In [1]:
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
import os
```

In [2]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

In [3]:
```
train_data_dir = r'C:\Gasket Dataset 1'

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    train_data_dir,
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 141 files belonging to 6 classes.
```

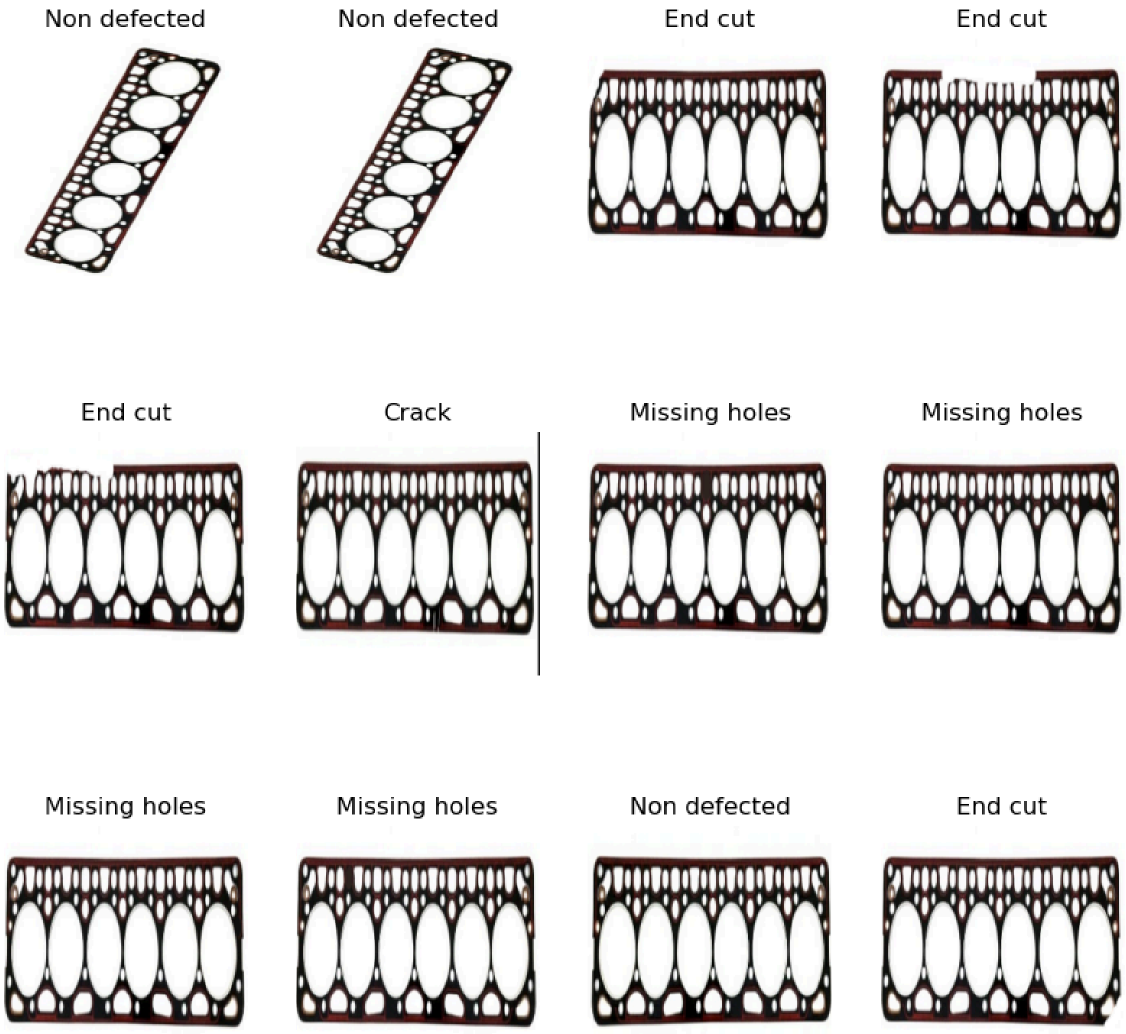
In [4]:
```
class_names = dataset.class_names
class_names
```

Out[4]:
```
['Burnt', 'Crack', 'End cut', 'Eyelet', 'Missing holes', 'Non defected']
```

In [5]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[4 1 4 0 2 4 2 4 4 4 5 0 2 3 4 4 5 0 0 4 4 4 1 4 5 5 2 4 1 5 2 4]
```

In [6]:
```
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")
```



In [7]:
```
len(dataset)
```

Out[7]: 5

In [8]:
```
train_size = 0.7
len(dataset)*train_size
```

Out[8]: 3.5

In [9]:
```
train_ds = dataset.take(3)
len(train_ds)
```

Out[9]: 3

In [10]:
```
test_ds = dataset.skip(3)
len(test_ds)
```

Out[10]: 2

In [11]:
```
val_size=0.2
len(dataset)*val_size
```

Out[11]: 1.0

In [12]:
```
val_ds = test_ds.take(1)
len(val_ds)
```

Out[12]: 1

In [13]:
```
test_ds = test_ds.skip(1)
len(test_ds)
```

Out[13]: 1

In [14]:
```
def get_dataset_partitions_tf(ds, train_split=0.7, val_split=0.2, test_split
    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
```

In [15]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [16]:
```
len(train_ds)
```

Out[16]: 3

In [17]:
```
len(val_ds)
```

Out[17]: 1

In [18]:
```
len(test_ds)
```

Out[18]: 1

In [19]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOT
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.AUTOTUN
```

In [20]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [21]:
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

In [22]:
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [23]:
```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 10

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=i
    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)
```

In [24]:
```
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 (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, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0         
 g2D)                                                            
                                                                 
 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928     
                                                                 
 max_pooling2d_5 (MaxPoolin  (32, 2, 2, 64)            0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (32, 256)                 0         
                                                                 
 dense (Dense)               (32, 64)                  16448     
                                                                 
 dense_1 (Dense)             (32, 10)                  650       
                                                                 
=================================================================
Total params: 184202 (719.54 KB)
Trainable params: 184202 (719.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [25]:
```
unique_labels = set()
for _, labels_batch in dataset:
    unique_labels.update(labels_batch.numpy())

print("Unique Label Values:", unique_labels)
```

```
Unique Label Values: {0, 1, 2, 3, 4, 5}
```

In [26]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [27]:
```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # or any {'0', '1', '2'}

# Your code here
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

```
3/3 [==============================] - 18s 7s/step - loss: 1.2145 - accu
racy: 0.5974 - val_loss: 1.2889 - val_accuracy: 0.5385
Epoch 45/50
3/3 [==============================] - 18s 5s/step - loss: 1.2087 - accu
racy: 0.5974 - val_loss: 1.1076 - val_accuracy: 0.5385
Epoch 46/50
3/3 [==============================] - 21s 6s/step - loss: 1.1647 - accu
racy: 0.5974 - val_loss: 1.1789 - val_accuracy: 0.5385
Epoch 47/50
3/3 [==============================] - 23s 7s/step - loss: 1.1558 - accu
racy: 0.5974 - val_loss: 1.3335 - val_accuracy: 0.5385
Epoch 48/50
3/3 [==============================] - 19s 5s/step - loss: 1.2387 - accu
racy: 0.5974 - val_loss: 1.2635 - val_accuracy: 0.5385
Epoch 49/50
3/3 [==============================] - 19s 6s/step - loss: 1.2293 - accu
racy: 0.5844 - val_loss: 1.1777 - val_accuracy: 0.5385
Epoch 50/50
3/3 [==============================] - 19s 8s/step - loss: 1.2092 - accu
racy: 0.5974 - val_loss: 1.2310 - val_accuracy: 0.5385
```

In [28]:
```
scores = model.evaluate(test_ds)
```

```
1/1 [==============================] - 12s 12s/step - loss: 1.1306 - accur
acy: 0.6923
```

In [29]:
```
# Assuming `model` is your Keras model and `test_ds` is your test dataset
model.summary()  # Check the model summary to verify the output shape

# Print out shapes for debugging
for batch in test_ds:
    features, labels = batch  # Assuming your test dataset yields features a
    print("Model Output Shape:", model.predict(features).shape)
    print("Labels Shape:", labels.shape)
    break  # Break after printing the first batch for inspection
```

...

In [30]:
```
scores
```
Out[30]: [1.1306419372558594, 0.692307710647583]

In [31]:
```
#Plotting the Accuracy and Loss Curves
history
```
Out[31]: <keras.src.callbacks.History at 0x1e4eec7b990>

In [32]:
```
history.params
```
Out[32]: {'verbose': 1, 'epochs': 50, 'steps': 3}

In [33]:
```
history.history.keys()
```
Out[33]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

In [34]:
```
type(history.history['loss'])
```
Out[34]: list

In [35]:
```
len(history.history['loss'])
```
Out[35]: 50

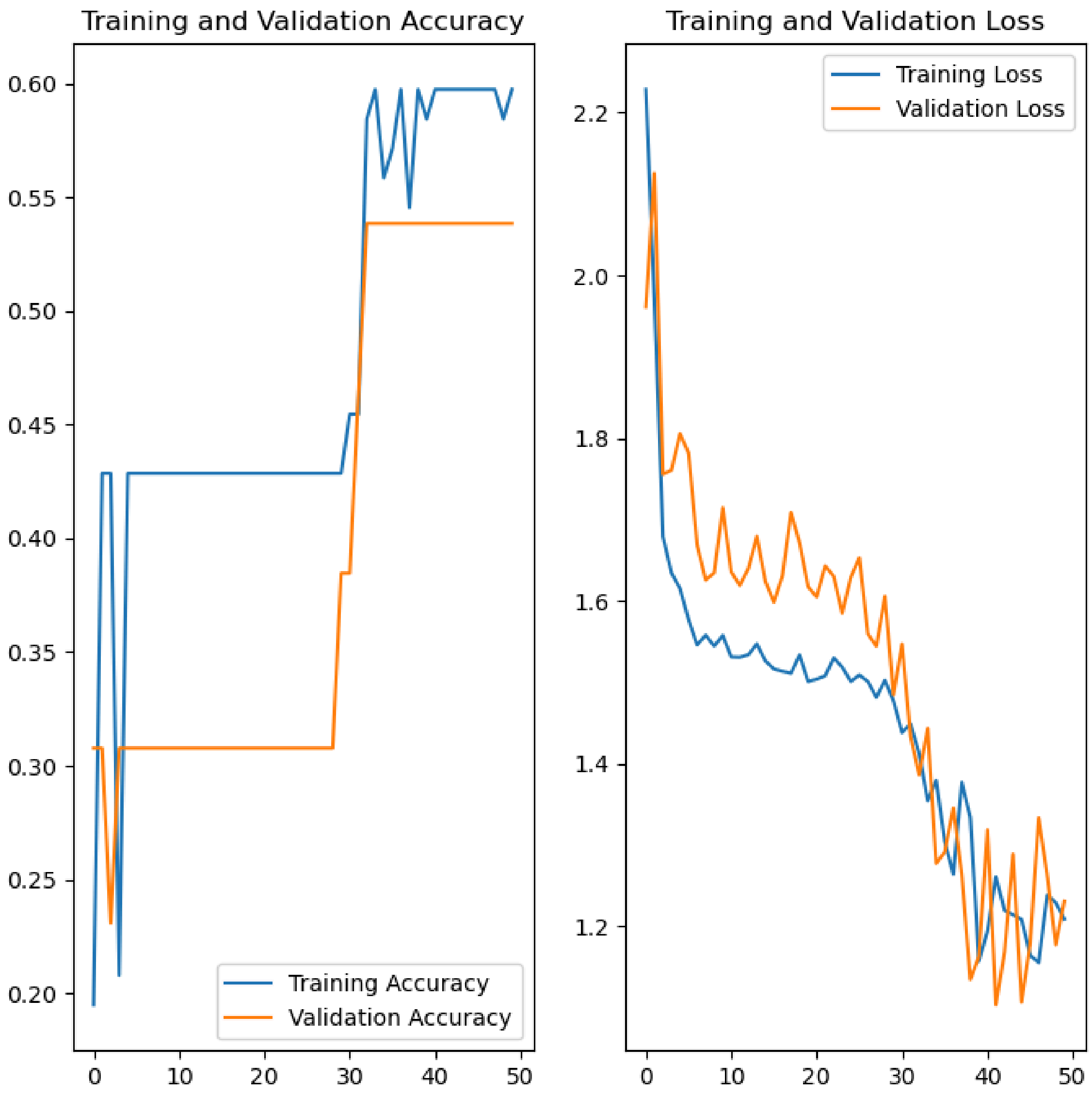
In [36]:
```
history.history['loss'][:5] # show loss for first 5 epochs
```
Out[36]:
```
[2.2282907962799072,
 1.9707003831863403,
 1.6791375875473022,
 1.6343554258346558,
 1.6151763200759888]
```

In [37]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

In [38]:
```
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()
```



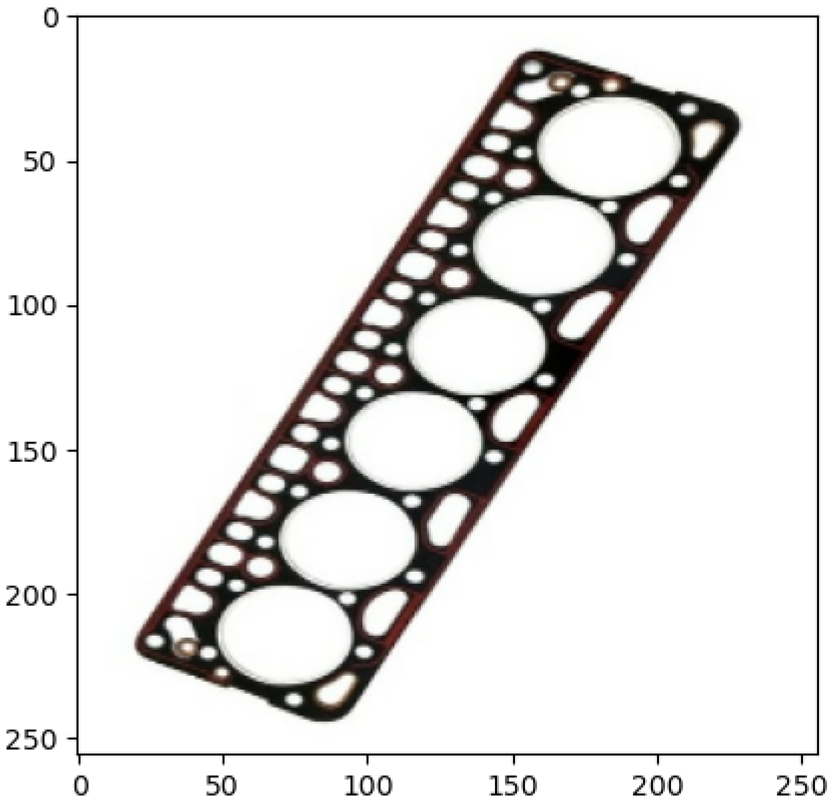
In [39]:
```
#Run prediction on a sample image
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

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

    print("first 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])])
```

```
first image to predict
actual label: Non defected
1/1 [==============================] - 1s 687ms/step
predicted label: Non defected
```



In [40]:
```
#Write a function for inference
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
```

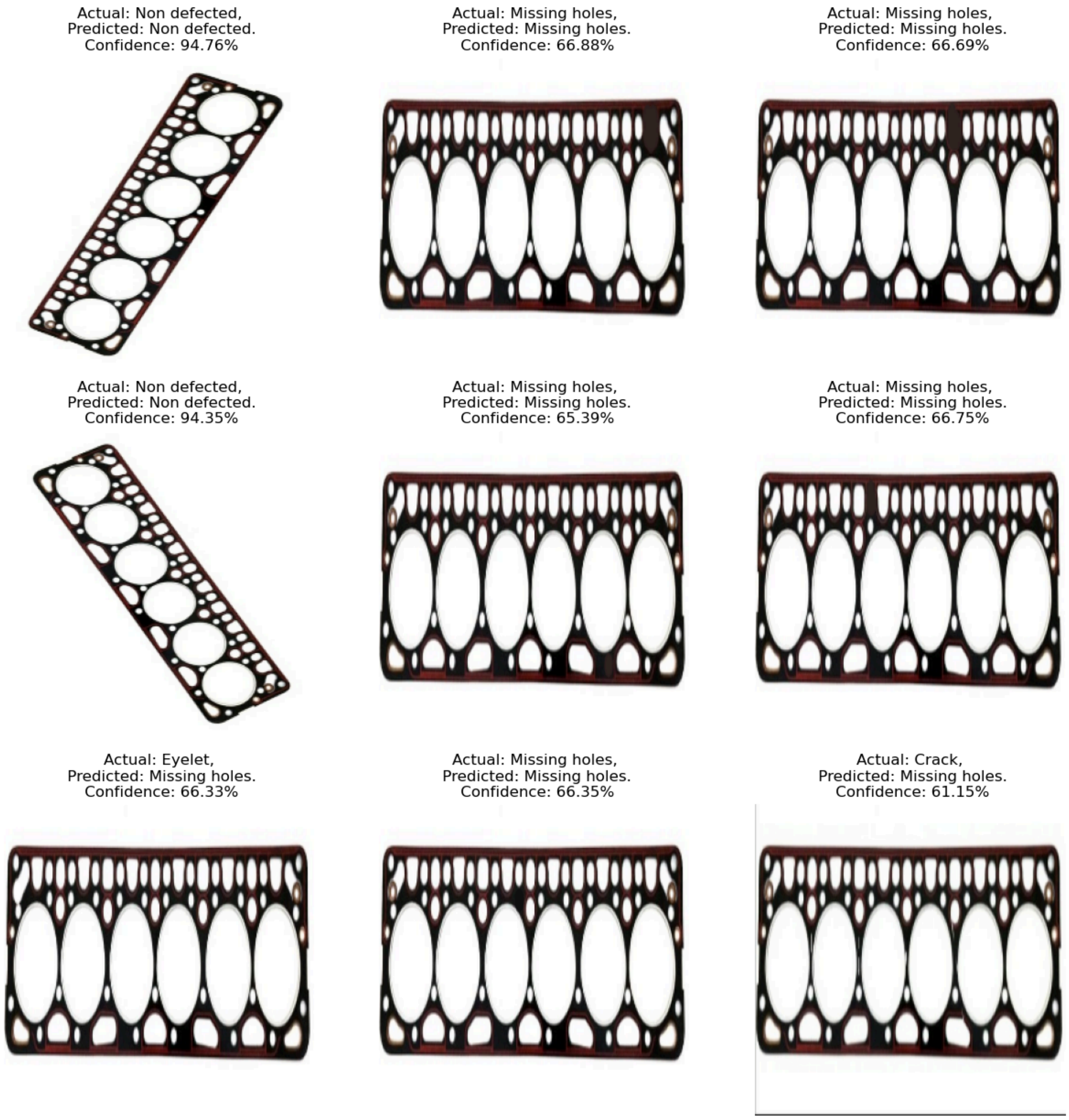
In [41]:
```
#Now run inference on few sample images
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

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

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

        plt.axis("off")
```

```
1/1 [==============================] - 0s 332ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 469ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step
```



In [42]:
```
#Saving the Model
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Load pre-trained MobileNetV2
base_model = MobileNetV2(weights='imagenet', include_top=False)

# Add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Add a fully-connected layer
x = Dense(1024, activation='relu')(x)

# Add a logistic layer with 10 classes (one for each type of defect)
predictions = Dense(10, activation='softmax')(x)

# Define the model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the base model (so they don't get trained)
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a

# Train the model
model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is
not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will b
e loaded as the default.
Epoch 1/50
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[42], line 31
     28 model.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
     30 # Train the model
---> 31 model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds)

File D:\New folder\Lib\site-packages\keras\src\utils\traceback_utils.py:7
0, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\AppData\Local\Temp\__autograph_generated_filedj8hhq3d.py:15, in out
er_factory.<locals>.inner_factory.<locals>.tf__train_function(iterator)
     13 try:
     14     do_return = True
---> 15     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld
(self), ag__.ld(iterator)), None, fscope)
     16 except:
     17     do_return = False

ValueError: in user code:

    File "D:\New folder\Lib\site-packages\keras\src\engine\training.py", l
ine 1377, in train_function  *
        return step_function(self, iterator)
    File "D:\New folder\Lib\site-packages\keras\src\engine\training.py", l
ine 1360, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "D:\New folder\Lib\site-packages\keras\src\engine\training.py", l
ine 1349, in run_step  **
        outputs = model.train_step(data)
    File "D:\New folder\Lib\site-packages\keras\src\engine\training.py", l
ine 1127, in train_step
        loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "D:\New folder\Lib\site-packages\keras\src\engine\training.py", l
ine 1185, in compute_loss
        return self.compiled_loss(
    File "D:\New folder\Lib\site-packages\keras\src\engine\compile_utils.p
y", line 277, in __call__
        loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "D:\New folder\Lib\site-packages\keras\src\losses.py", line 143,
in __call__
        losses = call_fn(y_true, y_pred)
    File "D:\New folder\Lib\site-packages\keras\src\losses.py", line 270,
in call  **
        return ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "D:\New folder\Lib\site-packages\keras\src\losses.py", line 2221,
in categorical_crossentropy
        return backend.categorical_crossentropy(
    File "D:\New folder\Lib\site-packages\keras\src\backend.py", line 557
5, in categorical_crossentropy
        target.shape.assert_is_compatible_with(output.shape)
```

```
ValueError: Shapes (None, 1) and (None, 10) are incompatible
```

In [ ]:

```
```

In [ ]:

```
```