```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
import numpy as np

BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "C:\Data Set\Plant Disease",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

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

```
class_names = dataset.class_names
print(class_names)
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
dataset.take(1)
```

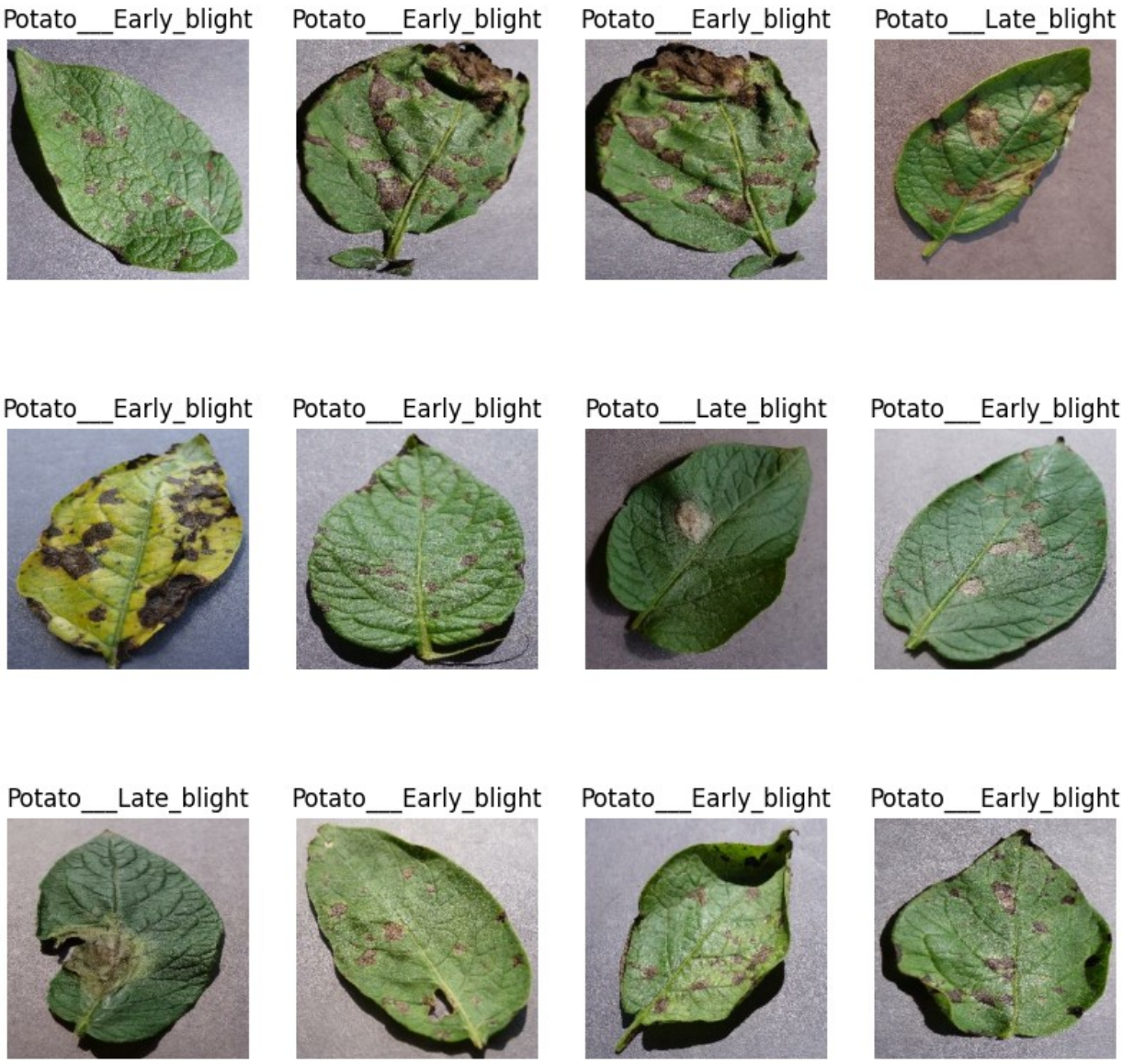
```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), 
dtype=tf.float32, name=None), TensorSpec(shape=(None,), 
dtype=tf.int32, name=None))>
```

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 2 0 0]
```

As you can see above, each element in the dataset is a tuple. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels

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




```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1,
test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

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

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
```

```
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

len(train_ds)
```

```
54
```

```
len(val_ds)
```

```
6
```

```
len(test_ds)
```

```
8
```

```
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds =
test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)


resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.Rescaling(1.0/255),
])

data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])

BATCH_SIZE
```

```
32
```

```
# train_ds = train_ds.map(
#     lambda x, y: (data_augmentation(x, training=True), y),
#     num_parallel_calls=tf.data.AUTOTUNE
# ).prefetch(buffer_size=tf.data.AUTOTUNE)



from tensorflow import keras
from tensorflow.keras import layers, models

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

```
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    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)
```

```
c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\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)
```

```
model.summary()
```

```
Model: "sequential_2"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ sequential (Sequential)              │ (32, 256, 256, 3)           │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ sequential_1 (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.compile(
    optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=1,
)
```

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 159s 3s/step - accuracy: 0.4699 - loss:
0.9468 - val_accuracy: 0.6042 - val_loss: 0.8214
```

```
scores = model.evaluate(test_ds)
```

```
1/8 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step - accuracy: 0.6562 - loss: 0.9234

8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 470ms/step - accuracy: 0.6306 - loss:
0.8488
```

```
scores
```

```
[0.8386331796646118, 0.6015625]
```

```
history
```

```
<keras.src.callbacks.history.History at 0x1a5bb9d6c20>
```

```
history.params
```

```
{'verbose': 1, 'epochs': 1, 'steps': 54}
```

```
history.history.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
type(history.history['loss'])
```

```
list
```

```
len(history.history['loss'])
```

```
1
```

```
history.history['loss'][:5] # show loss for first 5 epochs
```

```
[0.8919422030448914]
```

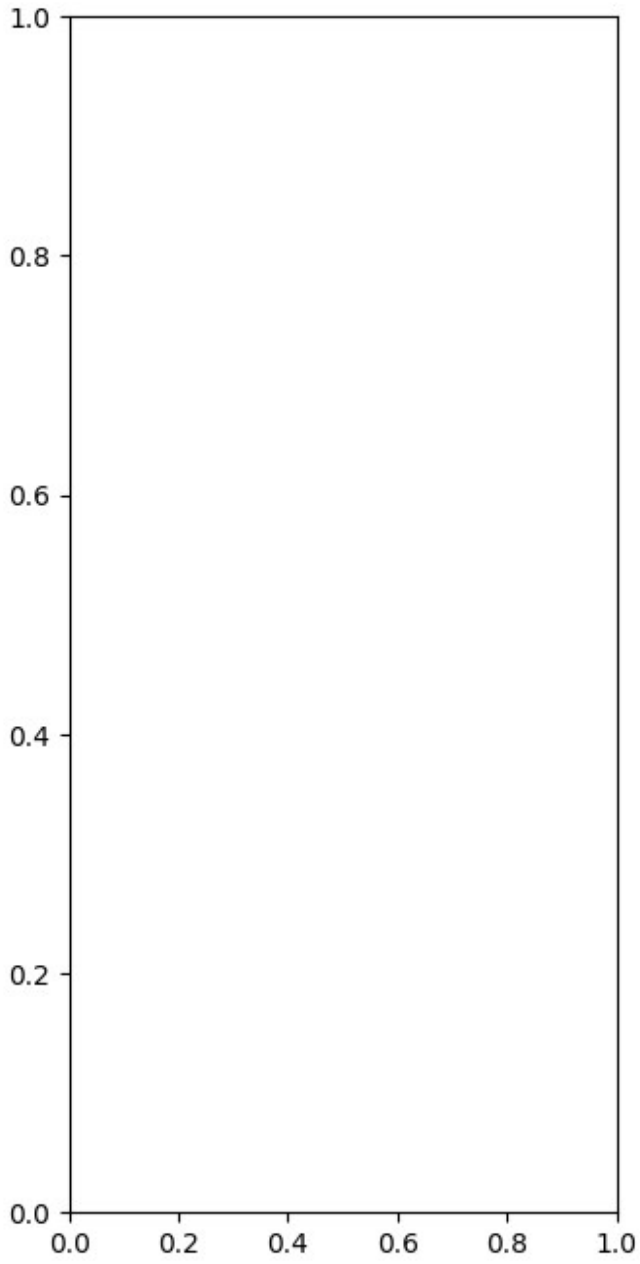
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[39], line 3
      1 plt.figure(figsize=(8, 8))
      2 plt.subplot(1, 2, 1)
----> 3 plt.plot(range(EPOCHS), acc_label='Training Accuracy')
      4 plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
      5 plt.legend(loc='lower right')

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\pyplot.py:3829, in plot(scalex, scaley, data, *args, **kwargs)
   3821 @_copy_docstring_and_deprecators(Axes.plot)
   3822 def plot(
```

```
   3823     *args: float | ArrayLike | str,
   (...)
   3827     **kwargs,
   3828 ) -> list[Line2D]:
-> 3829     return gca().plot(
   3830         *args,
   3831         scalex=scalex,
   3832         scaley=scaley,
   3833         **({"data": data} if data is not None else {}),
   3834         **kwargs,
   3835     )

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_axes.py:1777, in Axes.plot(self, scalex,
scaley, data, *args, **kwargs)
   1534 """
   1535 Plot y versus x as lines and/or markers.
   1536
   (...)
   1774 (``'green'``) or hex strings (``'#008000'``).
   1775 """
   1776 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1777 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1778 for line in lines:
   1779     self.add_line(line)

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_base.py:297, in
_process_plot_var_args.__call__(self, axes, data, return_kwargs,
*args, **kwargs)
    295     this += args[0],
    296     args = args[1:]
--> 297 yield from self._plot_args(
    298     axes, this, kwargs,
ambiguous_fmt_datakey=ambiguous_fmt_datakey,
    299     return_kwargs=return_kwargs
    300 )

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_base.py:546, in
_process_plot_var_args._plot_args(self, axes, tup, kwargs,
return_kwargs, ambiguous_fmt_datakey)
    544     return list(result)
    545 else:
--> 546     return [l[0] for l in result]

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_base.py:546, in <listcomp>(.0)
    544     return list(result)
    545 else:
```

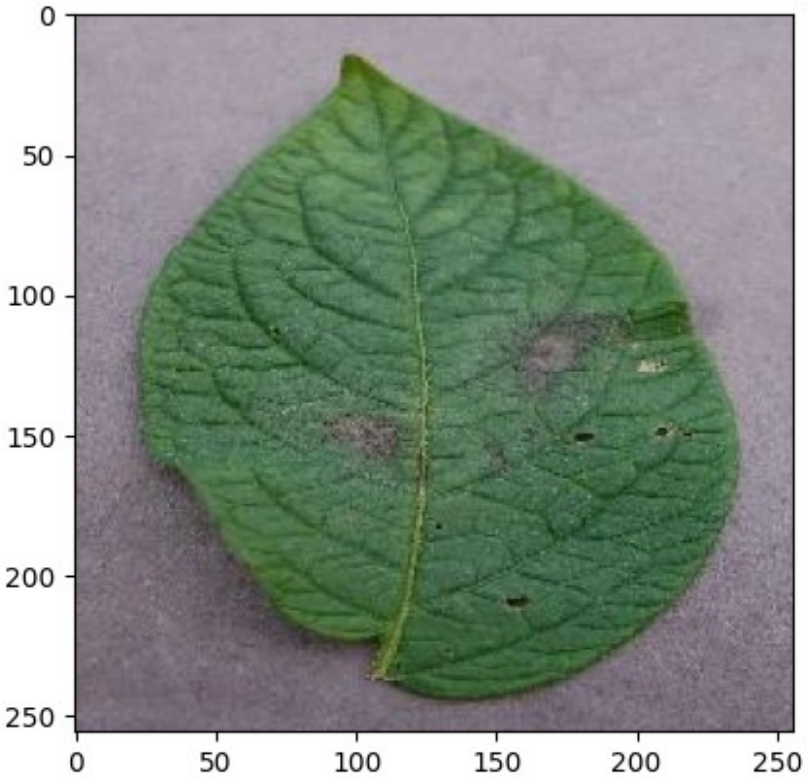
```
--> 546     return [l[0] for l in result]

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_base.py:539, in <genexpr>(.0)
    534 else:
    535     raise ValueError(
    536         f"label must be scalar or have the same length as the
input "
    537         f"data, but found {len(label)} for {n_datasets}
datasets.")
--> 539 result = (make_artist(axes, x[:, j % ncx], y[:, j % ncy], kw,
    540                         {**kwargs, 'label': label})
    541           for j, label in enumerate(labels))
    543 if return_kwargs:
    544     return list(result)

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\axes\_base.py:338, in
_process_plot_var_args._make_line(self, axes, x, y, kw, kwargs)
    336 kw = {**kw, **kwargs}  # Don't modify the original kw.
    337 self._setdefaults(self._getdefaults(kw), kw)
--> 338 seg = mlines.Line2D(x, y, **kw)
    339 return seg, kw

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\lines.py:407, in Line2D.__init__(self, xdata,
ydata, linewidth, linestyle, color, gapcolor, marker, markersize,
markeredgewidth, markeredgecolor, markerfacecolor, markerfacecoloralt,
fillstyle, antialiased, dash_capstyle, solid_capstyle, dash_joinstyle,
solid_joinstyle, pickradius, drawstyle, markevery, **kwargs)
    403 self.set_markeredgewidth(markeredgewidth)
    405 # update kwargs before updating data to give the caller a
    406 # chance to init axes (and hence unit support)
--> 407 self._internal_update(kwargs)
    408 self.pickradius = pickradius
    409 self.ind_offset = 0

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\artist.py:1233, in Artist._internal_update(self,
kwargs)
   1226 def _internal_update(self, kwargs):
   1227     """
   1228     Update artist properties without prenormalizing them, but
generating
   1229     errors as if calling `set`.
   1230
   1231     The lack of prenormalization is to maintain
backcompatibility.
   1232     """
-> 1233     return self._update_props(
```

```
   1234         kwargs, "{cls.__name__}.set() got an unexpected
keyword argument "
   1235         "{prop_name!r}")

File c:\Users\Prince\AppData\Local\Programs\Python\Python310\lib\site-
packages\matplotlib\artist.py:1206, in Artist._update_props(self,
props, errfmt)
   1204             func = getattr(self, f"set_{k}", None)
   1205             if not callable(func):
-> 1206                 raise AttributeError(
   1207                     errfmt.format(cls=type(self),
prop_name=k),
   1208                     name=k)
   1209             ret.append(func(v))
   1210 if ret:

AttributeError: Line2D.set() got an unexpected keyword argument
'acc_label'
```




```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print(" image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
```

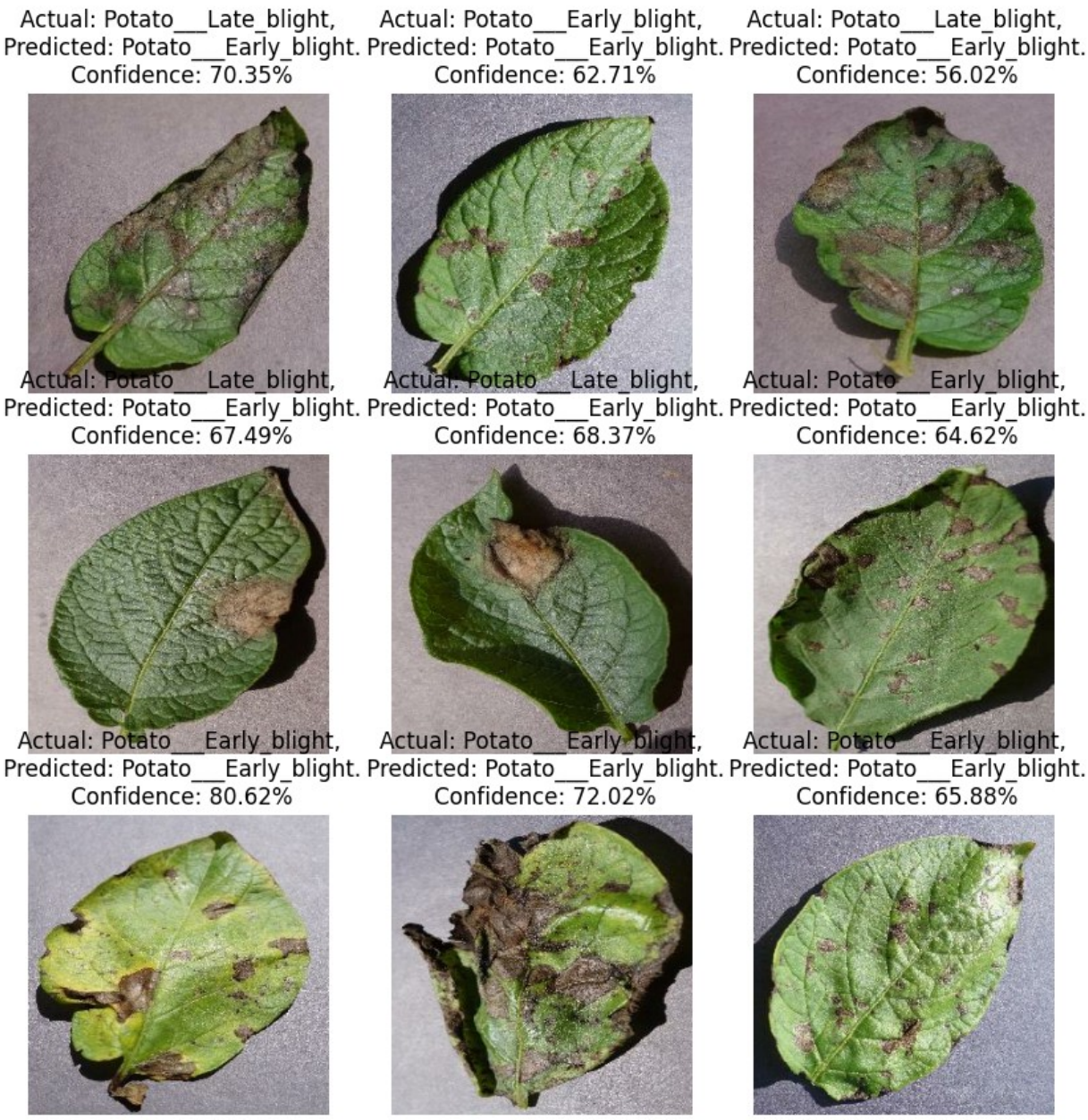
```
    print("predicted
label:",class_names[np.argmax(batch_prediction[0])])

 image to predict
actual label: Potato___Late_blight
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
predicted label: Potato___Late_blight
```




```
def predict(model, img):
    img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

plt.figure(figsize=(10,10))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
```

```
        predicted_class, confidence = predict(model,
images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted:
{predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
```




```
model.save("../potatoes.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 considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```