```
from google.colab import drive
drive.mount('/content/drive')
```

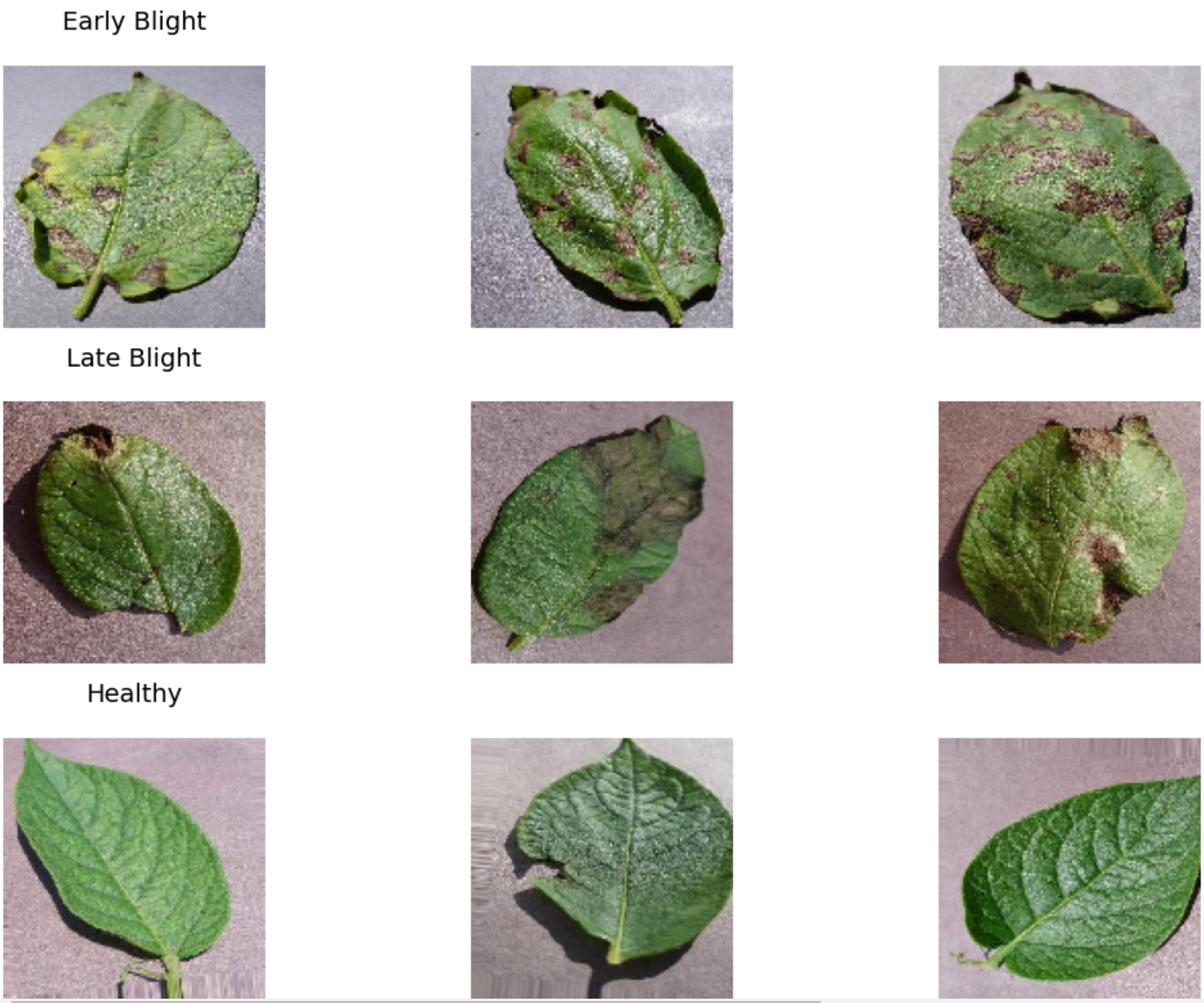
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Step 1: Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
```

```
# Import necessary libraries for SVM
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
```

```
# Step 2: Set up paths to the data directories
train_dir = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Train'
test_dir = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Test'
valid_dir = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Valid'
```

```
# Function to display images from each category
def display_images(category_paths, labels, num_images=3):
    plt.figure(figsize=(12, 8))

    for i, (category_path, label) in enumerate(zip(category_paths, labels)):
        image_files = os.listdir(category_path)
        selected_images = random.sample(image_files, num_images)

        for j, img_name in enumerate(selected_images):
            img_path = os.path.join(category_path, img_name)
            img = load_img(img_path, target_size=(128, 128))
            ax = plt.subplot(len(category_paths), num_images, i * num_images + j + 1)
            plt.imshow(img)
            plt.axis('off')
            if j == 0:
                ax.set_title(label, fontsize=14, pad=20)

    plt.tight_layout()
    plt.show()

# Paths to each class in the Test set
early_blight_path = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Test/Potato___Early_blight'
late_blight_path = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Test/Potato___Late_blight'
healthy_path = '/content/drive/MyDrive/3.1/Ml/project/data/Potato/Test/Potato___healthy'

# Display sample images from each category
display_images(
    category_paths=[early_blight_path, late_blight_path, healthy_path],
    labels=['Early Blight', 'Late Blight', 'Healthy'],
    num_images=3  # Display 3 images from each category
)
```




```
# Step 3: Data Loading and Preprocessing
# Set up ImageDataGenerator for loading the images
datagen = ImageDataGenerator(rescale=1.0/255.0)

# Load training data
train_data = datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),  # Resize images to 128x128
    batch_size=32,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=True
)

# Load validation data
valid_data = datagen.flow_from_directory(
    valid_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=False
)

# Load test data
test_data = datagen.flow_from_directory(
    test_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=False
)
```

```
Found 900 images belonging to 3 classes.
Found 300 images belonging to 3 classes.
Found 300 images belonging to 3 classes.
```

```
# Step 4: Flatten the image data to use it in SVM
# Extract features and labels from the ImageDataGenerator objects
def extract_features(data):
    features = []
    labels = []
    for batch in data:
        X_batch, y_batch = batch
        for i in range(X_batch.shape[0]):
            features.append(X_batch[i].flatten())  # Flatten each image
            labels.append(np.argmax(y_batch[i]))  # Convert one-hot to label index
        if len(features) >= data.samples:  # Stop when all images are processed
            break
    return np.array(features), np.array(labels)

X_train, y_train = extract_features(train_data)
X_valid, y_valid = extract_features(valid_data)
X_test, y_test = extract_features(test_data)
```

```
# Step 5: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

```
# Step 6: Hyperparameter Tuning using GridSearchCV for SVM
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = grid_search.best_params_
best_svm = grid_search.best_estimator_

print(f"\nBest Hyperparameters: {best_params}")
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits

Best Hyperparameters: {'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
```

```
# Step 7: Model Training with the best parameters
best_svm.fit(X_train, y_train)
```

SVC

```
SVC(C=0.1, gamma=1, kernel='linear')
```

```
# Step 8: Predictions and Evaluation on Test Data
y_pred = best_svm.predict(X_test)
```

```
# Step 9: Performance Evaluation
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print(f"Accuracy Score: {accuracy_score(y_test, y_pred):.2f}")
```

```
Confusion Matrix:
[[98  2  0]
 [ 6 91  3]
 [ 0 14 86]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       100
           1       0.85      0.91      0.88       100
           2       0.97      0.86      0.91       100

    accuracy                           0.92       300
```

```
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300

Accuracy Score: 0.92
```

```
# Step 10: Cross-Validation Score
cv_scores = cross_val_score(best_svm, X_train, y_train, cv=5)
print(f"\n5-Fold Cross-Validation Accuracy: {cv_scores.mean():.2f} ± {cv_scores.std():.2f}")
```

```
5-Fold Cross-Validation Accuracy: 0.91 ± 0.01
```

```
# Step 11: Visualization of Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', xticklabels=train_data.class_indices, yticklabels=trair
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```