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

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

In [3]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "train",
    shuffle= True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

```
Found 2900 files belonging to 4 classes.
```

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

Out[4]:
```
['Closed', 'Open', 'no_yawn', 'yawn']
```

In [5]:
```
len(dataset)
```

Out[5]:
```
91
```

In [6]:
```
for image_batch, label_batch in dataset.take(1):
    # print(image_batch.shape)
    # print(label_batch.numpy())
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")
```

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

Out[7]: 91

In [8]:
```
train_size = 0.8
train_ds = int(len(dataset)*train_size)
train_ds = dataset.take(train_ds)
len(train_ds)
```

Out[8]: 72

In [9]:
```
test_ds = dataset.skip(len(train_ds))
len(test_ds)
```

Out[9]: 19

In [10]:
```
val_size = 0.1
val_ds = int(len(dataset)*val_size)
val_ds = dataset.take(val_ds)
len(val_ds)
```

Out[10]: 9

In [11]:
```
test_ds = test_ds.skip(len(val_ds))
len(test_ds)
```

Out[11]: 10

In [12]:
```
def get_dataset_partitions_tf(ds,train_split = 0.8, val_split=0.1, shuffle=True, sh

    if shuffle :
        ds = ds.shuffle(shuffle_size, seed=12)
    ds_size = len(ds)
    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 [13]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [14]:
```
len(train_ds), len(val_ds), len(test_ds)
```

Out[14]: (72, 9, 10)

In [15]:
```
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)
```

In [16]:
```
resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.Rescaling(1.0/255)
])
```

In [17]:
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2)
])
```

In [18]:
```
input_shape = (BATCH_SIZE,IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 4

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, (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')
])
```

```
C:\Users\techno\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: 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__(
```

In [19]:
```
model.build(input_shape=input_shape)
```

In [20]:
```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	
sequential (Sequential)	(32, 256, 256, 3)	
sequential_1 (Sequential)	(32, 256, 256, 3)	
conv2d (Conv2D)	(32, 254, 254, 32)	
max_pooling2d (MaxPooling2D)	(32, 127, 127, 32)	
conv2d_1 (Conv2D)	(32, 125, 125, 64)	
max_pooling2d_1 (MaxPooling2D)	(32, 62, 62, 64)	
conv2d_2 (Conv2D)	(32, 60, 60, 64)	
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	
flatten (Flatten)	(32, 57600)	
dense (Dense)	(32, 64)	
dense_1 (Dense)	(32, 4)	

Total params: 3,743,044 (14.28 MB)

Trainable params: 3,743,044 (14.28 MB)

Non-trainable params: 0 (0.00 B)

In [21]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy']
             )
```

In [22]:
```
history = model.fit(
    train_ds,
    epochs= EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=val_ds
)
```

```
Epoch 1/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 488s 5s/step - accuracy: 0.5749 - loss: 0.9277 - val_accu
racy: 0.7257 - val_loss: 0.5205
Epoch 2/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 262s 4s/step - accuracy: 0.7517 - loss: 0.4705 - val_accu
racy: 0.7361 - val_loss: 0.4648
Epoch 3/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 204s 3s/step - accuracy: 0.7653 - loss: 0.4496 - val_accu
racy: 0.7778 - val_loss: 0.4475
Epoch 4/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 226s 3s/step - accuracy: 0.7620 - loss: 0.4352 - val_accu
racy: 0.8021 - val_loss: 0.3979
Epoch 5/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 206s 3s/step - accuracy: 0.7720 - loss: 0.4044 - val_accu
racy: 0.8368 - val_loss: 0.3797
Epoch 6/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 210s 3s/step - accuracy: 0.7946 - loss: 0.3950 - val_accu
racy: 0.7951 - val_loss: 0.4126
Epoch 7/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 198s 3s/step - accuracy: 0.7855 - loss: 0.4293 - val_accu
racy: 0.8160 - val_loss: 0.3664
Epoch 8/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 248s 3s/step - accuracy: 0.8218 - loss: 0.3616 - val_accu
racy: 0.7569 - val_loss: 0.3898
Epoch 9/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 275s 4s/step - accuracy: 0.7994 - loss: 0.3701 - val_accu
racy: 0.8160 - val_loss: 0.3438
Epoch 10/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 339s 5s/step - accuracy: 0.8131 - loss: 0.3610 - val_accu
racy: 0.8576 - val_loss: 0.3118
Epoch 11/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 229s 3s/step - accuracy: 0.8220 - loss: 0.3607 - val_accu
racy: 0.8438 - val_loss: 0.3230
Epoch 12/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 177s 2s/step - accuracy: 0.8147 - loss: 0.3598 - val_accu
racy: 0.8264 - val_loss: 0.3282
Epoch 13/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 179s 2s/step - accuracy: 0.8400 - loss: 0.3392 - val_accu
racy: 0.8507 - val_loss: 0.2953
Epoch 14/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 206s 3s/step - accuracy: 0.8458 - loss: 0.3157 - val_accu
racy: 0.8611 - val_loss: 0.3392
Epoch 15/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 179s 2s/step - accuracy: 0.8509 - loss: 0.3133 - val_accu
racy: 0.8750 - val_loss: 0.2754
Epoch 16/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 167s 2s/step - accuracy: 0.8329 - loss: 0.3342 - val_accu
racy: 0.8507 - val_loss: 0.3027
Epoch 17/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 205s 3s/step - accuracy: 0.8842 - loss: 0.2706 - val_accu
racy: 0.9097 - val_loss: 0.2561
Epoch 18/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 173s 2s/step - accuracy: 0.8614 - loss: 0.3042 - val_accu
racy: 0.8889 - val_loss: 0.2551
Epoch 19/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 203s 3s/step - accuracy: 0.8962 - loss: 0.2506 - val_accu
```

```
racy: 0.8264 - val_loss: 0.3437
Epoch 20/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 169s 2s/step - accuracy: 0.8686 - loss: 0.2946 - val_accu
racy: 0.8819 - val_loss: 0.2458
Epoch 21/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 259s 4s/step - accuracy: 0.8874 - loss: 0.2582 - val_accu
racy: 0.8125 - val_loss: 0.4129
Epoch 22/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 185s 3s/step - accuracy: 0.8639 - loss: 0.2878 - val_accu
racy: 0.8299 - val_loss: 0.3385
Epoch 23/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 181s 3s/step - accuracy: 0.8777 - loss: 0.2608 - val_accu
racy: 0.9132 - val_loss: 0.2192
Epoch 24/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 156s 2s/step - accuracy: 0.9002 - loss: 0.2190 - val_accu
racy: 0.9097 - val_loss: 0.2553
Epoch 25/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 156s 2s/step - accuracy: 0.9128 - loss: 0.2245 - val_accu
racy: 0.9132 - val_loss: 0.2072
Epoch 26/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 178s 2s/step - accuracy: 0.9125 - loss: 0.2082 - val_accu
racy: 0.9271 - val_loss: 0.2158
Epoch 27/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 197s 3s/step - accuracy: 0.9242 - loss: 0.1926 - val_accu
racy: 0.9479 - val_loss: 0.1534
Epoch 28/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 165s 2s/step - accuracy: 0.9386 - loss: 0.1588 - val_accu
racy: 0.8819 - val_loss: 0.2550
Epoch 29/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 176s 2s/step - accuracy: 0.9517 - loss: 0.1485 - val_accu
racy: 0.9583 - val_loss: 0.1362
Epoch 30/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 181s 3s/step - accuracy: 0.9369 - loss: 0.1549 - val_accu
racy: 0.9549 - val_loss: 0.1416
Epoch 31/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 162s 2s/step - accuracy: 0.9319 - loss: 0.1708 - val_accu
racy: 0.9132 - val_loss: 0.2402
Epoch 32/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 170s 2s/step - accuracy: 0.9519 - loss: 0.1356 - val_accu
racy: 0.9444 - val_loss: 0.1313
Epoch 33/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 212s 3s/step - accuracy: 0.9469 - loss: 0.1352 - val_accu
racy: 0.9271 - val_loss: 0.1671
Epoch 34/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 176s 2s/step - accuracy: 0.9625 - loss: 0.1115 - val_accu
racy: 0.9444 - val_loss: 0.1390
Epoch 35/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 161s 2s/step - accuracy: 0.9470 - loss: 0.1456 - val_accu
racy: 0.9236 - val_loss: 0.1727
Epoch 36/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 179s 2s/step - accuracy: 0.9500 - loss: 0.1405 - val_accu
racy: 0.9549 - val_loss: 0.1147
Epoch 37/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 194s 3s/step - accuracy: 0.9612 - loss: 0.1053 - val_accu
racy: 0.9340 - val_loss: 0.1374
Epoch 38/50
```

```
72/72 ━━━━━━━━━━━━━━━━━━━━ 166s 2s/step - accuracy: 0.9513 - loss: 0.1347 - val_accu
racy: 0.9271 - val_loss: 0.1856
Epoch 39/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 175s 2s/step - accuracy: 0.9469 - loss: 0.1307 - val_accu
racy: 0.9583 - val_loss: 0.1223
Epoch 40/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 187s 3s/step - accuracy: 0.9520 - loss: 0.1171 - val_accu
racy: 0.9653 - val_loss: 0.1172
Epoch 41/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 170s 2s/step - accuracy: 0.9660 - loss: 0.0905 - val_accu
racy: 0.9653 - val_loss: 0.1108
Epoch 42/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 260s 4s/step - accuracy: 0.9542 - loss: 0.1138 - val_accu
racy: 0.9340 - val_loss: 0.1628
Epoch 43/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 173s 2s/step - accuracy: 0.9612 - loss: 0.0957 - val_accu
racy: 0.9583 - val_loss: 0.1092
Epoch 44/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 181s 3s/step - accuracy: 0.9689 - loss: 0.0917 - val_accu
racy: 0.9583 - val_loss: 0.1206
Epoch 45/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 200s 3s/step - accuracy: 0.9607 - loss: 0.1190 - val_accu
racy: 0.9271 - val_loss: 0.1825
Epoch 46/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 192s 3s/step - accuracy: 0.9548 - loss: 0.1469 - val_accu
racy: 0.9583 - val_loss: 0.1120
Epoch 47/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 195s 3s/step - accuracy: 0.9629 - loss: 0.0925 - val_accu
racy: 0.9549 - val_loss: 0.0963
Epoch 48/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 186s 3s/step - accuracy: 0.9577 - loss: 0.1072 - val_accu
racy: 0.9514 - val_loss: 0.1327
Epoch 49/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 203s 3s/step - accuracy: 0.9491 - loss: 0.1258 - val_accu
racy: 0.9583 - val_loss: 0.1111
Epoch 50/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 191s 3s/step - accuracy: 0.9715 - loss: 0.0932 - val_accu
racy: 0.9722 - val_loss: 0.0856
```

In [23]:
```
scores = model.evaluate(test_ds)
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 795ms/step - accuracy: 0.9741 - loss: 0.1117
```

In [25]:
```
scores
```

Out[25]:
```
[0.11559893935918808, 0.971875011920929]
```

In [26]:
```
history
```

Out[26]:
```
<keras.src.callbacks.history.History at 0x2a0f756f390>
```

In [27]:
```
history.params
```

Out[27]:
```
{'verbose': 'auto', 'epochs': 50, 'steps': 72}
```

In [28]:
```
history.history.keys()
```

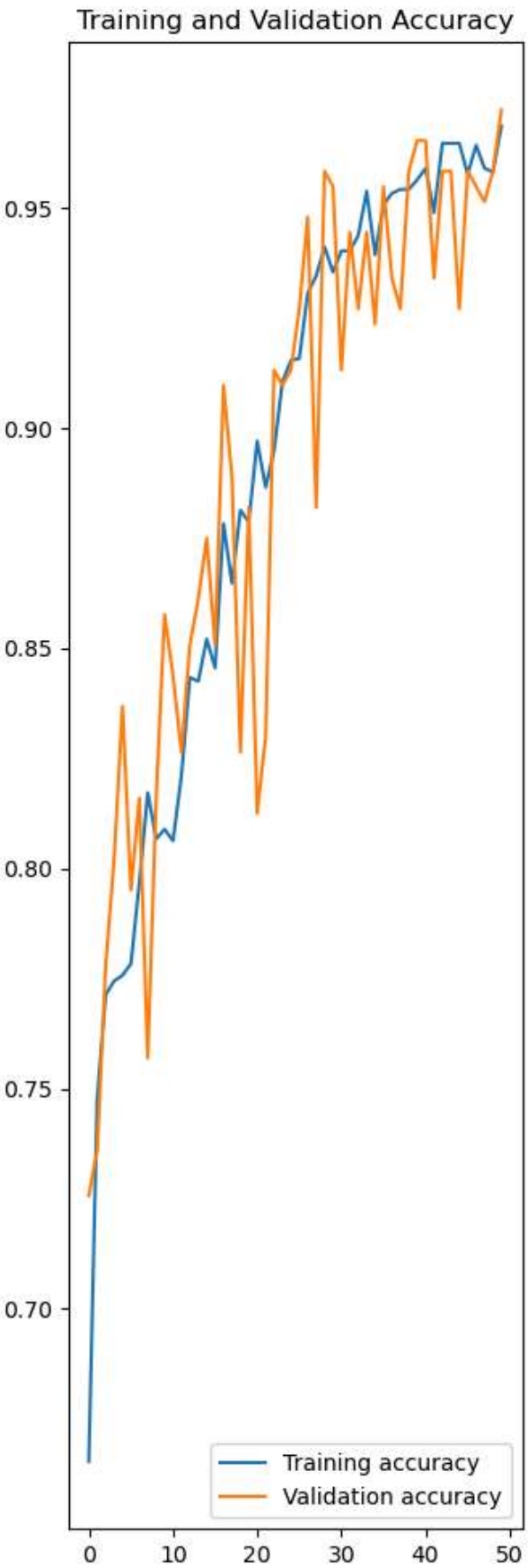
Out[28]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

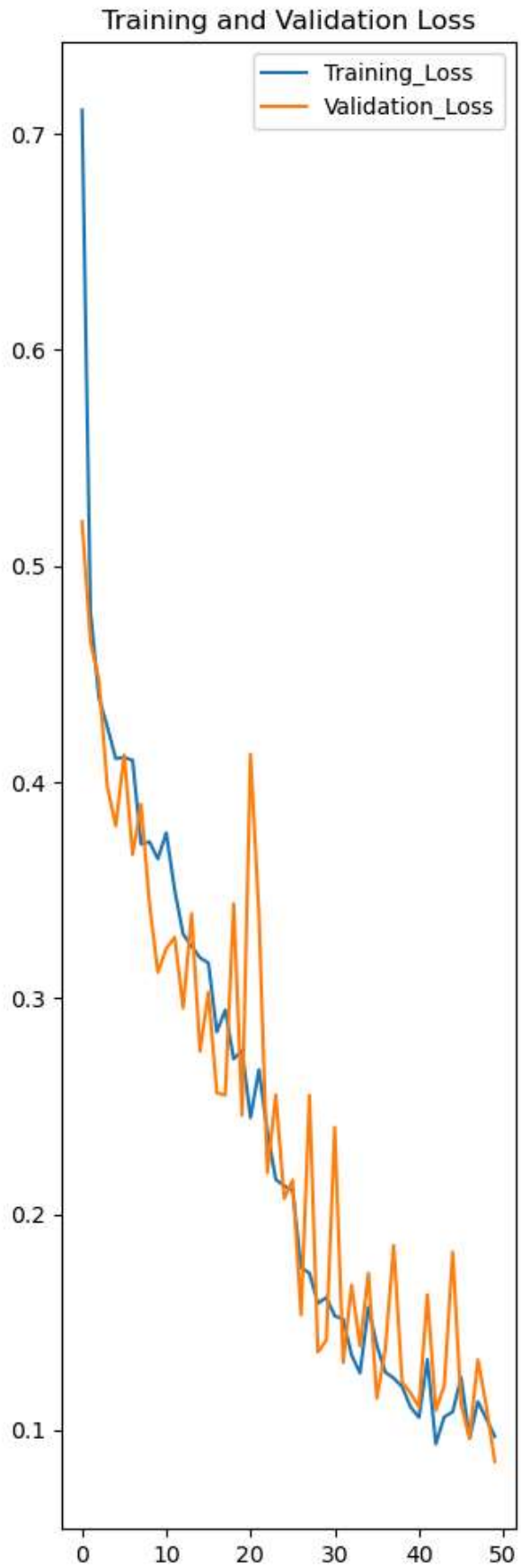
In [29]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

In [39]:
```
plt.figure(figsize=(8,12))
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()
```





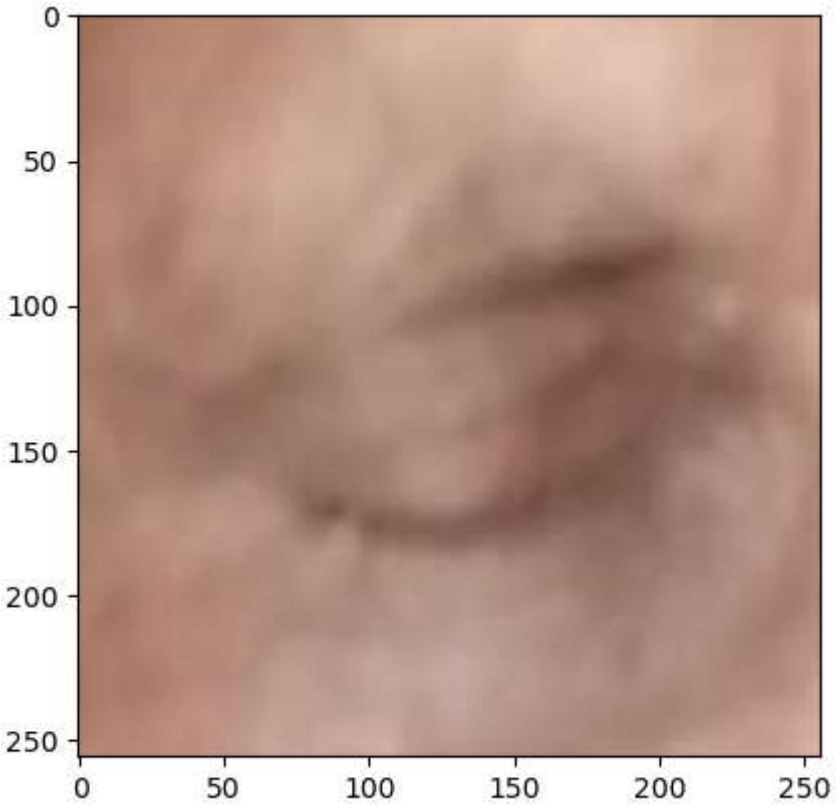
In [54]:
```
import numpy as np
```

In [71]:
```
for image_batch, labels_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

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

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

```
first image to predict
first image's actual label:  Closed
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 520ms/step
predicted label:  Closed
```



In [86]:
```
def predict(model, img):
    img_array = tf.keras.utils.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array,0) # create a batch

    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 [91]:
```
plt.figure(figsize=(12,12))
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 C
        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
```

In [95]:
```
model_version = 2
model.export(f"../Models/{model_version}")
```

```
INFO:tensorflow:Assets written to: ../Models/1\assets
INFO:tensorflow:Assets written to: ../Models/1\assets
Saved artifact at '../Models/1'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32,
name='keras_tensor')
Output Type:
  TensorSpec(shape=(None, 4), dtype=tf.float32, name=None)
Captures:
  2890395907376: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395907552: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395908080: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395909488: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395909136: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395908960: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395908784: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890395908432: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890478764976: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890478765328: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890478764800: TensorSpec(shape=(), dtype=tf.resource, name=None)
  2890478765680: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

In [1]:
```
print("Saurav")
```

```
Saurav
```