```
! pip install opencv-python
```

```
import cv2
from tqdm import tqdm
from sklearn.utils import shuffle
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("")

print("Path to dataset files:", path)
```

```
import shutil

shutil.make_archive("potato_dataset", 'zip', path)
```

```
# Path to the folder where all images are kept (without train/test split)
data_path = '/content/potato_dataset.zip'
```

```
labels = ['early_bliht','late_blight','healthy']
CLASS_NAMES = labels
```

```
!unzip /content/potato_dataset.zip -d /content
```

```
data_path="/content/Potato"
```

```
from collections import defaultdict
category_counts = defaultdict(int)

# Loop over each category folder
for category in os.listdir(data_path):
    category_folder = os.path.join(data_path, category)
    # Check if the item is a directory before processing
    if os.path.isdir(category_folder):
        image_count = len(os.listdir(category_folder))
        category_counts[category] = image_count

# Prepare for plotting
labels = list(category_counts.keys())
counts = list(category_counts.values())
plt.figure(figsize=(12, 6))
sns.barplot(x=counts, y=labels, palette="viridis", orient="h")

for i, count in enumerate(counts):
    plt.text(count + 0.5, i, str(count), va='center', fontsize=10, fontweight='bold', color='black')
```

```
plt.title("Number of Images Per Category", fontsize=18, fontweight='bold')
plt.xlabel("Number of Images", fontsize=14)
plt.ylabel("Categories", fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the chart
plt.show()
```

```
#List of all labels (categories)
labels = os.listdir(data_path)

# Lists to hold images and their labels
X = []
y = []

image_size = 224

# Loop through each category folder
for i in labels:
    folderPath = os.path.join(data_path, i)
    if os.path.isdir(folderPath):
        # Loop through each image in the category
        for j in tqdm(os.listdir(folderPath)):
            img_path = os.path.join(folderPath, j)
            img = cv2.imread(img_path)
            img = cv2.resize(img, (image_size, image_size))  # Resize to 224x224
            X.append(img)
            y.append(i)

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

print(f"Training set size: {len(X_train)} images")
print(f"Testing set size: {len(X_test)} images")


colors_dark = ["#1F1F1F", "#313131", '#636363', '#AEAEAE', '#DADADA']
colors_red = ["#331313", "#582626", '#9E1717', '#D35151', '#E9B4B4']
colors_green = ['#01411C','#4B6F44','#4F7942','#74C365','#D0F0C0']


k=0
fig, ax = plt.subplots(1,3,figsize=(20,20))
fig.text(s='Sample Image From Each Class',size=18,fontweight='bold',
             fontname='monospace',color=colors_dark[1],y=0.62,x=0.4,alpha=0.8)
for i in labels:
    j=0
    while True :
        if y_train[j]==i:
            ax[k].imshow(X_train[j])
            ax[k].set_title(y_train[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1



train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    data_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    data_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

#  Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(len(CLASS_NAMES), activation='softmax')
```

```
])

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

#  Train
model.fit(train_generator, epochs=2, validation_data=val_generator)

#  Evaluate
loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")

#  Confusion matrix
y_pred = model.predict(val_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator.classes

cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES, cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Optional classification report
print(classification_report(y_true, y_pred_classes, target_names=CLASS_NAMES))
```

```
import random
import numpy as np
import matplotlib.pyplot as plt

# Randomly select one test image
random_idx = random.randint(0, len(X_test) - 1)
test_image = X_test[random_idx]
true_label = y_test[random_idx]
image_size=64
# Resize to model input size and normalize
img = cv2.resize(test_image, (image_size, image_size))
img = img.astype("float32") / 255.0

# Expand dimensions to match model input shape: (1, 64, 64, 3)
img = np.expand_dims(img, axis=0)

# Predict the class
pred_prob = model.predict(img)
pred_class = np.argmax(pred_prob, axis=1)[0]

#Display the image
plt.imshow(test_image)
plt.axis("off")
plt.title(f"Predicted: {CLASS_NAMES[pred_class]}, Actual: {true_label}")
plt.show()
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```