - val_accuracy: 0.1379 - val_loss: 5.0772 - learning_rate: 0
Epoch 3/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6003 - loss: 1.1777 - val_accuracy: 0.2305 - val_loss: 4.4204 - learning_rate: 0
Epoch 4/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6515 - loss: 1.0221 - val_accuracy: 0.2768 - val_loss: 2.4546 - learning_rate: 0
Epoch 5/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6663 - loss: 0.9538 - val_accuracy: 0.2705 - val_loss: 4.1410 - learning_rate: 0
Epoch 6/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7116 - loss: 0.8386 - val_accuracy: 0.6568 - val_loss: 1.0374 - learning_rate: 0
Epoch 7/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7142 - loss: 0.8209 - val_accuracy: 0.5916 - val_loss: 1.2596 - learning_rate: 0
Epoch 8/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7442 - loss: 0.7345 - val_accuracy: 0.5832 - val_loss: 1.2209 - learning_rate: 0
Epoch 9/30
116/119 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7421 - loss: 0.7157
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.7422 - loss: 0.7160 - val_accuracy: 0.6137 - val_loss: 1.1949 - learning_rate: 0
Epoch 10/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.7701 - loss: 0.6325 - val_accuracy: 0.6242 - val_loss: 1.2712 - learning_rate: 5
Epoch 11/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.7963 - loss: 0.5643 - val_accuracy: 0.6221 - val_loss: 1.1765 - learning_rate: 5
Epoch 12/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.7918 - loss: 0.5661 - val_accuracy: 0.7568 - val_loss: 0.7202 - learning_rate: 5
Epoch 13/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8089 - loss: 0.5431 - val_accuracy: 0.7989 - val_loss: 0.5931 - learning_rate: 5
Epoch 14/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8197 - loss: 0.5159 - val_accuracy: 0.7379 - val_loss: 0.7803 - learning_rate: 5
Epoch 15/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8073 - loss: 0.5379 - val_accuracy: 0.4684 - val_loss: 2.0335 - learning_rate: 5
Epoch 16/30
116/119 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8143 - loss: 0.5236
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8142 - loss: 0.5239 - val_accuracy: 0.6400 - val_loss: 0.9960 - learning_rate: 5
Epoch 17/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8293 - loss: 0.4987 - val_accuracy: 0.7674 - val_loss: 0.6631 - learning_rate: 2
Epoch 18/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8231 - loss: 0.4675 - val_accuracy: 0.7600 - val_loss: 0.7036 - learning_rate: 2
Epoch 19/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8384 - loss: 0.4549 - val_accuracy: 0.8347 - val_loss: 0.4964 - learning_rate: 2
Epoch 20/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8500 - loss: 0.4304 - val_accuracy: 0.8063 - val_loss: 0.5472 - learning_rate: 2
Epoch 21/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8484 - loss: 0.4104 - val_accuracy: 0.7453 - val_loss: 0.7920 - learning_rate: 2
Epoch 22/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8427 - loss: 0.4323
Epoch 22: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8427 - loss: 0.4324 - val_accuracy: 0.7853 - val_loss: 0.6328 - learning_rate: 2
Epoch 23/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8547 - loss: 0.4173 - val_accuracy: 0.8537 - val_loss: 0.4499 - learning_rate: 1
Epoch 24/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8517 - loss: 0.3963 - val_accuracy: 0.8421 - val_loss: 0.4612 - learning_rate: 1
Epoch 25/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8649 - loss: 0.3751 - val_accuracy: 0.8547 - val_loss: 0.4543 - learning_rate: 1
Epoch 26/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8501 - loss: 0.4031 - val_accuracy: 0.8379 - val_loss: 0.4475 - learning_rate: 1
Epoch 27/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8547 - loss: 0.3802 - val_accuracy: 0.8284 - val_loss: 0.5654 - learning_rate: 1
Epoch 28/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8756 - loss: 0.3694 - val_accuracy: 0.8284 - val_loss: 0.5465 - learning_rate: 1
Epoch 29/30
117/119 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8599 - loss: 0.3962
Epoch 29: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8598 - loss: 0.3959 - val_accuracy: 0.8242 - val_loss: 0.4869 - learning_rate: 1
Epoch 30/30
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8681 - loss: 0.3658 - val_accuracy: 0.8474 - val_loss: 0.4476 - learning_rate: 6
```





Step 9: Model Performance Evaluation python Copy code

```
#View test accuracy, validation accuracy, and confusion matrix
test_loss, test_accuracy = model.evaluate(X_test, y_test_one_hot)
print(f"Test Accuracy: {test_accuracy:.2f}, Test Loss: {test_loss:.2f}")

y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, y_pred_labels)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels['Label'].unique(), yticklabels=labels['Label'].unique())
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

print(classification_report(y_test, y_pred_labels, target_names=labels['Label'].unique()))
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8547 - loss: 0.3959
Test Accuracy: 0.84, Test Loss: 0.45
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step
```