National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Raj Shrikant Sonawane
Student ID: x21155054

School of Computing
National College of Ireland

Supervisor: Prof. Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Raj Shrikant Sonawane
Student ID:	x21155054
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Prof. Noel Cosgrave
Submission Due Date:	15/12/2022
Project Title:	Skin-Cancer Classification Using Deep Learning with DenseNet and VGG with Streamlit-Framework Implementation
Word Count:	915
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	30th January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Raj Shrikant Sonawane
x21155054

1 Introduction

This research effort uses Deep Learning methods like DenseNet and VGG to categorize skin cancer as benign or malignant and constructed a web-based application using Python's Streamlit-framework. In this configuration manual, all the processes that might be necessary for replication are listed. An explanation of the project design flow from data gathering to model evaluation. As needed, Streamlit implementation and code samples from various sections have also been added.

2 System Configuration:

This research work made advantage of the Kaggle Notebook environment because it was practical to run two Python notebooks simultaneously on the cloud. The configurations available consist of 16 Gigabytes of RAM and 13 Gigabytes of GPU. Kaggle Notebooks offered the GPUP100. For building Streamlit application, Anaconda had to be installed that comprises of Anaconda Prompt, Python Jupyter Notebook. Microsoft office Word and Excel have also been used for Table creation. For pictorial description of work flow of the project and to create image collage draw.io software has been used.

3 Data Selection:

The title of the dataset used in the research was "Skin Cancer: Malignant vs. Benign." After being retrieved from open repositories on Kaggle [1] , it was given this name after the discovery. The data is organized into two unique folders called "Test" and "Train," and each folder contains 1800 photographs. The names of the folders reflect their contents. Each folder contains an additional set of two folders, one for each of the categories referred to as benign and malignant. The initial source of the data that was obtained through Kaggle was the International Skin Imaging Collaboration. The dataset included 1800 images at a resolution of 224 by 224 pixels each. The figure 1 shows importing data images into train and test.

4 Exploratory Data Analysis:

Exploratory data analysis was carried out an to determine how the classification was broken down. Both the training dataset and the testing data have the same ratios as

[1] https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign

```
benign = "../input/skin-cancer-malignant-vs-benign/train/benign/"
malignant = "../input/skin-cancer-malignant-vs-benign/train/malignant/"

benign_paths = os.listdir(benign)

malignant_paths = os.listdir(malignant)

l = []

for x in benign_paths :
    l.append(['../input/skin-cancer-malignant-vs-benign/train/benign/'+x,"Benign"])

for x in malignant_paths :
    l.append(['../input/skin-cancer-malignant-vs-benign/train/malignant/'+x,"Malignant"])

train = pd.DataFrame(l,columns = ["IMAGE_PATH","DISEASE_TYPE"])

train
```

(a) Code for Train Data

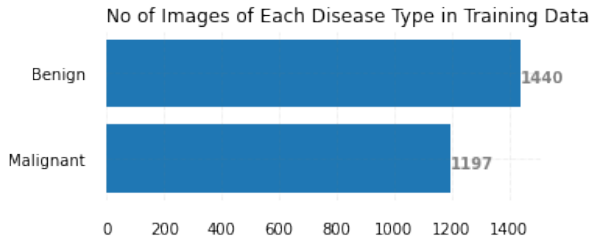
Out[3]:

	IMAGE_PATH	DISEASE_TYPE
0	../input/skin-cancer-malignant-vs-benign/train...	Benign
1	../input/skin-cancer-malignant-vs-benign/train...	Benign
2	../input/skin-cancer-malignant-vs-benign/train...	Benign
3	../input/skin-cancer-malignant-vs-benign/train...	Benign
4	../input/skin-cancer-malignant-vs-benign/train...	Benign
...	...	...
2632	../input/skin-cancer-malignant-vs-benign/train...	Malignant
2633	../input/skin-cancer-malignant-vs-benign/train...	Malignant
2634	../input/skin-cancer-malignant-vs-benign/train...	Malignant
2635	../input/skin-cancer-malignant-vs-benign/train...	Malignant
2636	../input/skin-cancer-malignant-vs-benign/train...	Malignant

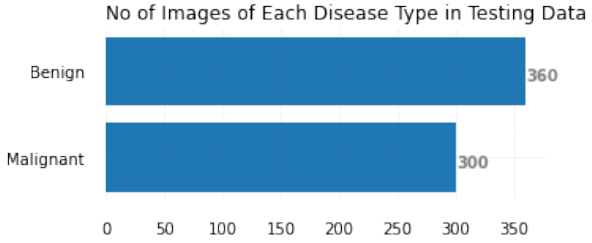
2637 rows × 2 columns

(b) Output for Train Data

Figure 1: Fetching Train data from Kaggle



(a) Class Balance for Training data



(b) Class Balance for Training data

Figure 2: Class Balance of Dataset

shown in 2. Visual representation of the data was the most efficient way to determine how the facts were balanced. Bar graph was plotted to depict the distribution of the categories. According to statistics, 45% of the data for the train set and 55% of the data for the test set respectively, are photos of malignant skin tumors.

5 Cleaning the Data:

The image dataset was downloaded from the Kaggle repositories using the appropriate software. The information that is related to the datasets that are made available by the ISIC is examined in order to determine whether or not any values are absent and whether or not there are any files present in the directory. The dataset that was retrieved contains no null or missing values, and this is the case for both the train and test files. This is because there are no missing or null values in the original dataset.

6 Importing Libraries.

The necessary python libraries required have been download using ‘pip’ and ‘pip3’ commands. For example, streamlit framework had to be installed in the system. The Anacondra Prompt tool had to be launched and command “pip install streamlit” and been used. Similarly to run Streamlit on local machine, OpenCV, Matlpotlib and TensorFLow had to be installed with same command such as ‘pip install package name’.

```python
# importing the required libraries and modules
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os
```

```python
# importing required layers from tensoforlow library
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation,
MaxPool2D, UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
import tensorflow as tf
from tensorflow.keras.models import Model
```

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# data generators are python concept to pass data to model and defining
#data augmentation techniques as well to increase the training data set
```

```python
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
red = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, verbose=1, min_lr=1e-3)
checkpoint = ModelCheckpoint('best_epoch_model.h5', verbose=1, save_best_only=True)
```

```python
from math import *
from sklearn import metrics
from math import sqrt

from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report

import numpy as np
import cv2
from tqdm import tqdm
from skimage.transform import resize
```

(a) Class Balance for Training data

```python
import streamlit as st
import altair as altc
import pandas as pd
import numpy as np
import os
import urllib
import cv2
from PIL import Image
import cv2
import matplotlib.pyplot as plt
import PIL

import tensorflow as tf


from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, AveragePooling2D,
GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import Model, Sequential, Input
from tensorflow.keras.models import load_model


from tensorflow.keras.applications import VGG19


from tensorflow.keras.applications import DenseNet121
```

(b) Class Balance for Training data

Figure 3: Importing Libraries

Figure 4: Data Augmentation

For building Deep Learning models, Kaggle notebook was used. Kaggle notebooks already have packages installed and only importing code is required. See figure 3

7 Data Augmentation:

The Keras data generator is implemented for data augmentation in the code snippet Figure 4. Data augmentation was accomplished by using the ImageDataGenerator module. The idea of an ImageDataGenerator in Python is useful for both explaining data augmentation techniques and providing input to a model.

```python
from tensorflow.keras.applications import DenseNet201
# DenseNet201 model

pre_trained = DenseNet201(weights='imagenet', include_top=False)
# calling densenet201 model by removing top layer of 1000 neuron

for layer in pre_trained.layers:
    # making all layers of pretrained model non trainable
    layer.trainable = False


input_l = Input(shape=(224, 224, 3))
# defining input layer with shape 224,224,3

pre_trained_output = pre_trained(input_l)
# passing input layer to pretrained layer to extract convolutional filter maps

l = GlobalAveragePooling2D()(pre_trained_output)
# applying global average pooling layer to the extracted output from pre trained model

l = Dense(256, activation='relu')(l)
# dense layer with 256 neurons and activation relu (because to learn non linear patterns)

l = Dropout(0.25)(l)
# drop out layer with drop out rate 0.25 to avoid overfitting and underfitting

l = BatchNormalization()(l)
# adding batch normalization layer to speed up the training
#process and in less number of epochs to obtain optimized value

l = Dense(128, activation='relu')(l)
# dense layer with 128 neurons and activation relu

output_l = Dense(1,activation = 'sigmoid')(l)
# dense layer with 1 neurons and activation sigmoid (because of binary classification)
```

Figure 5: DenseNet-201

```python
from tensorflow.keras.applications import DenseNet121

pre_trained = DenseNet121(weights='imagenet', include_top=False)

for layer in pre_trained.layers:
    layer.trainable = False


input_l = Input(shape=(224, 224, 3))

pre_trained_output = pre_trained(input_l)

l = GlobalAveragePooling2D()(pre_trained_output)

l = Dense(512, activation='relu')(l)

l = Dense(256, activation='relu')(l)

l = Dropout(0.25)(l)

l = BatchNormalization()(l)

l = Dense(128, activation='relu')(l)

output_l = Dense(1,activation = 'sigmoid')(l)

model = Model(input_l,output_l)
```

(a) DenseNet-121

```python
from tensorflow.keras.applications import DenseNet169

pre_trained = DenseNet169(weights='imagenet', include_top=False)

for layer in pre_trained.layers:
    layer.trainable = False


input_l = Input(shape=(224, 224, 3))

pre_trained_output = pre_trained(input_l)

l = GlobalAveragePooling2D()(pre_trained_output)

l = Dense(256, activation='relu')(l)

l = Dropout(0.25)(l)

l = BatchNormalization()(l)

l = Dense(128, activation='relu')(l)

output_l = Dense(1,activation = 'sigmoid')(l)

model = Model(input_l,output_l)
```

(b) DenseNet-169

Figure 6: DenseNet

8 Building Model Architecture:

This section will show code snippet of only top model architecture and layer configuration from DensetNet121, DenseNet169, DenseNet201, VGG-16 and VGG-19 in figure **??** and 7.

8.1 Model Summary

Figure 8 shows model summary of any deep learning architecture.

8.2 Model Configuration

Code snippet in figure 9 shows loss function, optimizer and metrics set for the model.

8.3 Epochs setting

Figure 10 shows the number of Epochs set with the EarlyStopping function

```python
from tensorflow.keras.applications import VGG16

pre_trained = VGG16(weights='imagenet', include_top=False)

for layer in pre_trained.layers:
    layer.trainable = False


input_1 = Input(shape=(224, 224, 3))

pre_trained_output = pre_trained(input_1)

l = GlobalAveragePooling2D()(pre_trained_output)

l = Dense(256, activation='relu')(l)

l = Dense(128, activation='relu')(l)

l = Dropout(0.25)(l)

l = BatchNormalization()(l)

l = Dense(64, activation='relu')(l)

output_1 = Dense(1,activation = 'sigmoid')(l)

model = Model(input_1,output_1)
```

(a) VGG-16

```python
from tensorflow.keras.applications import VGG19

pre_trained =VGG19(weights='imagenet', include_top=False)

for layer in pre_trained.layers:
    layer.trainable = False


input_1 = Input(shape=(224, 224, 3))

pre_trained_output = pre_trained(input_1)

l = GlobalAveragePooling2D()(pre_trained_output)

l = Dense(512, activation='relu')(l)

l = Dense(256, activation='relu')(l)

l = Dropout(0.25)(l)

l = BatchNormalization()(l)

l = Dense(128, activation='relu')(l)

output_1 = Dense(1,activation = 'sigmoid')(l)

model = Model(input_1,output_1)
```

(b) VGG-19

Figure 7: VGG

```
[10]: model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
densenet201 (Functional)     (None, None, None, 1920)  18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 256)               491776
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 18,847,809
Trainable params: 525,313
Non-trainable params: 18,322,496
```

Figure 8: Model Summary

```python
# configuring the model to train

# loss function is binary cross entropy because of binary classification
# optimizer is adam
# metrics are accuracy , recall , precision
model.compile(loss='binary_crossentropy', optimizer="adam",
              metrics=['accuracy',tf.keras.metrics.Recall(),tf.keras.metrics.Precision(),tf.keras.metrics.AUC()])
```

Figure 9: Model Configuration

```
# model training should be started with fit method

# number of epochs are defined to 1000 useful for cost function optimizing and can be changed to other value
#as well but limiting the computation time kept 1000
#while applying EarlyStopping Function if patience is not increased

red = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, verbose=1, min_lr=1e-3)
checkpoint = ModelCheckpoint('best_epoch_model.h5', verbose=1, save_best_only=True)
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
epochs=1000
modelhistory = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=test_generator,
    callbacks=[red, checkpoint,callback]
    )
```

Figure 10: Epochs settings

```
def performance_metrics(y_true,y_pred):

    cm = metrics.confusion_matrix(y_true, y_pred)

    fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(5, 5), cmap=plt.cm.Greens ,class_names=['Benign','
    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix', fontsize=18)
    plt.show()

    cm_df = pd.DataFrame(cm,
            columns = ['Predicted Negative', 'Predicted Positive'],
            index = ['Actual Negative', 'Actual Positive'])

    TP = cm[1][1]
    TN = cm[0][0]
    FP = cm[0][1]
    FN = cm[1][0]
```

```
    # accuracy
    Accuracy = ((TP+TN) / (TP + TN + FP + FN))*100

    #FAR represented the probability in which a record is incorrectly classif
    FAR =(FP + FN)/(FP + FN + TP + TN)*100

    ## sensitivity
    Sensitivity = (TP / (TP + FN))*100

    # specificity
    Specificity = (TN / (TN + FP))*100

    # False positive rate (FPR)
    FPR = (FP/(FP + TN))*100

    # False negative rate (FNR)
    FNR = (FN/(TP + FN))*100

    # predict probabilities
    Auc = roc_auc_score(y_true, y_pred)*100

    # precision tp / (tp + fp)
    Precision = precision_score(y_true, y_pred)*100

    # recall: tp / (tp + fn)
    Recall = recall_score(y_true, y_pred)*100

    # f1: 2 tp / (2 tp + fp + fn)
    F1 = f1_score(y_true, y_pred)*100

    return(Accuracy,FAR,FPR,FNR,Sensitivity,Specificity,Auc,Precision,Recall,F1)
```

Figure 11: Evaltuation Metrics

9 Evaluation Metrics

To evaluate the model performance, a code snippet from Figure 11 has been created.

9.1 Evaluation Metrics Graph

Model's train and test evaluation metrics such as Accuracy, Precision, Recall, AUC, and Loss are shown in figure 12. This configuration Manual document has only mentioned the Evaluation Metrics Graph of the top-performing model which is DenseNet-121 with 3 layers.

9.2 Confusion Matrix

The confusion matrix created for Training and Testing data is shown in figure 13 .

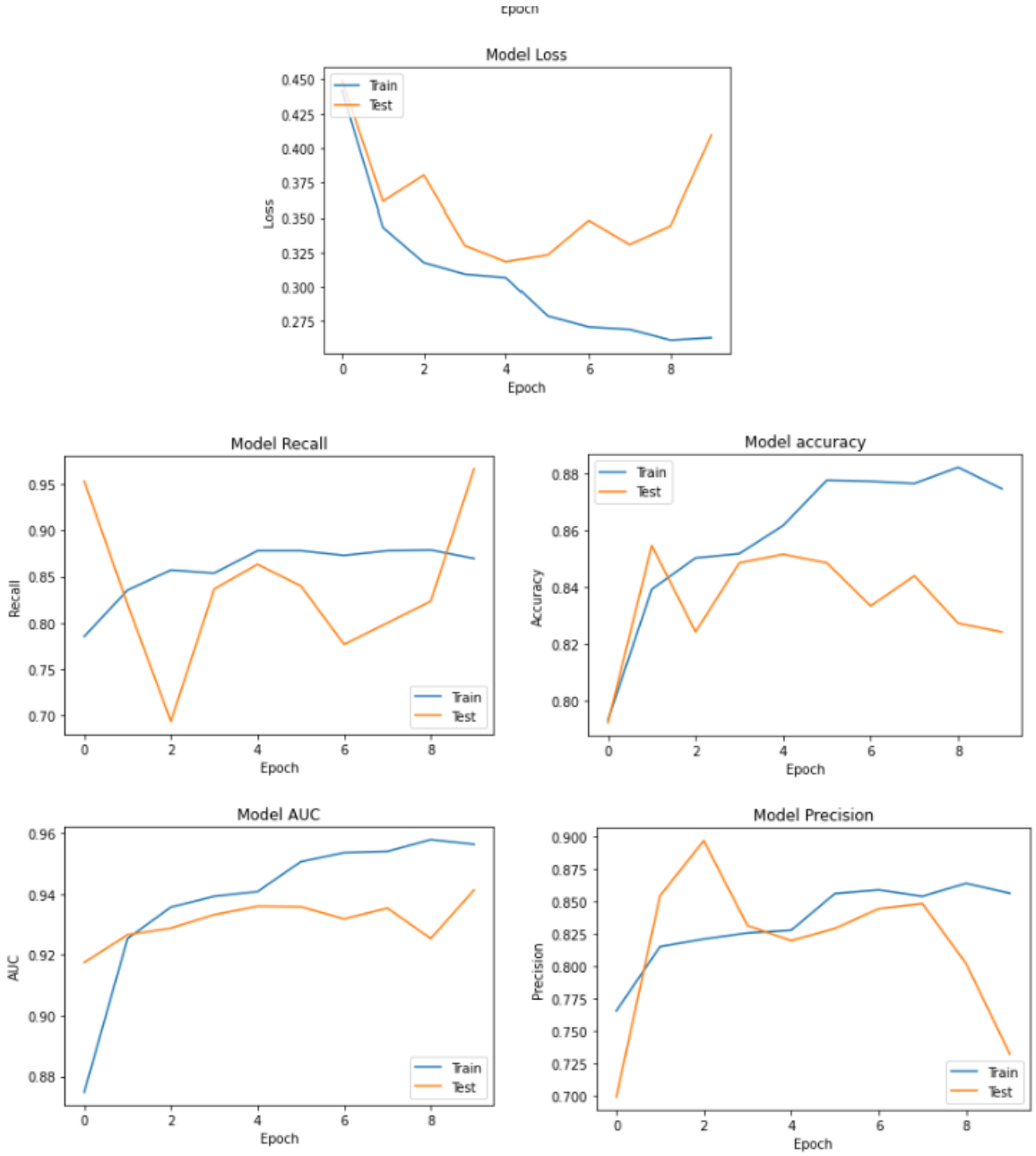


Figure 12: Evaluation Metrics Graphs

Figure 13: Confusion Matrix for Training and Testing Data

10 Streamlit-Framework Architecture

This streamlit app has 2 pages: one to describe the project, disease, and why the app is used, and another to make predictions by uploading skin cancer photographs to a pre-trained model that predicts the class and probability as shown in Figure 14

The snapshot of page 1 and page 2 are shown in figure 15

Following are the steps to launch Streamlit application:

- Open CMD
- Change the directory to folder where Python and H5 file is present.
- Run command 'streamlit run app.py'

```
st.sidebar.header("Skin Classification")
page = st.sidebar.selectbox(
    "Select Activity", ["Disease Information", "Skin Cancer Prediction"])


if page == "Disease Information":

    st.header("Benign vs Melignant Skin Classification")
    st.write("")
    st.write("")
    st.subheader("Cancer of the skin is among the most dangerous forms of the disease, as it is one of the most likely to result in
mortality due to DNA damage. This faulty DNA causes cells to proliferate uncontrollably, a phenomenon that is becoming increasingly common
in modern times. There has been some study done on computerized methods for analyzing skin lesions for signs of cancer. Streamlit is a web-
based interface that employs the weights of Deep Learning Models to categorize a given image as either benign or malignant along with a
probability score.")
    st.write("")

if page == "Skin Cancer Prediction":

    uploaded_file = st.file_uploader("Upload Skin image file")

    if uploaded_file is not None:

        image = Image.open(uploaded_file)

        c = image.copy()

        st.subheader("Skin Image")
        st.image(PIL.Image.fromarray(
            np.uint8(np.asarray(c))).resize((224, 224)))

        image = np.asarray(image)

        norm_img = image/255

        norm_img = norm_img.astype(np.float32)

        final_img = norm_img.reshape((1, 224, 224, 3))

        prob = model.predict(final_img)[0]

        if prob >= 0.5:
            p = "Malignant"
            r = model.predict(final_img)[0]
        else:
            p = "Benign"
            r = 1 - model.predict(final_img)[0]

        st.subheader(f"Predicted Disease : {p}")

        st.subheader("Predicted " + p+" Probability : "+str(r))
```

Figure 14: Streamlit-Framework Architecture

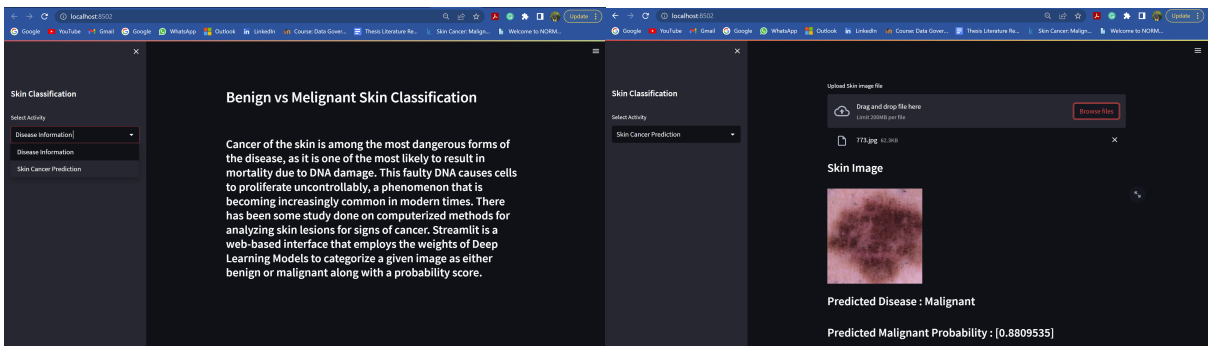


Figure 15: Streamlit Web-application for Skin Cancer Classification