Plant Disease Detection Using Convolutional Neural Networks

January 24, 2025

1 Plant Disease Detection Using Convolutional Neural Networks

This project aims to develop a Convolutional Neural Network (CNN) model to detect plant diseases from images. The dataset used is the PlantVillage dataset, which contains images of healthy and diseased plant leaves. The project involves the following steps:

1. Data Loading and Preprocessing: Load the dataset, resize, and rescale the images.
2. Model Architecture: Build a CNN model using TensorFlow's Sequential API.
3. Model Training: Train the model on the training dataset and validate it on the validation dataset.
4. Model Evaluation: Evaluate the model's performance on the test dataset.
5. Visualization: Plot the training and validation accuracy and loss over epochs.
6. Predictions: Make predictions on the test dataset and visualize the results.
7. Model Saving: Save the trained model for future use.

[1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
```

1. **Data Loading and Preprocessing**: Load the dataset, resize, and rescale the images.

[2]:
```
# Constants Variables
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 50
```

[3]:
```
# Load the dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
)
```

```
Found 2152 files belonging to 3 classes.
```

```
2025-01-24 23:29:44.886048: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M3 Pro
2025-01-24 23:29:44.886073: I metal_plugin/src/device/metal_device.cc:296]
```

```
systemMemory: 18.00 GB
2025-01-24 23:29:44.886076: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 6.00 GB
2025-01-24 23:29:44.886103: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2025-01-24 23:29:44.886112: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

[4]:

```
# Print the class names
class_names = dataset.class_names

# Get the class names
print("Dataset Classes:")
for i, class_name in enumerate(class_names):
    print(f"{i + 1}. {class_name.replace('_', ' ')}")
```

```
Dataset Classes:
1. Potato Early blight
2. Potato Late blight
3. Potato healthy
```

[5]:

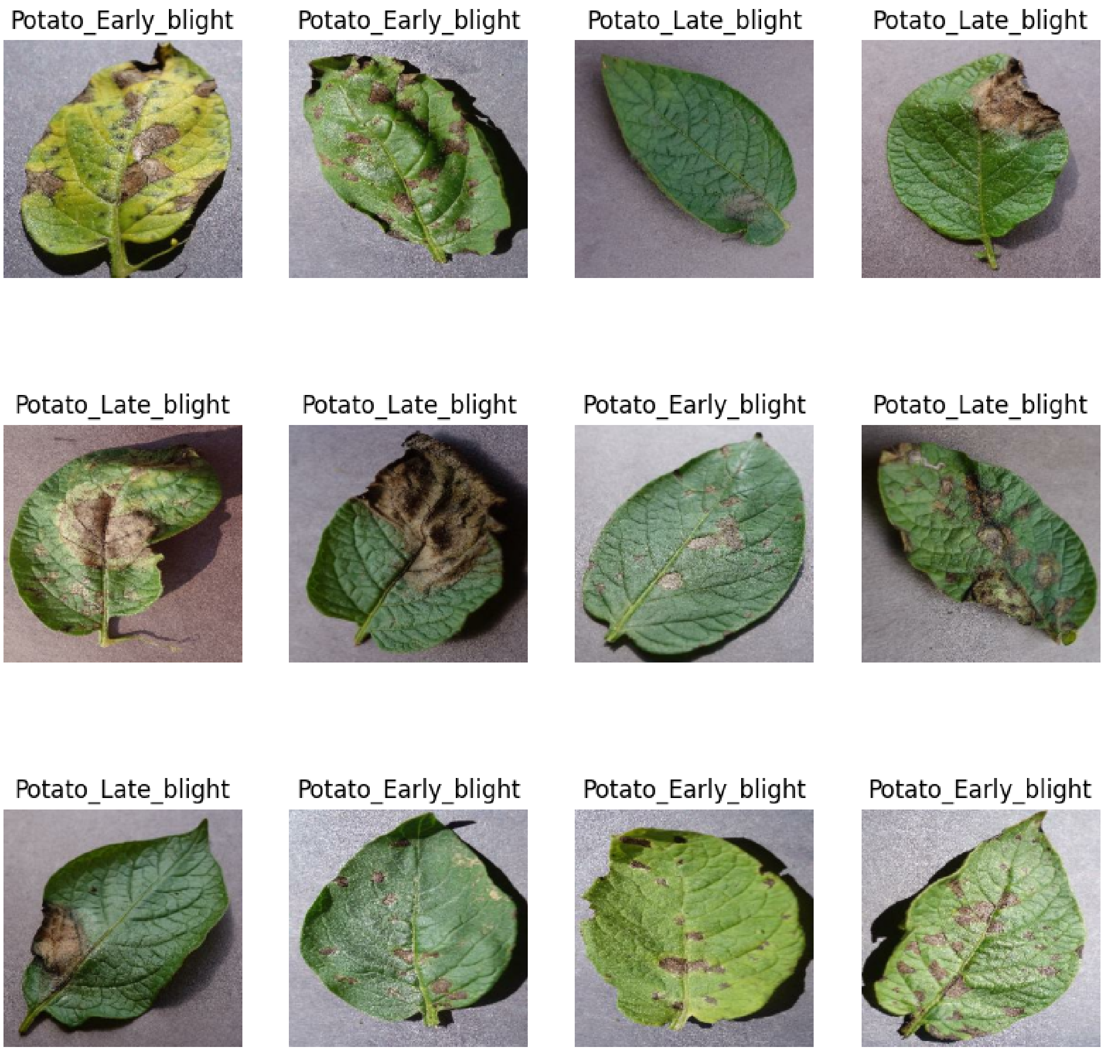
```
# Length of the dataset
dataset_length = len(dataset)
print(f"Dataset Length: {dataset_length}")
```

```
Dataset Length: 68
```

[6]:

```
plt.figure(figsize=(10, 10))
for iamge_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(iamge_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```

```
2025-01-24 23:29:45.320216: W tensorflow/core/framework/local_rendezvous.cc:404]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```



[7]:
```python
# Train, Validation and Test Split
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    # Verify that split ratios sum to 1
    assert (train_split + test_split + val_split) == 1

    # Get the total size of the dataset
    ds_size = len(ds)

    # Shuffle the dataset if specified
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    # Calculate sizes for train and validation sets
```

```
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    # Split dataset into train, validation and test sets
    train_ds = ds.take(train_size)    # Take first train_size elements for␣
 ↪training
    val_ds = ds.skip(train_size).take(val_size)    # Skip training data, take␣
 ↪val_size elements for validation
    test_ds = ds.skip(train_size).skip(val_size)    # Skip training and␣
 ↪validation data for test set

    return train_ds, val_ds, test_ds
```

[8]:
```
# Get the partitions
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

# Print the length of the datasets
print("Train Dataset: ", len(train_ds))
print("Validation Dataset: ", len(val_ds))
print("Test Dataset: ", len(test_ds))
```

```
Train Dataset:  54
Validation Dataset:  6
Test Dataset:  8
```

[9]:
```
# Train DS Cache, Prefetch and Shuffle
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.
 ↪experimental.AUTOTUNE)

# Validation DS Cache, Prefetch and Shuffle
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experimental.
 ↪AUTOTUNE)

# Test DS Cache, Prefetch and Shuffle
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.
 ↪experimental.AUTOTUNE)
```

[10]:
```
# Resize and Rescale the images using Sequential API
resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.Rescaling(1./255),
])
```

[11]:
```
# Data Augmentation using Sequential API
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
```

```
])
```

Model Architecture: Build a CNN model using TensorFlow's Sequential API.

[12]:
```
# Nural Network Architecture

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

# Create the model using Sequential API
model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',␣
 ↪input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

```
/opt/anaconda3/envs/metal-env/lib/python3.10/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)
```

[13]:
```
# Print the model summary
model.summary()
```

```
Model: "sequential_2"


 Layer (type)                     Output Shape                  Param #

 sequential (Sequential)          (32, 256, 256, 3)                   0
```

conv2d (Conv2D)	(32, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(32, 127, 127, 32)	0
conv2d_1 (Conv2D)	(32, 125, 125, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_2 (Conv2D)	(32, 60, 60, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	0
conv2d_3 (Conv2D)	(32, 28, 28, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(32, 14, 14, 64)	0
conv2d_4 (Conv2D)	(32, 12, 12, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(32, 6, 6, 64)	0
conv2d_5 (Conv2D)	(32, 4, 4, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(32, 2, 2, 64)	0
flatten (Flatten)	(32, 256)	0
dense (Dense)	(32, 64)	16,448
dense_1 (Dense)	(32, 3)	195

Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)

Model Training: Train the model on the training dataset and validate it on the validation dataset.

[14]:
```
# Compile the model
model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
```

```
)
```

[15]:
```python
# Train the model using fit
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    batch_size = BATCH_SIZE,
    verbose=1,
)
```

```
Epoch 1/50
```

```
/opt/anaconda3/envs/metal-env/lib/python3.10/site-
packages/keras/src/backend/tensorflow/nn.py:708: UserWarning:
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `output`
argument was produced by a Softmax activation and thus does not represent
logits. Was this intended?
  output, from_logits = _get_logits(
2025-01-24 23:29:46.428625: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
```

```
54/54              6s 87ms/step -
accuracy: 0.4855 - loss: 0.9233 - val_accuracy: 0.7344 - val_loss: 0.7751
Epoch 2/50
54/54              4s 73ms/step -
accuracy: 0.7159 - loss: 0.6528 - val_accuracy: 0.8594 - val_loss: 0.3000
Epoch 3/50
54/54              4s 71ms/step -
accuracy: 0.8052 - loss: 0.4436 - val_accuracy: 0.7865 - val_loss: 0.5052
Epoch 4/50
54/54              4s 71ms/step -
accuracy: 0.8497 - loss: 0.3531 - val_accuracy: 0.9375 - val_loss: 0.1730
Epoch 5/50
54/54              4s 70ms/step -
accuracy: 0.9453 - loss: 0.1370 - val_accuracy: 0.9583 - val_loss: 0.1683
Epoch 6/50
54/54              4s 71ms/step -
accuracy: 0.9409 - loss: 0.1371 - val_accuracy: 0.9792 - val_loss: 0.0892
Epoch 7/50
54/54              4s 70ms/step -
accuracy: 0.9610 - loss: 0.1148 - val_accuracy: 0.9688 - val_loss: 0.0801
Epoch 8/50
54/54              4s 70ms/step -
accuracy: 0.9460 - loss: 0.1307 - val_accuracy: 0.9740 - val_loss: 0.0645
Epoch 9/50
54/54              4s 70ms/step -
accuracy: 0.9705 - loss: 0.0788 - val_accuracy: 0.9635 - val_loss: 0.0870
```

```
Epoch 10/50
54/54            4s 70ms/step -
accuracy: 0.9739 - loss: 0.0710 - val_accuracy: 0.8333 - val_loss: 0.3802
Epoch 11/50
54/54            4s 71ms/step -
accuracy: 0.9418 - loss: 0.1550 - val_accuracy: 0.9792 - val_loss: 0.0554
Epoch 12/50
54/54            4s 70ms/step -
accuracy: 0.9943 - loss: 0.0251 - val_accuracy: 0.9531 - val_loss: 0.1640
Epoch 13/50
54/54            4s 70ms/step -
accuracy: 0.9852 - loss: 0.0492 - val_accuracy: 0.9948 - val_loss: 0.0251
Epoch 14/50
54/54            4s 70ms/step -
accuracy: 0.9947 - loss: 0.0151 - val_accuracy: 0.9479 - val_loss: 0.1734
Epoch 15/50
54/54            4s 71ms/step -
accuracy: 0.9725 - loss: 0.0758 - val_accuracy: 0.9792 - val_loss: 0.0718
Epoch 16/50
54/54            4s 70ms/step -
accuracy: 0.9883 - loss: 0.0382 - val_accuracy: 0.9688 - val_loss: 0.1030
Epoch 17/50
54/54            4s 71ms/step -
accuracy: 0.9876 - loss: 0.0384 - val_accuracy: 0.9948 - val_loss: 0.0159
Epoch 18/50
54/54            4s 70ms/step -
accuracy: 0.9972 - loss: 0.0101 - val_accuracy: 0.9948 - val_loss: 0.0248
Epoch 19/50
54/54            4s 71ms/step -
accuracy: 0.9986 - loss: 0.0040 - val_accuracy: 0.9948 - val_loss: 0.0058
Epoch 20/50
54/54            4s 70ms/step -
accuracy: 0.9959 - loss: 0.0062 - val_accuracy: 1.0000 - val_loss: 0.0030
Epoch 21/50
54/54            4s 70ms/step -
accuracy: 0.9998 - loss: 0.0016 - val_accuracy: 1.0000 - val_loss: 0.0056
Epoch 22/50
54/54            4s 74ms/step -
accuracy: 1.0000 - loss: 8.0519e-04 - val_accuracy: 1.0000 - val_loss:
6.1671e-04
Epoch 23/50
54/54            4s 80ms/step -
accuracy: 1.0000 - loss: 3.2719e-04 - val_accuracy: 1.0000 - val_loss:
3.3542e-04
Epoch 24/50
54/54            4s 80ms/step -
accuracy: 1.0000 - loss: 1.7377e-04 - val_accuracy: 1.0000 - val_loss:
3.1924e-04
```

```
Epoch 25/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 2.1863e-04 - val_accuracy: 1.0000 - val_loss:
2.3263e-04
Epoch 26/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 1.4914e-04 - val_accuracy: 1.0000 - val_loss:
1.7398e-04
Epoch 27/50
54/54            4s 76ms/step -
accuracy: 1.0000 - loss: 1.2335e-04 - val_accuracy: 1.0000 - val_loss:
1.3393e-04
Epoch 28/50
54/54            4s 75ms/step -
accuracy: 1.0000 - loss: 1.0239e-04 - val_accuracy: 1.0000 - val_loss:
1.1247e-04
Epoch 29/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 6.0933e-05 - val_accuracy: 1.0000 - val_loss:
1.0076e-04
Epoch 30/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 8.4621e-05 - val_accuracy: 1.0000 - val_loss:
9.1918e-05
Epoch 31/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 6.9011e-05 - val_accuracy: 1.0000 - val_loss:
8.0663e-05
Epoch 32/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 6.3797e-05 - val_accuracy: 1.0000 - val_loss:
7.0260e-05
Epoch 33/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 5.1162e-05 - val_accuracy: 1.0000 - val_loss:
6.1019e-05
Epoch 34/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 5.5293e-05 - val_accuracy: 1.0000 - val_loss:
5.3250e-05
Epoch 35/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 2.6842e-05 - val_accuracy: 1.0000 - val_loss:
4.5389e-05
Epoch 36/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 3.7752e-05 - val_accuracy: 1.0000 - val_loss:
4.2019e-05
```

```
Epoch 37/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 2.6252e-05 - val_accuracy: 1.0000 - val_loss:
3.6705e-05
Epoch 38/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 2.2441e-05 - val_accuracy: 1.0000 - val_loss:
3.2829e-05
Epoch 39/50
54/54            4s 76ms/step -
accuracy: 1.0000 - loss: 1.9143e-05 - val_accuracy: 1.0000 - val_loss:
2.7075e-05
Epoch 40/50
54/54            4s 83ms/step -
accuracy: 1.0000 - loss: 1.4802e-05 - val_accuracy: 1.0000 - val_loss:
2.3498e-05
Epoch 41/50
54/54            4s 76ms/step -
accuracy: 1.0000 - loss: 1.5437e-05 - val_accuracy: 1.0000 - val_loss:
2.0372e-05
Epoch 42/50
54/54            4s 76ms/step -
accuracy: 1.0000 - loss: 1.3965e-05 - val_accuracy: 1.0000 - val_loss:
1.7390e-05
Epoch 43/50
54/54            4s 71ms/step -
accuracy: 1.0000 - loss: 1.2252e-05 - val_accuracy: 1.0000 - val_loss:
1.3574e-05
Epoch 44/50
54/54            4s 82ms/step -
accuracy: 1.0000 - loss: 1.1145e-05 - val_accuracy: 1.0000 - val_loss:
1.1673e-05
Epoch 45/50
54/54            4s 72ms/step -
accuracy: 1.0000 - loss: 1.0230e-05 - val_accuracy: 1.0000 - val_loss:
1.0245e-05
Epoch 46/50
54/54            4s 71ms/step -
accuracy: 1.0000 - loss: 1.1499e-05 - val_accuracy: 1.0000 - val_loss:
9.1297e-06
Epoch 47/50
54/54            4s 71ms/step -
accuracy: 1.0000 - loss: 6.5025e-06 - val_accuracy: 1.0000 - val_loss:
6.6589e-06
Epoch 48/50
54/54            4s 70ms/step -
accuracy: 1.0000 - loss: 7.4407e-06 - val_accuracy: 1.0000 - val_loss:
7.2669e-06
```

```
Epoch 49/50
54/54              4s 70ms/step -
accuracy: 1.0000 - loss: 5.4190e-06 - val_accuracy: 1.0000 - val_loss:
6.3819e-06
Epoch 50/50
54/54              4s 71ms/step -
accuracy: 1.0000 - loss: 5.7029e-06 - val_accuracy: 1.0000 - val_loss:
4.8648e-06
```

Model Evaluation: Evaluate the model's performance on the test dataset.

[16]:
```python
# Print the Score
score = model.evaluate(test_ds)

# Print the History
print("History: ", history.history)

# Print the History Parameters
print("History Parameters: ", history.params)

# Print the History Keys
print("History Keys: ", history.history.keys())
```

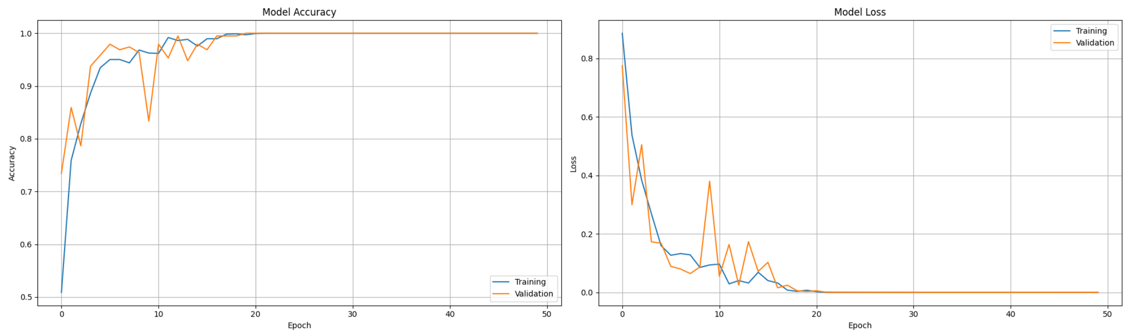
```
8/8             1s 23ms/step -
accuracy: 1.0000 - loss: 2.3917e-05
History:  {'accuracy': [0.5086805820465088, 0.7586805820465088,
0.8287037014961243, 0.8865740895271301, 0.9346064925193787, 0.9502314925193787,
0.9502314925193787, 0.9438657164573669, 0.9681712985038757, 0.9623842835426331,
0.9618055820465088, 0.9918981194496155, 0.9861111044883728, 0.9884259104728699,
0.9756944179534912, 0.9895833134651184, 0.9895833134651184, 0.9982638955116272,
0.9988425970077515, 0.9971064925193787, 0.9994212985038757, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], 'loss': [0.8861762881278992,
0.5380606055259705, 0.38371923565864563, 0.26927873492240906,
0.15991713106632233, 0.12705282866954803, 0.13288123905658722,
0.12859532237052917, 0.08547067642211914, 0.0939326211810112,
0.0967046245932579, 0.029325591400265694, 0.04042663425207138,
0.03214135766029358, 0.06893835216760635, 0.040527280420064926,
0.03230319917201996, 0.00788001250475645, 0.003639115719124675,
0.007490135263651609, 0.0026996140368282795, 0.000605887093115598,
0.0002898953389376402, 0.0002095387171721086, 0.00020424139802344143,
0.0001545710110804066, 0.00011307737440802157, 9.909149230225012e-05,
8.22138026705943e-05, 7.984159310581163e-05, 6.455412949435413e-05,
5.523480285773985e-05, 4.985101622878574e-05, 4.510027065407485e-05,
3.7614281609421596e-05, 3.357157766004093e-05, 2.938467514468357e-05,
2.6195879399892874e-05, 2.2167097995406948e-05, 1.9590112060541287e-05,
1.8399536202196032e-05, 1.5167736819421407e-05, 1.3569278053182643e-05,
1.0944606401608326e-05, 1.0249548722640611e-05, 8.511266059940681e-06,
6.82707286614459e-06, 7.577144060633145e-06, 5.3504022616834845e-06,
```

```
6.363929060171358e-06], 'val_accuracy': [0.734375, 0.859375, 0.7864583134651184,
0.9375, 0.9583333134651184, 0.9791666865348816, 0.96875, 0.9739583134651184,
0.9635416865348816, 0.8333333134651184, 0.9791666865348816, 0.953125,
0.9947916865348816, 0.9479166865348816, 0.9791666865348816, 0.96875,
0.9947916865348816, 0.9947916865348816, 0.9947916865348816, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], 'val_loss':
[0.7751259207725525, 0.2999683916568756, 0.5052155256271362,
0.17300207912921906, 0.1683455854654312, 0.08924367278814316,
0.08008333295583725, 0.06446651369333267, 0.08698394149541855,
0.3802114427089691, 0.05536839738488197, 0.1639692485332489,
0.025141308084130287, 0.17344509065151215, 0.07181329280138016,
0.10299161821603775, 0.015894034877419472, 0.02477092854678631,
0.0058439853601157665, 0.0030389565508812666, 0.005625460296869278,
0.0006167085375636816, 0.00033541544689796865, 0.00031924361246638,
0.0002326324611203745, 0.00017397967167198658, 0.0001339261798420921,
0.000112465291749686, 0.00010075725003844127, 9.191766002913937e-05,
8.066268492257223e-05, 7.026007369859144e-05, 6.101901817601174e-05,
5.3249805205268785e-05, 4.5388605940388516e-05, 4.2019204556709155e-05,
3.670513251563534e-05, 3.282892794231884e-05, 2.7075473553850316e-05,
2.3497637812397443e-05, 2.037180274783168e-05, 1.738955506880302e-05,
1.3573971955338493e-05, 1.1673376320686657e-05, 1.0244657460134476e-05,
9.12974701350322e-06, 6.658910479018232e-06, 7.266888133017346e-06,
6.381856564985355e-06, 4.8648334995959885e-06]}
History Parameters:  {'verbose': 1, 'epochs': 50, 'steps': 54}
History Keys:  dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Visualization: Plot the training and validation accuracy and loss over epochs.

[17]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Create figure and axis objects with a single subplot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# Plot training & validation accuracy values
ax1.plot(history.history['accuracy'], label='Training')
ax1.plot(history.history['val_accuracy'], label='Validation')
ax1.set_title('Model Accuracy')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.legend(loc='lower right')
ax1.grid(True)

# Plot training & validation loss values
ax2.plot(history.history['loss'], label='Training')
ax2.plot(history.history['val_loss'], label='Validation')
ax2.set_title('Model Loss')
```

```
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Loss')
ax2.legend(loc='upper right')
ax2.grid(True)

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```

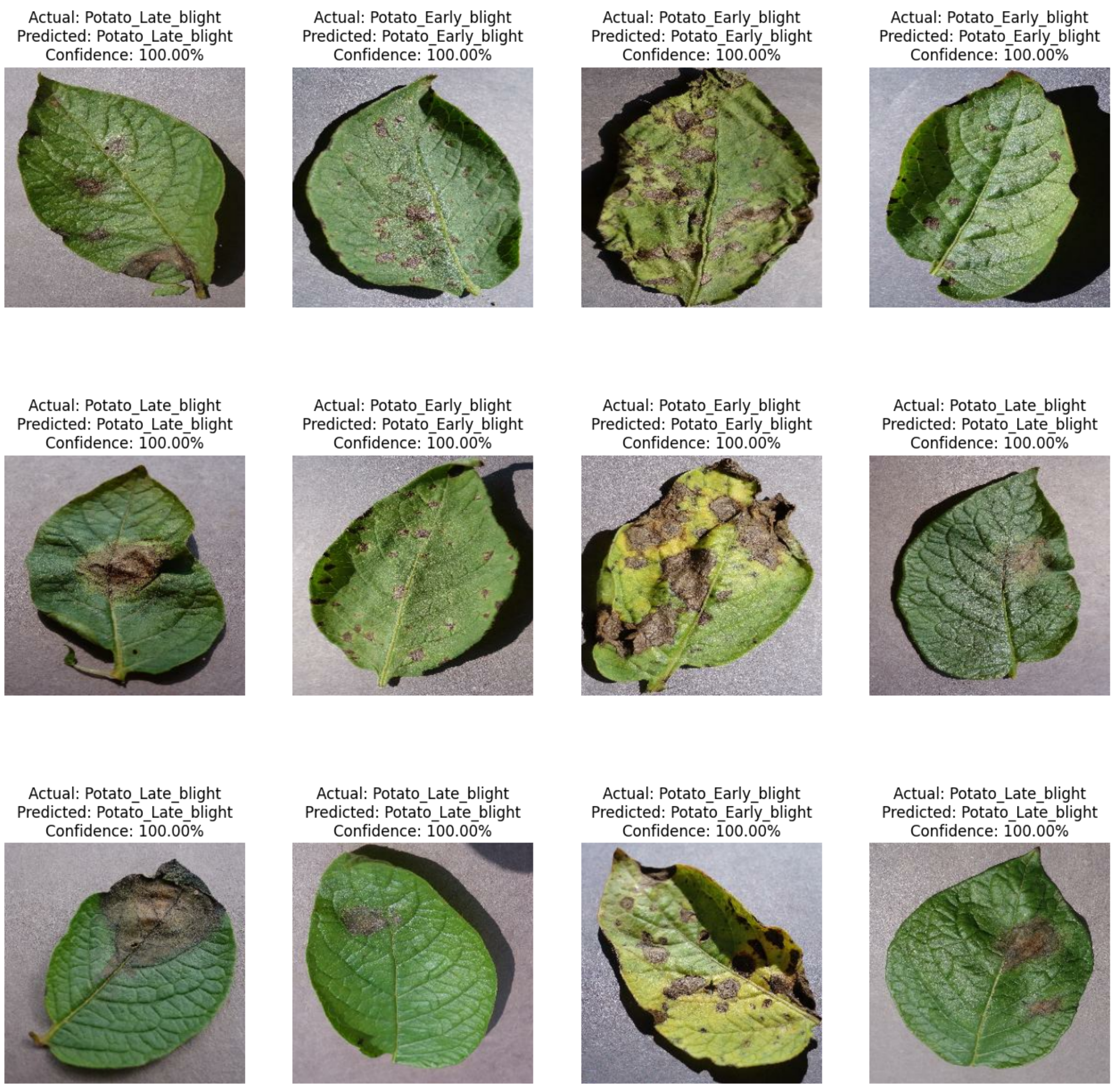


Predictions: Make predictions on the test dataset and visualize the results.

[18]:
```
# Predictions using the model
for image_batch, labels_batch in test_ds.take(1):  # Take one batch from test␣
↪dataset
    ps = model.predict(image_batch)  # Get model predictions for the batch
    images = image_batch.numpy().astype("uint8")  # Convert images to numpy␣
↪array
    labels = labels_batch.numpy()  # Convert labels to numpy array

    plt.figure(figsize=(16, 16))  # Create a figure with specified size
    for i in range(12):  # Loop through first 12 images
        ax = plt.subplot(3, 4, i + 1)  # Create a 3x4 subplot grid
        plt.imshow(images[i])  # Display the image

        # Get prediction confidence
        confidence = ps[i][ps[i].argmax()] * 100

        # Create title with actual label, predicted label and confidence
        title = f"Actual: {class_names[labels[i]]}\nPredicted:␣
↪{class_names[ps[i].argmax()]}\nConfidence: {confidence:.2f}%"
        plt.title(title)  # Show title
        plt.axis("off")  # Hide axes
```

```
1/1           0s 82ms/step

2025-01-24 23:33:05.373316: W tensorflow/core/framework/local_rendezvous.cc:404]
```

```
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```



Model Saving: Save the trained model for future use.

[19]:
```python
# Save the model
model_version = "V0.1"
model_name = "PlantDiseaseDetection"
model.save(f"./models/{model_name}_{model_version}.keras")
```