Multimodal Custom CNN

DOWNLOADING DATA FROM KAGGLE

```
#Downloading dataset HAM10000 using kaggle -

from google.colab import files
files.upload()
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets list
!chmod 600 /root/.kaggle/kaggle.json
```

```
!kaggle datasets download -d kmader/skin-cancer-mnist-ham10000
!unzip /content/skin-cancer-mnist-ham10000.zip -d /content/data_folder
```

```
#Importing required libraries,modules and checking versions
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import os, cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D
# Import modules and check versions
import google
import glob
import itertools
import os
import pickle
import PIL
import seaborn as sns
from glob import glob
from google.colab import drive
from PIL import Image
from platform import python_version
from google.colab import drive
from keras.applications.resnet import ResNet50
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, History
from keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
from keras.metrics import categorical_accuracy
from keras.models import Sequential,Model,load_model
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from numpy import expand_dims
from platform import python_version

%matplotlib inline

# Versions
print("Version Python:",python_version())
print()
print("Version Matplotlib:",matplotlib.__version__)
print("Version NumPy:",np.__version__)
print("Version Pandas:",pd.__version__)
print("Version PIL:",PIL.__version__)
print("Version Seaborn:",sns.__version__)
print("Version TensorFlow:",tf.__version__)
```

```
Version Python: 3.10.11

Version Matplotlib: 3.7.1
Version NumPy: 1.22.4
Version Pandas: 1.5.3
Version PIL: 8.4.0
Version Seaborn: 0.12.2
Version TensorFlow: 2.12.0
```

PRE-PROCESSING

```
import pandas as pd
metadata = pd.read_csv('/content/data_folder/HAM10000_metadata.csv')
metadata.head()
```

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

```
# To check Unique values per column
for col in metadata:
    print(metadata[col].nunique(),"The unique values present in " + col + ":",np.sort(metadata[col].unique()))
print()
```

```
# Missing values (in form of 'NA' or 'unknown')
print("Number of 'NA' in 'age':", metadata['age'].isna().sum())
print("Number of 'unknown' in 'sex':", len(metadata[metadata['sex']=='unknown']))
print("Number of 'unknown' in 'localization':", len(metadata[metadata['localization']=='unknown']))
print()
```

We have 57 null values in age.Dropping these values can cause loss of metadata. Let's replace them with mean age

```
#Remove rows with missing values for 'age', 'sex', and 'localization'
initial_length_metadata = len(metadata)
metadata['age'].fillna(metadata['age'].mean(), inplace=True)
metadata = metadata.drop(metadata[(metadata['localization'] == 'unknown') & (metadata['sex'] == 'unknown')].index)
#Count missing values ('NA' or 'unknown') in each column
unknown_sex_count = len(metadata[metadata['sex'] == 'unknown'])
unknown_localization_count = len(metadata[metadata['localization'] == 'unknown'])
na_age_count = metadata['age'].isna().sum()

print("Number of 'unknown' values in 'sex' column:", unknown_sex_count)
print("Number of 'unknown' values in 'localization' column:", unknown_localization_count)
print("Number of 'NA' values in 'age' column:", na_age_count)
print()
```

```
Number of 'unknown' values in 'sex' column: 10
Number of 'unknown' values in 'localization' column: 187
Number of 'NA' values in 'age' column: 0
```

```
# Percentage missing (after initial drop)
print("Number of instances with multiple missing values :",sum((len(metadata[(metadata['sex'] == 'unknown') & (metadata['localization'] == 'unknown')].index),1
print("Percentage of missing cases for 'age':", round((metadata['age'].isna().sum()/initial_length_metadata*100),1),"%")
print("Percentage of missing cases for 'localization':", round((len(metadata[metadata['localization'] == 'unknown'])/initial_length_metadata*100),1),"%")
print("Percentage of missing cases for 'sex':", round((len(metadata[metadata['sex'] == 'unknown'])/initial_length_metadata*100),1),"%")
print()
# Drop all remaining rows containing NA's (10) and rows containing value 'unknown' (10 + 187)
metadata = metadata.dropna()
metadata = metadata[metadata.localization != 'unknown']
metadata = metadata[metadata.sex != 'unknown']
print("Percentage of complete cases:", round((len(metadata)/initial_length_metadata*100),1),"%")
```

```
Number of instances with multiple missing values : 0
Percentage of missing cases for 'age': 0.0 %
Percentage of missing cases for 'localization': 1.9 %
Percentage of missing cases for 'sex': 0.1 %

Percentage of complete cases: 97.6 %
```

```
# Creating dictionary to map lesion code with skin lesion type
lesion_classes_dict ={
    0:'akiec',
    1:'bcc',
    2:'bkl',
    3:'df',
    4:'nv',
    5:'mel',
    6:'vasc'
}
#Creating dictionary to determine full name of the skin lesion
lesions = {
    'akiec': 'Actinic keratoses',
    'bcc': 'Basal cell carcinoma',
    'bkl': 'Benign keratosis-like lesions ',
    'df': 'Dermatofibroma',
    'mel': 'Melanoma',
    'nv': 'Melanocytic nevi',
    'vasc': 'Vascular lesions'
}

#malignant skin lesion can cause cancer .
lesion_type = {
    'akiec': 'Malignant',
    'bcc': 'Malignant',
    'bkl': 'Benign',
    'df': 'Benign',
    'mel': 'Malignant',
    'nv': 'Benign',
    'vasc': 'Benign'
}
#Sun exposed and no-sun areas of the body
localizations = {
    'abdomen': 'No Sun',
    'acral': 'No Sun',
    'back': 'No Sun',
    'chest': 'No Sun',
    'ear': 'Sun',
    'face': 'Sun',
    'foot': 'No Sun',
    'genital': 'No Sun',
    'hand': 'Sun',
    'lower extremity': 'Sun',
    'neck': 'Sun',
```

```
    'scalp': 'Sun',
    'trunk': 'No Sun',
    'upper extremity': 'Sun'
}
```

```
ham_data_folder = '../content/data_folder'

# Merge images from both folders into one dictionary

imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(ham_data_folder, '*', '*.jpg'))}
```

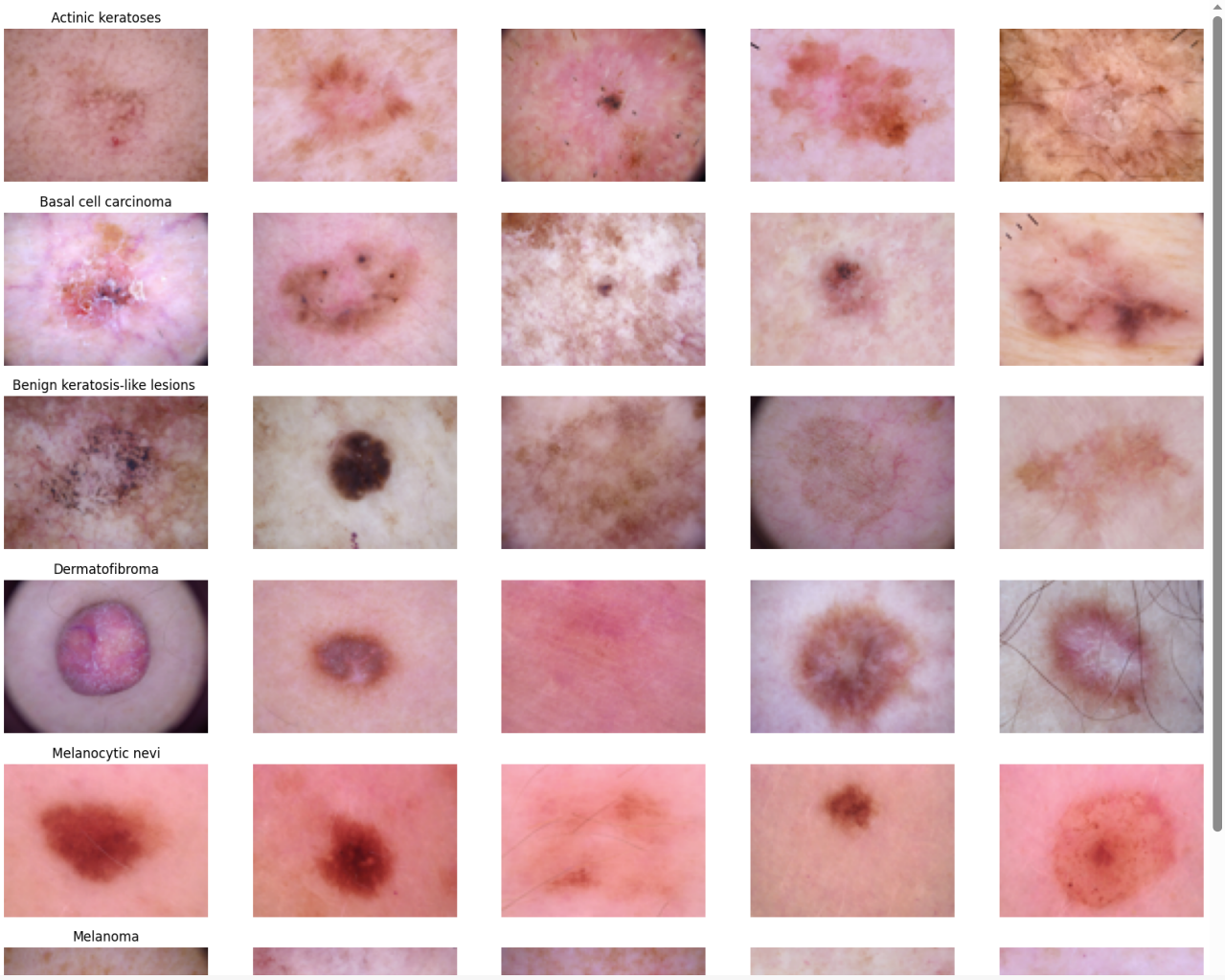
```
#Map 'dx' codes to lesion names and create a categorical code column
metadata['dx_full'] = metadata['dx'].map(lesions.get)
metadata['dx_cat'] = pd.Categorical(metadata['dx_full']).codes
#Map 'image_id' to image file paths and map 'dx' codes to lesion types and create a categorical code column
metadata['image_path'] = metadata['image_id'].map(imageid_path_dict.get)
metadata['lesion_type'] = metadata['dx'].map(lesion_type.get)
metadata['lesion_type_cat'] = pd.Categorical(metadata['lesion_type']).codes
#Create a categorical code column for 'sex' and map 'localization' to localization groups and create a categorical code column
metadata['sex_cat'] = pd.Categorical(metadata['sex']).codes
metadata['loc_group'] = metadata['localization'].map(localizations.get)
metadata['loc_cat'] = pd.Categorical(metadata['loc_group']).codes
#Create an 'age_group' column based on age and create a categorical code column
metadata.loc[metadata['age'] >= 50, 'age_group'] = 'Older'
metadata['age_group'] = metadata['age_group'].fillna('Younger')
metadata['age_cat'] = pd.Categorical(metadata['age_group']).codes
```

```
#Count the number of images per lesion and create a 'unique' column to indicate if a lesion has only one image
images_per_lesion = metadata["lesion_id"].value_counts()
def is_unique(x):
  if images_per_lesion[x] > 1:
    return False
  else:
    return True

metadata["unique"] = metadata["lesion_id"].map(is_unique)

#Rearrange the columns of metadata
metadata = metadata[['lesion_id', 'image_id', 'unique', 'dx', 'dx_full', 'dx_cat', 'lesion_type', 'lesion_type_cat', 'dx_type', 'age', 'age_group', 'age_cat',
```

```
metadata.head(4)
```

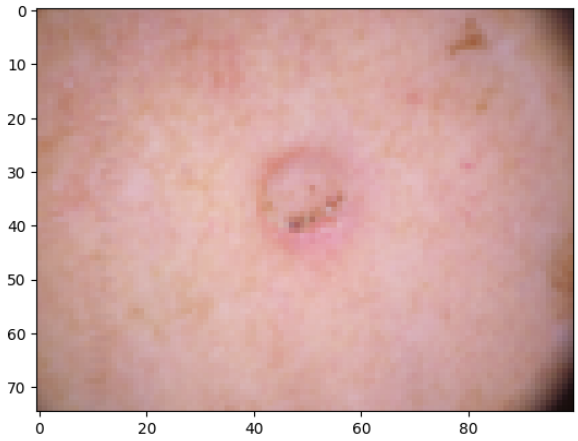
	lesion_id	image_id	unique	dx	dx_full	dx_cat	lesion_type	lesion_type_cat	dx_type	age	age_group	age_cat	sex	sex_cat	localization	l
0	HAM_0000118	ISIC_0027419	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	80.0	Older	0	male	1	scalp	
1	HAM_0000118	ISIC_0025030	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	80.0	Older	0	male	1	scalp	
2	HAM_0002730	ISIC_0026769	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	80.0	Older	0	male	1	scalp	
3	HAM_0002730	ISIC_0025661	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	80.0	Older	0	male	1	scalp	

```
#Load and resize images from 600x450 to 100x75into a numpy array and add it as a new 'image' column in metadata
metadata['image'] = metadata['image_path'].map(lambda x: np.asarray(Image.open(x).resize((100, 75))))
#Display samples of each lesion type in a grid of subplots
fig, m_axs = plt.subplots(7, 5, figsize=(20, 21))
for n_axs, (type_name, type_rows) in zip(m_axs, metadata.sort_values(['dx_full']).groupby('dx_full')):
  n_axs[0].set_title(type_name)
  for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(5, random_state=42).iterrows()):
    c_ax.imshow(c_row['image'])
    c_ax.axis('off')
```




```
plt.imshow(metadata['image'][7])
```

```
<matplotlib.image.AxesImage at 0x7f3615c4d240>
```




```
#Pickle serializes the python objects so they can be saved in a file (like byte stream)and loaded in a program whenever we want in formats

pd.to_pickle(metadata, os.path.join('/content/data_folder', 'data.pkl'))
```

Model

```
# Set seeds
from numpy.random import seed
import tensorflow as tf
seed(42)
tf.random.set_seed(42)
```

```
metadata = pd.read_pickle(os.path.join('/content/data_folder', 'data.pkl'))
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import minmax_scale
```

```
# Encode the relevant features for the metadata model
labelencoder_sex = LabelEncoder()
```

```
metadata['sex'] = labelencoder_sex.fit_transform(metadata['sex'])
metadata['age'] = minmax_scale(metadata['age'])
dummies = pd.get_dummies(metadata.localization)
metadata = pd.concat([metadata, dummies], axis=1)
```

```
metadata.head(2)
```

	lesion_id	image_id	unique	dx	dx_full	dx_cat	lesion_type	lesion_type_cat	dx_type	age	...	ear	face	foot	genital	hand	lower extremity
0	HAM_0000118	ISIC_0027419	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	0.941176	...	0	0	0	0	0	0
1	HAM_0000118	ISIC_0025030	False	bkl	Benign keratosis-like lesions	2	Benign	0	histo	0.941176	...	0	0	0	0	0	0

2 rows × 33 columns

```
# Split 70%, 10%, 20%. Test and validation, together 30%, should be from unique ID's only
size_val_test = 0.3 * len(metadata)
val_test_ratio_from_unique = size_val_test/len(metadata[metadata.unique == True])

# Training metadata = remainder of unique ID's + non-uniques. Split is stratified by lesion type (benign/malignant)
mb_metadata_train_unique, mb_metadata_val_test = train_test_split(metadata[metadata["unique"] == True], test_size = val_test_ratio_from_unique, stratify=metada
mb_metadata_train = pd.concat((mb_metadata_train_unique, metadata[metadata["unique"] == False]), axis = 0)

# Split validation and test sets. Split is stratified by lesion type (benign/malignant)
mb_metadata_validation, mb_metadata_test = train_test_split(mb_metadata_val_test, test_size = 0.6667, stratify=mb_metadata_val_test["lesion_type_cat"], random_

# Class balancing (random oversampling)
X = mb_metadata_train.drop(['lesion_type_cat'], axis=1)
y = mb_metadata_train['lesion_type_cat']

X_test = mb_metadata_test.drop(['lesion_type_cat'], axis=1)
y_test = mb_metadata_test['lesion_type_cat']

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)

ros = RandomOverSampler(random_state=42)
X_test_resampled, y_test_resampled = ros.fit_resample(X_test, y_test)

mb_metadata_train = pd.concat([X_resampled, y_resampled], axis=1)
mb_metadata_test = pd.concat([X_test_resampled, y_test_resampled], axis=1)
```

```
mb_metadata_train.head(2)
```

	lesion_id	image_id	unique	dx	dx_full	dx_cat	lesion_type	dx_type	age	age_group	...	face	foot	genital	hand	lower extremity	neck	s
0	HAM_0000202	ISIC_0027219	True	bkl	Benign keratosis-like lesions	2	Benign	histo	0.588235	Older	...	1	0	0	0	0	0	
1	HAM_0001581	ISIC_0027288	True	nv	Melanocytic nevi	4	Benign	follow_up	0.411765	Younger	...	0	0	0	0	0	0	

2 rows × 33 columns

```
# Create variables for feature (x) training, validation and test sets for the ResNet model
mb_metadata_train_x = np.asarray(mb_metadata_train['image'].tolist())
mb_metadata_validation_x = np.asarray(mb_metadata_validation['image'].tolist())
mb_metadata_test_x = np.asarray(mb_metadata_test['image'].tolist())

# Create variables for feature (x) training, validation and test sets for the ANN model
mb_metadata_train_meta = mb_metadata_train.iloc[:, [8,11,18,19,20,21,22,23,24,25,26,27,28,29,30,31]]
mb_metadata_validation_meta = mb_metadata_validation.iloc[:, [9,12,19,20,21,22,23,24,25,26,27,28,29,30,31,32]]
mb_metadata_test_meta = mb_metadata_test.iloc[:, [8,11,18,19,20,21,22,23,24,25,26,27,28,29,30,31]]

# Create variables for target (y)
mb_metadata_train_y = np.asarray(mb_metadata_train['lesion_type_cat'].tolist())
mb_metadata_validation_y = np.asarray(mb_metadata_validation['lesion_type_cat'].tolist())
mb_metadata_test_y = np.asarray(mb_metadata_test['lesion_type_cat'].tolist())

# One-hot encoding of target variable
mb_num_classes = len(np.sort(metadata['lesion_type_cat'].unique()))

mb_metadata_train_y = to_categorical(mb_metadata_train_y, num_classes = mb_num_classes)
mb_metadata_validation_y = to_categorical(mb_metadata_validation_y, num_classes = mb_num_classes)
mb_metadata_test_y = to_categorical(mb_metadata_test_y, num_classes = mb_num_classes)
```

```
for col in mb_metadata_test.columns:
  print(col)
```

```
lesion_id
image_id
unique
dx
dx_full
```

```
dx_cat
lesion_type
dx_type
age
age_group
age_cat
sex
sex_cat
localization
loc_group
loc_cat
image_path
image
abdomen
acral
back
chest
ear
face
foot
genital
hand
lower extremity
neck
scalp
trunk
upper extremity
lesion_type_cat
```

```
mb_metadata_test_meta.head(3)
```

	age	sex	abdomen	acral	back	chest	ear	face	foot	genital	hand	lower extremity	neck	scalp	trunk	upper extremity
0	0.705882	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0.470588	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.470588	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Performing Data Augmentation

```
# Setting data augmentation parameters
from keras.preprocessing.image import ImageDataGenerator
metadata_augmentation = ImageDataGenerator(
    rotation_range = 60,
    zoom_range = 0.2,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip = True,
    vertical_flip = True,
    shear_range = 10)
```

```
# Fit metadata augmentation
metadata_augmentation.fit(mb_metadata_train_x)
```

ARCHITECTURE

```
from tensorflow.keras.layers import concatenate
```

```
from keras.layers import Dense, Dropout, Flatten, Input,MaxPooling2D,Conv2D
from tensorflow.keras.utils import plot_model
from tensorflow.keras.optimizers.legacy import Adam

# Model parameters
input_shape = (75, 100, 3)
input_shape_meta = mb_metadata_train_meta.shape[1]

optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
epochs = 100
batch_size = 32

# Callbacks
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience=5, verbose=1, factor=0.5, min_lr=0.000001)
early_stopping_monitor = EarlyStopping(patience=20,monitor='val_accuracy')
history = History()

# Create CNN
inp1 = Input(input_shape)

# 1st convolutional layer

x1=Conv2D(96, kernel_size=(11,11), strides=(4,4), activation='relu')(inp1)
x1=BatchNormalization()(x1)
x1=MaxPooling2D(pool_size=(3,3), strides=(2,2))(x1)

# 2nd convolutional layer
x1=Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same")(x1)
x1=BatchNormalization()(x1)
x1=MaxPooling2D(pool_size=(3,3), strides=(2,2))(x1)

# 3rd convolutional layer
x1=Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same")(x1)
x1=BatchNormalization()(x1)

# 4th convolutional layer
x1=Conv2D(filters=384, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same")(x1)
```

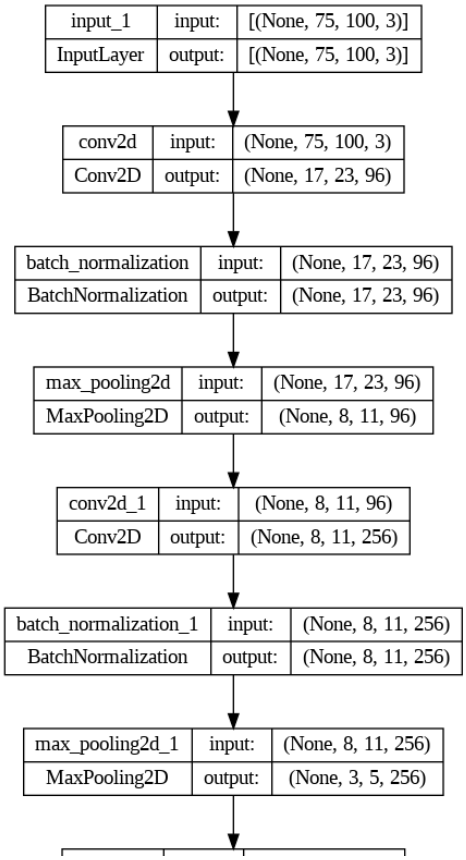
```
x1=BatchNormalization()(x1)

    # 5th convolutional layer
x1=Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same")(x1)
x1=BatchNormalization()(x1)
x1=MaxPool2D(pool_size=(3,3), strides=(2,2))(x1)


x1=Dense(4096, activation='relu')(x1)
x1=Dropout(0.2)(x1)

x1=Dense(4096, activation='relu')(x1)
x1=Dropout(0.2)(x1)
x1=Flatten()(x1)

# Create metadata ANN
inp2 = Input(input_shape_meta)
#.........................................(16 nodes only :) )
x2 = Flatten() (inp2)


# Merge model1 and ANN
merge = concatenate([x1, x2])
merge = Dropout(0.2)(merge)
output = Dense(2, activation='sigmoid')(merge)


# Create final model
Multimodal_binary_model = Model(inputs=[inp1, inp2], outputs=output)

# Layers in ResNet are pretrained (ImageNet)

# Compile model
Multimodal_binary_model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=['accuracy'])

# Output model architecture
print(Multimodal_binary_model.summary())
plot_model(Multimodal_binary_model, to_file=os.path.join('/content/data_folder', 'multimodal_architecture.png'),show_shapes = True)
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 75, 100, 3)]	0	[]
conv2d (Conv2D)	(None, 17, 23, 96)	34944	['input_1[0][0]']
batch_normalization (BatchNormalization)	(None, 17, 23, 96)	384	['conv2d[0][0]']
max_pooling2d (MaxPooling2D)	(None, 8, 11, 96)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 8, 11, 256)	614656	['max_pooling2d[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 8, 11, 256)	1024	['conv2d_1[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 3, 5, 256)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 3, 5, 384)	885120	['max_pooling2d_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 3, 5, 384)	1536	['conv2d_2[0][0]']
conv2d_3 (Conv2D)	(None, 3, 5, 384)	147840	['batch_normalization_2[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 3, 5, 384)	1536	['conv2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 3, 5, 256)	98560	['batch_normalization_3[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 3, 5, 256)	1024	['conv2d_4[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 1, 2, 256)	0	['batch_normalization_4[0][0]']
dense (Dense)	(None, 1, 2, 4096)	1052672	['max_pooling2d_2[0][0]']
dropout (Dropout)	(None, 1, 2, 4096)	0	['dense[0][0]']
dense_1 (Dense)	(None, 1, 2, 4096)	16781312	['dropout[0][0]']
dropout_1 (Dropout)	(None, 1, 2, 4096)	0	['dense_1[0][0]']
input_2 (InputLayer)	[(None, 16)]	0	[]
flatten (Flatten)	(None, 8192)	0	['dropout_1[0][0]']
flatten_1 (Flatten)	(None, 16)	0	['input_2[0][0]']
concatenate (Concatenate)	(None, 8208)	0	['flatten[0][0]', 'flatten_1[0][0]']
dropout_2 (Dropout)	(None, 8208)	0	['concatenate[0][0]']
dense_2 (Dense)	(None, 2)	16418	['dropout_2[0][0]']

```
Total params: 19,637,026
Trainable params: 19,634,274
Non-trainable params: 2,752
```

```
None
```



| conv2d_2 | input: | (None, 3, 5, 256) |

Model Architecture-

|

Train and save model

| batch_normalization_2 | input: | (None, 3, 5, 384) |

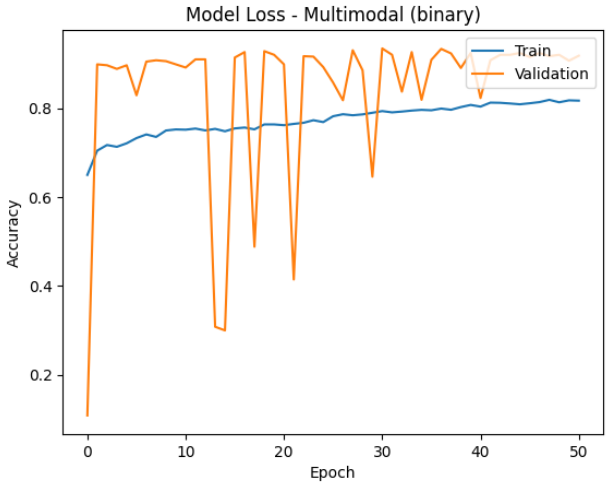
```
mb_history = Multimodal_binary_model.fit(metadata_augmentation.flow(x=[mb_metadata_train_x,mb_metadata_train_meta],y=mb_metadata_train_y,
                        batch_size=batch_size),epochs = epochs, validation_data = ([mb_metadata_validation_x,mb_metadata_validation_meta],mb_metadata_validatio
                        verbose = 1,
                        steps_per_epoch=mb_metadata_train_x.shape[0] // batch_size,
                        callbacks=[learning_rate_reduction,early_stopping_monitor, history])
```

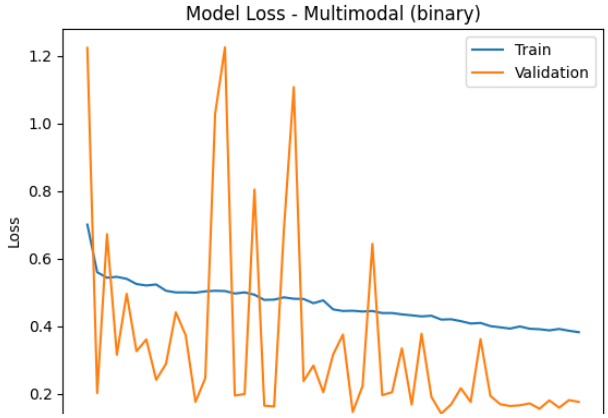
```
Epoch 27/100
325/325 [==============================] - 24s 75ms/step - loss: 0.4447 - accuracy: 0.7864 - val_loss: 0.3748 - val_accuracy: 0.8178 - lr: 5.0000e-04
Epoch 28/100
325/325 [==============================] - 23s 71ms/step - loss: 0.4454 - accuracy: 0.7841 - val_loss: 0.1456 - val_accuracy: 0.9304 - lr: 5.0000e-04
Epoch 29/100
325/325 [==============================] - 24s 73ms/step - loss: 0.4432 - accuracy: 0.7860 - val_loss: 0.2227 - val_accuracy: 0.8854 - lr: 5.0000e-04
Epoch 30/100
325/325 [==============================] - 24s 75ms/step - loss: 0.4447 - accuracy: 0.7897 - val_loss: 0.6434 - val_accuracy: 0.6459 - lr: 5.0000e-04
Epoch 31/100
325/325 [==============================] - 24s 74ms/step - loss: 0.4385 - accuracy: 0.7935 - val_loss: 0.1955 - val_accuracy: 0.9345 - lr: 5.0000e-04
Epoch 32/100
325/325 [==============================] - 25s 76ms/step - loss: 0.4386 - accuracy: 0.7906 - val_loss: 0.2044 - val_accuracy: 0.9202 - lr: 5.0000e-04
Epoch 33/100
325/325 [==============================] - 23s 72ms/step - loss: 0.4347 - accuracy: 0.7924 - val_loss: 0.3342 - val_accuracy: 0.8373 - lr: 5.0000e-04
Epoch 34/100
325/325 [==============================] - 24s 73ms/step - loss: 0.4319 - accuracy: 0.7946 - val_loss: 0.1675 - val_accuracy: 0.9263 - lr: 5.0000e-04
Epoch 35/100
325/325 [==============================] - 24s 73ms/step - loss: 0.4287 - accuracy: 0.7963 - val_loss: 0.3773 - val_accuracy: 0.8188 - lr: 5.0000e-04
Epoch 36/100
325/325 [==============================] - ETA: 0s - loss: 0.4306 - accuracy: 0.7953
Epoch 36: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
325/325 [==============================] - 24s 73ms/step - loss: 0.4306 - accuracy: 0.7953 - val_loss: 0.1908 - val_accuracy: 0.9089 - lr: 5.0000e-04
Epoch 37/100
325/325 [==============================] - 24s 74ms/step - loss: 0.4193 - accuracy: 0.7992 - val_loss: 0.1409 - val_accuracy: 0.9335 - lr: 2.5000e-04
Epoch 38/100
325/325 [==============================] - 24s 74ms/step - loss: 0.4202 - accuracy: 0.7965 - val_loss: 0.1677 - val_accuracy: 0.9232 - lr: 2.5000e-04
Epoch 39/100
325/325 [==============================] - 23s 72ms/step - loss: 0.4146 - accuracy: 0.8025 - val_loss: 0.2161 - val_accuracy: 0.8905 - lr: 2.5000e-04
Epoch 40/100
325/325 [==============================] - 25s 77ms/step - loss: 0.4076 - accuracy: 0.8073 - val_loss: 0.1755 - val_accuracy: 0.9243 - lr: 2.5000e-04
Epoch 41/100
325/325 [==============================] - ETA: 0s - loss: 0.4093 - accuracy: 0.8036
Epoch 41: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
325/325 [==============================] - 24s 74ms/step - loss: 0.4093 - accuracy: 0.8036 - val_loss: 0.3613 - val_accuracy: 0.8229 - lr: 2.5000e-04
Epoch 42/100
325/325 [==============================] - 24s 73ms/step - loss: 0.3999 - accuracy: 0.8125 - val_loss: 0.1936 - val_accuracy: 0.9079 - lr: 1.2500e-04
Epoch 43/100
325/325 [==============================] - 24s 73ms/step - loss: 0.3963 - accuracy: 0.8122 - val_loss: 0.1688 - val_accuracy: 0.9202 - lr: 1.2500e-04
Epoch 44/100
325/325 [==============================] - 24s 74ms/step - loss: 0.3926 - accuracy: 0.8106 - val_loss: 0.1634 - val_accuracy: 0.9202 - lr: 1.2500e-04
Epoch 45/100
325/325 [==============================] - 24s 75ms/step - loss: 0.3989 - accuracy: 0.8088 - val_loss: 0.1655 - val_accuracy: 0.9243 - lr: 1.2500e-04
Epoch 46/100
325/325 [==============================] - ETA: 0s - loss: 0.3922 - accuracy: 0.8111
Epoch 46: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
325/325 [==============================] - 24s 73ms/step - loss: 0.3922 - accuracy: 0.8111 - val_loss: 0.1712 - val_accuracy: 0.9150 - lr: 1.2500e-04
Epoch 47/100
325/325 [==============================] - 24s 74ms/step - loss: 0.3907 - accuracy: 0.8137 - val_loss: 0.1553 - val_accuracy: 0.9243 - lr: 6.2500e-05
Epoch 48/100
325/325 [==============================] - 24s 73ms/step - loss: 0.3873 - accuracy: 0.8188 - val_loss: 0.1801 - val_accuracy: 0.9171 - lr: 6.2500e-05
Epoch 49/100
325/325 [==============================] - 24s 73ms/step - loss: 0.3915 - accuracy: 0.8133 - val_loss: 0.1584 - val_accuracy: 0.9202 - lr: 6.2500e-05
Epoch 50/100
325/325 [==============================] - 24s 75ms/step - loss: 0.3861 - accuracy: 0.8177 - val_loss: 0.1810 - val_accuracy: 0.9069 - lr: 6.2500e-05
Epoch 51/100
325/325 [==============================] - ETA: 0s - loss: 0.3819 - accuracy: 0.8170
Epoch 51: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
325/325 [==============================] - 24s 74ms/step - loss: 0.3819 - accuracy: 0.8170 - val_loss: 0.1755 - val_accuracy: 0.9181 - lr: 6.2500e-05
```

```
Multimodal_binary_model.save(os.path.join('/content/data_folder',"Multimodal_binary_model"))
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_conv
```

```
# Accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Loss - Multimodal (binary)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
filename = str('Model Accuracy - Multimodal (binary).png')
plt.savefig(os.path.join('/content/data_folder',filename), dpi=600)
plt.show()

# Loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss - Multimodal (binary)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
filename = str('Model Loss - Multimodal (binary).png')
plt.savefig(os.path.join('/content/data_folder',filename), dpi=600)
plt.show()
```





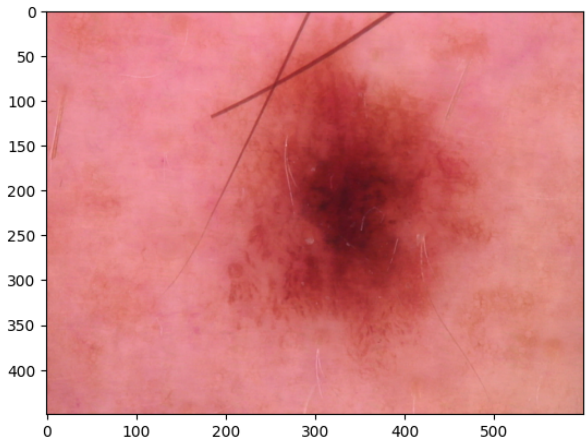
Evaluation

Testing on some images-

```
from PIL import Image as pil_image
from matplotlib.pyplot import imshow, imsave
from IPython.display import Image as Image
```

```
test_image = np.asarray(pil_image.open('test_nv.jpg'))
print('Original Shape of image is : ',test_image.shape)
plt.imshow(test_image)
```

```
Original Shape of image is :  (450, 600, 3)
<matplotlib.image.AxesImage at 0x7f3561d735b0>
```



```
resized_image = np.asarray(pil_image.open('test_nv.jpg').resize((100,75)))
image_array = np.asarray(resized_image.tolist())
test_image2 = image_array.reshape(1,75,100,3)
```

```
plt.imshow(resized_image)
print('New Shape of image is : ',resized_image.shape)
```

```
New Shape of image :  (75, 100, 3)
```

```
mb_metadata_test_meta
```

	age	sex	abdomen	acral	back	chest	ear	face	foot	genital	hand	lower extremity	neck	scalp	trunk	upper extremity
0	0.705882	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0.470588	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.470588	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
3	0.588235	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0.647059	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3499	0.647059	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3500	0.647059	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
3501	0.705882	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3502	0.647059	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3503	0.764706	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

3504 rows × 16 columns

```
mb_metadata_test_x
```

```
array([[[[221, 132, 148],
         [211, 128, 144],
         [225, 142, 160],
         ...,
         [220, 129, 153],
         [218, 129, 149],
         [217, 130, 149]],

        [[214, 128, 143],
         [214, 134, 150],
         [227, 144, 162],
         ...,
         [221, 130, 153],
         [219, 129, 149],
         [217, 128, 147]],

        [[211, 126, 140],
         [217, 137, 152],
         [227, 143, 160],
         ...,
         [224, 136, 155],
         [221, 131, 150],
         [220, 130, 148]],

        ...,

        [[207, 132, 131],
         [209, 131, 133],
         [208, 127, 129],
         ...,
         [207, 133, 137],
         [205, 133, 134],
         [206, 134, 137]],

        [[207, 131, 131],
         [209, 132, 134],
         [208, 131, 135],
         ...,
         [208, 134, 140],
         [205, 132, 136],
         [204, 132, 136]],

        [[210, 135, 139],
         [209, 133, 135],
         [208, 131, 135],
         ...,
         [206, 133, 142],
         [204, 132, 139],
         [202, 130, 139]]],


       [[[183, 178, 178],
         [185, 176, 178],
         [185, 177, 179],
         ...,
         [153, 150, 157],
         [150, 148, 155],
         [144, 143, 150]],
```

```
# Obtain train/validation/test accuracies
mb_loss_train, mb_accuracy_train = Multimodal_binary_model.evaluate([mb_metadata_train_x,mb_metadata_train_meta], mb_metadata_train_y, verbose=1)
```

```
mb_loss_train, mb_accuracy_train = Multimodal_binary_model.evaluate([mb_metadata_train_x,mb_metadata_train_meta], mb_metadata_train_y, verbose=1)
mb_loss_validation, mb_accuracy_validation = Multimodal_binary_model.evaluate([mb_metadata_validation_x,mb_metadata_validation_meta], mb_metadata_validation_y,
mb_loss_test, mb_accuracy_test = Multimodal_binary_model.evaluate([mb_metadata_test_x,mb_metadata_test_meta], mb_metadata_test_y, verbose=1)
```

```
326/326 [==============================] - 2s 7ms/step - loss: 0.3468 - accuracy: 0.8389
31/31 [==============================] - 0s 6ms/step - loss: 0.1755 - accuracy: 0.9181
110/110 [==============================] - 1s 7ms/step - loss: 0.2493 - accuracy: 0.9015
```

```
mb_y_test = np.argmax(mb_metadata_test_y, axis=1)
mb_y_pred_probs = Multimodal_binary_model.predict([mb_metadata_test_x,mb_metadata_test_meta])
mb_y_pred = np.array(list(map(lambda x: np.argmax(x), mb_y_pred_probs)))
```

```
110/110 [==============================] - 1s 5ms/step
```

```
from sklearn.metrics import classification_report
print(classification_report( mb_y_test,mb_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      1752
           1       0.90      0.90      0.90      1752

    accuracy                           0.90      3504
   macro avg       0.90      0.90      0.90      3504
weighted avg       0.90      0.90      0.90      3504
```

```
from sklearn.metrics import precision_score

print("Precision score: {}".format(precision_score(mb_y_test,mb_y_pred,average='weighted')))
```

```
Precision score: 0.9015422732403865
```

```
from sklearn.metrics import f1_score
print("F1 score: {}".format(f1_score(mb_y_test,mb_y_pred,average='weighted')))
```

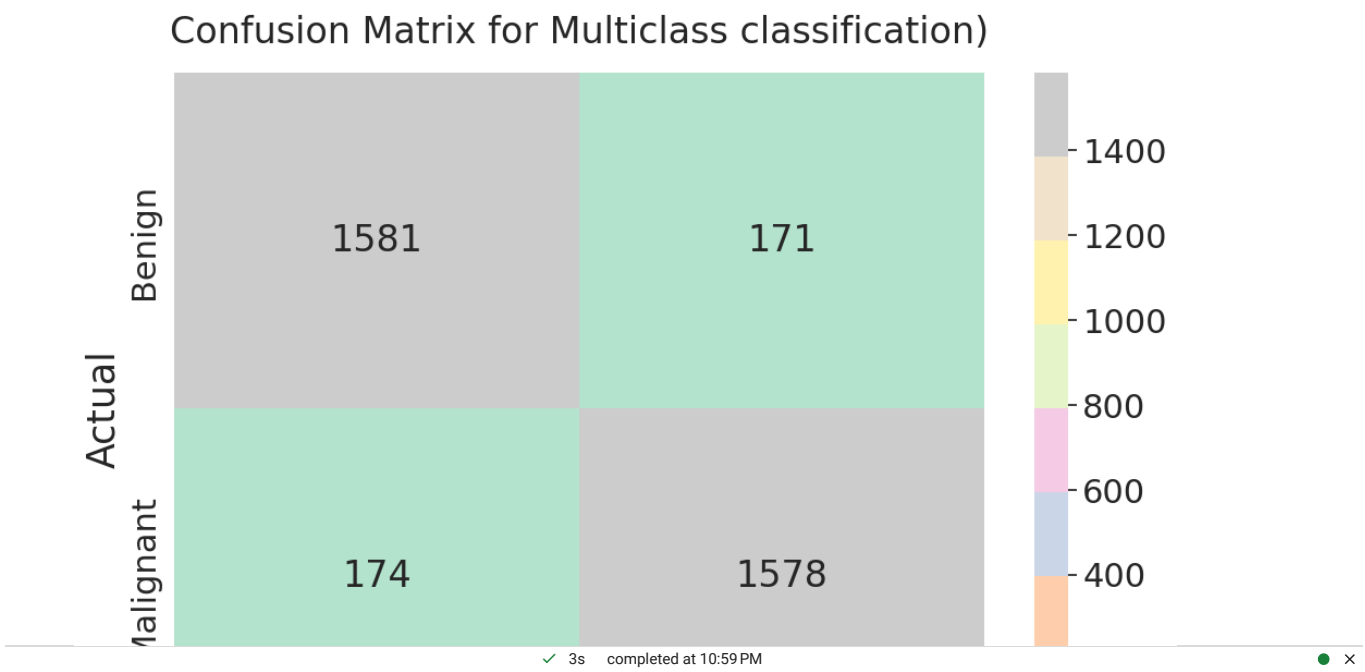
```
F1 score: 0.9015410237182631
```

```
from sklearn.metrics import recall_score
print("Recall score: {}".format(recall_score(mb_y_test,mb_y_pred,average='weighted')))
```

```
Recall score: 0.901541095890411
```

```
lesion_types = ['Benign','Malignant']
```

```
from sklearn.metrics import confusion_matrix
conf_matrix = pd.DataFrame(confusion_matrix(mb_y_test,mb_y_pred),columns = lesion_types, index = lesion_types)
plt.figure(figsize=(12,8))
sns.set(font_scale=2.0)
ax = sns.heatmap(conf_matrix, annot = True, fmt = 'g' ,vmin = 0,cmap = 'Pastel2')
ax.set_xlabel('Predicted',fontsize = 25)
ax.set_xticklabels(ax.get_xticklabels(),rotation =90);
ax.set_ylabel('Actual',fontsize =25)
ax.set_title('Confusion Matrix for Multiclass classification)',fontsize =24,pad=20);
filename = 'Confusion Matrix(binary)'
plt.savefig(os.path.join('/content/data_folder',filename), dpi=600)
plt.show()
```


Confusion Matrix for Multiclass classification)
Actual
Benign
Malignant
1581
171
174
1578
1400
1200
1000
800
600
400
3s completed at 10:59 PM