```
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Initialize the sequential model
model_custom = Sequential()

# First convolutional layer
model_custom.add(Conv2D(32, (3, 3), input_shape=(256, 256, 3), activation='relu'))
model_custom.add(BatchNormalization())
model_custom.add(MaxPooling2D(pool_size=(2, 2)))
model_custom.add(Dropout(0.25))

# Second convolutional layer
model_custom.add(Conv2D(64, (3, 3), activation='relu'))
model_custom.add(BatchNormalization())
model_custom.add(MaxPooling2D(pool_size=(2, 2)))
model_custom.add(Dropout(0.25))

# Third convolutional layer
model_custom.add(Conv2D(128, (3, 3), activation='relu'))
model_custom.add(BatchNormalization())
model_custom.add(MaxPooling2D(pool_size=(2, 2)))
model_custom.add(Dropout(0.25))

# Fourth convolutional layer
model_custom.add(Conv2D(128, (3, 3), activation='relu'))
model_custom.add(BatchNormalization())
model_custom.add(MaxPooling2D(pool_size=(2, 2)))
model_custom.add(Dropout(0.25))

# Fifth convolutional layer
model_custom.add(Conv2D(256, (3, 3), activation='relu'))
model_custom.add(BatchNormalization())
model_custom.add(MaxPooling2D(pool_size=(2, 2)))
model_custom.add(Dropout(0.25))

# Global average pooling
model_custom.add(GlobalAveragePooling2D())

# Dense layers
model_custom.add(Dense(256, activation='relu'))
model_custom.add(Dropout(0.5))
model_custom.add(Dense(128, activation='relu'))
model_custom.add(Dropout(0.5))
model_custom.add(Dense(3, activation='softmax'))

# Compile the model

compilation= model_custom.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy','confusion_metics'])

# Summary of the model
model_custom.summary()

# Data augmentation and normalization
train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    brightness_range=[0.5, 1.3],
    fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load the training, validation, and test datasets
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/potato_leaf_disease_classification/Train', target_size=(256, 256
validation_set = val_datagen.flow_from_directory('/content/drive/MyDrive/potato_leaf_disease_classification/Validation', target_size=(256
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/potato_leaf_disease_classification/Test', target_size=(256, 256), bat

# Train the model
history = model_custom.fit(training_set, steps_per_epoch=20, epochs=50, validation_data=validation_set, validation_steps=20)

# Evaluate the model
# Evaluate the model on the test set
# Print the results
#print("\n\n")
#print("Test Loss: \t", test_loss, "\n")
#print("Test Accuracy: \t", test_acc, "\n")
#print("Test Precision: \t", test_precision, "\n")
#print("Test Recall: \t", test_recall, "\n")
#print("Test F1 Score: \t", test_f1, "\n")
test_loss, test_acc = model_custom.evaluate(test_set, verbose=2)
print("\n\n")
print("Test Loss: \t", test_loss, "\n")
print("Test Accuracy: \t", test_acc, "\n")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃          Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 254, 254, 32)        │              896 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization                  │ (None, 254, 254, 32)        │              128 │
│ (BatchNormalization)                 │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 127, 127, 32)        │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout (Dropout)                    │ (None, 127, 127, 32)        │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 125, 125, 64)        │           18,496 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_1                │ (None, 125, 125, 64)        │              256 │
│ (BatchNormalization)                 │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 62, 62, 64)          │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_1 (Dropout)                  │ (None, 62, 62, 64)          │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 60, 60, 128)         │           73,856 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_2                │ (None, 60, 60, 128)         │              512 │
│ (BatchNormalization)                 │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 30, 30, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_2 (Dropout)                  │ (None, 30, 30, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 28, 28, 128)         │          147,584 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_3                │ (None, 28, 28, 128)         │              512 │
│ (BatchNormalization)                 │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 14, 14, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_3 (Dropout)                  │ (None, 14, 14, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 12, 12, 256)         │          295,168 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_4                │ (None, 12, 12, 256)         │            1,024 │
│ (BatchNormalization)                 │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 6, 6, 256)           │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_4 (Dropout)                  │ (None, 6, 6, 256)           │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ global_average_pooling2d             │ (None, 256)                 │                0 │
│ (GlobalAveragePooling2D)             │                             │                  │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense (Dense)                        │ (None, 256)                 │           65,792 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_5 (Dropout)                  │ (None, 256)                 │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_1 (Dense)                      │ (None, 128)                 │           32,896 │
```

```
y_pred = model_custom.predict(test_set)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_set.classes
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:\n", cm)
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step
Confusion Matrix:
 [[117  68  15]
 [115  75  10]
 [  8  19   3]]
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 218s 9s/step - accuracy: 0.5147 - loss: 1.2894 - val_accuracy: 0.4688 - val_loss: 0.9183
```

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model_custom.predict(test_set)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
y_true = test_set.classes
cm = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:\n", cm)
report = classification_report(y_true, y_pred_classes, target_names=test_set.class_indices.keys())
print("Classification Report:\n", report)
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step
Confusion Matrix:
 [[109  70  21]
 [113  80   7]
 [ 18  12   0]]
Classification Report:
                         precision    recall  f1-score   support

Potato___Early_blight       0.45      0.55      0.50       200
 Potato___Late_blight       0.49      0.40      0.44       200
     Potato___healthy       0.00      0.00      0.00        30

             accuracy                           0.44       430
            macro avg       0.32      0.32      0.31       430
         weighted avg       0.44      0.44      0.44       430

20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 328ms/step - accuracy: 0.8329 - loss: 0.4219 - val_accuracy: 0.5455 - val_loss: 2.7436
```

```
results = model_custom.evaluate(test_set, verbose=2)

# Extract loss and accuracy
test_loss = results[0]
test_acc = results[1]

# Predict probabilities for the test set
y_pred_proba = model_custom.predict(test_set)

# Get predicted classes
y_pred = np.argmax(y_pred_proba, axis=1)

# Get true classes
y_true = test_set.classes

# Calculate precision, recall, and F1-score
# Note: 'macro' average calculates metrics globally by considering each class independently
from sklearn.metrics import precision_score, recall_score, f1_score # Import necessary metrics
test_precision = precision_score(y_true, y_pred, average='macro')
test_recall = recall_score(y_true, y_pred, average='macro')
test_f1 = f1_score(y_true, y_pred, average='macro')

# Print the results
print("\n\n")
print("Test Loss: \t", test_loss, "\n")
print("Test Accuracy: \t", test_acc, "\n")
print("Test Precision: \t", test_precision, "\n")
print("Test Recall: \t", test_recall, "\n")
print("Test F1 Score: \t", test_f1, "\n")
```

```
27/27 - 2s - 68ms/step - accuracy: 0.8744 - loss: 0.3023
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step



Test Loss:       0.30227696895599365

Test Accuracy:   0.8744186162948608

Test Precision:          0.33628747795414465

Test Recall:     0.3377777777777778

Test F1 Score:   0.33392854050122106

Epoch 40/50
```

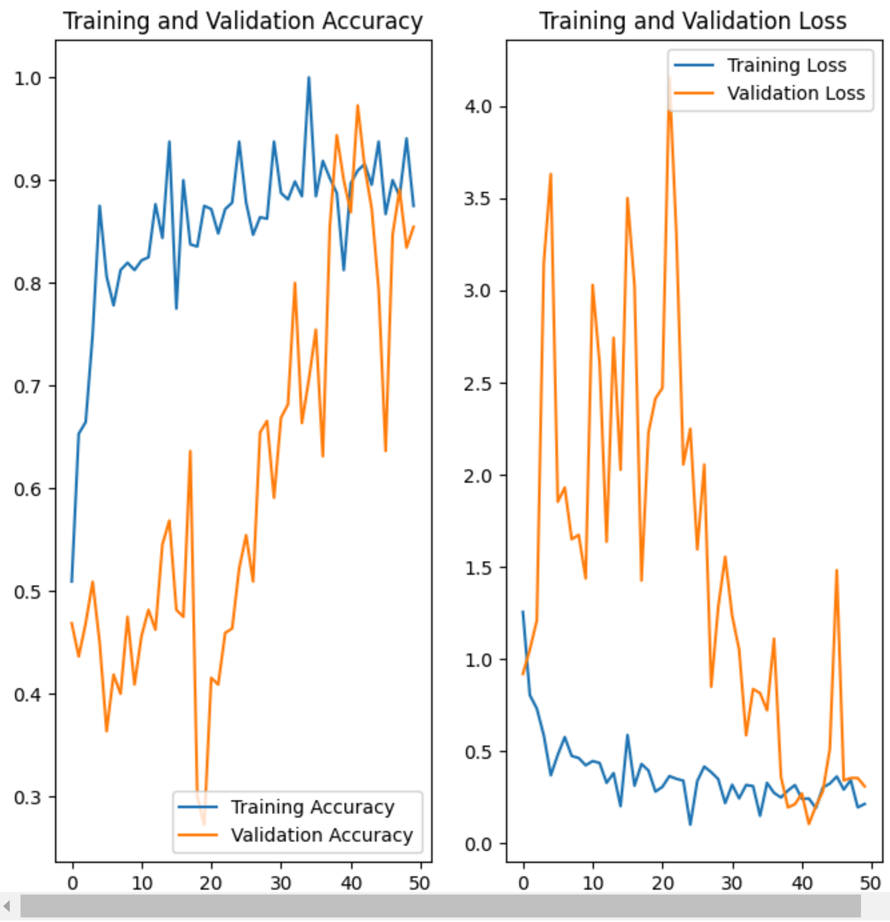
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.
```

```python
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).
```

```python
from matplotlib import pyplot as plt
EPOCHS = 50
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



```
model_custom.save('final_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is 
```

Start coding or generate with AI.

```
import numpy as np
from keras.preprocessing import image
import keras.utils as image

# Load and preprocess the test image
test_image = image.load_img('/content/drive/MyDrive/Potato_Healthy.jpg', target_size=(256, 256))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Predict the class of the test image
result = model_custom.predict(test_image)
print(result)

# Assuming the class indices mapping
# training_set.class_indices will return a dictionary like {'class1': 0, 'class2': 1, 'class3': 2}
class_indices = training_set.class_indices
class_labels = {v: k for k, v in class_indices.items()}  # Reverse the dictionary to get labels from indices

# Get the predicted class index
predicted_class_index = np.argmax(result, axis=1)[0]
prediction = class_labels[predicted_class_index]

print(prediction)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
[[0.0000000e+00 1.0000000e+00 1.2121233e-18]]
Potato___Late_blight
```

```
import numpy as np
from keras.preprocessing import image
import keras.utils as image

# Load and preprocess the test image
test_image = image.load_img('/content/drive/MyDrive/test_potato_early_blight.jpg', target_size=(256, 256))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
```

```
# Predict the class of the test image
result = model_custom.predict(test_image)
print(result)

# Assuming the class indices mapping
# training_set.class_indices will return a dictionary like {'class1': 0, 'class2': 1, 'class3': 2}
class_indices = training_set.class_indices
```