bell-pepper

February 26, 2024

1 Bell Pepper Disease Classification Using Neural Network

1.1 IMPORTNG LIBRARIES

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

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

1.2 CONVERTING IMAGES TO TENSORS

[2]:
```
''' For downloading dataset into tensorfolw dataset using keras '''
dataset = tf.keras.preprocessing.image_dataset_from_directory(
     r"C:\Users\raman\Desktop\mini-projects\pepper_bell class\data",
     shuffle = True,
     image_size = (256,256),
     batch_size = 32)
```

```
Found 2475 files belonging to 2 classes.
```

[3]:
```
class_names = dataset.class_names
class_names    # for printing class names
```

[3]:
```
['Pepper__bell___Bacterial_spot', 'Pepper__bell___healthy']
```

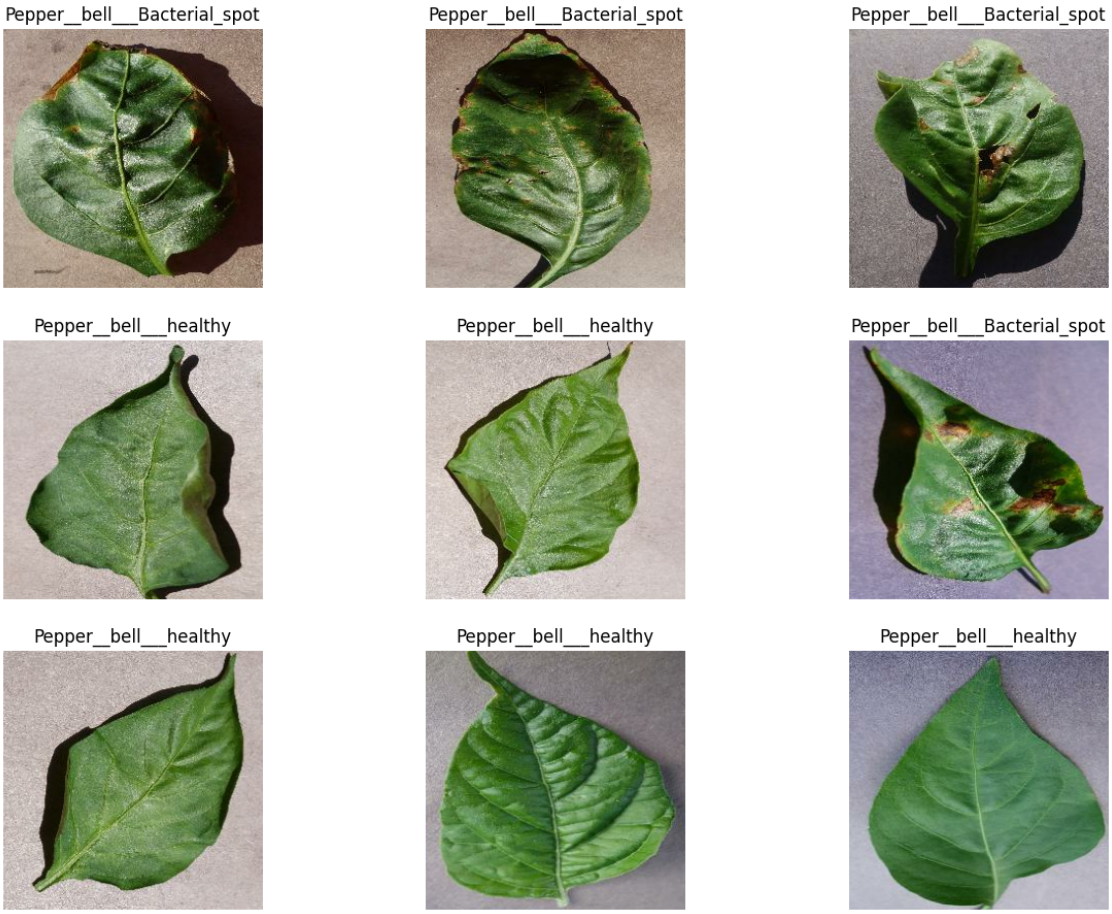
[4]:
```
len(dataset)  # so 78 batches of 32 in one group
```

[4]:
```
78
```

[5]:
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())  # here 0,1 -> are class labels
```

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

[6]:
```
plt.figure(figsize=(15,15))
for image_batch,label_batch in dataset.take(5):
    for i in range(9):
        plt.subplot(4,3,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```



2 DATA SPLITTING

[7]:
```
def train_test_split(ds,train_split=0.8,val_split=0.1,test_split=0.
↪1,shuffle=True,shuffle_size=10000):
    '''
    For Splitting data into training, validation and testing data
    '''
```

```
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size,seed=15)
    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
```

[8]:
```
train_ds , val_ds , test_ds = train_test_split(dataset)
```

[9]:
```
len(train_ds) , len(val_ds) , len(test_ds)
```

[9]: (62, 7, 9)

[10]:
```
## distributing workload on CPU and GPU
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)
```

2.1 PREPROCESSING

[11]:
```
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(256,256),
    layers.experimental.preprocessing.Rescaling(1.0/255)
]) # Standardizing size and scale of pixels
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.
```

DATA AUGMENTATION

[12]:
```
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.3)
]) # Generating different views of same leaf so that if anyone capture image at␣
 ↪different angle or side it should be able to predict
```

2.2 MODELLING USING CONVOLUTIONAL NEURAL NETWORK

[13]:
```
input_shape = (32,256,256,3) # (Batches , X, Y, Channel)  -->> Channel = 3 ->␣
 ↪RGB
n_classes = 10
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32,(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(128,kernel_size = (5,5),activation = "relu"),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128,kernel_size = (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)
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool
is deprecated. Please use tf.nn.max_pool2d instead.
```

[14]:
```
model.summary()  # summary of neural network architecture
```

```
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 (MaxPooling2  (32, 127, 127, 32)        0
 D)

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)
```

```
 conv2d_2 (Conv2D)           (32, 58, 58, 128)         204928

 max_pooling2d_2 (MaxPoolin  (32, 29, 29, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (32, 27, 27, 128)         147584

 max_pooling2d_3 (MaxPoolin  (32, 13, 13, 128)         0
 g2D)

 flatten (Flatten)           (32, 21632)               0

 dense (Dense)               (32, 64)                  1384512

 dense_1 (Dense)             (32, 10)                  650

=================================================================
Total params: 1757066 (6.70 MB)
Trainable params: 1757066 (6.70 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[15]:
```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']
)
## keras.losses.SparseCategoricalCrossentropy
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

[16]:
```
history = model.fit(
    train_ds,
    epochs = 30,
    batch_size = 32,
    verbose = 1,
    validation_data = val_ds
)
```

```
Epoch 1/30
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-
```

```
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

62/62 [==============================] - 158s 2s/step - loss: 0.7072 - accuracy:
0.6184 - val_loss: 1.8485 - val_accuracy: 0.4420
Epoch 2/30
62/62 [==============================] - 96s 2s/step - loss: 0.4909 - accuracy:
0.7881 - val_loss: 0.2865 - val_accuracy: 0.9018
Epoch 3/30
62/62 [==============================] - 118s 2s/step - loss: 0.2961 - accuracy:
0.8986 - val_loss: 0.2882 - val_accuracy: 0.8705
Epoch 4/30
62/62 [==============================] - 125s 2s/step - loss: 0.2434 - accuracy:
0.9215 - val_loss: 0.1925 - val_accuracy: 0.9375
Epoch 5/30
62/62 [==============================] - 104s 2s/step - loss: 0.1487 - accuracy:
0.9592 - val_loss: 0.0963 - val_accuracy: 0.9643
Epoch 6/30
62/62 [==============================] - 91s 1s/step - loss: 0.1065 - accuracy:
0.9791 - val_loss: 0.2628 - val_accuracy: 0.8929
Epoch 7/30
62/62 [==============================] - 110s 2s/step - loss: 0.1739 - accuracy:
0.9531 - val_loss: 0.1014 - val_accuracy: 0.9598
Epoch 8/30
62/62 [==============================] - 85s 1s/step - loss: 0.0862 - accuracy:
0.9822 - val_loss: 0.0681 - val_accuracy: 0.9732
Epoch 9/30
62/62 [==============================] - 82s 1s/step - loss: 0.0580 - accuracy:
0.9852 - val_loss: 0.0343 - val_accuracy: 0.9955
Epoch 10/30
62/62 [==============================] - 81s 1s/step - loss: 0.0532 - accuracy:
0.9878 - val_loss: 0.0802 - val_accuracy: 0.9732
Epoch 11/30
62/62 [==============================] - 78s 1s/step - loss: 0.0560 - accuracy:
0.9842 - val_loss: 0.0762 - val_accuracy: 0.9866
Epoch 12/30
62/62 [==============================] - 74s 1s/step - loss: 0.0447 - accuracy:
0.9918 - val_loss: 0.0240 - val_accuracy: 0.9911
Epoch 13/30
62/62 [==============================] - 83s 1s/step - loss: 0.0306 - accuracy:
0.9924 - val_loss: 0.0227 - val_accuracy: 0.9866
Epoch 14/30
62/62 [==============================] - 74s 1s/step - loss: 0.0556 - accuracy:
0.9929 - val_loss: 0.0793 - val_accuracy: 0.9732
Epoch 15/30
62/62 [==============================] - 84s 1s/step - loss: 0.0476 - accuracy:
0.9924 - val_loss: 0.0599 - val_accuracy: 0.9732
```

```
Epoch 16/30
62/62 [==============================] - 92s 1s/step - loss: 0.0283 - accuracy:
0.9944 - val_loss: 0.0950 - val_accuracy: 0.9821
Epoch 17/30
62/62 [==============================] - 86s 1s/step - loss: 0.0253 - accuracy:
0.9939 - val_loss: 0.1552 - val_accuracy: 0.9554
Epoch 18/30
62/62 [==============================] - 84s 1s/step - loss: 0.0376 - accuracy:
0.9873 - val_loss: 0.0103 - val_accuracy: 0.9955
Epoch 19/30
62/62 [==============================] - 83s 1s/step - loss: 0.0189 - accuracy:
0.9959 - val_loss: 0.0525 - val_accuracy: 0.9821
Epoch 20/30
62/62 [==============================] - 83s 1s/step - loss: 0.0332 - accuracy:
0.9908 - val_loss: 0.0095 - val_accuracy: 0.9955
Epoch 21/30
62/62 [==============================] - 83s 1s/step - loss: 0.0344 - accuracy:
0.9929 - val_loss: 0.0321 - val_accuracy: 0.9911
Epoch 22/30
62/62 [==============================] - 83s 1s/step - loss: 0.0202 - accuracy:
0.9944 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 23/30
62/62 [==============================] - 83s 1s/step - loss: 0.0175 - accuracy:
0.9944 - val_loss: 0.0093 - val_accuracy: 0.9955
Epoch 24/30
62/62 [==============================] - 83s 1s/step - loss: 0.0185 - accuracy:
0.9944 - val_loss: 0.0044 - val_accuracy: 1.0000
Epoch 25/30
62/62 [==============================] - 83s 1s/step - loss: 0.0088 - accuracy:
0.9975 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 26/30
62/62 [==============================] - 83s 1s/step - loss: 0.0076 - accuracy:
0.9985 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 27/30
62/62 [==============================] - 83s 1s/step - loss: 0.0084 - accuracy:
0.9969 - val_loss: 0.0100 - val_accuracy: 0.9955
Epoch 28/30
62/62 [==============================] - 83s 1s/step - loss: 0.0315 - accuracy:
0.9913 - val_loss: 0.0112 - val_accuracy: 1.0000
Epoch 29/30
62/62 [==============================] - 89s 1s/step - loss: 0.0099 - accuracy:
0.9964 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 30/30
62/62 [==============================] - 90s 1s/step - loss: 0.0107 - accuracy:
0.9975 - val_loss: 0.0572 - val_accuracy: 0.9821
```

[17]:
```
scores = model.evaluate(test_ds)
```

```
9/9 [==============================] - 9s 615ms/step - loss: 0.0505 - accuracy:
0.9861
```

[18]:
```
scores
```

[18]:
```
[0.050534799695014954, 0.9861111044883728]
```

[19]:
```
history
```

[19]:
```
<keras.src.callbacks.History at 0x13cfd5b9b10>
```

[20]:
```
history.params
```

[20]:
```
{'verbose': 1, 'epochs': 30, 'steps': 62}
```

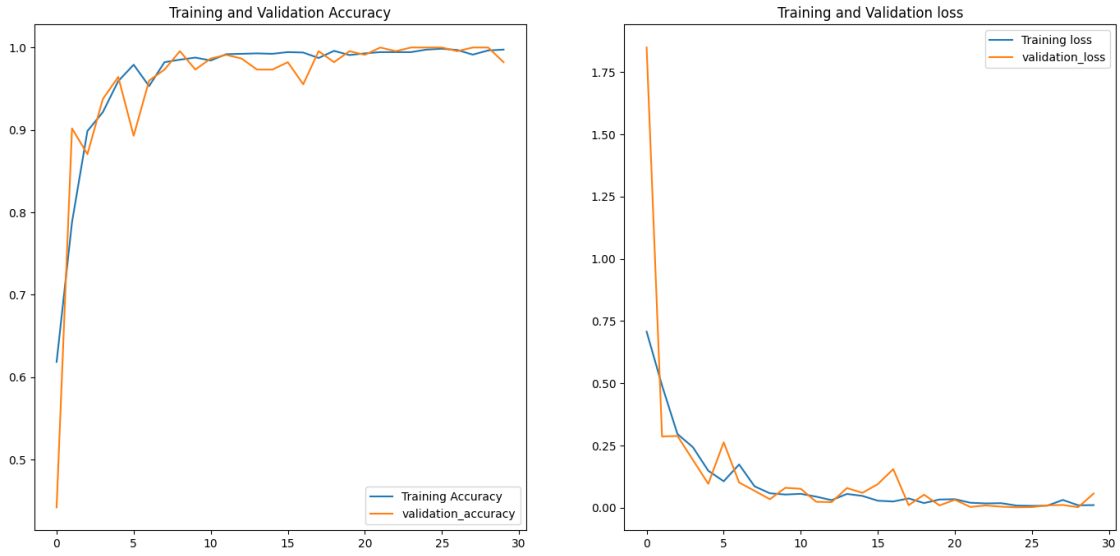
[21]:
```
history.history.keys()
```

[21]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[22]:
```
acc = history.history["accuracy"]
val_acc= history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]
```

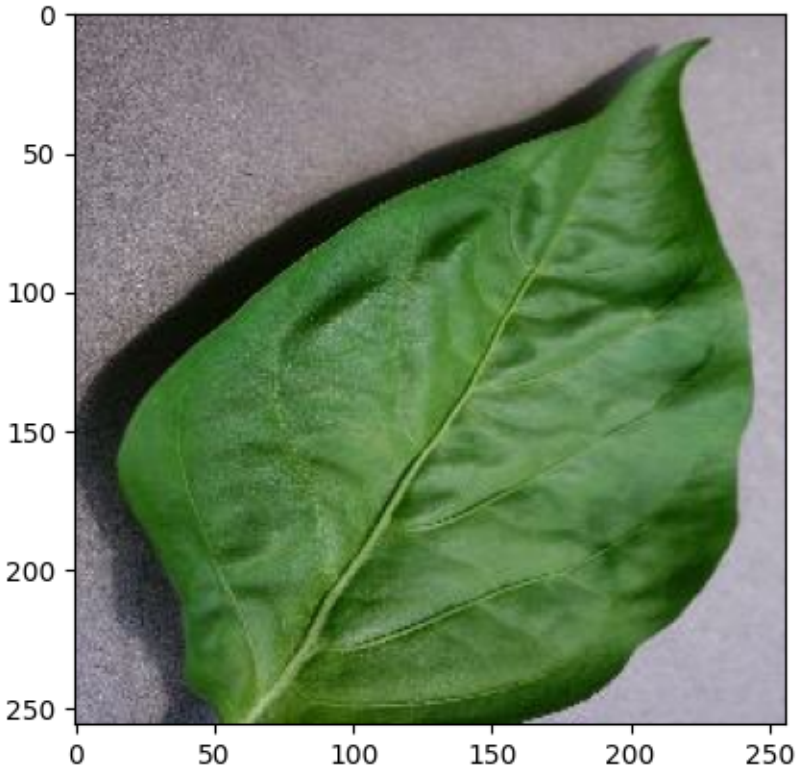
[24]:
```
plt.figure(figsize=(17,8))
plt.subplot(1,2,1)
plt.plot(range(30),acc,label="Training Accuracy")
plt.plot(range(30),val_acc,label = "validation_accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Accuracy")

plt.subplot(1,2,2)
plt.plot(range(30),loss,label="Training loss")
plt.plot(range(30),val_loss,label = "validation_loss")
plt.legend(loc="upper right")
plt.title("Training and Validation loss")
```

[24]:
```
Text(0.5, 1.0, 'Training and Validation loss')
```



[25]:
```
for image_batch,label_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype("uint8")
    first_label = label_batch[0].numpy()

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

    batch_prediction = model.predict(image_batch)
    print(class_names[numpy.argmax(batch_prediction[0])])
```

```
first image to predict
actual label :  Pepper__bell___healthy
1/1 [==============================] - 1s 835ms/step
Pepper__bell___healthy
```

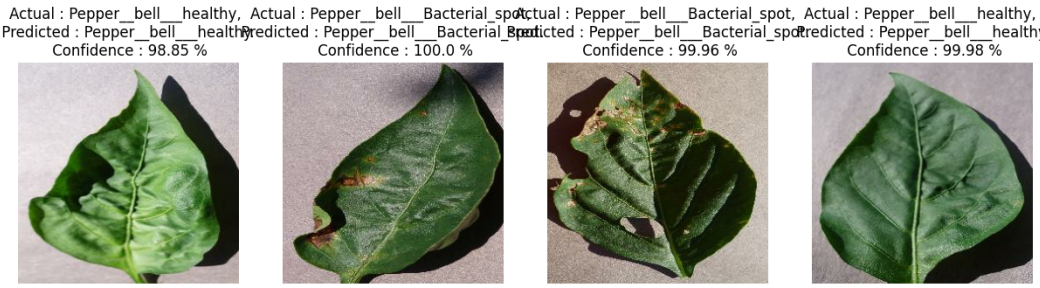


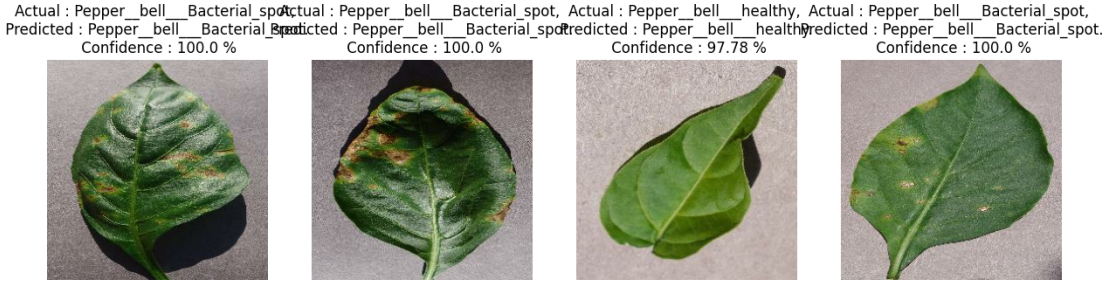
[26]:
```
## Defining functions for predicting test dataset results

def predict(model,img):
    img_arry = tf.keras.preprocessing.image.img_to_array((img))
    img_array = tf.expand_dims(img_arry,0)  ## creating a batch
    # prediction
    prediction = model.predict(img_array)

    pred_class = class_names[numpy.argmax(prediction[0])]
    confidence = round(100*(numpy.max(prediction[0])),2)
    return pred_class , confidence
```

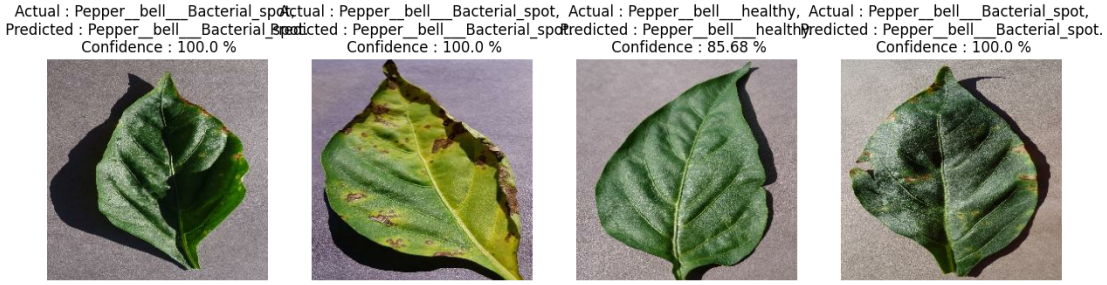
[27]:
```
plt.figure(figsize = (15,15))
for images,label in test_ds.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))

        pred_class , confidence = predict(model,images[i])
        actual_class = class_names[label[i]]
```

```
        plt.title(f"Actual : {actual_class}, \n Predicted : {pred_class}. \n␣
↪Confidence : {confidence} %")
        plt.axis('off')
```

```
1/1 [==============================] - 0s 218ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 46ms/step
```







[ ]: