LIBRARIES IMPORT

In [2]:
```
## Importing Libraries
###Tensorflow and Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras.utils import image_dataset_from_directory

###Image Processing Libraries
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import pickle
from PIL import Image
import cv2
import Augmentor

###Import other basic data processing libraries
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from pandas import Series

###Visualization
import matplotlib.pyplot as plt
import plotly
import nbformat
%matplotlib inline
import scikitplot as skplt

###Import system libraries

import os
from pathlib import Path
import logging, sys
import datetime
import time
```

Filepath Label generator Function

In [3]:
```
def _FilepathLabelGenerator_Test(filepath):
    filepathList=[]
    labelList=[]
    labels={}
    listedClass=os.listdir(filepath)
    for i in range(len(listedClass)):
        wrd=listedClass[i].split("_")
        j=0
        s=''
        while j<len(wrd):
            s=(s+wrd[j][0])
            j+=1
        labels.update({listedClass[i]:s.upper()})

    for class_ in listedClass:
            imagefolder = os.path.join(filepath,class_)
            image_paths = os.listdir(imagefolder)
            for imagepath in image_paths:
                filepathList.append(os.path.join(imagefolder,imagepath))
                labelList.append(labels.get(class_))
    return filepathList, labelList
```

In [4]:
```
##Function for Filepath and Labels
def _FilepathLabelGenerator_Trn(filepath):
    filepathList=[]
    labelList=[]
    labels={}
    listedClass=os.listdir(filepath)
    for i in range(len(listedClass)):
        wrd=listedClass[i].split("_")
        j=0
        s=''
        while j<len(wrd):
            s=(s+wrd[j][0])
            j+=1
        labels.update({listedClass[i]:s.upper()})

    for class_ in listedClass:
            imagefolder = os.path.join(filepath,class_)
            image_paths = os.listdir(imagefolder+"\output")
            for imagepath in image_paths:
                filepathList.append(os.path.join(imagefolder+"\output",imagepath))
                labelList.append(labels.get(class_))
    return filepathList, labelList
```

File path Generation (Edit ME)

In [5]:
```
rootpath = r'K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Pap_Smear'
```

In [6]:
```
imagepath, labels = _FilepathLabelGenerator_Test(rootpath)
df_imgpath_label = pd.DataFrame({'imagepath':imagepath, 'label':labels})
df_imgpath_label
```

Out[6]:

	imagepath	label
0	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
1	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
2	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
3	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
4	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
...	...	...
957	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
958	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
959	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
960	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
961	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC

962 rows × 2 columns

Test Images Transfer Function

In [7]:
```
import shutil

def testImage_transfer(test_df,transferfolderpath):
    test_df = test_df.reset_index()
    for img_num in range(len(test_df)):
        f = test_df['imagepath'][img_num].split('\\')[-1]
        new_path=os.path.join(transferfolderpath,f)
        shutil.move(test_df['imagepath'][img_num], new_path)
        test_df['imagepath'][img_num]=new_path
    return test_df
```

TRAIN TEST SPLIT FUNCTION (Run Only)

In [8]:
```
##Training and Test Split
ds_dum, ds_tst_Og = train_test_split(df_imgpath_label,  test_size= 0.2, shuffle= True, random_state= 31, stratify=df_imgpath_label['label'])
ds_tst_Og
```

Out[8]:

	imagepath	label
113	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
99	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
776	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
111	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
566	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
...	...	...
255	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
400	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
304	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
931	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
6	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL

193 rows × 2 columns

TEST FILES TRANSFER

In [9]:
```
rootpath_test = r'K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Test'
```

In [10]:
```
ds_tst_Og=testImage_transfer(ds_tst_Og,rootpath_test)
ds_tst_Og
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3040331425.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['imagepath'][img_num]=new_path
```

Out[10]:

	index	imagepath	label
0	113	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
1	99	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
2	776	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
3	111	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
4	566	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
...	...	...	...
188	255	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
189	400	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
190	304	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
191	931	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
192	6	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL

193 rows × 3 columns

AUGMENTATION FUNCTION

In [11]:

```
def Image_Oversample(path,sample_size):
    aug = Augmentor.Pipeline(path)
    aug.rotate(0.7,25,25)
    aug.flip_random(0.5)
    aug.skew(0.3)
    aug.crop_random(0.4,0.8)
    aug.sample(sample_size)
```

IMAGE AUGMENTATIONS ON TRAIN SET ONLY

In [12]:

```
hsil_count = ds_dum[ds_dum['label']=='HSIL'].count()[1]
lsil_count = ds_dum[ds_dum['label']=='LSIL'].count()[1]
scc_count = ds_dum[ds_dum['label']=='SCC'].count()[1]
```

```
neg_count = ds_dum[ds_dum['label']=='NEG'].count()[1]
total = len(ds_dum)

print("The count of samples in each class is: \n\nHSIL:{}({}%) \n\nLSIL:{}({}%) \n\nSCC:{}({}%) \n\nNEG:{}({}%)".format(hsil_count,(round(hsil_count/total*100,2)),
lsil_count,(round(lsil_count/total*100,2)),
scc_count,(round(scc_count/total*100,2)),
neg_count,(round(neg_count/total*100,2))))
```

```
The count of samples in each class is:

HSIL:130(16.91%)

LSIL:91(11.83%)

SCC:59(7.67%)

NEG:489(63.59%)
```

In [13]:
```
list_foders=os.listdir(rootpath)
path_list=[]
for fold in list_foders:
    path_list.append(os.path.join(rootpath,fold))

hsil=path_list[0]
lsil=path_list[1]
neg=path_list[2]
scc=path_list[3]
```

In [14]:
```
req_datapoints=5000
```

In [15]:
```
Image_Oversample(hsil,round(req_datapoints/4))
```

```
Initialised with 130 image(s) found.
Output directory set to K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Pap_Smear\High_squamous_intra-epithelial_lesion\output.
Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=2048x1536 at 0x276C705A700>: 100%|| 1250/1250 [02:37
```

In [16]:
```
Image_Oversample(scc,round(req_datapoints/4))
```

```
Initialised with 59 image(s) found.
Output directory set to K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Pap_Smear\Squamous_cell_carcinoma\output.
Processing <PIL.Image.Image image mode=RGB size=2048x1536 at 0x276B0EBB2B0>: 100%|| 1250/1250 [02:44<00:00,  7.61 Samp
```

In [17]:
```
Image_Oversample(lsil,round(req_datapoints/4))
```

```
Initialised with 91 image(s) found.
Output directory set to K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Pap_Smear\Low_squamous_intra-epithelial_lesion\output.
Processing <PIL.Image.Image image mode=RGB size=2048x1536 at 0x276AEB68D00>: 100%|| 1250/1250 [02:36<00:00,  7.97 Samp
```

In [18]:
```
Image_Oversample(neg,round(req_datapoints/4))
```

```
Initialised with 489 image(s) found.
Output directory set to K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel2.3\Pap_Smear\N_E_G\output.
Processing <PIL.Image.Image image mode=RGB size=2048x1536 at 0x276AEB35CD0>: 100%|| 1250/1250 [02:35<00:00,  8.05 Samp
```

In [19]:
```
trn_img, trn_label = _FilepathLabelGenerator_Trn(rootpath)
data_train=pd.DataFrame({"train_imgpath":trn_img,"train_label":trn_label})
```

In [20]:
```
##Training and Validation Split
ds_trn, ds_val = train_test_split(data_train,  test_size= 0.25, shuffle= True, random_state= 42, stratify= data_train['train_label'])
ds_trn
```

Out[20]:

	train_imgpath	train_label
423	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
4941	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	SCC
771	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	HSIL
1868	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
3115	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
...	...	...
1400	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
2791	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	NEG
2391	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
2480	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL
2227	K:\2023\NEU\IE7615_NNDL\NNDL_Project\BaseModel...	LSIL

3750 rows × 2 columns

In [21]:
```
ds_val, ds_tst = train_test_split(ds_val,  test_size= 0.4, shuffle= True, random_state= 42, stratify= ds_val['train_label'])
```

DATA GENERATOR FUNCTION ImageDataGenerator (Run Only)

In [22]:
```
# Define the image size and input shape
image_height = 128
image_width = 128

# Define the batch size
batch_size = 64

image_size = (image_height, image_width)
input_shape = (image_height, image_width, 3)
```

In [23]:
```
# Create an instance of ImageDataGenerator for Training Set Only

datagen_trn = ImageDataGenerator(rescale=1./255)
                                 #horizontal_flip=True,
                                 #vertical_flip=True)
                                 #height_shift_range=0.0,
                                 #rotation_range=180,
                                 #brightness_range=[0.5,1],
                                 #fill_mode='wrap',)

ds_trn = datagen_trn.flow_from_dataframe(ds_trn,
                                     x_col='train_imgpath',
                                     y_col='train_label',
                                     target_size=image_size,  # Resize images to the specified size
                                     class_mode='categorical',  # For multi-class classification
                                     color_mode='rgb',
                                     shuffle=False,  # Disable shuffling to maintain order
                                     batch_size=batch_size,
                                     keep_aspect_ratio=True)
```

```
Found 3750 validated image filenames belonging to 4 classes.
```

In [24]:
```
# Create a separate instance of ImageDataGenerator for Validation and Test Sets

datagen_tst = ImageDataGenerator(rescale=1./255)
```

```
                                #height_shift_range=0.0,
                                #rotation_range=180,
                                #brightness_range=[0.5,1],
                                #fill_mode='wrap',)

#For Validation Set
ds_val=datagen_tst.flow_from_dataframe(ds_val,
                                       x_col='train_imgpath',
                                       y_col='train_label',
                                       target_size=image_size,  # Resize images to the specified size
                                       class_mode='categorical',  # For multi-class classification
                                       color_mode='rgb',
                                       shuffle=True,  # Disable shuffling to maintain order
                                       batch_size=batch_size,
                                       keep_aspect_ratio=True)
```

```
Found 750 validated image filenames belonging to 4 classes.
```

In [25]:
```
ds_tst=datagen_tst.flow_from_dataframe(ds_tst,
                                       x_col='train_imgpath',
                                       y_col='train_label',
                                       target_size=image_size,  # Resize images to the specified size
                                       class_mode='categorical',  # For multi-class classification
                                       color_mode='rgb',
                                       shuffle=False,  # Disable shuffling to maintain order
                                       batch_size=batch_size,
                                       keep_aspect_ratio=True)
```

```
Found 500 validated image filenames belonging to 4 classes.
```

In [26]:
```
ds_tst_Og=datagen_tst.flow_from_dataframe(ds_tst_Og,
                                       x_col='imagepath',
                                       y_col='label',
                                       target_size=image_size,  # Resize images to the specified size
                                       class_mode='categorical',  # For multi-class classification
                                       color_mode='rgb',
                                       shuffle=False,  # Disable shuffling to maintain order
                                       batch_size=batch_size,
                                       keep_aspect_ratio=True)
```

```
Found 193 validated image filenames belonging to 4 classes.
```

SHOW IMAGES FUNCTION (Run Only)

In [27]:
```
def show_sampleImages(data,batchsize=25):
    # return classes , images to be displayed
    # data.class_indices is a dictionary
    classes = list(data.class_indices.keys())     # defines list of dictionary's kays (classes), classes names : string
    images, labels = next(data)        # get a batch size samples from the datagenerator

    # calculate number of displayed samples
    length = len(labels)        # length of batch size
    sample = min(length, batchsize)    # check if sample less than 25 images

    plt.figure(figsize= (round(batchsize*2), round(batchsize*2)))
    m=0
    b=batchsize
    while b>1:
        m+=1
        b=b/2

    for i in range(sample):
        plt.subplot(m, m, i + 1)
        image = images[i]
        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= m*6)
        plt.axis('off')

    plt.show()
```

CALLBACK FUNCTION (Run Only)

In [28]:
```
# Define CallBack
class FinalCallBack(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        print("\n****************\n\nThe average loss for epoch {} is{:7.3f} and accuracy is{:7.3f}.".format(epoch+1, logs["loss"], logs["accuracy"]),
              "\n\n***************\n")

EarlyStopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                 mode='max',
                                                 min_delta = 0.01,
                                                 baseline = 0.5,
                                                 patience = 10,
                                                 restore_best_weights = True)
```

CONFUSION MATRIX (Run Only)

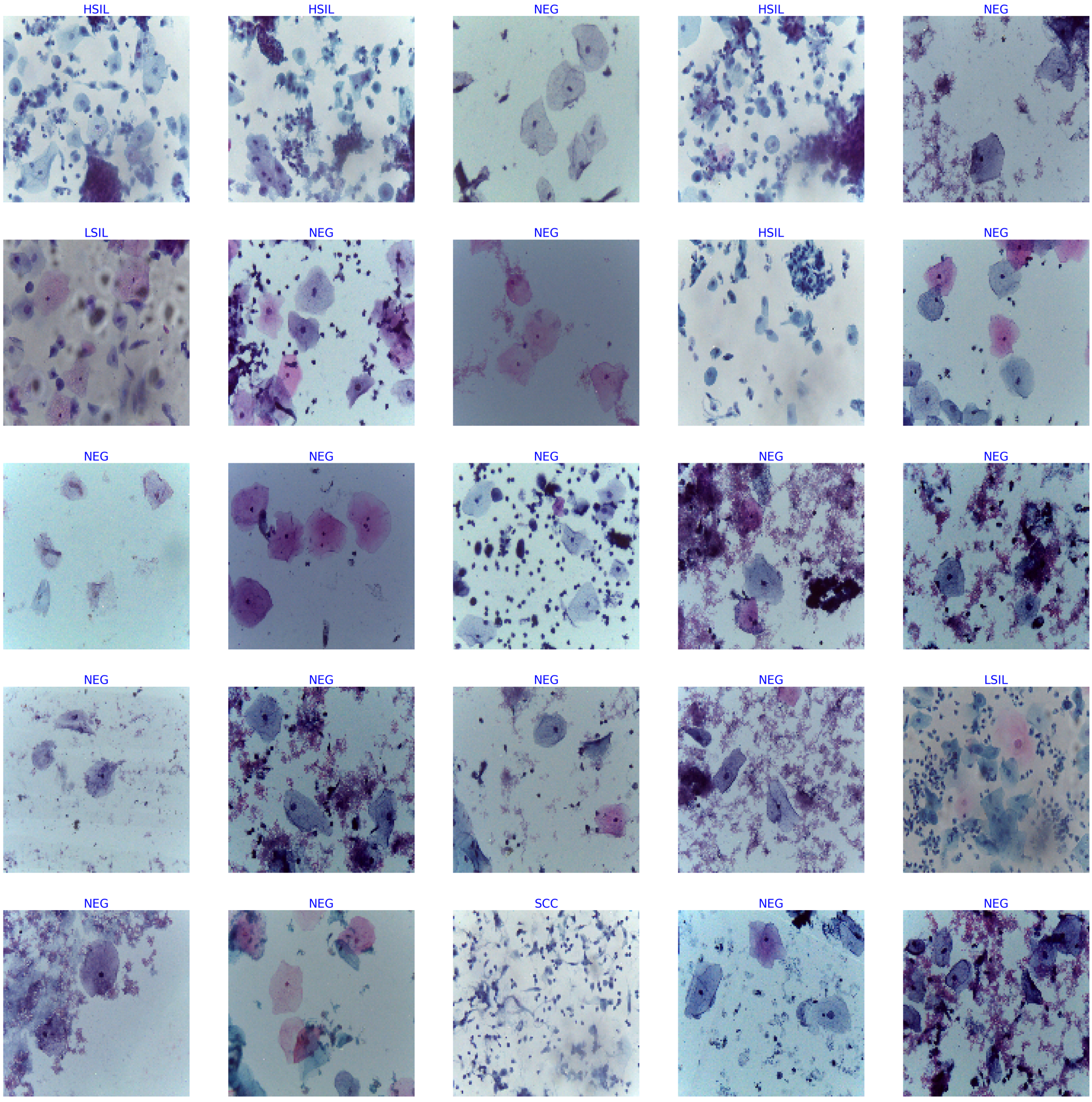
In [29]:
```
##Unnormalized and Normalized Confusion Matric Plots

def confusion_matrix_num(model, data):
    preds = model.predict_generator(data)
    y_pred = np.argmax(preds, axis = 1)

    temp_dict = {0: 'HSIL', 1: 'LSIL', 2: 'NEG', 3: 'SCC'}

    y_p = [temp_dict[k] for k in y_pred]
    y_t = [temp_dict[k] for k in data.classes]

    skplt.metrics.plot_confusion_matrix(y_t,
                                        y_p,
                                        normalize = False,
                                        title = "Count-wise Confusion Matrix",
                                        figsize = (10, 10))

def confusion_matrix_norm(model, data):
    preds = model.predict_generator(data)
    y_pred = np.argmax(preds, axis = 1)

    temp_dict = {0: 'HSIL', 1: 'LSIL', 2: 'NEG', 3: 'SCC'}


    y_p = [temp_dict[k] for k in y_pred]
    y_t = [temp_dict[k] for k in data.classes]

    skplt.metrics.plot_confusion_matrix(y_t,
                                        y_p,
                                        normalize = True,
                                        title = "Normalized Confusion Matrix",
                                        figsize = (10, 10))
```

PLOTTING DURING EPOCHS (Run Only)

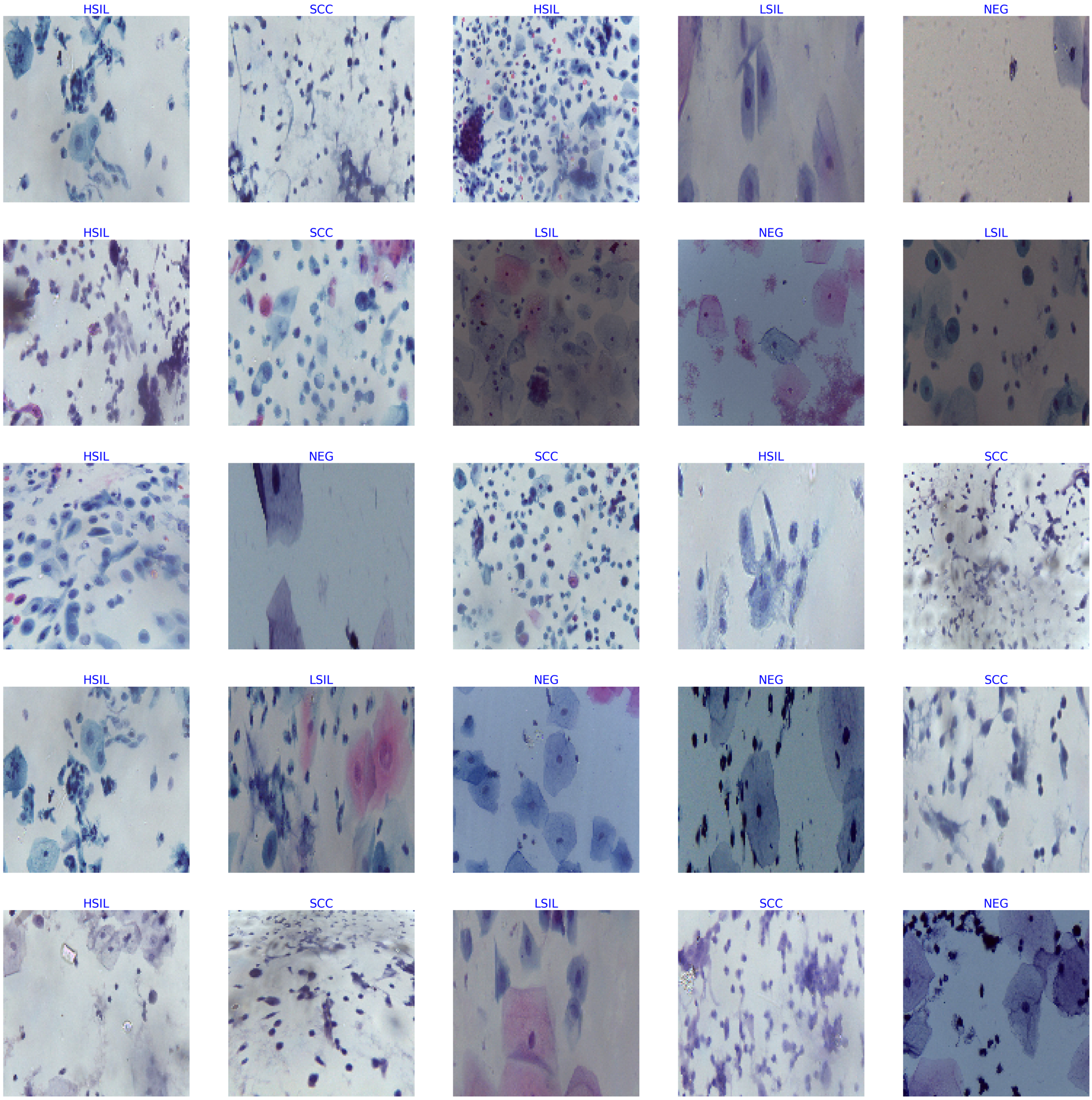
In [30]:
```
def plot_training(hist):
    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_loss = hist.history['loss']
    val_acc = hist.history['val_accuracy']
    val_loss = hist.history['val_loss']
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]
    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, 'orange', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'steelblue', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(Epochs, tr_acc, 'orange', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'steelblue', 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()
```

Checking Images

In [32]:
```
show_sampleImages(ds_tst_Og)
```



In [34]:
```
show_sampleImages(ds_trn)
```



MODEL

In [41]:
```
# Create the Sequential model with modified architecture
CNN_H2 = tf.keras.Sequential([
    #for equal regularization
    tf.keras.layers.SeparableConv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation="tanh", input_shape=input_shape),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation="relu"),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.SeparableConv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    tf.keras.layers.SpatialDropout2D(rate=0.1),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='elu'),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
], name='CNNModel')

# Compile the model with the modified optimizer
#sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
CNN_H2.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])

CNN_H2.summary()
```

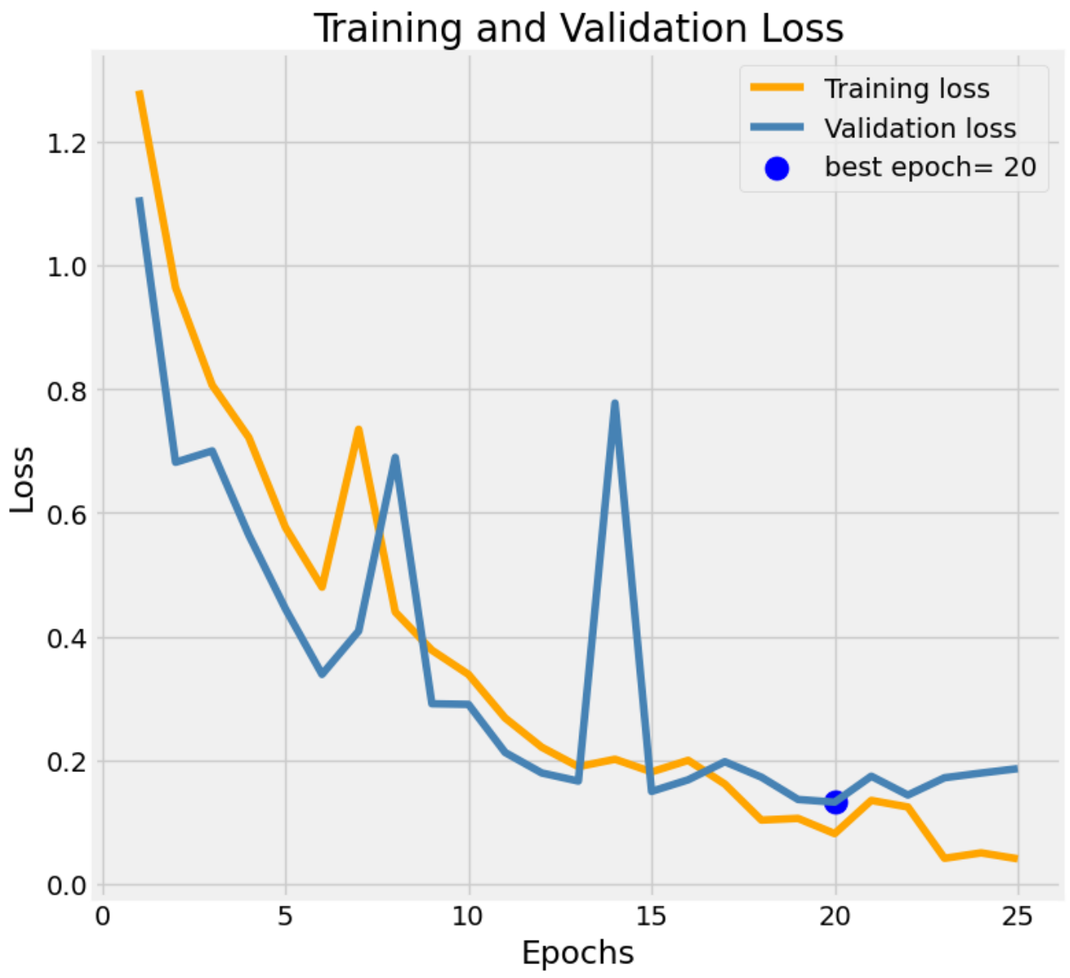
```
Model: "CNNModel"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 separable_conv2d_6 (Separab  (None, 126, 126, 32)     155
 leConv2D)

 max_pooling2d_16 (MaxPoolin  (None, 63, 63, 32)       0
 g2D)

 conv2d_10 (Conv2D)          (None, 61, 61, 64)        18496

 max_pooling2d_17 (MaxPoolin  (None, 30, 30, 64)       0
 g2D)

 separable_conv2d_7 (Separab  (None, 30, 30, 128)      8896
 leConv2D)

 max_pooling2d_18 (MaxPoolin  (None, 15, 15, 128)      0
 g2D)

 spatial_dropout2d_4 (Spatia  (None, 15, 15, 128)      0
 lDropout2D)

 conv2d_11 (Conv2D)          (None, 15, 15, 128)       147584

 max_pooling2d_19 (MaxPoolin  (None, 8, 8, 128)        0
 g2D)

 flatten_4 (Flatten)         (None, 8192)              0

 dense_12 (Dense)            (None, 128)               1048704

 dropout_3 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 32)                4128

 dense_14 (Dense)            (None, 4)                 132

=================================================================
Total params: 1,228,095
Trainable params: 1,228,095
Non-trainable params: 0
_________________________________________________________________
```

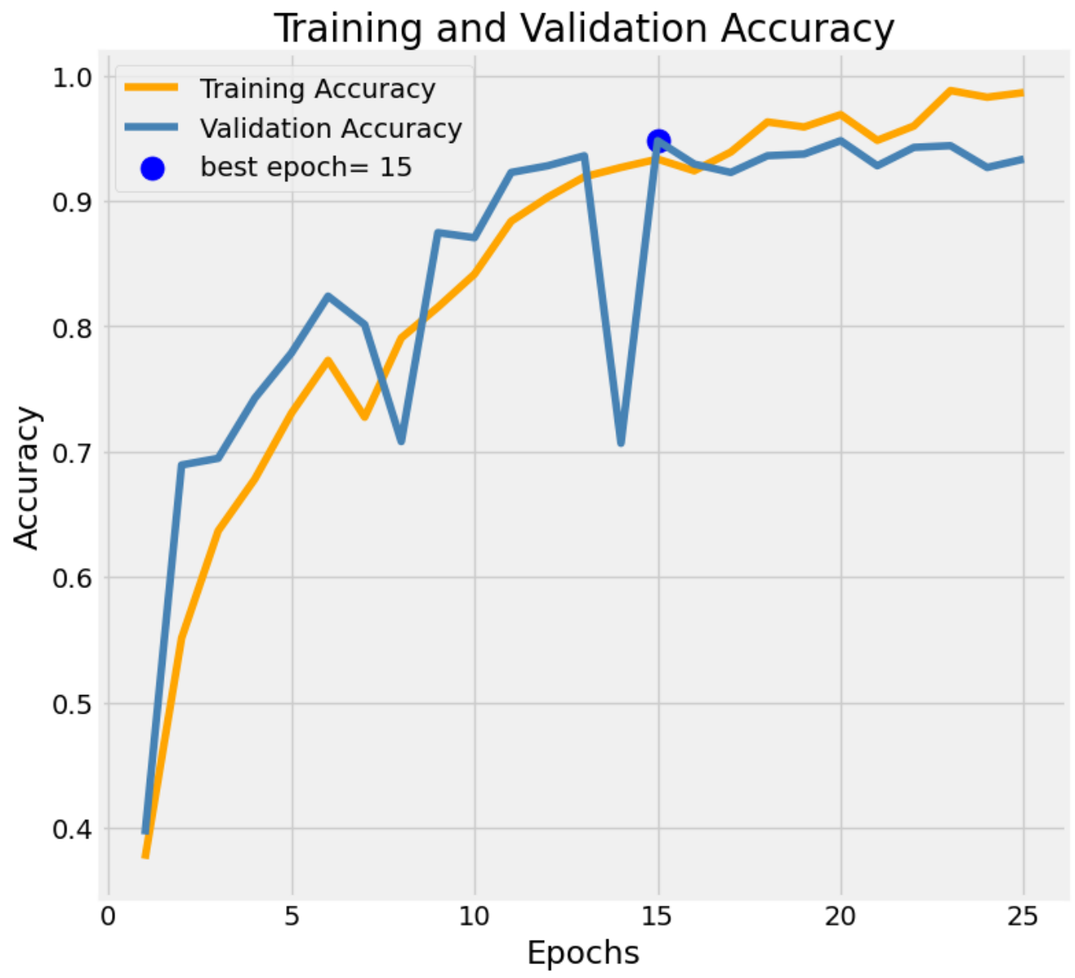
In [42]:
```
results_H2 = CNN_H2.fit(ds_trn,
                  epochs=50,
                  batch_size = 64,
                  verbose= 1,
                  callbacks=[FinalCallBack(), EarlyStopping],
                  validation_data= ds_val)
```

```
Epoch 1/50
59/59 [==============================] - ETA: 0s - loss: 1.2829 - accuracy: 0.3752
***************

The average loss for epoch 1 is  1.283 and accuracy is  0.375.

**************

59/59 [==============================] - 160s 3s/step - loss: 1.2829 - accuracy: 0.3752 - val_loss: 1.1103 - val_accuracy: 0.3947
Epoch 2/50
59/59 [==============================] - ETA: 0s - loss: 0.9645 - accuracy: 0.5512
***************

The average loss for epoch 2 is  0.965 and accuracy is  0.551.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.9645 - accuracy: 0.5512 - val_loss: 0.6817 - val_accuracy: 0.6893
Epoch 3/50
59/59 [==============================] - ETA: 0s - loss: 0.8066 - accuracy: 0.6368
***************

The average loss for epoch 3 is  0.807 and accuracy is  0.637.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.8066 - accuracy: 0.6368 - val_loss: 0.7002 - val_accuracy: 0.6947
Epoch 4/50
59/59 [==============================] - ETA: 0s - loss: 0.7219 - accuracy: 0.6781
***************

The average loss for epoch 4 is  0.722 and accuracy is  0.678.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.7219 - accuracy: 0.6781 - val_loss: 0.5641 - val_accuracy: 0.7427
Epoch 5/50
59/59 [==============================] - ETA: 0s - loss: 0.5765 - accuracy: 0.7307
***************

The average loss for epoch 5 is  0.577 and accuracy is  0.731.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.5765 - accuracy: 0.7307 - val_loss: 0.4448 - val_accuracy: 0.7787
Epoch 6/50
59/59 [==============================] - ETA: 0s - loss: 0.4799 - accuracy: 0.7728
***************

The average loss for epoch 6 is  0.480 and accuracy is  0.773.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.4799 - accuracy: 0.7728 - val_loss: 0.3388 - val_accuracy: 0.8240
Epoch 7/50
59/59 [==============================] - ETA: 0s - loss: 0.7351 - accuracy: 0.7275
***************

The average loss for epoch 7 is  0.735 and accuracy is  0.727.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.7351 - accuracy: 0.7275 - val_loss: 0.4089 - val_accuracy: 0.8013
Epoch 8/50
59/59 [==============================] - ETA: 0s - loss: 0.4397 - accuracy: 0.7907
***************

The average loss for epoch 8 is  0.440 and accuracy is  0.791.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.4397 - accuracy: 0.7907 - val_loss: 0.6897 - val_accuracy: 0.7080
Epoch 9/50
59/59 [==============================] - ETA: 0s - loss: 0.3779 - accuracy: 0.8152
***************

The average loss for epoch 9 is  0.378 and accuracy is  0.815.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.3779 - accuracy: 0.8152 - val_loss: 0.2913 - val_accuracy: 0.8747
Epoch 10/50
59/59 [==============================] - ETA: 0s - loss: 0.3387 - accuracy: 0.8416
***************

The average loss for epoch 10 is  0.339 and accuracy is  0.842.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.3387 - accuracy: 0.8416 - val_loss: 0.2902 - val_accuracy: 0.8707
Epoch 11/50
59/59 [==============================] - ETA: 0s - loss: 0.2682 - accuracy: 0.8837
***************

The average loss for epoch 11 is  0.268 and accuracy is  0.884.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.2682 - accuracy: 0.8837 - val_loss: 0.2125 - val_accuracy: 0.9227
Epoch 12/50
59/59 [==============================] - ETA: 0s - loss: 0.2210 - accuracy: 0.9029
***************

The average loss for epoch 12 is  0.221 and accuracy is  0.903.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.2210 - accuracy: 0.9029 - val_loss: 0.1793 - val_accuracy: 0.9280
Epoch 13/50
59/59 [==============================] - ETA: 0s - loss: 0.1896 - accuracy: 0.9192
***************

The average loss for epoch 13 is  0.190 and accuracy is  0.919.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1896 - accuracy: 0.9192 - val_loss: 0.1662 - val_accuracy: 0.9360
Epoch 14/50
59/59 [==============================] - ETA: 0s - loss: 0.2015 - accuracy: 0.9267
***************

The average loss for epoch 14 is  0.202 and accuracy is  0.927.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.2015 - accuracy: 0.9267 - val_loss: 0.7775 - val_accuracy: 0.7067
```

```
Epoch 15/50
59/59 [==============================] - ETA: 0s - loss: 0.1811 - accuracy: 0.9333
***************

The average loss for epoch 15 is  0.181 and accuracy is  0.933.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.1811 - accuracy: 0.9333 - val_loss: 0.1495 - val_accuracy: 0.9480
Epoch 16/50
59/59 [==============================] - ETA: 0s - loss: 0.1998 - accuracy: 0.9240
***************

The average loss for epoch 16 is  0.200 and accuracy is  0.924.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1998 - accuracy: 0.9240 - val_loss: 0.1683 - val_accuracy: 0.9293
Epoch 17/50
59/59 [==============================] - ETA: 0s - loss: 0.1614 - accuracy: 0.9389
***************

The average loss for epoch 17 is  0.161 and accuracy is  0.939.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1614 - accuracy: 0.9389 - val_loss: 0.1975 - val_accuracy: 0.9227
Epoch 18/50
59/59 [==============================] - ETA: 0s - loss: 0.1031 - accuracy: 0.9629
***************

The average loss for epoch 18 is  0.103 and accuracy is  0.963.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1031 - accuracy: 0.9629 - val_loss: 0.1725 - val_accuracy: 0.9360
Epoch 19/50
59/59 [==============================] - ETA: 0s - loss: 0.1058 - accuracy: 0.9589
***************

The average loss for epoch 19 is  0.106 and accuracy is  0.959.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.1058 - accuracy: 0.9589 - val_loss: 0.1364 - val_accuracy: 0.9373
Epoch 20/50
59/59 [==============================] - ETA: 0s - loss: 0.0812 - accuracy: 0.9688
***************

The average loss for epoch 20 is  0.081 and accuracy is  0.969.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0812 - accuracy: 0.9688 - val_loss: 0.1321 - val_accuracy: 0.9480
Epoch 21/50
59/59 [==============================] - ETA: 0s - loss: 0.1349 - accuracy: 0.9483
***************

The average loss for epoch 21 is  0.135 and accuracy is  0.948.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.1349 - accuracy: 0.9483 - val_loss: 0.1741 - val_accuracy: 0.9280
Epoch 22/50
59/59 [==============================] - ETA: 0s - loss: 0.1243 - accuracy: 0.9597
***************

The average loss for epoch 22 is  0.124 and accuracy is  0.960.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.1243 - accuracy: 0.9597 - val_loss: 0.1436 - val_accuracy: 0.9427
Epoch 23/50
59/59 [==============================] - ETA: 0s - loss: 0.0414 - accuracy: 0.9880
***************

The average loss for epoch 23 is  0.041 and accuracy is  0.988.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0414 - accuracy: 0.9880 - val_loss: 0.1716 - val_accuracy: 0.9440
Epoch 24/50
59/59 [==============================] - ETA: 0s - loss: 0.0501 - accuracy: 0.9827
***************

The average loss for epoch 24 is  0.050 and accuracy is  0.983.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0501 - accuracy: 0.9827 - val_loss: 0.1792 - val_accuracy: 0.9267
Epoch 25/50
59/59 [==============================] - ETA: 0s - loss: 0.0405 - accuracy: 0.9864
***************

The average loss for epoch 25 is  0.040 and accuracy is  0.986.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0405 - accuracy: 0.9864 - val_loss: 0.1863 - val_accuracy: 0.9333
```

In [43]:
```
plot_training(results_H2)
```





In [44]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = CNN_H2.evaluate(ds_tst)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))

# Generate classification report for the test data
y_pred = CNN_H2.predict(ds_tst)
y_pred_classes = np.argmax(y_pred, axis=1)
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst.classes, y_pred_classes, target_names=class_names))
```

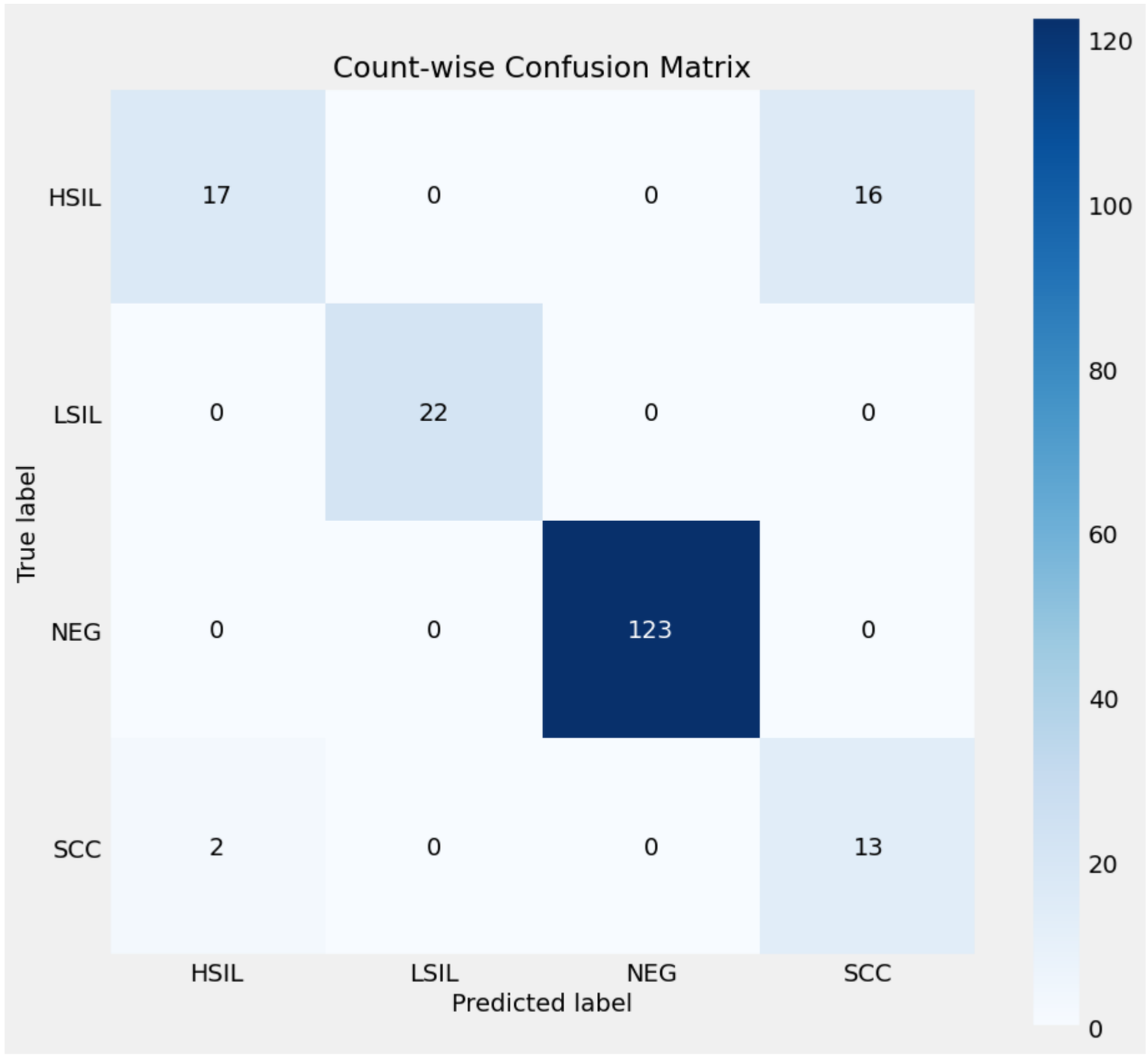
```
8/8 [==============================] - 14s 2s/step - loss: 0.1961 - accuracy: 0.9160
Test Accuracy: 91.60%
              precision    recall  f1-score   support

        HSIL       0.78      0.94      0.86       125
        LSIL       0.99      1.00      1.00       125
         NEG       1.00      0.97      0.98       125
         SCC       0.92      0.75      0.83       125

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500
```

In [47]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = CNN_H2.evaluate(ds_tst_Og)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))


# Generate classification report for the test dataz
y_pred = CNN_H2.predict(ds_tst_Og)
y_pred_classes = np.argmax(y_pred, axis=1)
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst_Og.classes, y_pred_classes, target_names=class_names))
```

```
4/4 [==============================] - 13s 3s/step - loss: 0.1207 - accuracy: 0.9637
Test Accuracy: 96.37%
              precision    recall  f1-score   support

        HSIL       0.91      0.91      0.91        33
        LSIL       0.96      1.00      0.98        22
         NEG       1.00      0.99      1.00       123
         SCC       0.80      0.80      0.80        15

    accuracy                           0.96       193
   macro avg       0.92      0.93      0.92       193
weighted avg       0.96      0.96      0.96       193
```

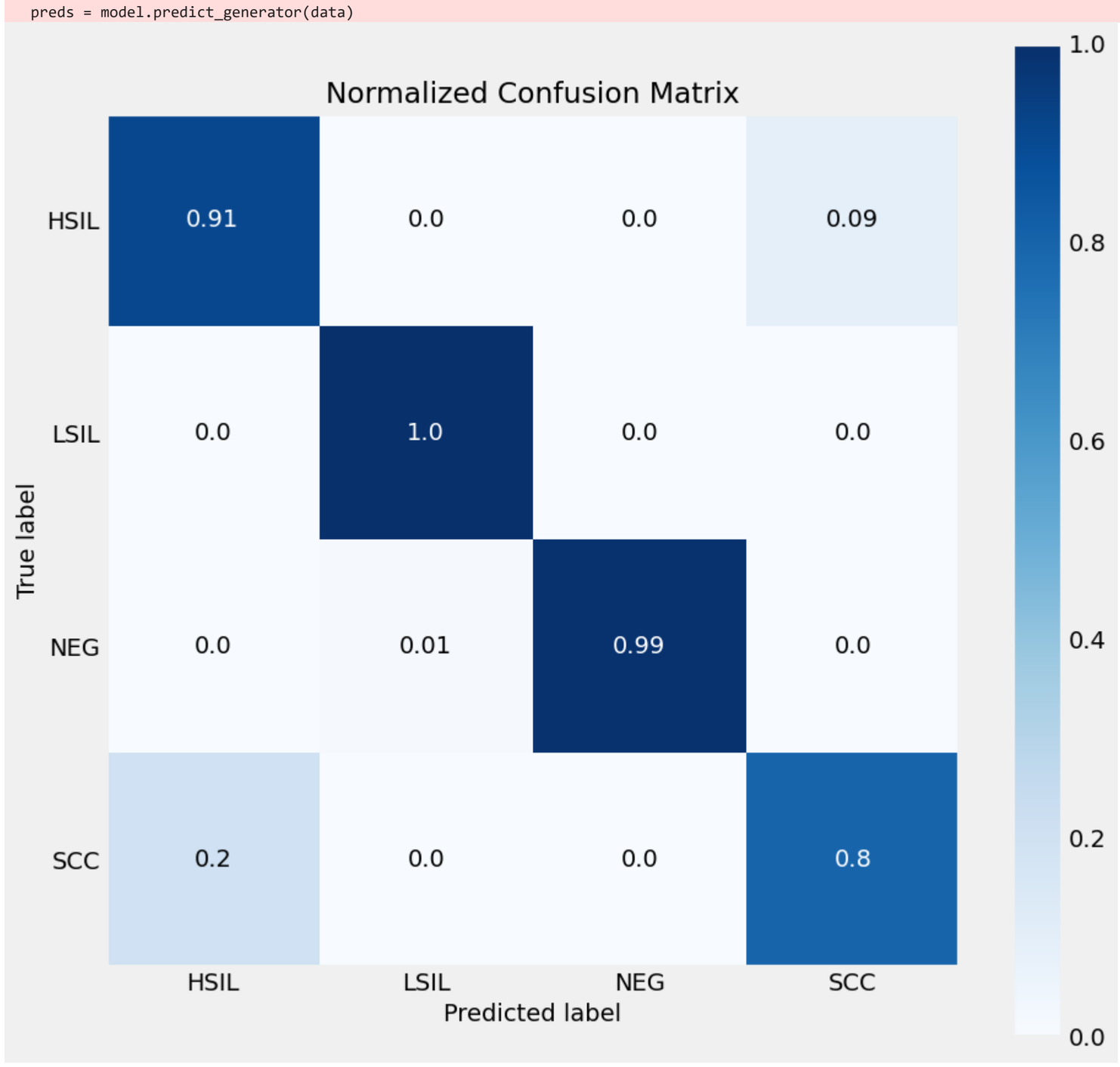
In [168…
```
confusion_matrix_num(CNN_H2, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:4: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, whi
ch supports generators.
  preds = model.predict_generator(data)
```



In [48]:
```
confusion_matrix_norm(CNN_H2, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:19: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, wh
ich supports generators.
  preds = model.predict_generator(data)
```



In [49]:
```
CNN_A1 = tf.keras.Sequential([
    #for equal regularization
    tf.keras.layers.SeparableConv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation="tanh", input_shape=input_shape),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    tf.keras.layers.SpatialDropout2D(rate=0.1),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='elu'),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
], name='CNNModel')

# Compile the model with the modified optimizer
#sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
CNN_A1.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
CNN_A1.summary()
```

```
Model: "CNNModel"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 separable_conv2d_8 (Separab  (None, 126, 126, 64)     283
 leConv2D)

 max_pooling2d_20 (MaxPoolin  (None, 63, 63, 64)       0
 g2D)

 conv2d_12 (Conv2D)          (None, 63, 63, 128)       73856

 max_pooling2d_21 (MaxPoolin  (None, 32, 32, 128)      0
 g2D)

 spatial_dropout2d_5 (Spatia  (None, 32, 32, 128)      0
 lDropout2D)

 conv2d_13 (Conv2D)          (None, 32, 32, 128)       147584

 max_pooling2d_22 (MaxPoolin  (None, 16, 16, 128)      0
 g2D)

 flatten_5 (Flatten)         (None, 32768)             0

 dense_15 (Dense)            (None, 128)               4194432

 dropout_4 (Dropout)         (None, 128)               0

 dense_16 (Dense)            (None, 32)                4128

 dense_17 (Dense)            (None, 4)                 132

=================================================================
Total params: 4,420,415
Trainable params: 4,420,415
Non-trainable params: 0
_________________________________________________________________
```

In [51]:
```
results_A1 = CNN_A1.fit(ds_trn,
                  epochs=50,
                  batch_size = 64,
                  verbose= 1,
                  callbacks=[FinalCallBack(), EarlyStopping],
                  validation_data= ds_val)
```

```
Epoch 1/50
59/59 [==============================] - ETA: 0s - loss: 1.3436 - accuracy: 0.3525
***************

The average loss for epoch 1 is  1.344 and accuracy is  0.353.

**************

59/59 [==============================] - 103s 2s/step - loss: 1.3436 - accuracy: 0.3525 - val_loss: 0.9143 - val_accuracy: 0.6547
Epoch 2/50
59/59 [==============================] - ETA: 0s - loss: 0.8166 - accuracy: 0.6283
***************

The average loss for epoch 2 is  0.817 and accuracy is  0.628.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.8166 - accuracy: 0.6283 - val_loss: 0.5663 - val_accuracy: 0.7467
Epoch 3/50
59/59 [==============================] - ETA: 0s - loss: 0.6363 - accuracy: 0.6904
***************

The average loss for epoch 3 is  0.636 and accuracy is  0.690.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.6363 - accuracy: 0.6904 - val_loss: 0.5407 - val_accuracy: 0.7493
Epoch 4/50
59/59 [==============================] - ETA: 0s - loss: 0.5612 - accuracy: 0.7211
***************

The average loss for epoch 4 is  0.561 and accuracy is  0.721.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.5612 - accuracy: 0.7211 - val_loss: 0.4805 - val_accuracy: 0.7680
Epoch 5/50
59/59 [==============================] - ETA: 0s - loss: 0.5242 - accuracy: 0.7565
***************

The average loss for epoch 5 is  0.524 and accuracy is  0.757.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.5242 - accuracy: 0.7565 - val_loss: 0.4598 - val_accuracy: 0.7627
Epoch 6/50
59/59 [==============================] - ETA: 0s - loss: 0.4194 - accuracy: 0.8083
***************

The average loss for epoch 6 is  0.419 and accuracy is  0.808.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.4194 - accuracy: 0.8083 - val_loss: 0.4206 - val_accuracy: 0.7827
Epoch 7/50
59/59 [==============================] - ETA: 0s - loss: 0.2905 - accuracy: 0.8749
***************

The average loss for epoch 7 is  0.291 and accuracy is  0.875.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.2905 - accuracy: 0.8749 - val_loss: 0.3132 - val_accuracy: 0.8720
Epoch 8/50
59/59 [==============================] - ETA: 0s - loss: 0.5107 - accuracy: 0.8464
***************

The average loss for epoch 8 is  0.511 and accuracy is  0.846.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.5107 - accuracy: 0.8464 - val_loss: 0.7512 - val_accuracy: 0.6053
Epoch 9/50
59/59 [==============================] - ETA: 0s - loss: 0.3275 - accuracy: 0.8720
***************

The average loss for epoch 9 is  0.328 and accuracy is  0.872.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.3275 - accuracy: 0.8720 - val_loss: 0.3237 - val_accuracy: 0.8587
Epoch 10/50
59/59 [==============================] - ETA: 0s - loss: 0.1992 - accuracy: 0.9179
***************

The average loss for epoch 10 is  0.199 and accuracy is  0.918.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1992 - accuracy: 0.9179 - val_loss: 0.3386 - val_accuracy: 0.8587
Epoch 11/50
59/59 [==============================] - ETA: 0s - loss: 0.1259 - accuracy: 0.9512
***************

The average loss for epoch 11 is  0.126 and accuracy is  0.951.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.1259 - accuracy: 0.9512 - val_loss: 0.3994 - val_accuracy: 0.8653
Epoch 12/50
59/59 [==============================] - ETA: 0s - loss: 0.0985 - accuracy: 0.9605
***************

The average loss for epoch 12 is  0.099 and accuracy is  0.961.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0985 - accuracy: 0.9605 - val_loss: 0.5186 - val_accuracy: 0.8200
Epoch 13/50
59/59 [==============================] - ETA: 0s - loss: 0.0665 - accuracy: 0.9755
***************

The average loss for epoch 13 is  0.067 and accuracy is  0.975.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0665 - accuracy: 0.9755 - val_loss: 0.4541 - val_accuracy: 0.8453
Epoch 14/50
59/59 [==============================] - ETA: 0s - loss: 0.0442 - accuracy: 0.9867
***************

The average loss for epoch 14 is  0.044 and accuracy is  0.987.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0442 - accuracy: 0.9867 - val_loss: 0.4555 - val_accuracy: 0.8600
```

```
Epoch 15/50
59/59 [==============================] - ETA: 0s - loss: 0.0256 - accuracy: 0.9931
***************

The average loss for epoch 15 is  0.026 and accuracy is  0.993.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0256 - accuracy: 0.9931 - val_loss: 0.7085 - val_accuracy: 0.8320
Epoch 16/50
59/59 [==============================] - ETA: 0s - loss: 0.0164 - accuracy: 0.9968
***************

The average loss for epoch 16 is  0.016 and accuracy is  0.997.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0164 - accuracy: 0.9968 - val_loss: 0.6370 - val_accuracy: 0.8613
Epoch 17/50
59/59 [==============================] - ETA: 0s - loss: 0.0104 - accuracy: 0.9960
***************

The average loss for epoch 17 is  0.010 and accuracy is  0.996.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0104 - accuracy: 0.9960 - val_loss: 0.6485 - val_accuracy: 0.8560
```

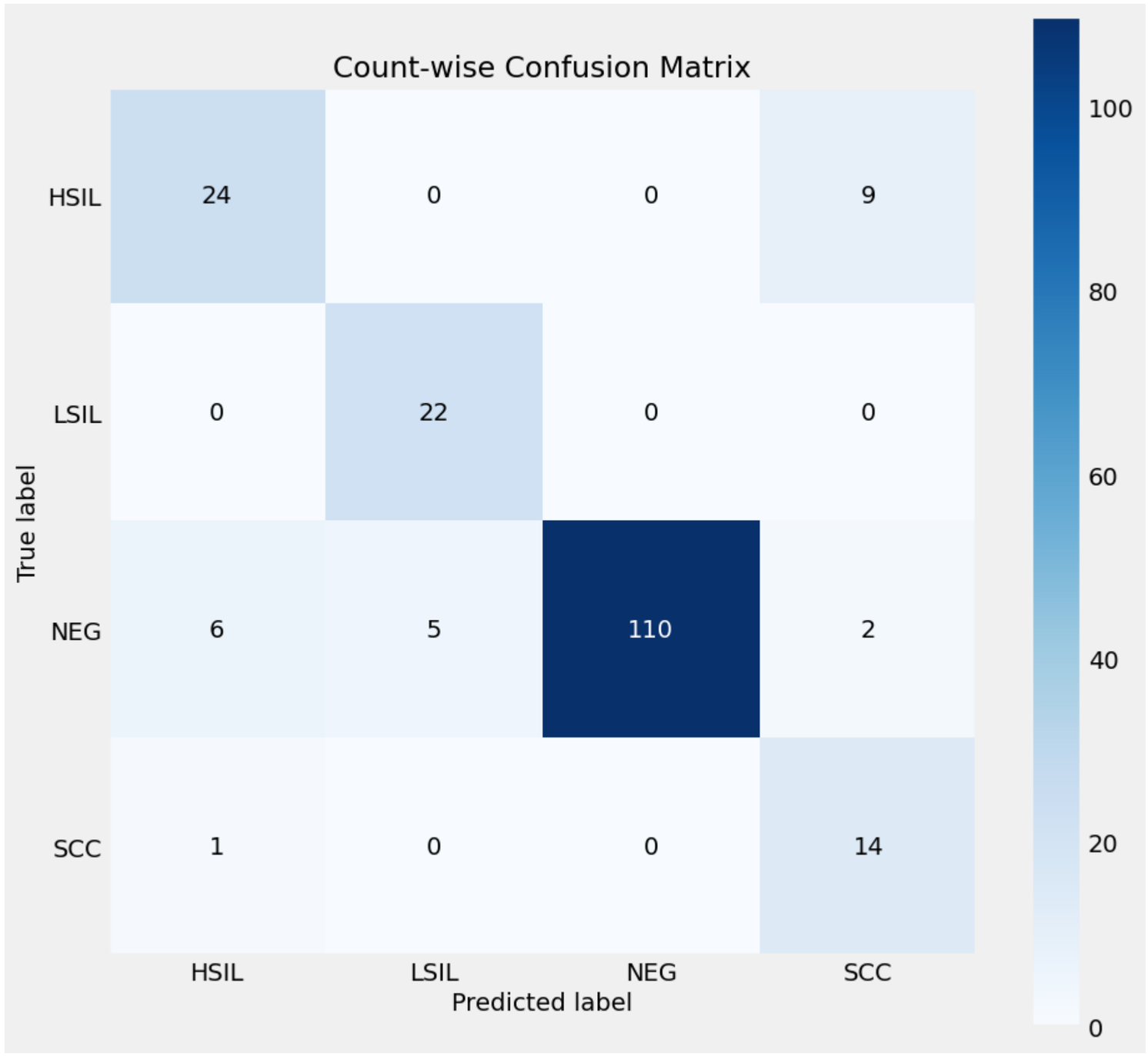
In [52]:
```
plot_training(results_sniA1)
```

In [83]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = CNN_A1.evaluate(ds_tst_Og)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))

# Generate classification report for the test data
y_pred = CNN_A1.predict(ds_tst_Og)
y_pred_classes = np.argmax(y_pred, axis=1)
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst_Og.classes, y_pred_classes, target_names=class_names))
```

```
4/4 [==============================] - 12s 3s/step - loss: 0.2863 - accuracy: 0.8808
Test Accuracy: 88.08%
              precision    recall  f1-score   support

        HSIL       0.77      0.73      0.75        33
        LSIL       0.81      1.00      0.90        22
         NEG       1.00      0.89      0.94       123
         SCC       0.56      0.93      0.70        15

    accuracy                           0.88       193
   macro avg       0.79      0.89      0.82       193
weighted avg       0.91      0.88      0.89       193
```

In [84]:
```
confusion_matrix_num(CNN_A1, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:4: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, whi
ch supports generators.
  preds = model.predict_generator(data)
```



H2 with Batch Normalization

In [64]:
```
CNN_H2N = tf.keras.Sequential([
    #for equal regularization
    tf.keras.layers.SeparableConv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation="tanh", input_shape=input_shape),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation="relu"),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    tf.keras.layers.SeparableConv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    BatchNormalization(),
    tf.keras.layers.SpatialDropout2D(rate=0.1),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides = (1,1), activation="relu",padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
    BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='elu'),
    BatchNormalization(),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(32,activation='relu'),
    BatchNormalization(),
    tf.keras.layers.Dense(4, activation='softmax')
], name='CNNModel')

# Compile the model with the modified optimizer
#sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
CNN_H2N.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])

CNN_H2N.summary()
```

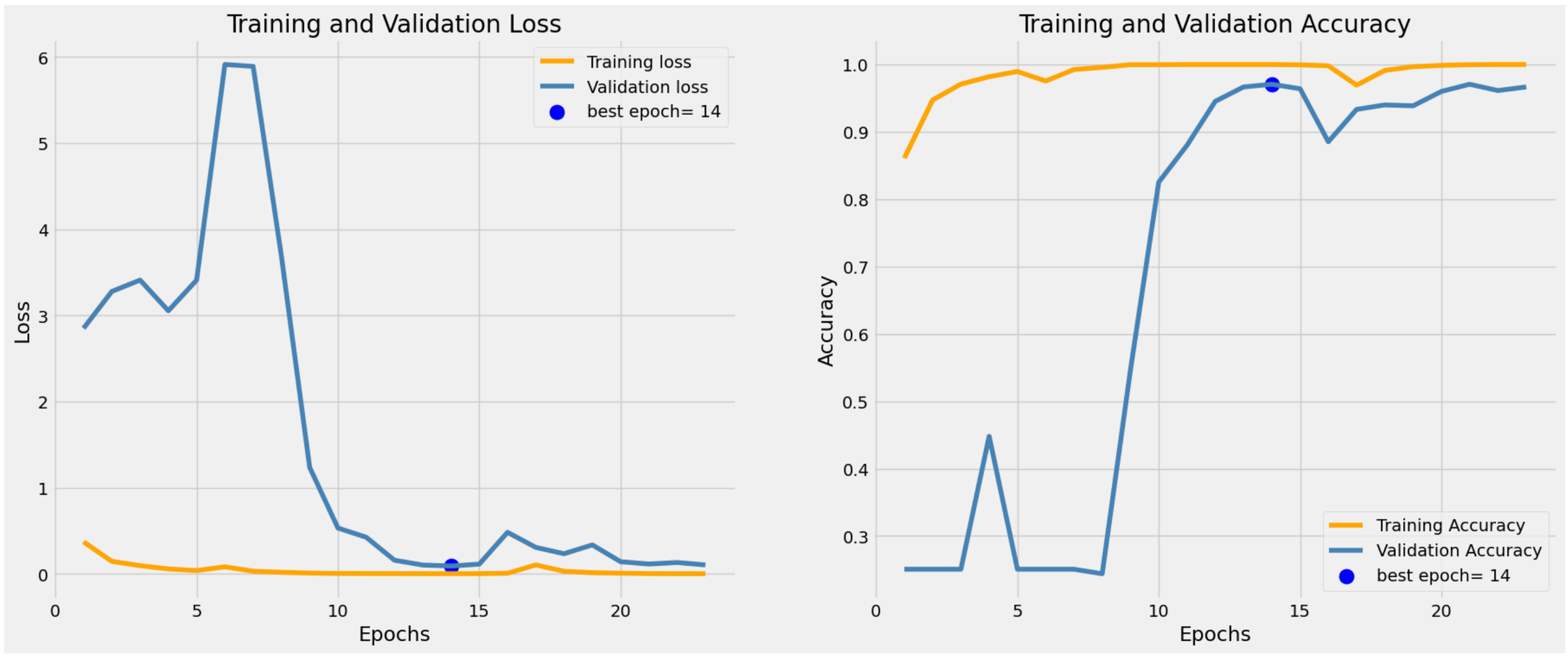
```
Model: "CNNModel"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 separable_conv2d_15 (Separa  (None, 126, 126, 32)     155
 bleConv2D)

 max_pooling2d_35 (MaxPoolin  (None, 63, 63, 32)       0
 g2D)

 batch_normalization_19 (Bat  (None, 63, 63, 32)       128
 chNormalization)

 conv2d_20 (Conv2D)          (None, 61, 61, 64)        18496

 max_pooling2d_36 (MaxPoolin  (None, 30, 30, 64)       0
 g2D)

 batch_normalization_20 (Bat  (None, 30, 30, 64)       256
 chNormalization)

 separable_conv2d_16 (Separa  (None, 30, 30, 128)      8896
 bleConv2D)

 max_pooling2d_37 (MaxPoolin  (None, 15, 15, 128)      0
 g2D)

 batch_normalization_21 (Bat  (None, 15, 15, 128)      512
 chNormalization)

 spatial_dropout2d_9 (Spatia  (None, 15, 15, 128)      0
 lDropout2D)

 conv2d_21 (Conv2D)          (None, 15, 15, 128)       147584

 max_pooling2d_38 (MaxPoolin  (None, 8, 8, 128)        0
 g2D)

 batch_normalization_22 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 flatten_9 (Flatten)         (None, 8192)              0

 dense_27 (Dense)            (None, 128)               1048704

 batch_normalization_23 (Bat  (None, 128)              512
 chNormalization)

 dropout_8 (Dropout)         (None, 128)               0

 dense_28 (Dense)            (None, 32)                4128

 batch_normalization_24 (Bat  (None, 32)               128
 chNormalization)

 dense_29 (Dense)            (None, 4)                 132

=================================================================
Total params: 1,230,143
Trainable params: 1,229,119
Non-trainable params: 1,024
_________________________________________________________________
```

In [65]:
```
results_H2N = CNN_H2N.fit(ds_trn,
                  epochs=50,
                  batch_size = 64,
                  verbose= 1,
                  callbacks=[FinalCallBack(), EarlyStopping],
                  validation_data= ds_val)
```

```
Epoch 1/50
59/59 [==============================] - ETA: 0s - loss: 0.3710 - accuracy: 0.8611
***************

The average loss for epoch 1 is  0.371 and accuracy is  0.861.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.3710 - accuracy: 0.8611 - val_loss: 2.8510 - val_accuracy: 0.2507
Epoch 2/50
59/59 [==============================] - ETA: 0s - loss: 0.1457 - accuracy: 0.9472
***************

The average loss for epoch 2 is  0.146 and accuracy is  0.947.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.1457 - accuracy: 0.9472 - val_loss: 3.2785 - val_accuracy: 0.2507
Epoch 3/50
59/59 [==============================] - ETA: 0s - loss: 0.0953 - accuracy: 0.9709
***************

The average loss for epoch 3 is  0.095 and accuracy is  0.971.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0953 - accuracy: 0.9709 - val_loss: 3.4107 - val_accuracy: 0.2507
Epoch 4/50
59/59 [==============================] - ETA: 0s - loss: 0.0575 - accuracy: 0.9819
***************

The average loss for epoch 4 is  0.058 and accuracy is  0.982.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0575 - accuracy: 0.9819 - val_loss: 3.0539 - val_accuracy: 0.4480
Epoch 5/50
59/59 [==============================] - ETA: 0s - loss: 0.0386 - accuracy: 0.9896
***************

The average loss for epoch 5 is  0.039 and accuracy is  0.990.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0386 - accuracy: 0.9896 - val_loss: 3.4106 - val_accuracy: 0.2507
Epoch 6/50
59/59 [==============================] - ETA: 0s - loss: 0.0820 - accuracy: 0.9755
***************

The average loss for epoch 6 is  0.082 and accuracy is  0.975.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.0820 - accuracy: 0.9755 - val_loss: 5.9131 - val_accuracy: 0.2507
Epoch 7/50
59/59 [==============================] - ETA: 0s - loss: 0.0311 - accuracy: 0.9925
***************

The average loss for epoch 7 is  0.031 and accuracy is  0.993.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0311 - accuracy: 0.9925 - val_loss: 5.8914 - val_accuracy: 0.2507
Epoch 8/50
59/59 [==============================] - ETA: 0s - loss: 0.0185 - accuracy: 0.9957
***************

The average loss for epoch 8 is  0.019 and accuracy is  0.996.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0185 - accuracy: 0.9957 - val_loss: 3.6880 - val_accuracy: 0.2440
Epoch 9/50
59/59 [==============================] - ETA: 0s - loss: 0.0095 - accuracy: 0.9997
***************

The average loss for epoch 9 is  0.010 and accuracy is  1.000.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0095 - accuracy: 0.9997 - val_loss: 1.2363 - val_accuracy: 0.5467
Epoch 10/50
59/59 [==============================] - ETA: 0s - loss: 0.0050 - accuracy: 0.9997
***************

The average loss for epoch 10 is  0.005 and accuracy is  1.000.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0050 - accuracy: 0.9997 - val_loss: 0.5323 - val_accuracy: 0.8253
Epoch 11/50
59/59 [==============================] - ETA: 0s - loss: 0.0041 - accuracy: 1.0000
***************

The average loss for epoch 11 is  0.004 and accuracy is  1.000.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0041 - accuracy: 1.0000 - val_loss: 0.4251 - val_accuracy: 0.8800
Epoch 12/50
59/59 [==============================] - ETA: 0s - loss: 0.0032 - accuracy: 1.0000
***************

The average loss for epoch 12 is  0.003 and accuracy is  1.000.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 0.1584 - val_accuracy: 0.9453
Epoch 13/50
59/59 [==============================] - ETA: 0s - loss: 0.0025 - accuracy: 1.0000
***************

The average loss for epoch 13 is  0.003 and accuracy is  1.000.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.1026 - val_accuracy: 0.9667
Epoch 14/50
59/59 [==============================] - ETA: 0s - loss: 0.0027 - accuracy: 1.0000
***************

The average loss for epoch 14 is  0.003 and accuracy is  1.000.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 0.0913 - val_accuracy: 0.9707
```

```
Epoch 15/50
59/59 [==============================] - ETA: 0s - loss: 0.0029 - accuracy: 0.9995
***************

The average loss for epoch 15 is  0.003 and accuracy is  0.999.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0029 - accuracy: 0.9995 - val_loss: 0.1133 - val_accuracy: 0.9640
Epoch 16/50
59/59 [==============================] - ETA: 0s - loss: 0.0074 - accuracy: 0.9981
***************

The average loss for epoch 16 is  0.007 and accuracy is  0.998.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0074 - accuracy: 0.9981 - val_loss: 0.4827 - val_accuracy: 0.8853
Epoch 17/50
59/59 [==============================] - ETA: 0s - loss: 0.1039 - accuracy: 0.9693
***************

The average loss for epoch 17 is  0.104 and accuracy is  0.969.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1039 - accuracy: 0.9693 - val_loss: 0.3074 - val_accuracy: 0.9333
Epoch 18/50
59/59 [==============================] - ETA: 0s - loss: 0.0301 - accuracy: 0.9912
***************

The average loss for epoch 18 is  0.030 and accuracy is  0.991.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0301 - accuracy: 0.9912 - val_loss: 0.2340 - val_accuracy: 0.9400
Epoch 19/50
59/59 [==============================] - ETA: 0s - loss: 0.0139 - accuracy: 0.9968
***************

The average loss for epoch 19 is  0.014 and accuracy is  0.997.

**************

59/59 [==============================] - 100s 2s/step - loss: 0.0139 - accuracy: 0.9968 - val_loss: 0.3363 - val_accuracy: 0.9387
Epoch 20/50
59/59 [==============================] - ETA: 0s - loss: 0.0081 - accuracy: 0.9987
***************

The average loss for epoch 20 is  0.008 and accuracy is  0.999.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0081 - accuracy: 0.9987 - val_loss: 0.1413 - val_accuracy: 0.9600
Epoch 21/50
59/59 [==============================] - ETA: 0s - loss: 0.0037 - accuracy: 0.9997
***************

The average loss for epoch 21 is  0.004 and accuracy is  1.000.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0037 - accuracy: 0.9997 - val_loss: 0.1146 - val_accuracy: 0.9707
Epoch 22/50
59/59 [==============================] - ETA: 0s - loss: 0.0020 - accuracy: 1.0000
***************

The average loss for epoch 22 is  0.002 and accuracy is  1.000.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.1320 - val_accuracy: 0.9613
Epoch 23/50
59/59 [==============================] - ETA: 0s - loss: 0.0017 - accuracy: 1.0000
***************

The average loss for epoch 23 is  0.002 and accuracy is  1.000.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.1040 - val_accuracy: 0.9667
```

In [67]:
```
plot_training(results_H2N)
```



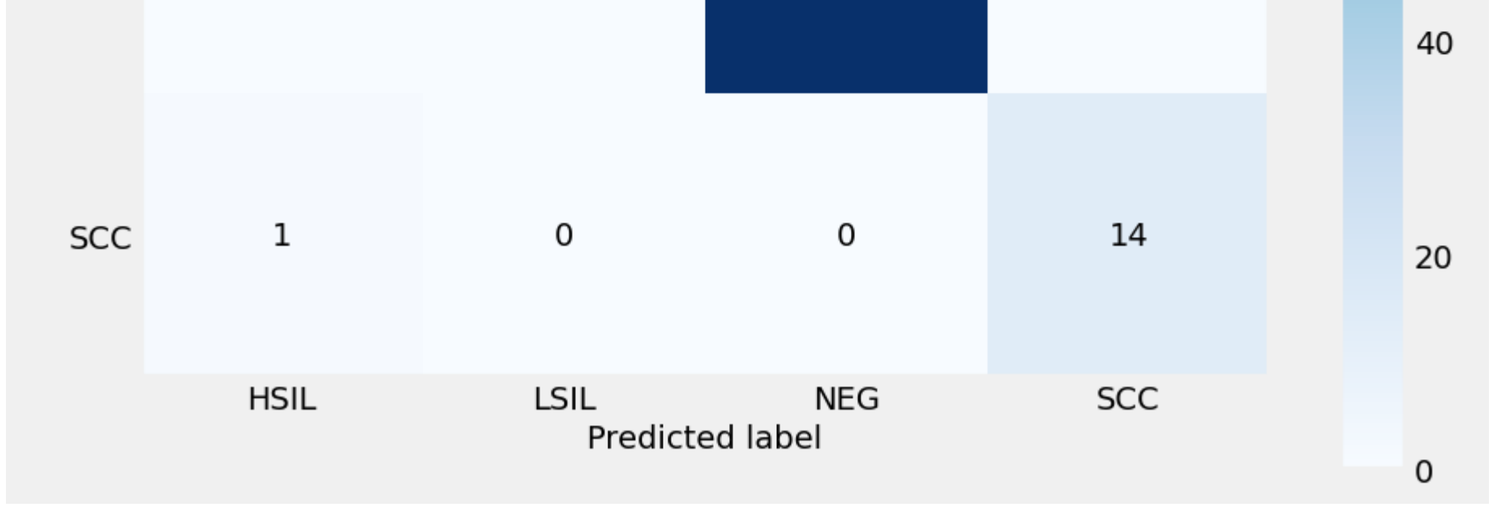
In [69]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = CNN_H2N.evaluate(ds_tst_Og)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))

# Generate classification report for the test data
y_pred = CNN_H2N.predict(ds_tst_Og)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst_Og.classes, y_pred_classes, target_names=class_names))
```

```
4/4 [==============================] - 11s 3s/step - loss: 0.0585 - accuracy: 0.9845
Test Accuracy: 98.45%
              precision    recall  f1-score   support

        HSIL       0.97      0.94      0.95        33
        LSIL       1.00      1.00      1.00        22
         NEG       1.00      1.00      1.00       123
         SCC       0.88      0.93      0.90        15

    accuracy                           0.98       193
   macro avg       0.96      0.97      0.96       193
weighted avg       0.98      0.98      0.98       193
```

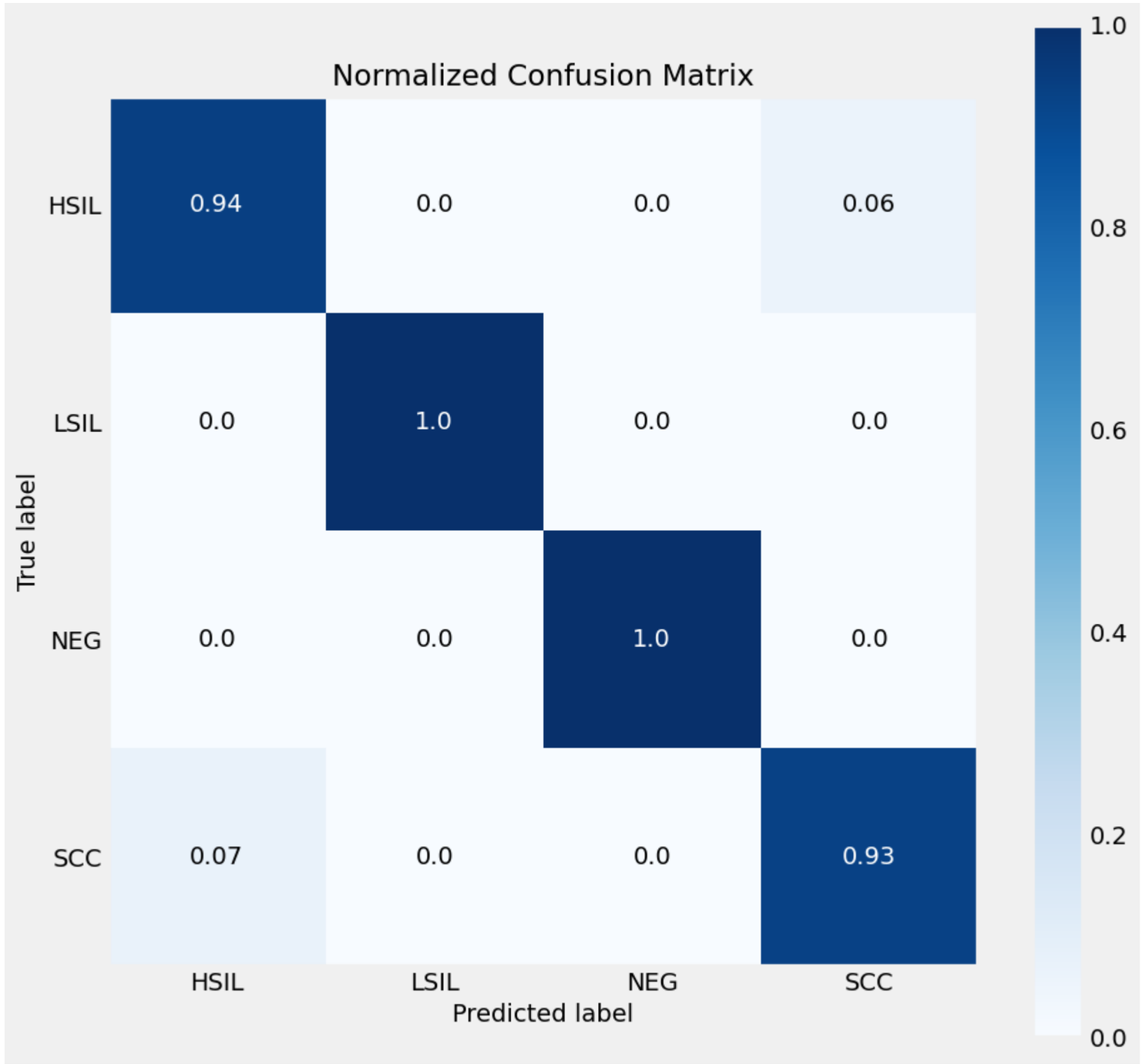
In [85]:
```
confusion_matrix_num(CNN_H2N, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:4: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, whi
ch supports generators.
  preds = model.predict_generator(data)
```



In [86]:
```
confusion_matrix_norm(CNN_H2N, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:19: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, wh
ich supports generators.
  preds = model.predict_generator(data)
```



VGGNet Model

In [72]:
```
from tensorflow.keras.applications import VGG16
from tensorflow.keras import models,optimizers, regularizers

# Define the batch size
batch_size = 64

# Define the number of classes in your dataset
num_classes = 4

# Create the VGGNet model with BatchNormalization and Dropout
def create_vgg_model(input_shape, num_classes):
    vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    # Freeze the VGG16 layers
    for layer in vgg_model.layers:
        layer.trainable = False

    model = models.Sequential()
    model.add(vgg_model)

    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Create the VGGNet model
model_vgg = create_vgg_model(input_shape, num_classes)
```

In [73]:
```
# Compile the model
#optimizer = optimizers.Adam(learning_rate=0.001)
model_vgg.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model_vgg.summary()
```

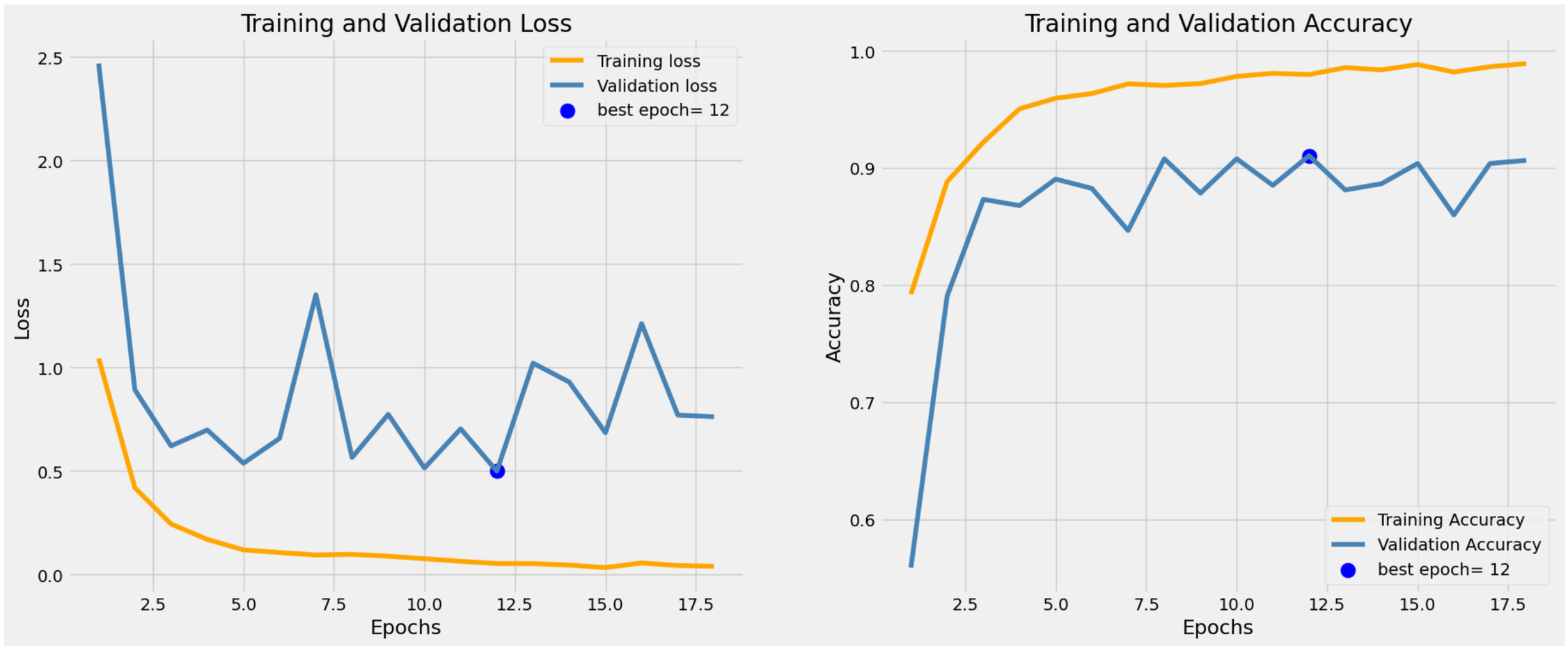
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten_10 (Flatten)        (None, 8192)              0

 dense_30 (Dense)            (None, 4096)              33558528

 batch_normalization_25 (Bat  (None, 4096)             16384
 chNormalization)

 dropout_9 (Dropout)         (None, 4096)              0

 dense_31 (Dense)            (None, 4096)              16781312

 batch_normalization_26 (Bat  (None, 4096)             16384
 chNormalization)

 dropout_10 (Dropout)        (None, 4096)              0

 dense_32 (Dense)            (None, 4)                 16388

=================================================================
Total params: 65,103,684
Trainable params: 50,372,612
Non-trainable params: 14,731,072
_________________________________________________________________
```

In [75]:
```
# Train the model
history = model_vgg.fit(
    ds_trn,
    epochs=50,
    batch_size = batch_size,
    verbose= 1,
    callbacks=[FinalCallBack(), EarlyStopping],
    validation_data= ds_val)
```

```
Epoch 1/50
59/59 [==============================] - ETA: 0s - loss: 1.0449 - accuracy: 0.7923
***************

The average loss for epoch 1 is  1.045 and accuracy is  0.792.

**************

59/59 [==============================] - 111s 2s/step - loss: 1.0449 - accuracy: 0.7923 - val_loss: 2.4688 - val_accuracy: 0.5587
Epoch 2/50
59/59 [==============================] - ETA: 0s - loss: 0.4201 - accuracy: 0.8885
***************

The average loss for epoch 2 is  0.420 and accuracy is  0.889.

**************

59/59 [==============================] - 107s 2s/step - loss: 0.4201 - accuracy: 0.8885 - val_loss: 0.8932 - val_accuracy: 0.7907
Epoch 3/50
59/59 [==============================] - ETA: 0s - loss: 0.2451 - accuracy: 0.9221
***************

The average loss for epoch 3 is  0.245 and accuracy is  0.922.

**************

59/59 [==============================] - 113s 2s/step - loss: 0.2451 - accuracy: 0.9221 - val_loss: 0.6225 - val_accuracy: 0.8733
Epoch 4/50
59/59 [==============================] - ETA: 0s - loss: 0.1708 - accuracy: 0.9507
***************

The average loss for epoch 4 is  0.171 and accuracy is  0.951.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.1708 - accuracy: 0.9507 - val_loss: 0.6993 - val_accuracy: 0.8680
Epoch 5/50
59/59 [==============================] - ETA: 0s - loss: 0.1200 - accuracy: 0.9597
***************

The average loss for epoch 5 is  0.120 and accuracy is  0.960.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.1200 - accuracy: 0.9597 - val_loss: 0.5387 - val_accuracy: 0.8907
Epoch 6/50
59/59 [==============================] - ETA: 0s - loss: 0.1073 - accuracy: 0.9637
***************

The average loss for epoch 6 is  0.107 and accuracy is  0.964.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.1073 - accuracy: 0.9637 - val_loss: 0.6589 - val_accuracy: 0.8827
Epoch 7/50
59/59 [==============================] - ETA: 0s - loss: 0.0956 - accuracy: 0.9720
***************

The average loss for epoch 7 is  0.096 and accuracy is  0.972.

**************

59/59 [==============================] - 105s 2s/step - loss: 0.0956 - accuracy: 0.9720 - val_loss: 1.3523 - val_accuracy: 0.8467
Epoch 8/50
59/59 [==============================] - ETA: 0s - loss: 0.0988 - accuracy: 0.9707
***************

The average loss for epoch 8 is  0.099 and accuracy is  0.971.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.0988 - accuracy: 0.9707 - val_loss: 0.5664 - val_accuracy: 0.9080
Epoch 9/50
59/59 [==============================] - ETA: 0s - loss: 0.0898 - accuracy: 0.9723
***************

The average loss for epoch 9 is  0.090 and accuracy is  0.972.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0898 - accuracy: 0.9723 - val_loss: 0.7746 - val_accuracy: 0.8787
Epoch 10/50
59/59 [==============================] - ETA: 0s - loss: 0.0776 - accuracy: 0.9784
***************

The average loss for epoch 10 is  0.078 and accuracy is  0.978.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0776 - accuracy: 0.9784 - val_loss: 0.5155 - val_accuracy: 0.9080
Epoch 11/50
59/59 [==============================] - ETA: 0s - loss: 0.0650 - accuracy: 0.9811
***************

The average loss for epoch 11 is  0.065 and accuracy is  0.981.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0650 - accuracy: 0.9811 - val_loss: 0.7048 - val_accuracy: 0.8853
Epoch 12/50
59/59 [==============================] - ETA: 0s - loss: 0.0542 - accuracy: 0.9800
***************

The average loss for epoch 12 is  0.054 and accuracy is  0.980.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0542 - accuracy: 0.9800 - val_loss: 0.5004 - val_accuracy: 0.9107
Epoch 13/50
59/59 [==============================] - ETA: 0s - loss: 0.0538 - accuracy: 0.9859
***************

The average loss for epoch 13 is  0.054 and accuracy is  0.986.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0538 - accuracy: 0.9859 - val_loss: 1.0220 - val_accuracy: 0.8813
Epoch 14/50
59/59 [==============================] - ETA: 0s - loss: 0.0468 - accuracy: 0.9840
***************

The average loss for epoch 14 is  0.047 and accuracy is  0.984.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0468 - accuracy: 0.9840 - val_loss: 0.9309 - val_accuracy: 0.8867
```

```
Epoch 15/50
59/59 [==============================] - ETA: 0s - loss: 0.0345 - accuracy: 0.9885
***************

The average loss for epoch 15 is  0.035 and accuracy is  0.989.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0345 - accuracy: 0.9885 - val_loss: 0.6861 - val_accuracy: 0.9040
Epoch 16/50
59/59 [==============================] - ETA: 0s - loss: 0.0572 - accuracy: 0.9821
***************

The average loss for epoch 16 is  0.057 and accuracy is  0.982.

**************

59/59 [==============================] - 102s 2s/step - loss: 0.0572 - accuracy: 0.9821 - val_loss: 1.2130 - val_accuracy: 0.8600
Epoch 17/50
59/59 [==============================] - ETA: 0s - loss: 0.0445 - accuracy: 0.9867
***************

The average loss for epoch 17 is  0.045 and accuracy is  0.987.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0445 - accuracy: 0.9867 - val_loss: 0.7713 - val_accuracy: 0.9040
Epoch 18/50
59/59 [==============================] - ETA: 0s - loss: 0.0409 - accuracy: 0.9893
***************

The average loss for epoch 18 is  0.041 and accuracy is  0.989.

**************

59/59 [==============================] - 101s 2s/step - loss: 0.0409 - accuracy: 0.9893 - val_loss: 0.7630 - val_accuracy: 0.9067
```

In [76]:
```
plot_training(history)
```

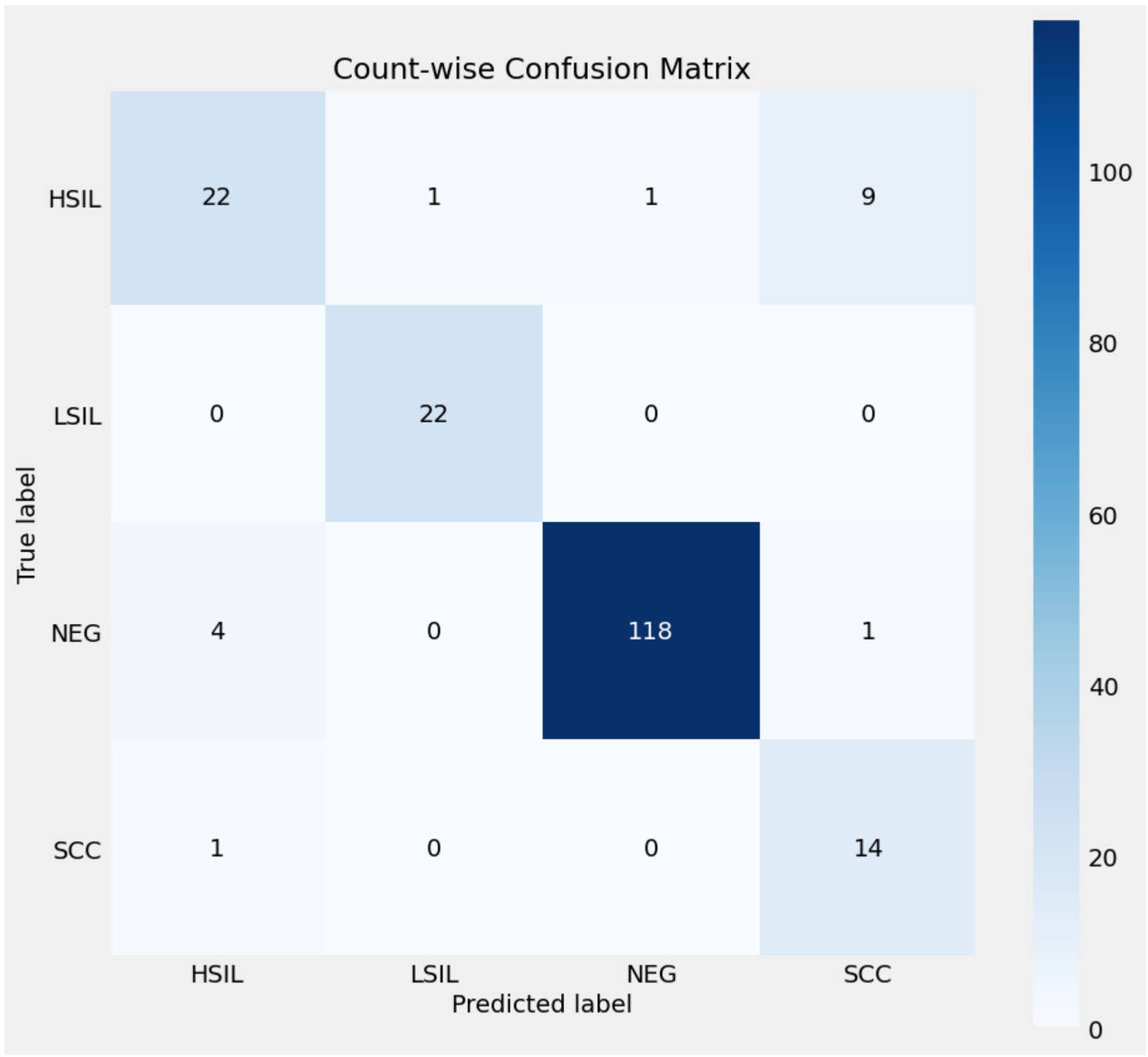


In [78]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = model_vgg.evaluate(ds_tst_Og)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))

# Generate classification report for the test data
y_pred = model_vgg.predict(ds_tst_Og)
y_pred_classes = np.argmax(y_pred, axis=1)
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst_Og.classes, y_pred_classes, target_names=class_names))
```

```
4/4 [==============================] - 14s 3s/step - loss: 0.4773 - accuracy: 0.9119
Test Accuracy: 91.19%
              precision    recall  f1-score   support

        HSIL       0.81      0.67      0.73        33
        LSIL       0.96      1.00      0.98        22
         NEG       0.99      0.96      0.98       123
         SCC       0.58      0.93      0.72        15

    accuracy                           0.91       193
   macro avg       0.84      0.89      0.85       193
weighted avg       0.93      0.91      0.91       193
```

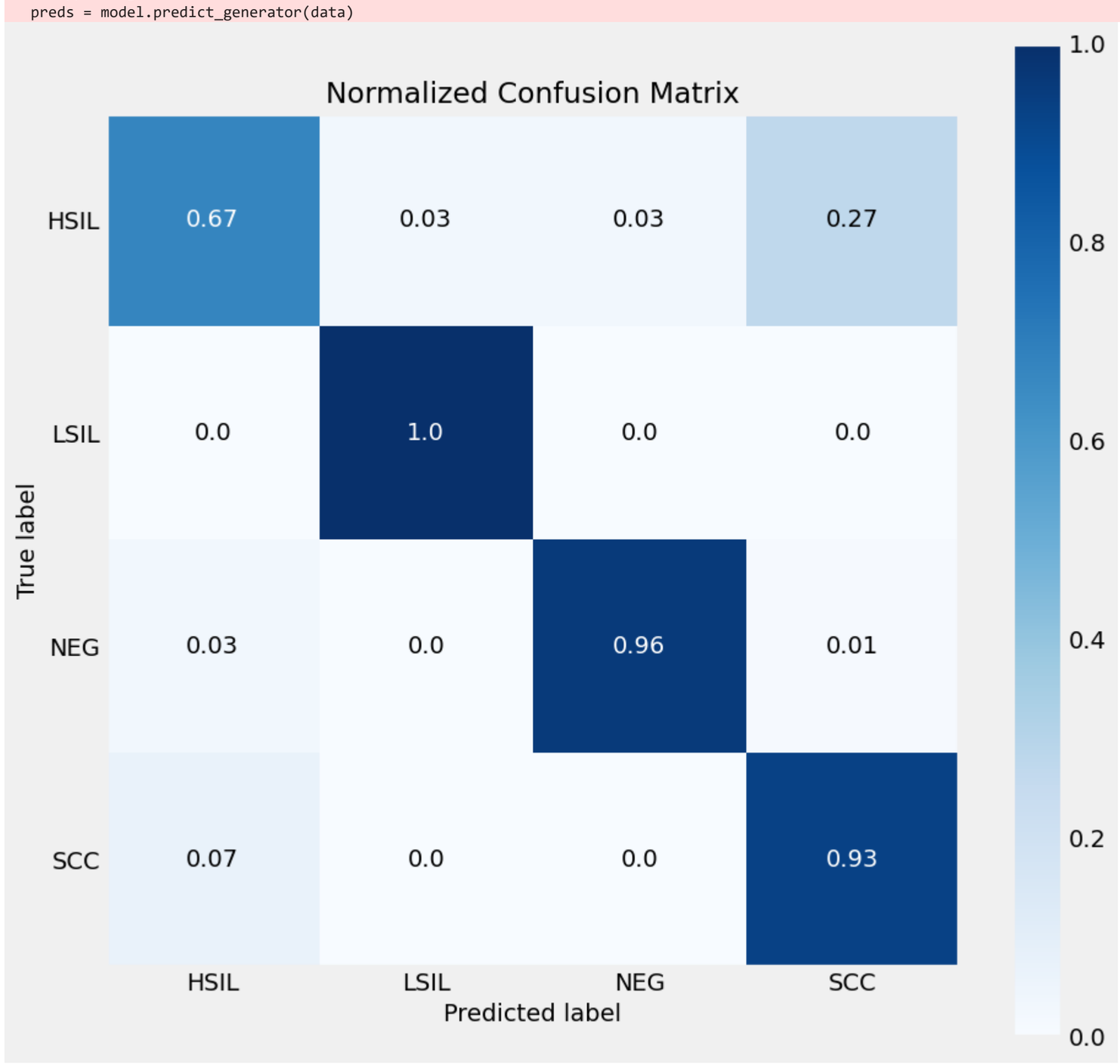
In [79]:
```
confusion_matrix_num(model_vgg, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:4: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, whi
ch supports generators.
  preds = model.predict_generator(data)
```



In [87]:
```
confusion_matrix_norm(model_vgg, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:19: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, wh
ich supports generators.
  preds = model.predict_generator(data)
```



RESNET50

Defining Blocks

In [126…
```
def identity_block(input, filters):
  filter1, filter2, filter3 = filters
  x = tf.keras.layers.Conv2D(filter1, (1,1))(input)
  x = tf.keras.layers.BatchNormalization()(x)
  x = tf.keras.activations.relu(x)

  x = tf.keras.layers.Conv2D(filter2, (3,3), padding='same')(x)
  x = tf.keras.layers.BatchNormalization()(x)
  x = tf.keras.activations.relu(x)

  x = tf.keras.layers.Conv2D(filter3, (1,1))(x)
  x = tf.keras.layers.BatchNormalization()(x)
```

```
  x = tf.keras.layers.add([x, input])
  x = tf.keras.activations.relu(x)
  return x

def conv_block(input, filters, strides):
  filter1, filter2, filter3 = filters
  x = tf.keras.layers.Conv2D(filter1, (1,1), strides=strides)(input)
  x = tf.keras.layers.BatchNormalization()(x)
  x = tf.keras.activations.relu(x)

  x = tf.keras.layers.Conv2D(filter2, (3,3), padding='same')(x)
  x = tf.keras.layers.BatchNormalization()(x)
  x = tf.keras.activations.relu(x)

  x = tf.keras.layers.Conv2D(filter3, (1,1))(x)
  x = tf.keras.layers.BatchNormalization()(x)

  tmp = tf.keras.layers.Conv2D(filter3, (1,1), strides=strides)(input)
  tmp = tf.keras.layers.BatchNormalization()(tmp)

  x = layers.add([x, tmp])
  x = tf.keras.activations.relu(x,return_sequences = True)
  return x
```

In [161…

```
model_res50 = tf.keras.Sequential()

def resnet50():
    input = tf.keras.Input(shape = input_shape)
    x = tf.keras.layers.ZeroPadding2D((3,3))(input)

    x = tf.keras.layers.Conv2D(64, (7,7), strides=(2,2))(input)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.activations.relu(x)
    x = tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides=(2,2))(x)

    x = conv_block(x, [64,64,256], strides=(1,1))
    x = identity_block(x, [64,64,256])
    x = identity_block(x, [64,64,256])

    x = conv_block(x, [128,128,512], strides=(2,2))
    x = identity_block(x, [128,128,512])
    x = identity_block(x, [128,128,512])
    x = identity_block(x, [128,128,512])

    x = conv_block(x, [256,256,1024], strides=(2,2))
    x = identity_block(x, [256,256,1024])
    x = identity_block(x, [256,256,1024])
    x = identity_block(x, [256,256,1024])
    x = identity_block(x, [256,256,1024])
    x = identity_block(x, [256,256,1024])

    x = conv_block(x, [512,512,2048], strides=(2,2))
    x = identity_block(x, [512,512,2048])
    x = identity_block(x, [512,512,2048])

    x = tf.keras.layers.AveragePooling2D(pool_size=(3,3),padding='same')(x)
    x = tf.keras.layers.Flatten()(x)

    output = tf.keras.layers.Dense(4, activation='softmax')(x)
    model = tf.keras.models.Model(input, output)
    return model
```

In [165…

```
resNet50 = resnet50()
resNet50.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])
resNet50.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_16 (InputLayer)          [(None, 128, 128, 3  0           []
                                )]

 conv2d_910 (Conv2D)            (None, 61, 61, 64)   9472        ['input_16[0][0]']

 batch_normalization_915 (Batch  (None, 61, 61, 64)  256         ['conv2d_910[0][0]']
 Normalization)

 tf.nn.relu_841 (TFOpLambda)    (None, 61, 61, 64)   0           ['batch_normalization_915[0][0]']

 max_pooling2d_50 (MaxPooling2D  (None, 30, 30, 64)  0           ['tf.nn.relu_841[0][0]']
 )

 conv2d_911 (Conv2D)            (None, 30, 30, 64)   4160        ['max_pooling2d_50[0][0]']

 batch_normalization_916 (Batch  (None, 30, 30, 64)  256         ['conv2d_911[0][0]']
 Normalization)

 tf.nn.relu_842 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_916[0][0]']

 conv2d_912 (Conv2D)            (None, 30, 30, 64)   36928       ['tf.nn.relu_842[0][0]']

 batch_normalization_917 (Batch  (None, 30, 30, 64)  256         ['conv2d_912[0][0]']
 Normalization)

 tf.nn.relu_843 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_917[0][0]']

 conv2d_913 (Conv2D)            (None, 30, 30, 256)  16640       ['tf.nn.relu_843[0][0]']

 conv2d_914 (Conv2D)            (None, 30, 30, 256)  16640       ['max_pooling2d_50[0][0]']

 batch_normalization_918 (Batch  (None, 30, 30, 256)  1024       ['conv2d_913[0][0]']
 Normalization)

 batch_normalization_919 (Batch  (None, 30, 30, 256)  1024       ['conv2d_914[0][0]']
 Normalization)

 add_279 (Add)                  (None, 30, 30, 256)  0           ['batch_normalization_918[0][0]',
                                                                  'batch_normalization_919[0][0]']

 tf.nn.relu_844 (TFOpLambda)    (None, 30, 30, 256)  0           ['add_279[0][0]']

 conv2d_915 (Conv2D)            (None, 30, 30, 64)   16448       ['tf.nn.relu_844[0][0]']

 batch_normalization_920 (Batch  (None, 30, 30, 64)  256         ['conv2d_915[0][0]']
 Normalization)

 tf.nn.relu_845 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_920[0][0]']

 conv2d_916 (Conv2D)            (None, 30, 30, 64)   36928       ['tf.nn.relu_845[0][0]']

 batch_normalization_921 (Batch  (None, 30, 30, 64)  256         ['conv2d_916[0][0]']
 Normalization)

 tf.nn.relu_846 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_921[0][0]']

 conv2d_917 (Conv2D)            (None, 30, 30, 256)  16640       ['tf.nn.relu_846[0][0]']

 batch_normalization_922 (Batch  (None, 30, 30, 256)  1024       ['conv2d_917[0][0]']
 Normalization)

 add_280 (Add)                  (None, 30, 30, 256)  0           ['batch_normalization_922[0][0]',
                                                                  'tf.nn.relu_844[0][0]']

 tf.nn.relu_847 (TFOpLambda)    (None, 30, 30, 256)  0           ['add_280[0][0]']

 conv2d_918 (Conv2D)            (None, 30, 30, 64)   16448       ['tf.nn.relu_847[0][0]']

 batch_normalization_923 (Batch  (None, 30, 30, 64)  256         ['conv2d_918[0][0]']
 Normalization)

 tf.nn.relu_848 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_923[0][0]']

 conv2d_919 (Conv2D)            (None, 30, 30, 64)   36928       ['tf.nn.relu_848[0][0]']

 batch_normalization_924 (Batch  (None, 30, 30, 64)  256         ['conv2d_919[0][0]']
 Normalization)

 tf.nn.relu_849 (TFOpLambda)    (None, 30, 30, 64)   0           ['batch_normalization_924[0][0]']

 conv2d_920 (Conv2D)            (None, 30, 30, 256)  16640       ['tf.nn.relu_849[0][0]']

 batch_normalization_925 (Batch  (None, 30, 30, 256)  1024       ['conv2d_920[0][0]']
 Normalization)

 add_281 (Add)                  (None, 30, 30, 256)  0           ['batch_normalization_925[0][0]',
                                                                  'tf.nn.relu_847[0][0]']

 tf.nn.relu_850 (TFOpLambda)    (None, 30, 30, 256)  0           ['add_281[0][0]']

 conv2d_921 (Conv2D)            (None, 15, 15, 128)  32896       ['tf.nn.relu_850[0][0]']

 batch_normalization_926 (Batch  (None, 15, 15, 128)  512       ['conv2d_921[0][0]']
 Normalization)

 tf.nn.relu_851 (TFOpLambda)    (None, 15, 15, 128)  0           ['batch_normalization_926[0][0]']

 conv2d_922 (Conv2D)            (None, 15, 15, 128)  147584      ['tf.nn.relu_851[0][0]']

 batch_normalization_927 (Batch  (None, 15, 15, 128)  512       ['conv2d_922[0][0]']
 Normalization)

 tf.nn.relu_852 (TFOpLambda)    (None, 15, 15, 128)  0           ['batch_normalization_927[0][0]']

 conv2d_923 (Conv2D)            (None, 15, 15, 512)  66048       ['tf.nn.relu_852[0][0]']

 conv2d_924 (Conv2D)            (None, 15, 15, 512)  131584      ['tf.nn.relu_850[0][0]']

 batch_normalization_928 (Batch  (None, 15, 15, 512)  2048      ['conv2d_923[0][0]']
 Normalization)

 batch_normalization_929 (Batch  (None, 15, 15, 512)  2048      ['conv2d_924[0][0]']
 Normalization)

 add_282 (Add)                  (None, 15, 15, 512)  0           ['batch_normalization_928[0][0]',
                                                                  'batch_normalization_929[0][0]']

 tf.nn.relu_853 (TFOpLambda)    (None, 15, 15, 512)  0           ['add_282[0][0]']

 conv2d_925 (Conv2D)            (None, 15, 15, 128)  65664       ['tf.nn.relu_853[0][0]']

 batch_normalization_930 (Batch  (None, 15, 15, 128)  512       ['conv2d_925[0][0]']
```

```
Normalization)

tf.nn.relu_854 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_930[0][0]']

conv2d_926 (Conv2D)            (None, 15, 15, 128)  147584     ['tf.nn.relu_854[0][0]']

batch_normalization_931 (Batch  (None, 15, 15, 128)  512       ['conv2d_926[0][0]']
Normalization)

tf.nn.relu_855 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_931[0][0]']

conv2d_927 (Conv2D)            (None, 15, 15, 512)  66048      ['tf.nn.relu_855[0][0]']

batch_normalization_932 (Batch  (None, 15, 15, 512)  2048      ['conv2d_927[0][0]']
Normalization)

add_283 (Add)                  (None, 15, 15, 512)  0          ['batch_normalization_932[0][0]',
                                                                'tf.nn.relu_853[0][0]']

tf.nn.relu_856 (TFOpLambda)    (None, 15, 15, 512)  0          ['add_283[0][0]']

conv2d_928 (Conv2D)            (None, 15, 15, 128)  65664      ['tf.nn.relu_856[0][0]']

batch_normalization_933 (Batch  (None, 15, 15, 128)  512       ['conv2d_928[0][0]']
Normalization)

tf.nn.relu_857 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_933[0][0]']

conv2d_929 (Conv2D)            (None, 15, 15, 128)  147584     ['tf.nn.relu_857[0][0]']

batch_normalization_934 (Batch  (None, 15, 15, 128)  512       ['conv2d_929[0][0]']
Normalization)

tf.nn.relu_858 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_934[0][0]']

conv2d_930 (Conv2D)            (None, 15, 15, 512)  66048      ['tf.nn.relu_858[0][0]']

batch_normalization_935 (Batch  (None, 15, 15, 512)  2048      ['conv2d_930[0][0]']
Normalization)

add_284 (Add)                  (None, 15, 15, 512)  0          ['batch_normalization_935[0][0]',
                                                                'tf.nn.relu_856[0][0]']

tf.nn.relu_859 (TFOpLambda)    (None, 15, 15, 512)  0          ['add_284[0][0]']

conv2d_931 (Conv2D)            (None, 15, 15, 128)  65664      ['tf.nn.relu_859[0][0]']

batch_normalization_936 (Batch  (None, 15, 15, 128)  512       ['conv2d_931[0][0]']
Normalization)

tf.nn.relu_860 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_936[0][0]']

conv2d_932 (Conv2D)            (None, 15, 15, 128)  147584     ['tf.nn.relu_860[0][0]']

batch_normalization_937 (Batch  (None, 15, 15, 128)  512       ['conv2d_932[0][0]']
Normalization)

tf.nn.relu_861 (TFOpLambda)    (None, 15, 15, 128)  0          ['batch_normalization_937[0][0]']

conv2d_933 (Conv2D)            (None, 15, 15, 512)  66048      ['tf.nn.relu_861[0][0]']

batch_normalization_938 (Batch  (None, 15, 15, 512)  2048      ['conv2d_933[0][0]']
Normalization)

add_285 (Add)                  (None, 15, 15, 512)  0          ['batch_normalization_938[0][0]',
                                                                'tf.nn.relu_859[0][0]']

tf.nn.relu_862 (TFOpLambda)    (None, 15, 15, 512)  0          ['add_285[0][0]']

conv2d_934 (Conv2D)            (None, 8, 8, 256)    131328     ['tf.nn.relu_862[0][0]']

batch_normalization_939 (Batch  (None, 8, 8, 256)   1024       ['conv2d_934[0][0]']
Normalization)

tf.nn.relu_863 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_939[0][0]']

conv2d_935 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_863[0][0]']

batch_normalization_940 (Batch  (None, 8, 8, 256)   1024       ['conv2d_935[0][0]']
Normalization)

tf.nn.relu_864 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_940[0][0]']

conv2d_936 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_864[0][0]']

conv2d_937 (Conv2D)            (None, 8, 8, 1024)   525312     ['tf.nn.relu_862[0][0]']

batch_normalization_941 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_936[0][0]']
Normalization)

batch_normalization_942 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_937[0][0]']
Normalization)

add_286 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_941[0][0]',
                                                                'batch_normalization_942[0][0]']

tf.nn.relu_865 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_286[0][0]']

conv2d_938 (Conv2D)            (None, 8, 8, 256)    262400     ['tf.nn.relu_865[0][0]']

batch_normalization_943 (Batch  (None, 8, 8, 256)   1024       ['conv2d_938[0][0]']
Normalization)

tf.nn.relu_866 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_943[0][0]']

conv2d_939 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_866[0][0]']

batch_normalization_944 (Batch  (None, 8, 8, 256)   1024       ['conv2d_939[0][0]']
Normalization)

tf.nn.relu_867 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_944[0][0]']

conv2d_940 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_867[0][0]']

batch_normalization_945 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_940[0][0]']
Normalization)

add_287 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_945[0][0]',
                                                                'tf.nn.relu_865[0][0]']

tf.nn.relu_868 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_287[0][0]']

conv2d_941 (Conv2D)            (None, 8, 8, 256)    262400     ['tf.nn.relu_868[0][0]']

batch_normalization_946 (Batch  (None, 8, 8, 256)   1024       ['conv2d_941[0][0]']
Normalization)
```

```
tf.nn.relu_869 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_946[0][0]']

conv2d_942 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_869[0][0]']

batch_normalization_947 (Batch  (None, 8, 8, 256)   1024       ['conv2d_942[0][0]']
Normalization)

tf.nn.relu_870 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_947[0][0]']

conv2d_943 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_870[0][0]']

batch_normalization_948 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_943[0][0]']
Normalization)

add_288 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_948[0][0]',
                                                                'tf.nn.relu_868[0][0]']

tf.nn.relu_871 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_288[0][0]']

conv2d_944 (Conv2D)            (None, 8, 8, 256)    262400     ['tf.nn.relu_871[0][0]']

batch_normalization_949 (Batch  (None, 8, 8, 256)   1024       ['conv2d_944[0][0]']
Normalization)

tf.nn.relu_872 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_949[0][0]']

conv2d_945 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_872[0][0]']

batch_normalization_950 (Batch  (None, 8, 8, 256)   1024       ['conv2d_945[0][0]']
Normalization)

tf.nn.relu_873 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_950[0][0]']

conv2d_946 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_873[0][0]']

batch_normalization_951 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_946[0][0]']
Normalization)

add_289 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_951[0][0]',
                                                                'tf.nn.relu_871[0][0]']

tf.nn.relu_874 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_289[0][0]']

conv2d_947 (Conv2D)            (None, 8, 8, 256)    262400     ['tf.nn.relu_874[0][0]']

batch_normalization_952 (Batch  (None, 8, 8, 256)   1024       ['conv2d_947[0][0]']
Normalization)

tf.nn.relu_875 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_952[0][0]']

conv2d_948 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_875[0][0]']

batch_normalization_953 (Batch  (None, 8, 8, 256)   1024       ['conv2d_948[0][0]']
Normalization)

tf.nn.relu_876 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_953[0][0]']

conv2d_949 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_876[0][0]']

batch_normalization_954 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_949[0][0]']
Normalization)

add_290 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_954[0][0]',
                                                                'tf.nn.relu_874[0][0]']

tf.nn.relu_877 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_290[0][0]']

conv2d_950 (Conv2D)            (None, 8, 8, 256)    262400     ['tf.nn.relu_877[0][0]']

batch_normalization_955 (Batch  (None, 8, 8, 256)   1024       ['conv2d_950[0][0]']
Normalization)

tf.nn.relu_878 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_955[0][0]']

conv2d_951 (Conv2D)            (None, 8, 8, 256)    590080     ['tf.nn.relu_878[0][0]']

batch_normalization_956 (Batch  (None, 8, 8, 256)   1024       ['conv2d_951[0][0]']
Normalization)

tf.nn.relu_879 (TFOpLambda)    (None, 8, 8, 256)    0          ['batch_normalization_956[0][0]']

conv2d_952 (Conv2D)            (None, 8, 8, 1024)   263168     ['tf.nn.relu_879[0][0]']

batch_normalization_957 (Batch  (None, 8, 8, 1024)  4096       ['conv2d_952[0][0]']
Normalization)

add_291 (Add)                  (None, 8, 8, 1024)   0          ['batch_normalization_957[0][0]',
                                                                'tf.nn.relu_877[0][0]']

tf.nn.relu_880 (TFOpLambda)    (None, 8, 8, 1024)   0          ['add_291[0][0]']

conv2d_953 (Conv2D)            (None, 4, 4, 512)    524800     ['tf.nn.relu_880[0][0]']

batch_normalization_958 (Batch  (None, 4, 4, 512)   2048       ['conv2d_953[0][0]']
Normalization)

tf.nn.relu_881 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_958[0][0]']

conv2d_954 (Conv2D)            (None, 4, 4, 512)    2359808    ['tf.nn.relu_881[0][0]']

batch_normalization_959 (Batch  (None, 4, 4, 512)   2048       ['conv2d_954[0][0]']
Normalization)

tf.nn.relu_882 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_959[0][0]']

conv2d_955 (Conv2D)            (None, 4, 4, 2048)   1050624    ['tf.nn.relu_882[0][0]']

conv2d_956 (Conv2D)            (None, 4, 4, 2048)   2099200    ['tf.nn.relu_880[0][0]']

batch_normalization_960 (Batch  (None, 4, 4, 2048)  8192       ['conv2d_955[0][0]']
Normalization)

batch_normalization_961 (Batch  (None, 4, 4, 2048)  8192       ['conv2d_956[0][0]']
Normalization)

add_292 (Add)                  (None, 4, 4, 2048)   0          ['batch_normalization_960[0][0]',
                                                                'batch_normalization_961[0][0]']

tf.nn.relu_883 (TFOpLambda)    (None, 4, 4, 2048)   0          ['add_292[0][0]']

conv2d_957 (Conv2D)            (None, 4, 4, 512)    1049088    ['tf.nn.relu_883[0][0]']

batch_normalization_962 (Batch  (None, 4, 4, 512)   2048       ['conv2d_957[0][0]']
Normalization)
```

```
tf.nn.relu_884 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_962[0][0]']

conv2d_958 (Conv2D)            (None, 4, 4, 512)    2359808    ['tf.nn.relu_884[0][0]']

batch_normalization_963 (Batch  (None, 4, 4, 512)   2048       ['conv2d_958[0][0]']
Normalization)

tf.nn.relu_885 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_963[0][0]']

conv2d_959 (Conv2D)            (None, 4, 4, 2048)   1050624    ['tf.nn.relu_885[0][0]']

batch_normalization_964 (Batch  (None, 4, 4, 2048)  8192       ['conv2d_959[0][0]']
Normalization)

add_293 (Add)                  (None, 4, 4, 2048)   0          ['batch_normalization_964[0][0]',
                                                               'tf.nn.relu_883[0][0]']

tf.nn.relu_886 (TFOpLambda)    (None, 4, 4, 2048)   0          ['add_293[0][0]']

conv2d_960 (Conv2D)            (None, 4, 4, 512)    1049088    ['tf.nn.relu_886[0][0]']

batch_normalization_965 (Batch  (None, 4, 4, 512)   2048       ['conv2d_960[0][0]']
Normalization)

tf.nn.relu_887 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_965[0][0]']

conv2d_961 (Conv2D)            (None, 4, 4, 512)    2359808    ['tf.nn.relu_887[0][0]']

batch_normalization_966 (Batch  (None, 4, 4, 512)   2048       ['conv2d_961[0][0]']
Normalization)

tf.nn.relu_888 (TFOpLambda)    (None, 4, 4, 512)    0          ['batch_normalization_966[0][0]']

conv2d_962 (Conv2D)            (None, 4, 4, 2048)   1050624    ['tf.nn.relu_888[0][0]']

batch_normalization_967 (Batch  (None, 4, 4, 2048)  8192       ['conv2d_962[0][0]']
Normalization)

add_294 (Add)                  (None, 4, 4, 2048)   0          ['batch_normalization_967[0][0]',
                                                               'tf.nn.relu_886[0][0]']

tf.nn.relu_889 (TFOpLambda)    (None, 4, 4, 2048)   0          ['add_294[0][0]']

average_pooling2d_10 (AverageP  (None, 2, 2, 2048)  0          ['tf.nn.relu_889[0][0]']
ooling2D)

flatten_20 (Flatten)           (None, 8192)         0          ['average_pooling2d_10[0][0]']

dense_47 (Dense)               (None, 4)            32772      ['flatten_20[0][0]']

==================================================================================================
Total params: 23,620,484
Trainable params: 23,567,364
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

In [166…

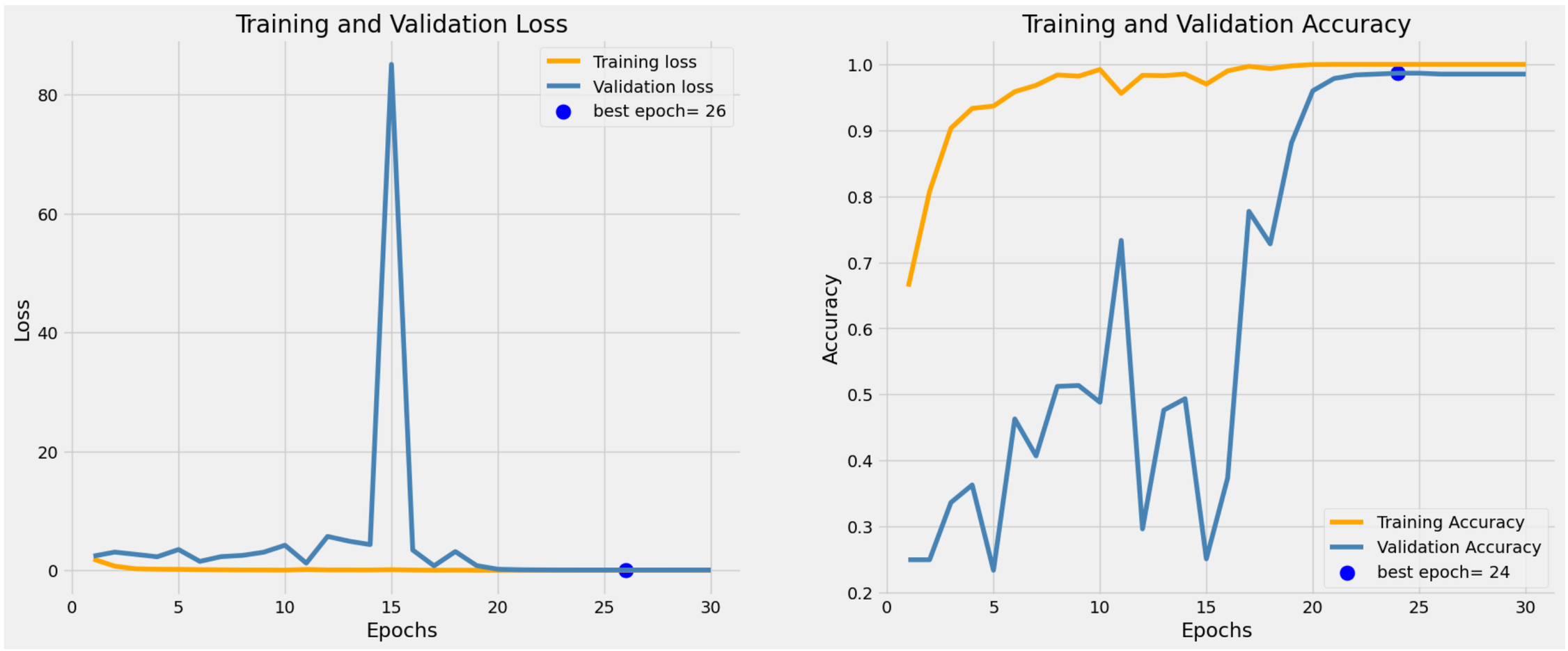
```
results_res50 = resNet50.fit(ds_trn,
                             epochs=30,
                             batch_size = 64,
                             verbose= 1,
                             validation_data=ds_val,
                             callbacks=[FinalCallBack(), EarlyStopping]
                            )
```

```
Epoch 1/30
59/59 [==============================] - ETA: 0s - loss: 1.8671 - accuracy: 0.6632
***************

The average loss for epoch 1 is  1.867 and accuracy is  0.663.

**************

59/59 [==============================] - 121s 2s/step - loss: 1.8671 - accuracy: 0.6632 - val_loss: 2.3804 - val_accuracy: 0.2493
Epoch 2/30
59/59 [==============================] - ETA: 0s - loss: 0.6953 - accuracy: 0.8077
***************

The average loss for epoch 2 is  0.695 and accuracy is  0.808.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.6953 - accuracy: 0.8077 - val_loss: 3.0588 - val_accuracy: 0.2493
Epoch 3/30
59/59 [==============================] - ETA: 0s - loss: 0.2447 - accuracy: 0.9032
***************

The average loss for epoch 3 is  0.245 and accuracy is  0.903.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.2447 - accuracy: 0.9032 - val_loss: 2.6664 - val_accuracy: 0.3360
Epoch 4/30
59/59 [==============================] - ETA: 0s - loss: 0.1735 - accuracy: 0.9333
***************

The average loss for epoch 4 is  0.173 and accuracy is  0.933.

**************

59/59 [==============================] - 106s 2s/step - loss: 0.1735 - accuracy: 0.9333 - val_loss: 2.2740 - val_accuracy: 0.3627
Epoch 5/30
59/59 [==============================] - ETA: 0s - loss: 0.1561 - accuracy: 0.9368
***************

The average loss for epoch 5 is  0.156 and accuracy is  0.937.

**************

59/59 [==============================] - 106s 2s/step - loss: 0.1561 - accuracy: 0.9368 - val_loss: 3.4880 - val_accuracy: 0.2333
Epoch 6/30
59/59 [==============================] - ETA: 0s - loss: 0.1076 - accuracy: 0.9587
***************

The average loss for epoch 6 is  0.108 and accuracy is  0.959.

**************

59/59 [==============================] - 107s 2s/step - loss: 0.1076 - accuracy: 0.9587 - val_loss: 1.4954 - val_accuracy: 0.4627
Epoch 7/30
59/59 [==============================] - ETA: 0s - loss: 0.0787 - accuracy: 0.9683
***************

The average loss for epoch 7 is  0.079 and accuracy is  0.968.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0787 - accuracy: 0.9683 - val_loss: 2.2994 - val_accuracy: 0.4067
Epoch 8/30
59/59 [==============================] - ETA: 0s - loss: 0.0525 - accuracy: 0.9840
***************

The average loss for epoch 8 is  0.052 and accuracy is  0.984.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0525 - accuracy: 0.9840 - val_loss: 2.5013 - val_accuracy: 0.5120
Epoch 9/30
59/59 [==============================] - ETA: 0s - loss: 0.0488 - accuracy: 0.9821
***************

The average loss for epoch 9 is  0.049 and accuracy is  0.982.

**************

59/59 [==============================] - 106s 2s/step - loss: 0.0488 - accuracy: 0.9821 - val_loss: 3.0411 - val_accuracy: 0.5133
Epoch 10/30
59/59 [==============================] - ETA: 0s - loss: 0.0243 - accuracy: 0.9923
***************

The average loss for epoch 10 is  0.024 and accuracy is  0.992.

**************

59/59 [==============================] - 108s 2s/step - loss: 0.0243 - accuracy: 0.9923 - val_loss: 4.2118 - val_accuracy: 0.4880
Epoch 11/30
59/59 [==============================] - ETA: 0s - loss: 0.1354 - accuracy: 0.9560
***************

The average loss for epoch 11 is  0.135 and accuracy is  0.956.

**************

59/59 [==============================] - 107s 2s/step - loss: 0.1354 - accuracy: 0.9560 - val_loss: 1.2047 - val_accuracy: 0.7333
Epoch 12/30
59/59 [==============================] - ETA: 0s - loss: 0.0562 - accuracy: 0.9835
***************

The average loss for epoch 12 is  0.056 and accuracy is  0.983.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0562 - accuracy: 0.9835 - val_loss: 5.6946 - val_accuracy: 0.2960
Epoch 13/30
59/59 [==============================] - ETA: 0s - loss: 0.0551 - accuracy: 0.9829
***************

The average loss for epoch 13 is  0.055 and accuracy is  0.983.

**************

59/59 [==============================] - 104s 2s/step - loss: 0.0551 - accuracy: 0.9829 - val_loss: 4.8997 - val_accuracy: 0.4760
Epoch 14/30
59/59 [==============================] - ETA: 0s - loss: 0.0439 - accuracy: 0.9853
***************

The average loss for epoch 14 is  0.044 and accuracy is  0.985.

**************

59/59 [==============================] - 103s 2s/step - loss: 0.0439 - accuracy: 0.9853 - val_loss: 4.3197 - val_accuracy: 0.4933
```

```
Epoch 15/30
59/59 [==============================] - ETA: 0s - loss: 0.0999 - accuracy: 0.9701
**********

The average loss for epoch 15 is  0.100 and accuracy is  0.970.

**********

59/59 [==============================] - 104s 2s/step - loss: 0.0999 - accuracy: 0.9701 - val_loss: 85.0651 - val_accuracy: 0.2507
Epoch 16/30
59/59 [==============================] - ETA: 0s - loss: 0.0379 - accuracy: 0.9901
**********

The average loss for epoch 16 is  0.038 and accuracy is  0.990.

**********

59/59 [==============================] - 106s 2s/step - loss: 0.0379 - accuracy: 0.9901 - val_loss: 3.4078 - val_accuracy: 0.3733
Epoch 17/30
59/59 [==============================] - ETA: 0s - loss: 0.0106 - accuracy: 0.9971
**********

The average loss for epoch 17 is  0.011 and accuracy is  0.997.

**********

59/59 [==============================] - 105s 2s/step - loss: 0.0106 - accuracy: 0.9971 - val_loss: 0.7470 - val_accuracy: 0.7773
Epoch 18/30
59/59 [==============================] - ETA: 0s - loss: 0.0211 - accuracy: 0.9936
**********

The average loss for epoch 18 is  0.021 and accuracy is  0.994.

**********

59/59 [==============================] - 104s 2s/step - loss: 0.0211 - accuracy: 0.9936 - val_loss: 3.1474 - val_accuracy: 0.7280
Epoch 19/30
59/59 [==============================] - ETA: 0s - loss: 0.0075 - accuracy: 0.9979
**********

The average loss for epoch 19 is  0.007 and accuracy is  0.998.

**********

59/59 [==============================] - 104s 2s/step - loss: 0.0075 - accuracy: 0.9979 - val_loss: 0.7899 - val_accuracy: 0.8813
Epoch 20/30
59/59 [==============================] - ETA: 0s - loss: 5.6691e-04 - accuracy: 0.9997
**********

The average loss for epoch 20 is  0.001 and accuracy is  1.000.

**********

59/59 [==============================] - 104s 2s/step - loss: 5.6691e-04 - accuracy: 0.9997 - val_loss: 0.1777 - val_accuracy: 0.9600
Epoch 21/30
59/59 [==============================] - ETA: 0s - loss: 1.5299e-04 - accuracy: 1.0000
**********

The average loss for epoch 21 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 108s 2s/step - loss: 1.5299e-04 - accuracy: 1.0000 - val_loss: 0.0917 - val_accuracy: 0.9787
Epoch 22/30
59/59 [==============================] - ETA: 0s - loss: 1.0216e-04 - accuracy: 1.0000
**********

The average loss for epoch 22 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 104s 2s/step - loss: 1.0216e-04 - accuracy: 1.0000 - val_loss: 0.0638 - val_accuracy: 0.9840
Epoch 23/30
59/59 [==============================] - ETA: 0s - loss: 7.7440e-05 - accuracy: 1.0000
**********

The average loss for epoch 23 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 107s 2s/step - loss: 7.7440e-05 - accuracy: 1.0000 - val_loss: 0.0536 - val_accuracy: 0.9853
Epoch 24/30
59/59 [==============================] - ETA: 0s - loss: 6.1659e-05 - accuracy: 1.0000
**********

The average loss for epoch 24 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 105s 2s/step - loss: 6.1659e-05 - accuracy: 1.0000 - val_loss: 0.0512 - val_accuracy: 0.9867
Epoch 25/30
59/59 [==============================] - ETA: 0s - loss: 5.0937e-05 - accuracy: 1.0000
**********

The average loss for epoch 25 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 104s 2s/step - loss: 5.0937e-05 - accuracy: 1.0000 - val_loss: 0.0508 - val_accuracy: 0.9867
Epoch 26/30
59/59 [==============================] - ETA: 0s - loss: 4.2979e-05 - accuracy: 1.0000
**********

The average loss for epoch 26 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 116s 2s/step - loss: 4.2979e-05 - accuracy: 1.0000 - val_loss: 0.0507 - val_accuracy: 0.9853
Epoch 27/30
59/59 [==============================] - ETA: 0s - loss: 3.6863e-05 - accuracy: 1.0000
**********

The average loss for epoch 27 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 114s 2s/step - loss: 3.6863e-05 - accuracy: 1.0000 - val_loss: 0.0510 - val_accuracy: 0.9853
Epoch 28/30
59/59 [==============================] - ETA: 0s - loss: 3.2108e-05 - accuracy: 1.0000
**********

The average loss for epoch 28 is  0.000 and accuracy is  1.000.

**********

59/59 [==============================] - 110s 2s/step - loss: 3.2108e-05 - accuracy: 1.0000 - val_loss: 0.0514 - val_accuracy: 0.9853
```

```
Epoch 29/30
59/59 [==============================] - ETA: 0s - loss: 2.8152e-05 - accuracy: 1.0000
***************

The average loss for epoch 29 is  0.000 and accuracy is  1.000.

**************

59/59 [==============================] - 112s 2s/step - loss: 2.8152e-05 - accuracy: 1.0000 - val_loss: 0.0520 - val_accuracy: 0.9853
Epoch 30/30
59/59 [==============================] - ETA: 0s - loss: 2.4935e-05 - accuracy: 1.0000
***************

The average loss for epoch 30 is  0.000 and accuracy is  1.000.

**************

59/59 [==============================] - 110s 2s/step - loss: 2.4935e-05 - accuracy: 1.0000 - val_loss: 0.0523 - val_accuracy: 0.9853
```

In [167…
```
plot_training(results_res50)
```

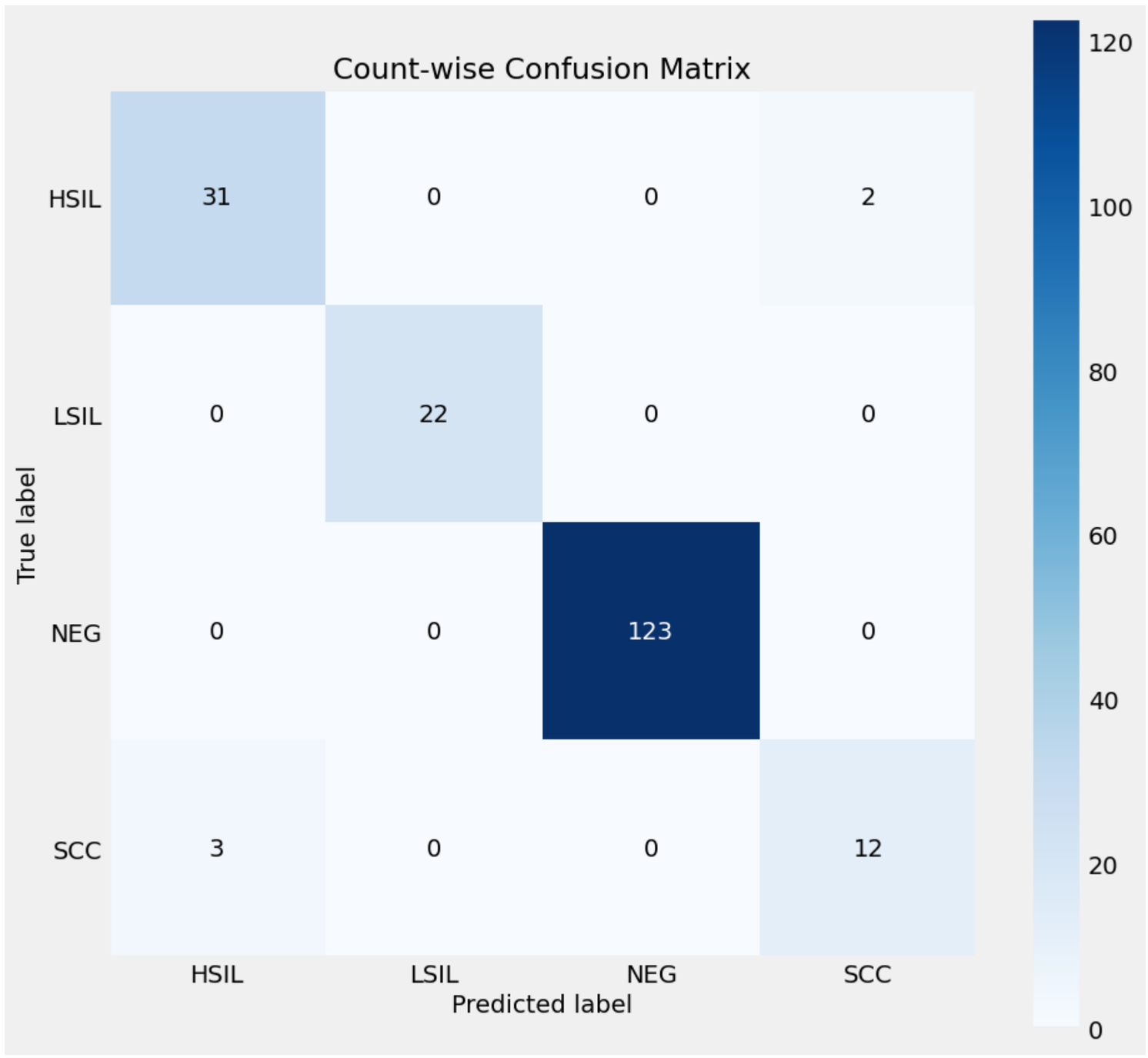


In [172…
```
# Evaluate the model on the test data
test_loss, test_accuracy = resNet50.evaluate(ds_tst_Og)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))

# Generate classification report for the test data
y_pred = resNet50.predict(ds_tst_Og)
y_pred_classes = np.argmax(y_pred, axis=1)
class_names = ds_trn.class_indices.keys()
print(classification_report(ds_tst_Og.classes, y_pred_classes, target_names=class_names))
```

```
4/4 [==============================] - 13s 3s/step - loss: 0.1752 - accuracy: 0.9741
Test Accuracy: 97.41%
              precision    recall  f1-score   support

        HSIL       0.91      0.94      0.93        33
        LSIL       1.00      1.00      1.00        22
         NEG       1.00      1.00      1.00       123
         SCC       0.86      0.80      0.83        15

    accuracy                           0.97       193
   macro avg       0.94      0.93      0.94       193
weighted avg       0.97      0.97      0.97       193
```

In [173…
```
confusion_matrix_num(resNet50, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:4: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, whi
ch supports generators.
  preds = model.predict_generator(data)
```



In [174…
```
confusion_matrix_norm(resNet50, ds_tst_Og)
```

```
C:\Users\Komal\AppData\Local\Temp\ipykernel_18088\3923782795.py:19: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, wh
ich supports generators.
  preds = model.predict_generator(data)
```