oral-cancer-effcientnet-classification

April 17, 2024

1 Import Libraries

[4]:
```
!pip install missingno
```

```
Collecting missingno
  Downloading missingno-0.5.2-py3-none-any.whl.metadata (639 bytes)
Requirement already satisfied: numpy in
/Users/raja/anaconda3/lib/python3.11/site-packages (from missingno) (1.26.4)
Requirement already satisfied: matplotlib in
/Users/raja/anaconda3/lib/python3.11/site-packages (from missingno) (3.8.4)
Requirement already satisfied: scipy in
/Users/raja/anaconda3/lib/python3.11/site-packages (from missingno) (1.13.0)
Requirement already satisfied: seaborn in
/Users/raja/anaconda3/lib/python3.11/site-packages (from missingno) (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(23.1)
Requirement already satisfied: pillow>=8 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->missingno)
(2.8.2)
```

```
Requirement already satisfied: pandas>=0.25 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from seaborn->missingno)
(2.2.2)
Requirement already satisfied: pytz>=2020.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from
pandas>=0.25->seaborn->missingno) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from
pandas>=0.25->seaborn->missingno) (2023.3)
Requirement already satisfied: six>=1.5 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from python-
dateutil>=2.7->matplotlib->missingno) (1.16.0)
Downloading missingno-0.5.2-py3-none-any.whl (8.7 kB)
Installing collected packages: missingno
Successfully installed missingno-0.5.2
```

[5]:

```
import os
import time
import shutil
import pathlib
import itertools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
import missingno as msno
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from plotly.offline import iplot
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.resnet50 import ResNet50
import warnings
warnings.filterwarnings("ignore")
```

```
print ('modules imported')
```

```
modules imported
```

2 Load Data

[6]:
```
tf.__version__
```

[6]:
```
'2.16.1'
```

[7]:
```
train_data_path = 'dataset/train'
test_data_path = 'dataset/test'
valid_data_path = 'dataset/val'
```

[8]:
```
labels = os.listdir(valid_data_path)
```

Creating data working directory

[11]:
```
data_path = 'data'

if not os.path.exists(data_path):
    os.mkdir(data_path)
    print("Created Succesfulley!")
else:
    print("Folder already exist")
```

```
Created Succesfulley!
```

[12]:
```
normal_data_path = 'data/Normal'
oscc_data_path = 'data/OSCC'

if not os.path.exists(normal_data_path):
    os.mkdir(normal_data_path)
    print("Created Succesfulley!")
else:
    print("Folder already exist")

if not os.path.exists(oscc_data_path):
    os.mkdir(oscc_data_path)
    print("Created Succesfulley!")
else:
    print("Folder already exist")
```

```
Created Succesfulley!
Created Succesfulley!
```

Moving all the images to the data working directory

[13]:
```python
def move_data(d_path, saved_path):
    for i in labels:
        images = os.listdir(d_path + '/' + i)
        for j in images:
            path = d_path +'/' + i
            img = cv2.imread(path + '/' + j)
            s_path = saved_path + '/' + i + '/' + j
            cv2.imwrite(s_path, img)
```

[14]:
```python
move_data(train_data_path, data_path)
```

[15]:
```python
norm_path = len(os.listdir(data_path + '/' + labels[0]))
oscc_path = len(os.listdir(data_path + '/' + labels[1]))
print(norm_path+oscc_path)
```

```
4946
```

[16]:
```python
move_data(test_data_path, data_path)
```

[17]:
```python
norm_path = len(os.listdir(data_path + '/' + labels[0]))
oscc_path = len(os.listdir(data_path + '/' + labels[1]))
print(norm_path+oscc_path)
```

```
5072
```

[18]:
```python
move_data(valid_data_path, data_path)
```

[19]:
```python
norm_path = len(os.listdir(data_path + '/' + labels[0]))
oscc_path = len(os.listdir(data_path + '/' + labels[1]))
print(norm_path+oscc_path)
```

```
5192
```

3 EDA

3.0.1 Define data path and dataset name

[20]:
```python
data_dir = 'data'
ds_name = 'Oral Cancer'
```

3.0.2 Create Dataframe

[21]:
```python
# Let's Generate data paths with labels

def generate_data_paths(data_dir):

    filepaths = []
    labels = []
```

```
    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
            labels.append(fold)

    return filepaths, labels


filepaths, labels = generate_data_paths(data_dir)
```

[22]:
```
def create_df(filepaths, labels):

    Fseries = pd.Series(filepaths, name= 'filepaths')
    Lseries = pd.Series(labels, name='labels')
    df = pd.concat([Fseries, Lseries], axis= 1)
    return df

df = create_df(filepaths, labels)
```

[23]:
```
df.head()
```

[23]:
```
                     filepaths labels
0  data/OSCC/OSCC_400x_426.jpg   OSCC
1  data/OSCC/OSCC_400x_340.jpg   OSCC
2  data/OSCC/OSCC_400x_354.jpg   OSCC
3   data/OSCC/aug_219_8212.jpg   OSCC
4  data/OSCC/OSCC_400x_432.jpg   OSCC
```

3.0.3 Number of Examples in the dataset

[24]:
```
def num_of_examples(df, name='df'):
    print(f"{name} dataset has {df.shape[0]} images.")

num_of_examples(df, ds_name)
```

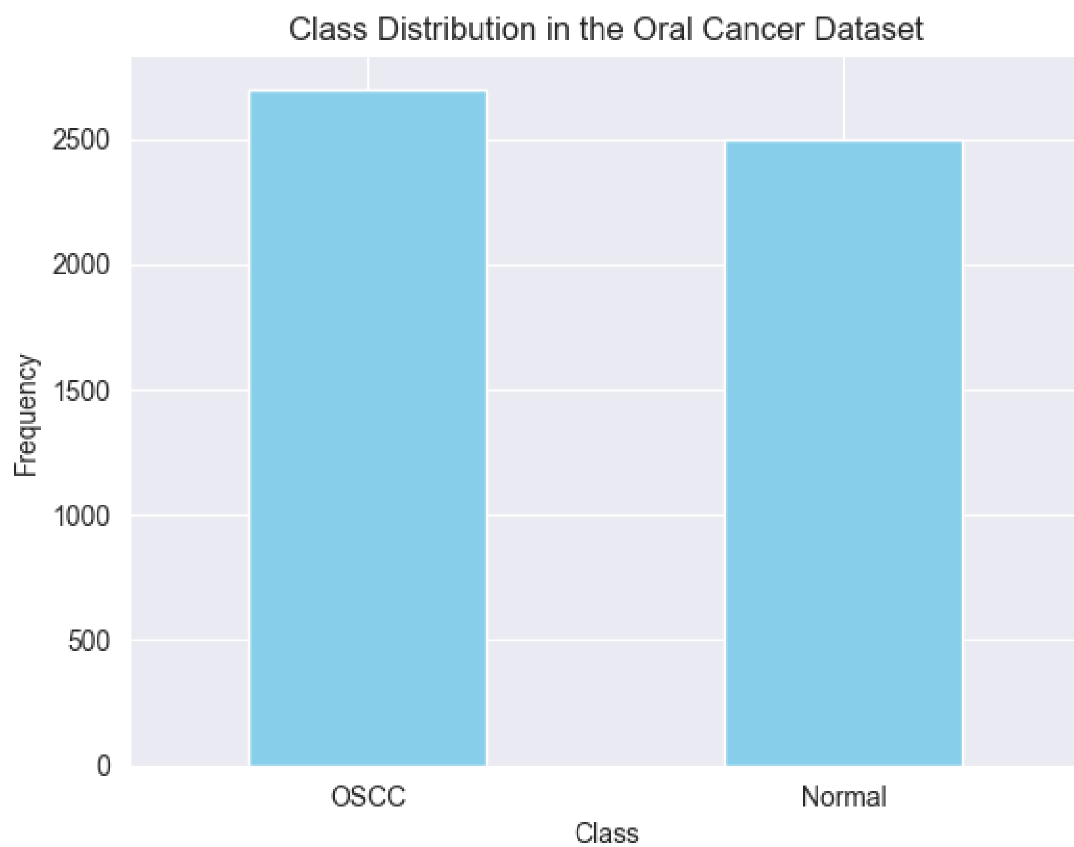
```
Oral Cancer dataset has 5192 images.
```

3.0.4 Number of Classes in the dataset

[27]:
```
import pandas as pd
import matplotlib.pyplot as plt

def num_of_classes(df, name='dataset'):
    print(f"The {name} dataset has {len(df['labels'].unique())} classes")
```

```
    return df['labels'].value_counts()

ds_name = "Oral Cancer"
class_counts = num_of_classes(df, ds_name)

# Plotting
fig, ax = plt.subplots()
class_counts.plot(kind='bar', ax=ax, color='skyblue')
ax.set_title(f'Class Distribution in the {ds_name} Dataset')
ax.set_xlabel('Class')
ax.set_ylabel('Frequency')
plt.xticks(rotation=0)  # Rotates labels to make them readable
plt.show()
```

```
The Oral Cancer dataset has 2 classes
```



3.0.5 No of images in each class of the dataset

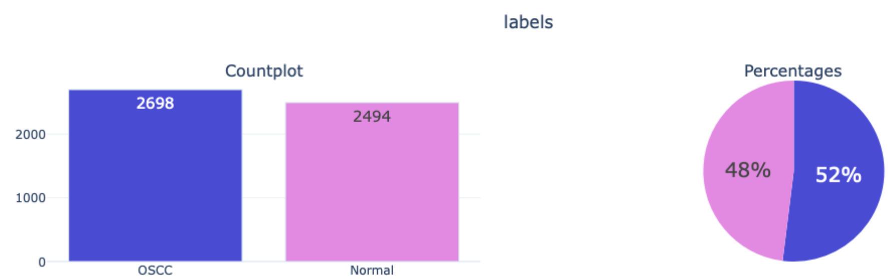
[28]:
```
def classes_count(df, name='df'):

    print(f"The {name} dataset has: ")
    print("="*70)
    print()
    for name in df['labels'].unique():
        num_class = len(df['labels'][df['labels'] == name])
        print(f"Class '{name}' has {num_class} images")
        print('-'*70)

classes_count(df, ds_name)
```

```
The Oral Cancer dataset has:
======================================================================

Class 'OSCC' has 2698 images
----------------------------------------------------------------------
Class 'Normal' has 2494 images
----------------------------------------------------------------------
```

3.0.6 Let's Visualize Each Class in the dataset

[29]:
```
def cat_summary_with_graph(dataframe, col_name):
    fig = make_subplots(rows=1, cols=2,
                        subplot_titles=('Countplot', 'Percentages'),
                        specs=[[{"type": "xy"}, {'type': 'domain'}]])

    fig.add_trace(go.Bar(y=dataframe[col_name].value_counts().values.tolist(),
                         x=[str(i) for i in dataframe[col_name].value_counts().
 ↪index],
                         text=dataframe[col_name].value_counts().values.
 ↪tolist(),
                         textfont=dict(size=15),
                         name=col_name,
                         textposition='auto',
                         showlegend=False,
                         marker=dict(color=colors,
                                     line=dict(color='#DBE6EC',
                                               width=1))),
                  row=1, col=1)

    fig.add_trace(go.Pie(labels=dataframe[col_name].value_counts().keys(),
                         values=dataframe[col_name].value_counts().values,
                         textfont=dict(size=20),
                         textposition='auto',
                         showlegend=False,
```

```
                            name=col_name,
                            marker=dict(colors=colors)),
                    row=1, col=2)

    fig.update_layout(title={'text': col_name,
                             'y': 0.9,
                             'x': 0.5,
                             'xanchor': 'center',
                             'yanchor': 'top'},
                        template='plotly_white')

    iplot(fig)


colors = ['#494BD3', '#E28AE2', '#F1F481', '#79DB80', '#DF5F5F',
              '#69DADE', '#C2E37D', '#E26580', '#D39F49', '#B96FE3']

cat_summary_with_graph(df,'labels')
```



3.0.7 Checking Null values in the dataframe

[30]:
```
def check_null_values(df, name='df'):

    num_null_vals = sum(df.isnull().sum().values)

    if not num_null_vals:
        print(f"The {name} dataset has no null values")

    else:
        print(f"The {name} dataset has {num_null_vals} null values")
        print('-'*70)
        print('Total null values in each column:\n')
        print(df.isnull().sum())
```

```
check_null_values(df, ds_name)
```

```
The Oral Cancer dataset has no null values
```

3.1 Split dataframe into train, valid, and test

[31]:
```
# train dataframe
train_df, dummy_df = train_test_split(df,  train_size= 0.7, shuffle= True,␣
↪random_state= 123)

# valid and test dataframe
valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= True,␣
↪random_state= 123)
```

[32]:
```
num_of_classes(train_df, "Training "+ds_name)
num_of_classes(valid_df, "Validation "+ds_name)
num_of_classes(test_df, "Testing "+ds_name)
```

```
The Training Oral Cancer dataset has 2 classes
The Validation Oral Cancer dataset has 2 classes
The Testing Oral Cancer dataset has 2 classes
```

[32]:
```
labels
OSCC      390
Normal    389
Name: count, dtype: int64
```

[33]:
```
classes_count(train_df, 'Training '+ds_name)
```

```
The Training Oral Cancer dataset has:
======================================================================

Class 'OSCC' has 1911 images
----------------------------------------------------------------------
Class 'Normal' has 1723 images
----------------------------------------------------------------------
```

[34]:
```
classes_count(valid_df, 'Validation '+ds_name)
```

```
The Validation Oral Cancer dataset has:
======================================================================

Class 'Normal' has 382 images
----------------------------------------------------------------------
Class 'OSCC' has 397 images
----------------------------------------------------------------------
```

[35]:
```
classes_count(test_df, 'Testing '+ds_name)
```

```
The Testing Oral Cancer dataset has:
======================================================================

Class 'OSCC' has 390 images
----------------------------------------------------------------------
Class 'Normal' has 389 images
----------------------------------------------------------------------
```

3.2 Let's Create Image Data Generator

[36]:
```python
# crobed image size
batch_size = 16
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

# Recommended : use custom function for test data batch size, else we can use␣
 ↪normal batch size.
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1)␣
 ↪if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

# This function which will be used in image data generator for data␣
 ↪augmentation, it just take the image and return it again.
def scalar(img):
    return img

tr_gen = ImageDataGenerator(preprocessing_function= scalar,
                           horizontal_flip=True)

ts_gen = ImageDataGenerator(preprocessing_function= scalar)

train_gen = tr_gen.flow_from_dataframe(train_df,
                                       x_col= 'filepaths',
                                       y_col= 'labels',
                                       target_size= img_size,
                                       class_mode= 'categorical',
                                       color_mode= 'rgb',
                                       shuffle= True,
                                       batch_size= batch_size)

valid_gen = ts_gen.flow_from_dataframe(valid_df,
                                       x_col= 'filepaths',
                                       y_col= 'labels',
                                       target_size= img_size,
                                       class_mode= 'categorical',
```

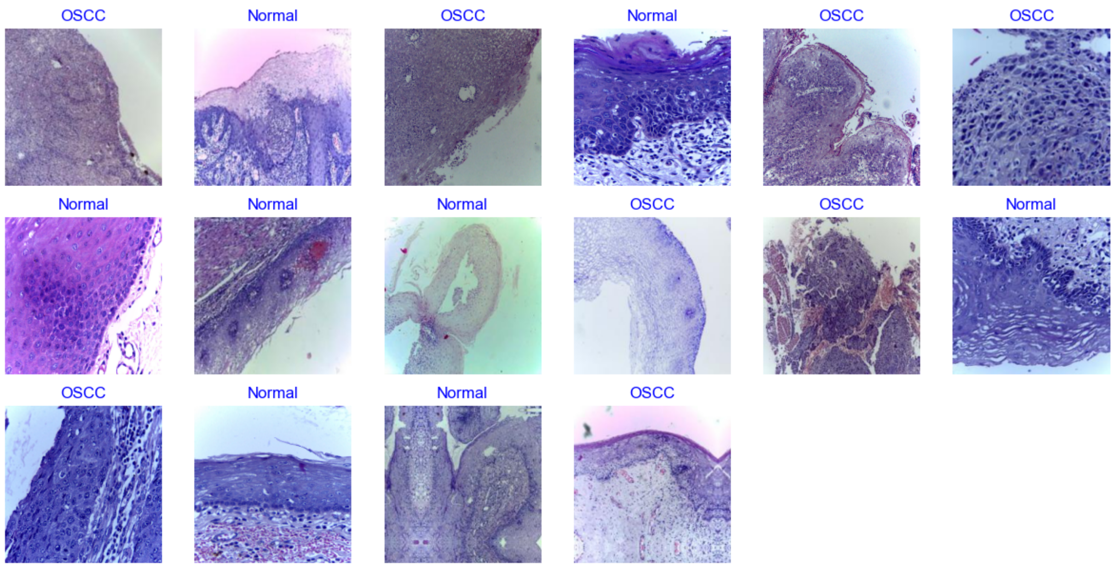
```
                                       color_mode= 'rgb',
                                       shuffle= True,
                                       batch_size= batch_size)

# Note: we will use custom test_batch_size, and make shuffle= false
test_gen = ts_gen.flow_from_dataframe(test_df,
                                      x_col= 'filepaths',
                                      y_col= 'labels',
                                      target_size= img_size,
                                      class_mode= 'categorical',
                                      color_mode= 'rgb',
                                      shuffle= False,
                                      batch_size= test_batch_size)
```

```
Found 3634 validated image filenames belonging to 2 classes.
Found 779 validated image filenames belonging to 2 classes.
Found 779 validated image filenames belonging to 2 classes.
```

3.2.1 Visualize Training dataset

[41]:

```
g_dict = train_gen.class_indices
classes = list(g_dict.keys())
images, labels = next(train_gen)

plt.figure(figsize= (15, 15))

for i in range(16):
    plt.subplot(6, 6, i + 1)
    image = images[i] / 255       # scales data to range (0 - 255)
    plt.imshow(image)
    index = np.argmax(labels[i])  # get image index
    class_name = classes[index]   # get class of image
    plt.title(class_name, color= 'blue', fontsize= 12)
    plt.axis('off')

plt.show()
```



4 Models

4.0.1 Generic Model

[43]:
```
# Correct model structure
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)
class_count = len(list(train_gen.class_indices.keys()))

base_model = tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=img_shape,
    pooling='max'
)

efficentNet_model = Sequential([
    base_model,
    BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001),
    Dense(256, kernel_regularizer=regularizers.l2(0.016),  # Corrected here
          activity_regularizer=regularizers.l1(0.006),
          bias_regularizer=regularizers.l1(0.006), activation='relu'),
    Dropout(rate=0.45, seed=123),
    Dense(class_count, activation='softmax')
])
```

```python
efficentNet_model.compile(Adamax(learning_rate=0.001),
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

efficentNet_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
efficientnetb3 (Functional)	?	10,783,535
batch_normalization_1 (BatchNormalization)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)

Total params: 10,783,535 (41.14 MB)

Trainable params: 10,696,232 (40.80 MB)

Non-trainable params: 87,303 (341.03 KB)

4.0.2 Early Stop

[44]:
```python
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=10,
                               restore_best_weights=True,
                               mode='min',
                              )
```

4.0.3 Let's Train the Model

[45]:
```python
batch_size = 128   # set batch size for training
epochs = 100   # number of all epochs in training

history = efficentNet_model.fit(x=train_gen,
                    epochs= epochs,
                    verbose= 1,
```

```
                    validation_data= valid_gen,
                    validation_steps= None,
                    shuffle= False,
                    batch_size= batch_size)
```

```
Epoch 1/100

2024-04-16 20:12:00.701906: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.

228/228              571s 2s/step -
accuracy: 0.5989 - loss: 12.1926 - val_accuracy: 0.5353 - val_loss: 5.6244
Epoch 2/100
228/228              443s 2s/step -
accuracy: 0.5414 - loss: 5.1147 - val_accuracy: 0.5148 - val_loss: 4.0747
Epoch 3/100
228/228              395s 2s/step -
accuracy: 0.5265 - loss: 3.7716 - val_accuracy: 0.5160 - val_loss: 3.0698
Epoch 4/100
228/228              414s 2s/step -
accuracy: 0.5357 - loss: 2.8107 - val_accuracy: 0.5173 - val_loss: 2.3946
Epoch 5/100
228/228              389s 2s/step -
accuracy: 0.5362 - loss: 2.1731 - val_accuracy: 0.5148 - val_loss: 2.0061
Epoch 6/100
228/228              368s 2s/step -
accuracy: 0.5264 - loss: 1.7572 - val_accuracy: 0.5276 - val_loss: 1.6311
Epoch 7/100
228/228              354s 2s/step -
accuracy: 0.5349 - loss: 1.4499 - val_accuracy: 0.5160 - val_loss: 1.3807
Epoch 8/100
228/228              340s 1s/step -
accuracy: 0.5364 - loss: 1.2213 - val_accuracy: 0.5173 - val_loss: 1.1269
Epoch 9/100
228/228              373s 2s/step -
accuracy: 0.5314 - loss: 1.0624 - val_accuracy: 0.5160 - val_loss: 1.0149
Epoch 10/100
228/228              361s 2s/step -
accuracy: 0.5263 - loss: 0.9513 - val_accuracy: 0.5096 - val_loss: 0.9411
Epoch 11/100
228/228              355s 2s/step -
accuracy: 0.5158 - loss: 0.8854 - val_accuracy: 0.5096 - val_loss: 0.8688
Epoch 12/100
228/228              348s 2s/step -
accuracy: 0.5347 - loss: 0.8257 - val_accuracy: 0.5032 - val_loss: 0.8162
Epoch 13/100
228/228              354s 2s/step -
accuracy: 0.5310 - loss: 0.7860 - val_accuracy: 0.5109 - val_loss: 0.8068
```

```
Epoch 14/100
228/228          363s 2s/step -
accuracy: 0.5309 - loss: 0.7590 - val_accuracy: 0.5096 - val_loss: 0.7693
Epoch 15/100
228/228          365s 2s/step -
accuracy: 0.5285 - loss: 0.7391 - val_accuracy: 0.5096 - val_loss: 0.7457
Epoch 16/100
228/228          356s 2s/step -
accuracy: 0.5187 - loss: 0.7241 - val_accuracy: 0.5071 - val_loss: 0.7379
Epoch 17/100
228/228          365s 2s/step -
accuracy: 0.5324 - loss: 0.7183 - val_accuracy: 0.5135 - val_loss: 0.7224
Epoch 18/100
228/228          367s 2s/step -
accuracy: 0.5227 - loss: 0.7113 - val_accuracy: 0.5199 - val_loss: 0.7169
Epoch 19/100
228/228          385s 2s/step -
accuracy: 0.5351 - loss: 0.7031 - val_accuracy: 0.5096 - val_loss: 0.7300
Epoch 20/100
228/228          378s 2s/step -
accuracy: 0.5289 - loss: 0.7057 - val_accuracy: 0.5148 - val_loss: 0.7062
Epoch 21/100
228/228          373s 2s/step -
accuracy: 0.5216 - loss: 0.7043 - val_accuracy: 0.5096 - val_loss: 0.7003
Epoch 22/100
228/228          382s 2s/step -
accuracy: 0.5214 - loss: 0.7005 - val_accuracy: 0.5122 - val_loss: 0.7187
Epoch 23/100
228/228          379s 2s/step -
accuracy: 0.5186 - loss: 0.6988 - val_accuracy: 0.5096 - val_loss: 0.7011
Epoch 24/100
228/228          376s 2s/step -
accuracy: 0.5257 - loss: 0.6962 - val_accuracy: 0.5148 - val_loss: 0.6983
Epoch 25/100
228/228          368s 2s/step -
accuracy: 0.5249 - loss: 0.6969 - val_accuracy: 0.5096 - val_loss: 0.6991
Epoch 26/100
228/228          383s 2s/step -
accuracy: 0.5243 - loss: 0.6955 - val_accuracy: 0.5096 - val_loss: 0.6974
Epoch 27/100
228/228          380s 2s/step -
accuracy: 0.5282 - loss: 0.6940 - val_accuracy: 0.5096 - val_loss: 0.7102
Epoch 28/100
228/228          403s 2s/step -
accuracy: 0.5311 - loss: 0.6938 - val_accuracy: 0.5096 - val_loss: 0.6979
Epoch 29/100
228/228          390s 2s/step -
accuracy: 0.5257 - loss: 0.6949 - val_accuracy: 0.5096 - val_loss: 0.6971
```

```
Epoch 30/100
228/228 403s 2s/step -
accuracy: 0.5119 - loss: 0.6959 - val_accuracy: 0.5096 - val_loss: 0.7443
Epoch 31/100
228/228 379s 2s/step -
accuracy: 0.5302 - loss: 0.6973 - val_accuracy: 0.5096 - val_loss: 0.7018
Epoch 32/100
228/228 410s 2s/step -
accuracy: 0.5318 - loss: 0.6939 - val_accuracy: 0.5135 - val_loss: 0.6987
Epoch 33/100
228/228 378s 2s/step -
accuracy: 0.5203 - loss: 0.6955 - val_accuracy: 0.5096 - val_loss: 0.7008
Epoch 34/100
228/228 389s 2s/step -
accuracy: 0.5369 - loss: 0.6935 - val_accuracy: 0.5096 - val_loss: 0.7122
Epoch 35/100
228/228 386s 2s/step -
accuracy: 0.5293 - loss: 0.6975 - val_accuracy: 0.5096 - val_loss: 0.6973
Epoch 36/100
228/228 419s 2s/step -
accuracy: 0.5155 - loss: 0.6956 - val_accuracy: 0.5096 - val_loss: 0.7086
Epoch 37/100
228/228 356s 2s/step -
accuracy: 0.5311 - loss: 0.6983 - val_accuracy: 0.5956 - val_loss: 0.6969
Epoch 38/100
228/228 365s 2s/step -
accuracy: 0.6361 - loss: 0.6911 - val_accuracy: 0.7381 - val_loss: 0.6718
Epoch 39/100
228/228 359s 2s/step -
accuracy: 0.6538 - loss: 0.6771 - val_accuracy: 0.8280 - val_loss: 0.6354
Epoch 40/100
228/228 372s 2s/step -
accuracy: 0.6907 - loss: 0.6500 - val_accuracy: 0.8652 - val_loss: 0.5819
Epoch 41/100
228/228 364s 2s/step -
accuracy: 0.7048 - loss: 0.6225 - val_accuracy: 0.8716 - val_loss: 0.5588
Epoch 42/100
228/228 367s 2s/step -
accuracy: 0.7261 - loss: 0.5906 - val_accuracy: 0.8973 - val_loss: 0.5286
Epoch 43/100
228/228 367s 2s/step -
accuracy: 0.7479 - loss: 0.5582 - val_accuracy: 0.9024 - val_loss: 0.4864
Epoch 44/100
228/228 370s 2s/step -
accuracy: 0.7386 - loss: 0.5388 - val_accuracy: 0.8947 - val_loss: 0.4637
Epoch 45/100
228/228 372s 2s/step -
accuracy: 0.7348 - loss: 0.5358 - val_accuracy: 0.9127 - val_loss: 0.4530
```

```
Epoch 46/100
228/228              368s 2s/step -
accuracy: 0.7358 - loss: 0.5227 - val_accuracy: 0.9204 - val_loss: 0.4243
Epoch 47/100
228/228              381s 2s/step -
accuracy: 0.7433 - loss: 0.5100 - val_accuracy: 0.8909 - val_loss: 0.4234
Epoch 48/100
228/228              375s 2s/step -
accuracy: 0.7384 - loss: 0.5119 - val_accuracy: 0.9089 - val_loss: 0.4268
Epoch 49/100
228/228              374s 2s/step -
accuracy: 0.7462 - loss: 0.5085 - val_accuracy: 0.9204 - val_loss: 0.4085
Epoch 50/100
228/228              384s 2s/step -
accuracy: 0.7567 - loss: 0.4891 - val_accuracy: 0.9255 - val_loss: 0.3909
Epoch 51/100
228/228              375s 2s/step -
accuracy: 0.7618 - loss: 0.4713 - val_accuracy: 0.9127 - val_loss: 0.3888
Epoch 52/100
228/228              369s 2s/step -
accuracy: 0.7577 - loss: 0.4786 - val_accuracy: 0.9076 - val_loss: 0.3824
Epoch 53/100
228/228              368s 2s/step -
accuracy: 0.7504 - loss: 0.4742 - val_accuracy: 0.9384 - val_loss: 0.4237
Epoch 54/100
228/228              373s 2s/step -
accuracy: 0.7682 - loss: 0.4700 - val_accuracy: 0.9153 - val_loss: 0.3726
Epoch 55/100
228/228              396s 2s/step -
accuracy: 0.7774 - loss: 0.4609 - val_accuracy: 0.9307 - val_loss: 0.3917
Epoch 56/100
228/228              401s 2s/step -
accuracy: 0.7621 - loss: 0.4595 - val_accuracy: 0.9448 - val_loss: 0.3567
Epoch 57/100
228/228              375s 2s/step -
accuracy: 0.7502 - loss: 0.4630 - val_accuracy: 0.9474 - val_loss: 0.3568
Epoch 58/100
228/228              370s 2s/step -
accuracy: 0.7641 - loss: 0.4564 - val_accuracy: 0.9512 - val_loss: 0.3446
Epoch 59/100
228/228              389s 2s/step -
accuracy: 0.7619 - loss: 0.4459 - val_accuracy: 0.9281 - val_loss: 0.3679
Epoch 60/100
228/228              385s 2s/step -
accuracy: 0.7511 - loss: 0.4569 - val_accuracy: 0.9461 - val_loss: 0.3472
Epoch 61/100
228/228              380s 2s/step -
accuracy: 0.7679 - loss: 0.4419 - val_accuracy: 0.9384 - val_loss: 0.3473
```

```
Epoch 62/100
228/228             383s 2s/step -
accuracy: 0.7727 - loss: 0.4442 - val_accuracy: 0.9448 - val_loss: 0.3267
Epoch 63/100
228/228             382s 2s/step -
accuracy: 0.7698 - loss: 0.4401 - val_accuracy: 0.9371 - val_loss: 0.3402
Epoch 64/100
228/228             390s 2s/step -
accuracy: 0.7691 - loss: 0.4298 - val_accuracy: 0.9384 - val_loss: 0.3551
Epoch 65/100
228/228             369s 2s/step -
accuracy: 0.7626 - loss: 0.4440 - val_accuracy: 0.9320 - val_loss: 0.3361
Epoch 66/100
228/228             371s 2s/step -
accuracy: 0.7696 - loss: 0.4235 - val_accuracy: 0.9307 - val_loss: 0.3157
Epoch 67/100
228/228             380s 2s/step -
accuracy: 0.7649 - loss: 0.4283 - val_accuracy: 0.9294 - val_loss: 0.3297
Epoch 68/100
228/228             380s 2s/step -
accuracy: 0.7670 - loss: 0.4350 - val_accuracy: 0.9397 - val_loss: 0.3118
Epoch 69/100
228/228             397s 2s/step -
accuracy: 0.7675 - loss: 0.4269 - val_accuracy: 0.9281 - val_loss: 0.3355
Epoch 70/100
228/228             376s 2s/step -
accuracy: 0.7481 - loss: 0.4419 - val_accuracy: 0.9602 - val_loss: 0.3454
Epoch 71/100
228/228             384s 2s/step -
accuracy: 0.7441 - loss: 0.4413 - val_accuracy: 0.9320 - val_loss: 0.3246
Epoch 72/100
228/228             379s 2s/step -
accuracy: 0.7416 - loss: 0.4365 - val_accuracy: 0.9371 - val_loss: 0.3224
Epoch 73/100
228/228             377s 2s/step -
accuracy: 0.7654 - loss: 0.4380 - val_accuracy: 0.9178 - val_loss: 0.3206
Epoch 74/100
228/228             372s 2s/step -
accuracy: 0.8272 - loss: 0.4287 - val_accuracy: 0.9294 - val_loss: 0.3312
Epoch 75/100
228/228             375s 2s/step -
accuracy: 0.8509 - loss: 0.4272 - val_accuracy: 0.9268 - val_loss: 0.4376
Epoch 76/100
228/228             407s 2s/step -
accuracy: 0.8598 - loss: 0.4149 - val_accuracy: 0.9435 - val_loss: 0.3123
Epoch 77/100
228/228             381s 2s/step -
accuracy: 0.8571 - loss: 0.3982 - val_accuracy: 0.9345 - val_loss: 0.3214
```

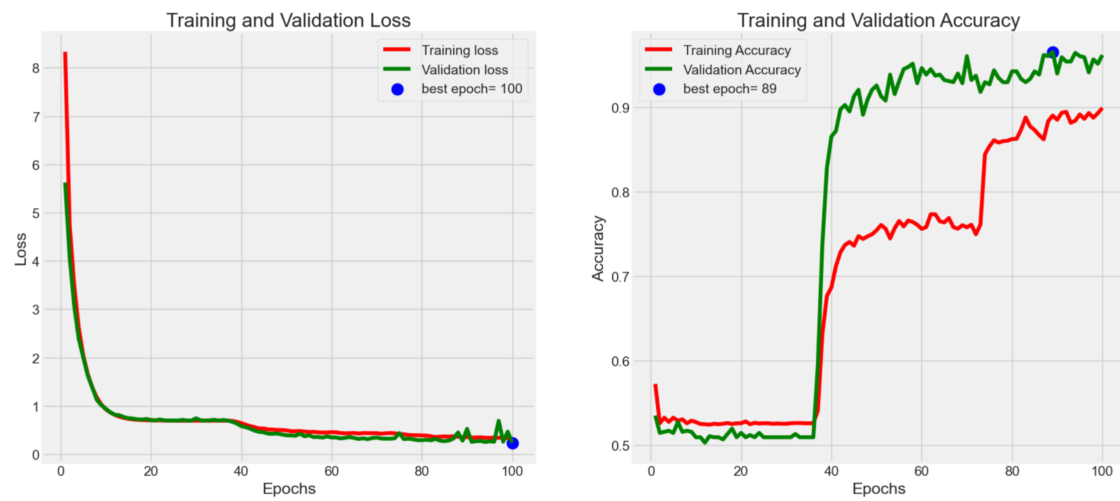
```
Epoch 78/100
228/228            398s 2s/step -
accuracy: 0.8479 - loss: 0.3984 - val_accuracy: 0.9294 - val_loss: 0.3090
Epoch 79/100
228/228            376s 2s/step -
accuracy: 0.8549 - loss: 0.4035 - val_accuracy: 0.9294 - val_loss: 0.2930
Epoch 80/100
228/228            371s 2s/step -
accuracy: 0.8594 - loss: 0.3951 - val_accuracy: 0.9422 - val_loss: 0.2906
Epoch 81/100
228/228            353s 2s/step -
accuracy: 0.8643 - loss: 0.3838 - val_accuracy: 0.9422 - val_loss: 0.2999
Epoch 82/100
228/228            407s 2s/step -
accuracy: 0.8669 - loss: 0.3691 - val_accuracy: 0.9332 - val_loss: 0.2892
Epoch 83/100
228/228            492s 2s/step -
accuracy: 0.8818 - loss: 0.3567 - val_accuracy: 0.9294 - val_loss: 0.3158
Epoch 84/100
228/228            517s 2s/step -
accuracy: 0.8790 - loss: 0.3639 - val_accuracy: 0.9332 - val_loss: 0.2818
Epoch 85/100
228/228            444s 2s/step -
accuracy: 0.8795 - loss: 0.3624 - val_accuracy: 0.9422 - val_loss: 0.2700
Epoch 86/100
228/228            401s 2s/step -
accuracy: 0.8661 - loss: 0.3664 - val_accuracy: 0.9384 - val_loss: 0.2855
Epoch 87/100
228/228            425s 2s/step -
accuracy: 0.8585 - loss: 0.3690 - val_accuracy: 0.9615 - val_loss: 0.3246
Epoch 88/100
228/228            387s 2s/step -
accuracy: 0.8796 - loss: 0.3731 - val_accuracy: 0.9602 - val_loss: 0.4503
Epoch 89/100
228/228            397s 2s/step -
accuracy: 0.8926 - loss: 0.3429 - val_accuracy: 0.9653 - val_loss: 0.2803
Epoch 90/100
228/228            407s 2s/step -
accuracy: 0.8810 - loss: 0.3633 - val_accuracy: 0.9397 - val_loss: 0.5252
Epoch 91/100
228/228            412s 2s/step -
accuracy: 0.8978 - loss: 0.3410 - val_accuracy: 0.9589 - val_loss: 0.2564
Epoch 92/100
228/228            376s 2s/step -
accuracy: 0.8944 - loss: 0.3474 - val_accuracy: 0.9538 - val_loss: 0.2696
Epoch 93/100
228/228            397s 2s/step -
accuracy: 0.8796 - loss: 0.3465 - val_accuracy: 0.9538 - val_loss: 0.2743
```

```
Epoch 94/100
228/228              372s 2s/step -
accuracy: 0.8806 - loss: 0.3442 - val_accuracy: 0.9641 - val_loss: 0.2545
Epoch 95/100
228/228              390s 2s/step -
accuracy: 0.8979 - loss: 0.3352 - val_accuracy: 0.9602 - val_loss: 0.2674
Epoch 96/100
228/228              403s 2s/step -
accuracy: 0.8880 - loss: 0.3413 - val_accuracy: 0.9589 - val_loss: 0.2585
Epoch 97/100
228/228              391s 2s/step -
accuracy: 0.8908 - loss: 0.3441 - val_accuracy: 0.9409 - val_loss: 0.6928
Epoch 98/100
228/228              399s 2s/step -
accuracy: 0.8840 - loss: 0.3517 - val_accuracy: 0.9564 - val_loss: 0.2602
Epoch 99/100
228/228              395s 2s/step -
accuracy: 0.9002 - loss: 0.3350 - val_accuracy: 0.9512 - val_loss: 0.4680
Epoch 100/100
228/228              396s 2s/step -
accuracy: 0.8949 - loss: 0.3296 - val_accuracy: 0.9615 - val_loss: 0.2390
```

4.0.4 Model Evaluation

[46]:
```python
# Define needed variables
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
Epochs = [i+1 for i in range(len(tr_acc))]
loss_label = f'best epoch= {str(index_loss + 1)}'
acc_label = f'best epoch= {str(index_acc + 1)}'

# Plot training history

plt.figure(figsize= (20, 8))
plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')
```

```python
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()
```



4.0.5 Let's calculate the model accuray

[47]:
```python
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = efficentNet_model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = efficentNet_model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = efficentNet_model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
```

```
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
19/19                 2s 121ms/step -
accuracy: 0.9701 - loss: 0.2293
19/19                 3s 130ms/step -
accuracy: 0.9392 - loss: 0.2831
19/19                 14s 440ms/step -
accuracy: 0.9786 - loss: 0.3343
Train Loss:  0.20888178050518036
Train Accuracy:  0.9835526347160339
--------------------
Validation Loss:  0.26151010394096375
Validation Accuracy:  0.9539473652839661
--------------------
Test Loss:  0.3370945155620575
Test Accuracy:  0.973042368888855
```

4.0.6 Get Prediction

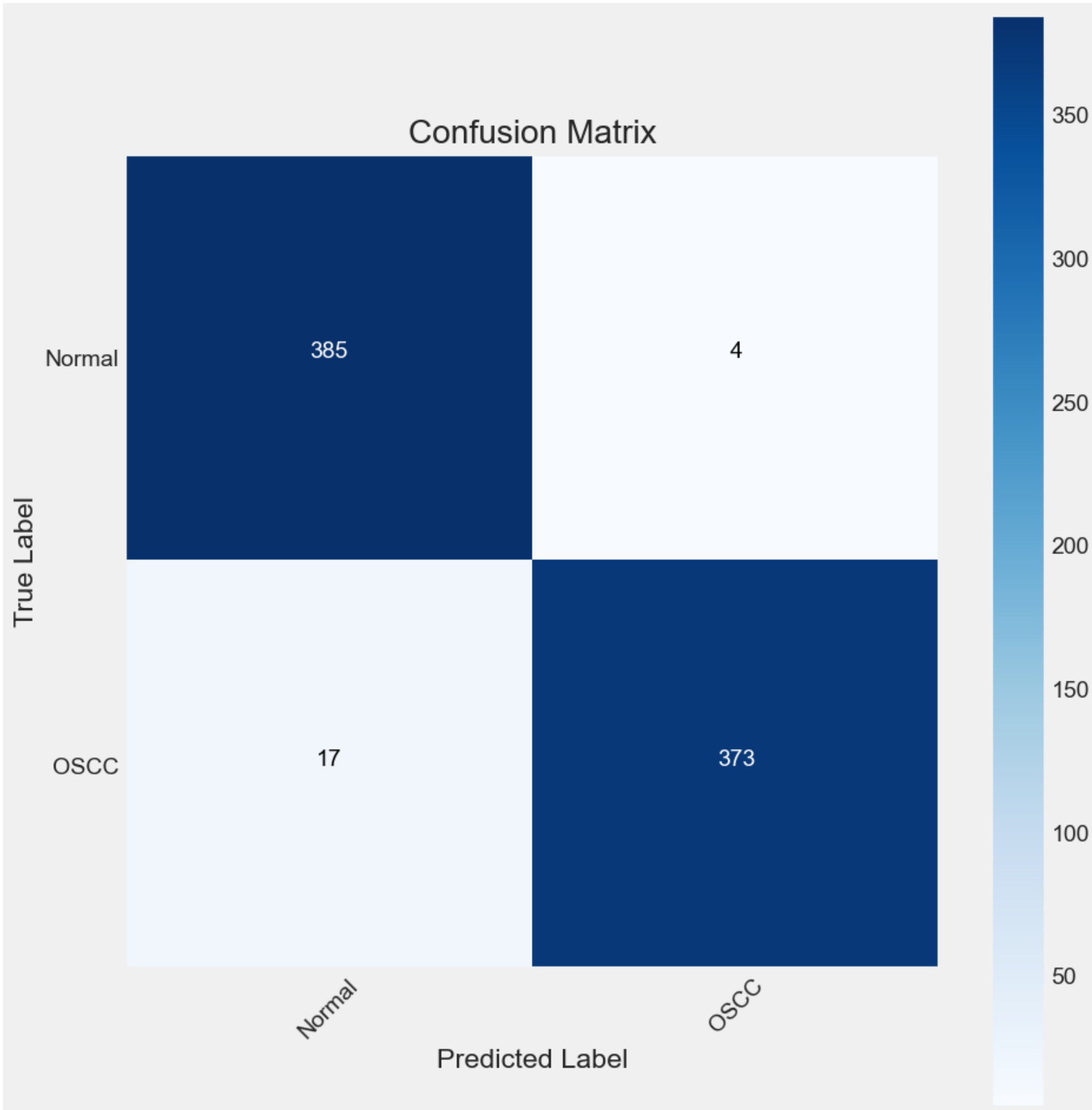
[53]:
```
preds = efficentNet_model.predict(test_gen, steps=test_steps)
y_pred = np.argmax(preds, axis=1)
```

```
19/19                 29s 447ms/step
```

4.0.7 Confussion Matrix

[54]:
```
#'test_gen' is testing data generator and 'y_pred' is the array of predictions
g_dict = test_gen.class_indices
classes = list(g_dict.keys())

# Calculate the confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)

# Create the plot
plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()

# Setting tick marks and labels for classes
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
```

```
plt.yticks(tick_marks, classes)

# Determine text color based on background
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j],
             horizontalalignment='center',
             color='white' if cm[i, j] > thresh else 'black')

# Additional styling
plt.gca().set_facecolor('white')
plt.grid(False)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')

plt.tight_layout()  # Adjust layout to make room for rotated x-labels
plt.show()
```



4.0.8 Classification Report

[55]:
```
# Classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

```
              precision    recall  f1-score   support

      Normal       0.96      0.99      0.97       389
        OSCC       0.99      0.96      0.97       390

    accuracy                           0.97       779
   macro avg       0.97      0.97      0.97       779
weighted avg       0.97      0.97      0.97       779
```

4.0.9 Let's Save the Model For Future Use

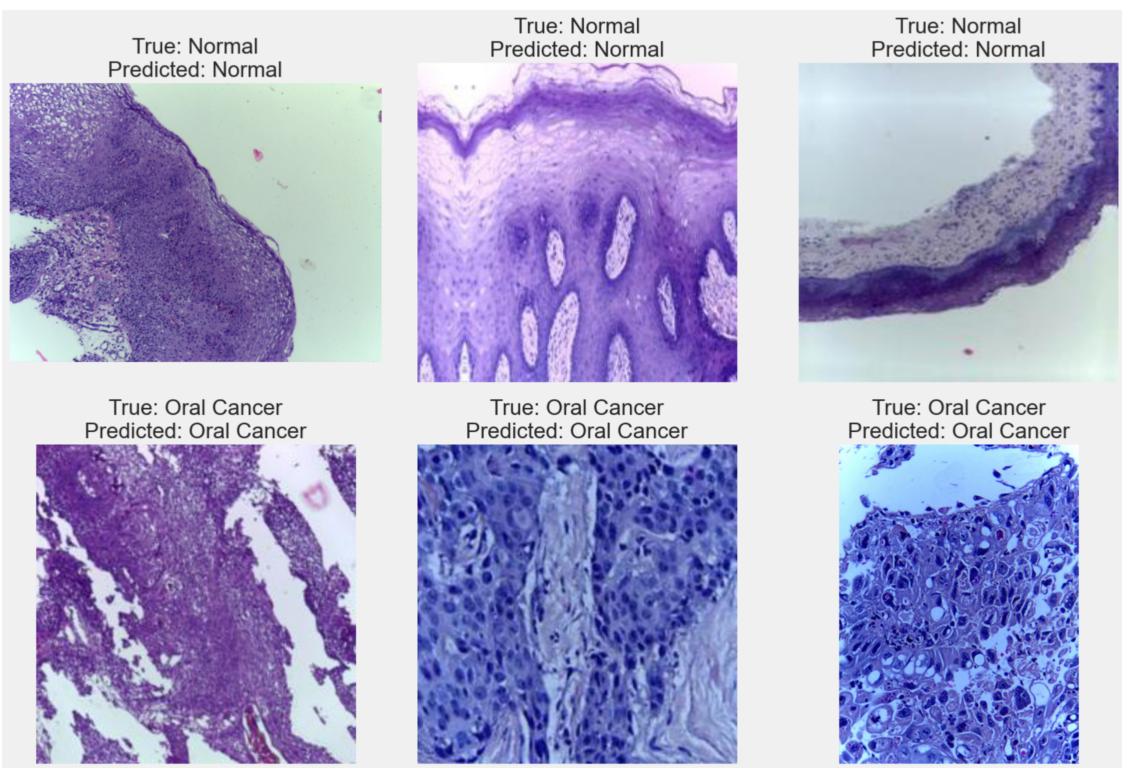
[56]:
```
efficentNet_model.save_weights('my_model.weights.h5')
```

4.0.10 Predictions

[57]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.models import load_model
import random

def predict_and_display(directory_normal, directory_cancer, model,␣
 ↪num_images=3):
    # List files in each directory
    normal_images = [os.path.join(directory_normal, img) for img in os.
 ↪listdir(directory_normal)]
    cancer_images = [os.path.join(directory_cancer, img) for img in os.
 ↪listdir(directory_cancer)]

    # Randomly select images
    selected_normal = random.sample(normal_images, num_images)
    selected_cancer = random.sample(cancer_images, num_images)

    # Concatenate all selected images
    selected_images = selected_normal + selected_cancer
    true_labels = ['Normal'] * num_images + ['Oral Cancer'] * num_images

    plt.figure(figsize=(15, 10))

    for i, (img_path, true_label) in enumerate(zip(selected_images,␣
 ↪true_labels), 1):
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = preprocess_input(img_array)

        prediction = model.predict(img_array)
        predicted_class_index = np.argmax(prediction)
        predicted_class_label = class_labels[predicted_class_index]

        # Plotting
        ax = plt.subplot(2, 3, i)
```

```python
        plt.imshow(image.load_img(img_path))
        plt.title(f"True: {true_label}\nPredicted: {predicted_class_label}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# 'efficentNet_model' is  compiled
class_labels = ['Normal', 'Oral Cancer']
directory_normal = 'data/Normal/'
directory_cancer = 'data/OSCC/'

# Predict and display images
predict_and_display(directory_normal, directory_cancer, efficentNet_model,␣
 ↪num_images=3)
```

```
1/1              0s 350ms/step
1/1              0s 57ms/step
1/1              0s 59ms/step
1/1              1s 1s/step
1/1              0s 54ms/step
1/1              0s 54ms/step
```



```
[ ]:
```