SKIN CANCER DETECTION USING TRANSFER LEARNING & ENSEMBLE MODELLING

Under the Guidance of

Dr. Agilandeeswari L

(Associate Professor, School of Information Technology and Engineering)

For the Course

ITE1015 – Soft Computing

For Winter Semester 2020-21

Project Team:

Aashish Bansal 19BIT0346

Perumalla Sasank 19BIT0338

Keerthi Yasasvi 19BIT0335

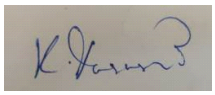
DECLARATION

We, hereby, declare that the thesis entitled "Skin Cancer Detection Using Transfer Learning and Ensemble Modelling" submitted by us, for the award of the degree of *Bachelor of Technology in Information Technology* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Agilandeeswari L.

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other Degree or Diploma in this Institute or any other Institute or University.

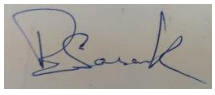
Place: Vellore Date: 17th May 2021

15-05-2021

X 

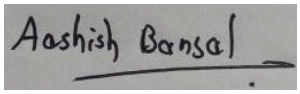
Keerthi Yasasvi 19BIT0335
Interface Developer
Signed by: 47316915-f1de-4b88-8c64-c2cc8414067c

15-05-2021

X 

Perumalla Sasank 19BIT0338
Interface Developer
Signed by: 47316915-f1de-4b88-8c64-c2cc8414067c

15-05-2021

X 

Aashish Bansal 19BIT0346
Learning Model Developer
Signed by: 47316915-f1de-4b88-8c64-c2cc8414067c

Signature of the Candidates

ACKNOWLEDGEMENTS

We would like to thank our mentor and the faculty in-charge for the course ITE1015 – Soft Computing, Dr. Agilandeeshwari L. for approving the project and being a mentor and guide for the project and helping us create, improve and bring out the best of our abilities to improve our skill and knowledge for learning, implementing and executing the project.

For the Project Team, we would to thank each other for helping each other in the hardships faced and sharing the joy of reaching a milestone and being a true support for each other at every step of the project. We had,

Aashish Bansal for Preparing and Training the Model and Documentation

Perumalla Sasank and Keerthi Yasasvi for Interface Development, Design and Documentation

EXECUTIVE SUMMARY

Vellore Institute of Technology, Vellore

Faculty Name: Agilandeeshwari L.

10th May, 2021

EXECUTIVE SUMMARY: SKIN CANCER

Skin Cancer is categorized into two categories namely: malignant melanoma(lethal) and benign(non-lethal/treatable). Evolution in technology has brought upon many boon's as well as banes and Cancer is one of them, skin cancer becoming a common side effect due to exposure to a lethal level of radiation or due to genetic disorder from exposure to radiation. The opportunity for machine learning models to be implemented, recommendations and costs are mentioned in this article.

OPPORTUNITY

With boons of advancement in technology machine learning models can be implemented and trained to detect, diagnose and possibly prevent any benign case from turning into a malignant case thus, advancing the medical infrastructure. Currently the main challenges are the cost-effective models to train, economic viability and error detection rates varying from ethnicity/race and a few specific cases.

SOLUTION/RECOMMENDATION

Training of a model with Mobilenet, Inception and Xception Architecture gives an economically viable solution and a detection rate of above 60% which as per the IEEE standards is viable for use practically. Recurring training of various Skin Cancer training data based on specific individuals with previously known other skin disorders and others on their race/ethnicity as to increase the detection rate without overfitting the model.

Table of Contents

TABLE OF FIGURES

TABLE OF TABULAR INFORMATION

ABBREVIATIONS

- CNN – Convolution Neural network
- GUI – Graphics User interface
- UV light – Ultra Violet light
- OS – Operating System
- ResNet – Residential Energy Services Network
- CSV – Comma Separated Values
- ROC – Receiver Operating Circuit
- EDA – Exploratory Data Analysis
- MCAR - Missing Completely At Random
- MAR - Missing At Random
- MNAR - Missing Not at Random
- ISIC – international Standard industrial classification
- SIIM – Society for Imaging Informatics in Medicine
- U.S. – United States of America
- SPF – Sun Protection Factors
- Exe File – Executable File
- VGG - Visual Geometry Group
- ROC - Receiving Operating Characteristic Curve
- EDA – Exploratory Data Analysis
- np – Numpy
- 4D – Four Dimensional
- ML – Machine Learning
- H5 - Hierarchical Data Format
- TFLITE – Tensorflow Lite
- API – Application Programming Interface

SYMBOLS AND NOTATIONS

Symbol	Meaning
-	Subtraction
+	Addition
/	Division
()/x	Multiplication (multiplication of enclosed value and value outside the enclosed bracket)

SKIN CANCER DETECTION USING TRANSFER LEARNING AND ENSEMBLE MODELING

PROPOSED ARCHITECTURE

ABSTRACT

The project is a Transfer Learning and Ensemble Learning trained model which consists of the MobileNet, Inception and Xception Learning pre-trained Models which can predict whether the patient has a suffering from Skin Cancer or not by checking the scanning and predicting on the images of the infected areas on the body. The model has been trained on a variety of images through which it predicts the required.

In this project, the image file of the patient is upload into a general public use software, which is a GUI-based interface, developed with the help of Tkinter and Python, and it consists of the model saved as a file and the software passes the image through the function and uses the results obtained that to analyse the image and give the prediction which can help people and doctors to start with the medication way earlier instead of waiting for the laboratory tests and reports for the confirmation, which are also very expensive.

We know about Skin Cancer that Skin cancer is an abnormal growth of skin cells. Most skin cancers are caused by exposure to ultraviolet (UV) light. When the skin is not protected, UV rays from sunlight or tanning beds can damage and alter skin's DNA that leads to the cancer.

Now, Deep learning model has been built to classify and identify the binary diagnostic group of melanocytic images obtained through dermoscopy. And based on the model, disease detection through dermal cell images has been investigated, and classifications on dermal cell images have been performed.

INTRODUCTION

OBJECTIVE

In this project, we plan on taking the image of the skin disease of a patient and upload and pass it through the machine learning model which we have created and and based on the analysis of the image by the model, we plan on predicting whether the person is suffering from Skin Cancer or not.

We intent to make a free software in the interest of the benefit of the general public and can be used by anyone and everyone and they can use it to check for the possibility of Skin Cancer with the ease of being at there home by uploading a clear image of the infected area.

The image file of the patient is upload into a software, which is GUI-based interface, developed with the help of Tkinter, and it consists of the model saved as a file and the software uses that to analyse the image and give the prediction which can help doctors to start with the medication way faster instead of waiting for the laboratory reports for the confirmation.

MOTIVATION

Skin cancer is an alarming disease for mankind. The necessity of early diagnosis of the skin cancer have been increased because of the rapid growth rate of Melanoma skin cancer, its high treatment costs, and death rate. This cancer cells are detected manually and it takes time to cure in most of the cases. We even intend to help people in the pandemic time so that people do not have to risk there lives on getting the laboratory tests.

The features of the affected skin cells are extracted after the segmentation of the dermoscopic images using feature extraction technique.

We want to build an application which we can deploy in the production for the general public and can be used by anyone and everyone for their benefit to start early with the medication and precautions.

BACKGROUND

Skin cancer is an abnormal growth of skin cells. Most skin cancers are caused by exposure to ultraviolet (UV) light. When the skin is not protected, UV rays from sunlight or tanning beds can damage and alter skin's DNA that leads to the cancer.

Deep learning model has been built to classify and identify the binary diagnostic group of melanocytic images obtained through dermoscopy. Based on the model, disease detection through dermal cell images has been investigated, and classifications on dermal cell images have been performed.

ABOUT THE DISEASE

Skin cancer is the most prevalent type of cancer. Melanoma, specifically, is responsible for 75% of skin cancer deaths, despite being the least common skin cancer. The American Cancer Society estimates over 100,000 new melanoma cases will be diagnosed in 2020. It's also expected that almost 7,000 people will die from the disease. As with other cancers, early and accurate detection—potentially aided by data science—can make treatment more effective.

Currently, dermatologists evaluate every one of a patient's moles to identify outlier lesions or “ugly ducklings” that are most likely to be melanoma. Existing AI approaches have not adequately considered this clinical frame of reference. Dermatologists could enhance their diagnostic accuracy if detection algorithms take into account “contextual” images within the same patient to determine which images represent a melanoma. If successful, classifiers would be more accurate and could better support dermatological clinic work.

As the leading healthcare organization for informatics in medical imaging, the Society for Imaging Informatics in Medicine (SIIM)'s mission is to advance medical imaging informatics through education, research, and innovation in a multi-disciplinary community. SIIM is joined by the International Skin Imaging Collaboration (ISIC), an international effort to improve melanoma diagnosis. The ISIC Archive contains the largest publicly available collection of quality-controlled dermoscopic images of skin lesions.

In this competition, you’ll identify melanoma in images of skin lesions. In particular, you’ll use images within the same patient and determine which are likely to represent a melanoma. Using patient-level contextual information may help the development of image analysis tools, which could better support clinical dermatologists.

Melanoma is a deadly disease, but if caught early, most melanomas can be cured with minor surgery. Image analysis tools that automate the diagnosis of melanoma will improve dermatologists' diagnostic accuracy. Better detection of melanoma has the opportunity to positively impact millions of people.

FACTS ON MELANOMA SKIN CANCER

- It's estimated that the number of new melanoma cases diagnosed in 2021 will increase by 5.8 percent.
- The number of melanoma deaths is expected to increase by 4.8 percent in 2021.
- An estimated 207,390 cases of melanoma will be diagnosed in the U.S. in 2021. Of those, 106,110 cases will be in situ (non-invasive), confined to the epidermis (the top layer of skin), and 101,280 cases will be invasive, penetrating the epidermis into the skin's second layer (the dermis). Of the invasive cases, 62,260 will be men and 43,850 will be women.
- In the past decade (2011 – 2021), the number of new invasive melanoma cases diagnosed annually increased by 44 percent
- An estimated 7,180 people will die of melanoma in 2021. Of those, 4,600 will be men and 2,580 will be women.
- The vast majority of melanomas are caused by the sun. In fact, one UK study found that about 86 percent of melanomas can be attributed to exposure to ultraviolet (UV) radiation from the sun.
- Compared with stage I melanoma patients treated within 30 days of being biopsied, those treated 30 to 59 days after biopsy have a 5 percent higher risk of dying from the disease, and those treated more than 119 days after biopsy have a 41 percent higher risk.
- Across all stages of melanoma, the average five-year survival rate in the U.S. is 93 percent. The estimated five-year survival rate for patients whose melanoma is detected early is about 99 percent. The survival rate falls to 66 percent when the disease reaches the lymph nodes and 27 percent when the disease metastasizes to distant organs.
- Only 20 to 30 percent of melanomas are found in existing moles, while 70 to 80 percent arise on apparently normal skin.
- On average, a person's risk for melanoma doubles if they have had more than five sunburns, but just one blistering sunburn in childhood or adolescence more than doubles a person's chances of developing melanoma later in life.
- Regular daily use of an SPF 15 or higher sunscreen reduces the risk of developing melanoma by 50 percent.
- Melanoma accounts for 6 percent of new cancer cases in men, and 5 percent of new cancer cases in women.
- Men age 49 and under have a higher probability of developing melanoma than any other cancer.
- From ages 15 to 39, men are 55 percent more likely to die of melanoma than women in the same age group.
- Women age 49 and under are more likely to develop melanoma than any other cancer except breast and thyroid cancers.
- From age 50 on, significantly more men develop melanoma than women. The majority of people who develop melanoma are white men over age 55. But until age 49, significantly more non-Hispanic white women develop melanoma than white men (one in 156 women versus one in 230 men). Overall, one in 27 white men and one in 40 white women will develop melanoma in their lifetime.

STATISTICS ON SKIN CANCER

SKIN CANCER RATES: BOTH GENDERS

Australia had the highest rate of melanoma in 2018, followed by New Zealand.

Table 1: Skin Cancer Rates: Both Gender

Rank	Country	Age-standardised rate per 100,000
1	Australia	33.6
2	New Zealand	33.3
3	Norway	29.6
4	Denmark	27.6
5	Netherlands	25.7
6	Sweden	24.7
7	Germany	21.6
8	Switzerland	21.3
9	Belgium	19.9
10	Slovenia	18.6
11	Luxembourg	16.5
12	Ireland	16.3
13	Finland	15.8
14	UK	15.0
15=	Austria	13.6
15=	France (metropolitan)	13.6
17	US	12.7

18	Czech Republic	12.6
19=	Canada	12.4
19=	Italy	12.4

SKIN CANCER RATES IN MEN

Australia had the highest rate of melanoma in men in 2018, followed by New Zealand.

Table 2: Skin Cancer Rates in Men

Rank	Country	Age-standardised rate per 100,000
1	Australia	40.4
2	**New Zealand**	**35.8**
3	Norway	29.0
4	**Netherlands**	**26.4**
5	Sweden	23.5
6	**Switzerland**	**23.4**
7	Denmark	22.4
8	**Germany**	**19.6**
9	Luxembourg	18.1
10	**Slovenia**	**18.0**
11=	Belgium	16.2
11=	**Finland**	**16.2**
13=	Austria	15.0
13=	**UK**	**15.0**
15	US	14.9

16	**France (metropolitan)**	**14.4**
17	Italy	14.0
18	**Ireland**	**13.6**
19	Canada	13.4
20	**Czech Republic**	**13.3**

SKIN CANCER RATES IN WOMEN

Denmark had the highest rate of melanoma in women in 2018, followed by New Zealand.

Table 3: Skin Cancer Rates in Women

Rank	Country	Age-standardised rate per 100,000
1	Denmark	33.1
2	New Zealand	31.1
3	Norway	30.7
4	Australia	27.5
5	Sweden	26.2
6	Netherlands	25.4
7	Germany	24.0
8	Belgium	23.9
9	Slovenia	19.7
10	Switzerland	19.5
11	Ireland	19.0
12	Finland	15.9
13	Luxembourg	15.4

14	UK	15.3
15	France (metropolitan)	12.9
16	Austria	12.6
17	Czech Republic	12.4
18=	Canada	11.7
18=	Iceland	11.7
20	Estonia	11.4
21=	Italy	11.0
21=	US	11.0
23	Greece	10.3
24	Hungary	10.1
25	Lithuania	9.0

PROJECT DESCRIPTION AND GOALS

As mentioned before, in this project, the image file of the patient is upload into a software, which is GUI-based interface, developed with the help of Tkinter, and it consists of the model saved as a file and the software uses that to analyze the image and give the prediction which can help doctors to start with the medication way faster instead of waiting for the laboratory reports for the confirmation.

The learning model is created with the help of the following pre-trained architectures: MobileNet, Inception and Xception.

The MobileNet Architecture is based on a streamlined architecture that uses a depth-wise separable convolution to build a light-weight deep neural network. We introduce two simple global hyper-parameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

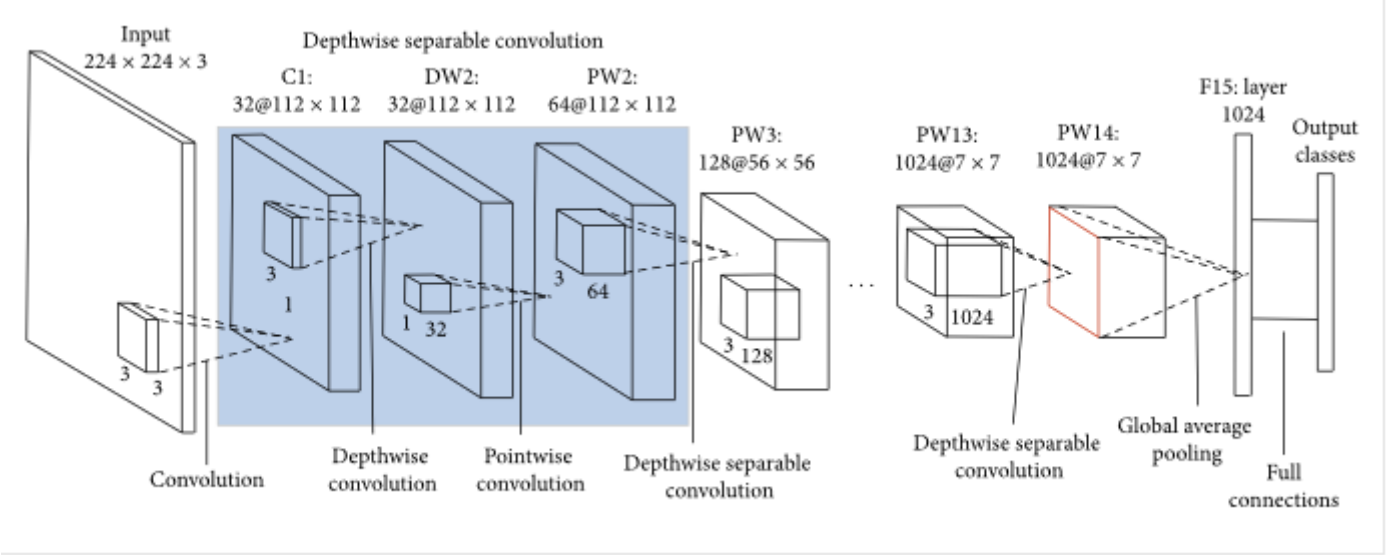


Figure 1: MobileNet Architeure (Source: Analytics Vidya)

The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.

The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.

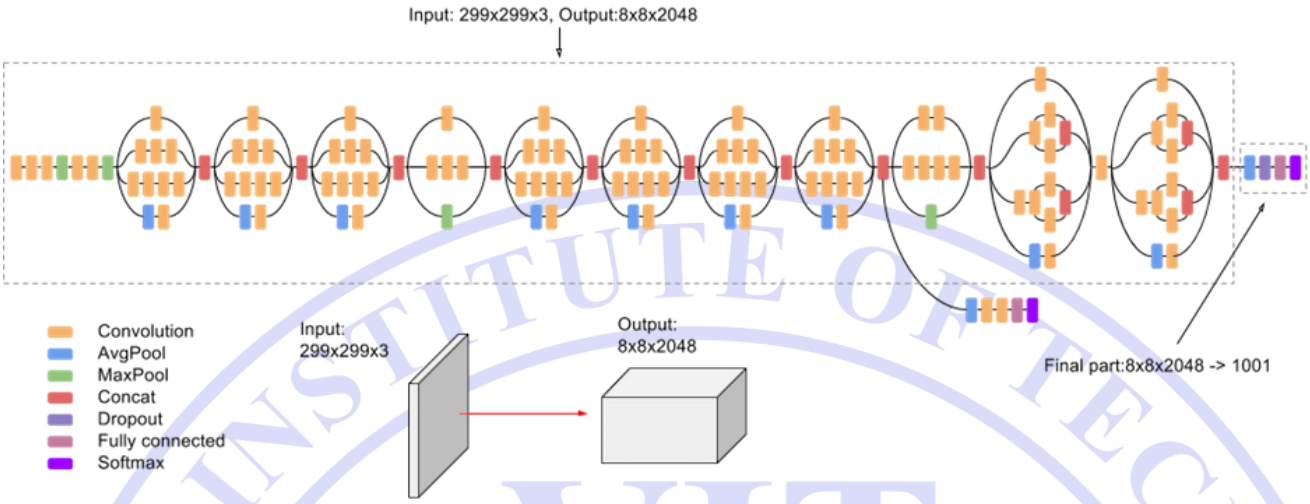


Figure 2: InceptionV3 Architecture (Source: Google Cloud)

Xception stands for "extreme inception." Rather like our previous two architectures, it reframes the way we look at neural nets — conv nets in particular. And, as the name suggests, it takes the principles of Inception to an extreme. Here's the hypothesis: "*cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly*."

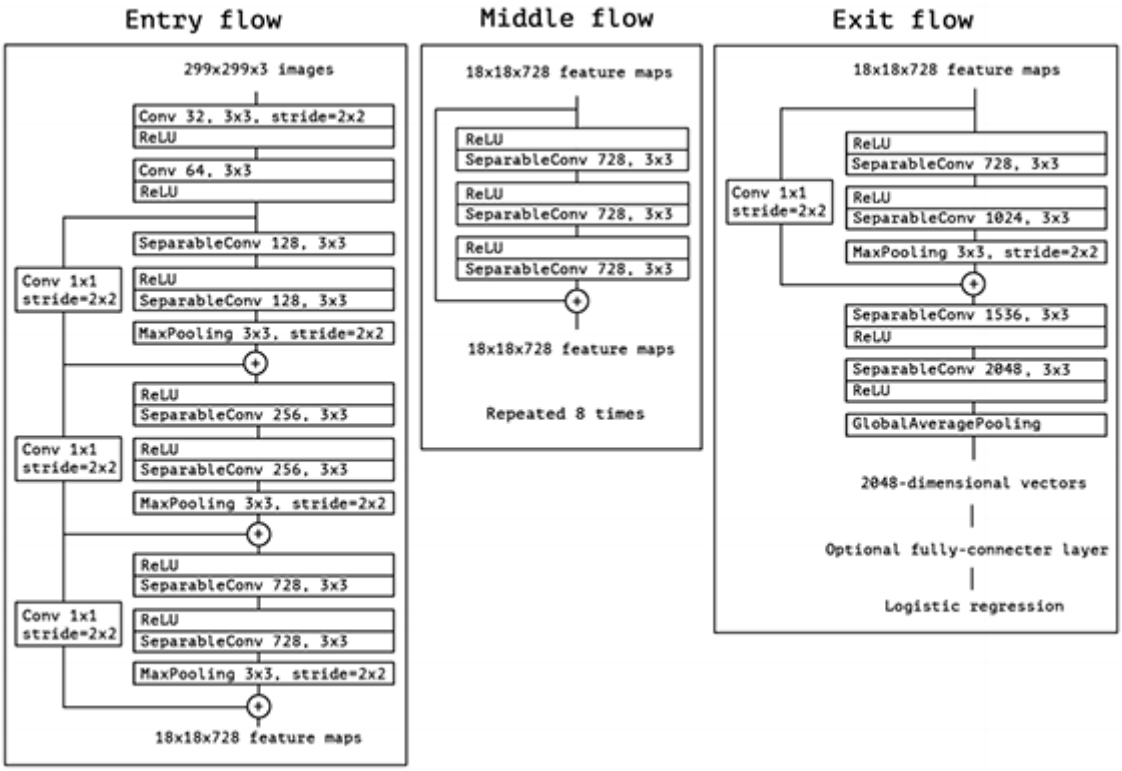


Figure 3: Xception Architecture (Source: pyimagesearch.com)

The goal of this project is in the favour of the help of the general public as anyone and everyone can use this to check personally so that the medications and precautions can be started early.

TECHNICAL SPECIFICATIONS

MODEL DEVELOPMENT AND TRAINING

Development of the Machine Learning/Deep Learning Model requires the knowledge and skill of using Jupyter Notebook (either installed via Anaconda Studio or directly as an individual component) or Google Colab along with high-powered Graphical Processing Unit for better, reduced and efficient processing time.

Use of Google Colab gives more advantages for writing and execution of code in Python, sharing notebook, Integrating Libraries and Dependencies and also to increase the available processing power through Cloud Computing. The advantage of Cloud Computing allows the model to still train even if the system crashes due to some unforeseen reason and does not cause a halt to the training.

INTERFACE DEVELOPMENT AND TESTING

The interface is converted into an executable file (exe File) so that the general public regardless of with or without any technical skills can simply install the software on their system and use the model to check if the patient suffering from the disease has Skin Cancer or not, so that the patient can think and take action at an early stage and start with the precautions and medications which can help save his/her life.

After uploading an image of the infected area, the final result is that whether the person is suffering from skin cancer or not. The final result is shown through an alert box.

INTERFACE REQUIREMENTS

There are two interfaces namely – User Interface and Software Interface.

- User Interface: The user interface will be implemented using any desktop running on Windows OS. This interface will be very user friendly so that people from different strata can use it to detect their disease without any difficulty by just uploading their medical test image.
- Software Interface: A software interface running on Windows OS. It should have Python compiler.

PLATFORM, CONFIGURATION AND SPECIFICATION

Platform: Google Colab (Pro version recommended)

Standard RAM: 12.69 GB

High-RAM Configuration: 25.46 GB (recommended with Pro version)

Normal Disk Space: 89.15 GB

Pro Version Disk Space: 147.15 GB

Engine: Google 3 Python Compute Engine backed (GPU)

GPU: NVIDIA Tesla K80

DESIGN APPROACH AND DETAILS

DESIGN APPROACH/MATERIALS AND METHODS

As suggested by the mentor to use the pre-trained models, for example, VGG, ResNet, MobileNet, etc., we planned on using these models and building an architecture. The proposed model includes the use of MobileNet, Inception and Xception model which are very well-known pre-trained architectures. These architectures have been brought together using Ensemble Modelling in which we have created a list of these models through which we join the Neural Net and further join their weights for one single model and to used and updated.

CODES AND STANDARDS

For the implementation of this model, the code was written using the programming language Python. Python also has several advantages due to its hundreds of open-source libraries which are available for almost every functionality to improve and increase the performance. The program for interface is also written using Python.

PROPOSED ARCHITECTURE

PRE-PROCESSING

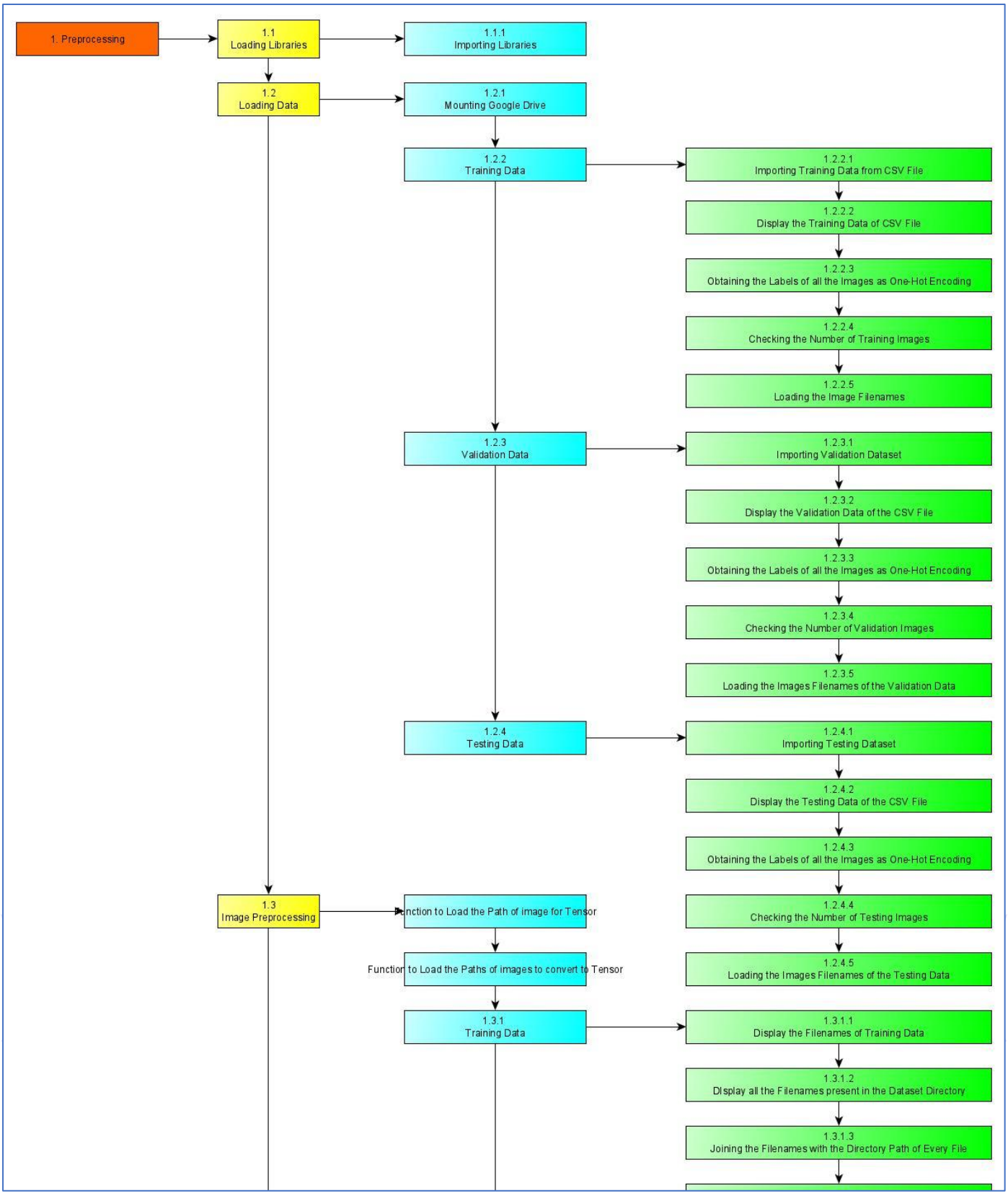


Figure 4: Proposed Architecture - Pre-processing(i)

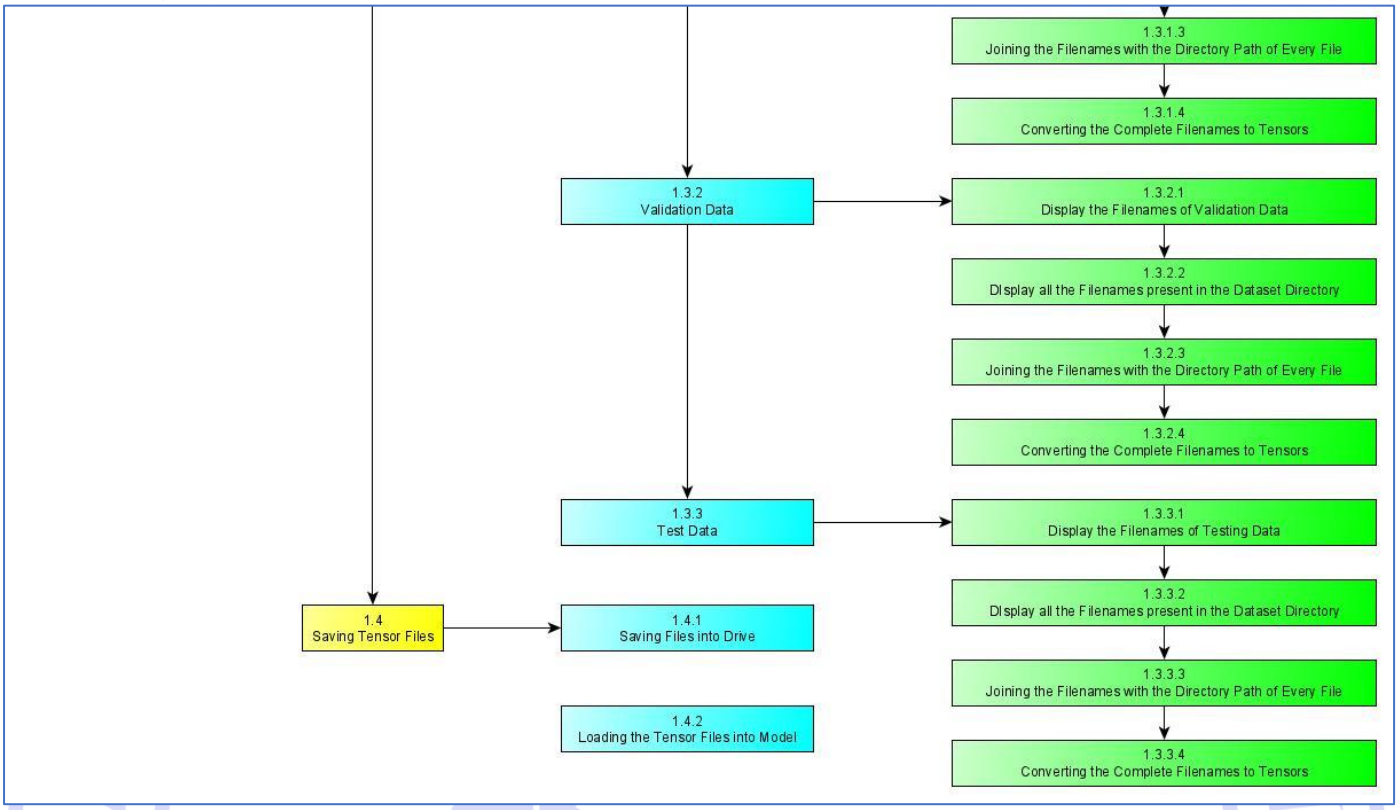


Figure 5: Proposed Architecture - Pre-processing(ii)

All the required libraries are first loaded into the model and then the dataset is also loaded into the model using the Google Drive as the Dataset has been uploaded to the Google Drive it can be accessed directly through it.

After the Google Drive is connected and authorised for use, the CSV files of the Training, Validation and Testing Data are read one-by-one and the data which they contain is displayed. Further, it is required for the dataset images to have their directory location attached to the filenames because then the images can be accessed directly with it without changing the directory which might interrupt any further operation.

After the directory locations are attached to the filenames, the images are loaded into the model one-by-one and they are converted into tensors. After obtaining the tensors, they are stored in a separate location so that these tensors can be used directly and it eliminates the need for loading the dataset and processing it again and again and instead these tensor files can just be loaded into the model and used for further training and processing. This also saves a lot of time because we do not need to create the tensors again and again as the tensors will be same each time they are being created because it is the same images which might be used during the development of the project.

TRAINING MODEL

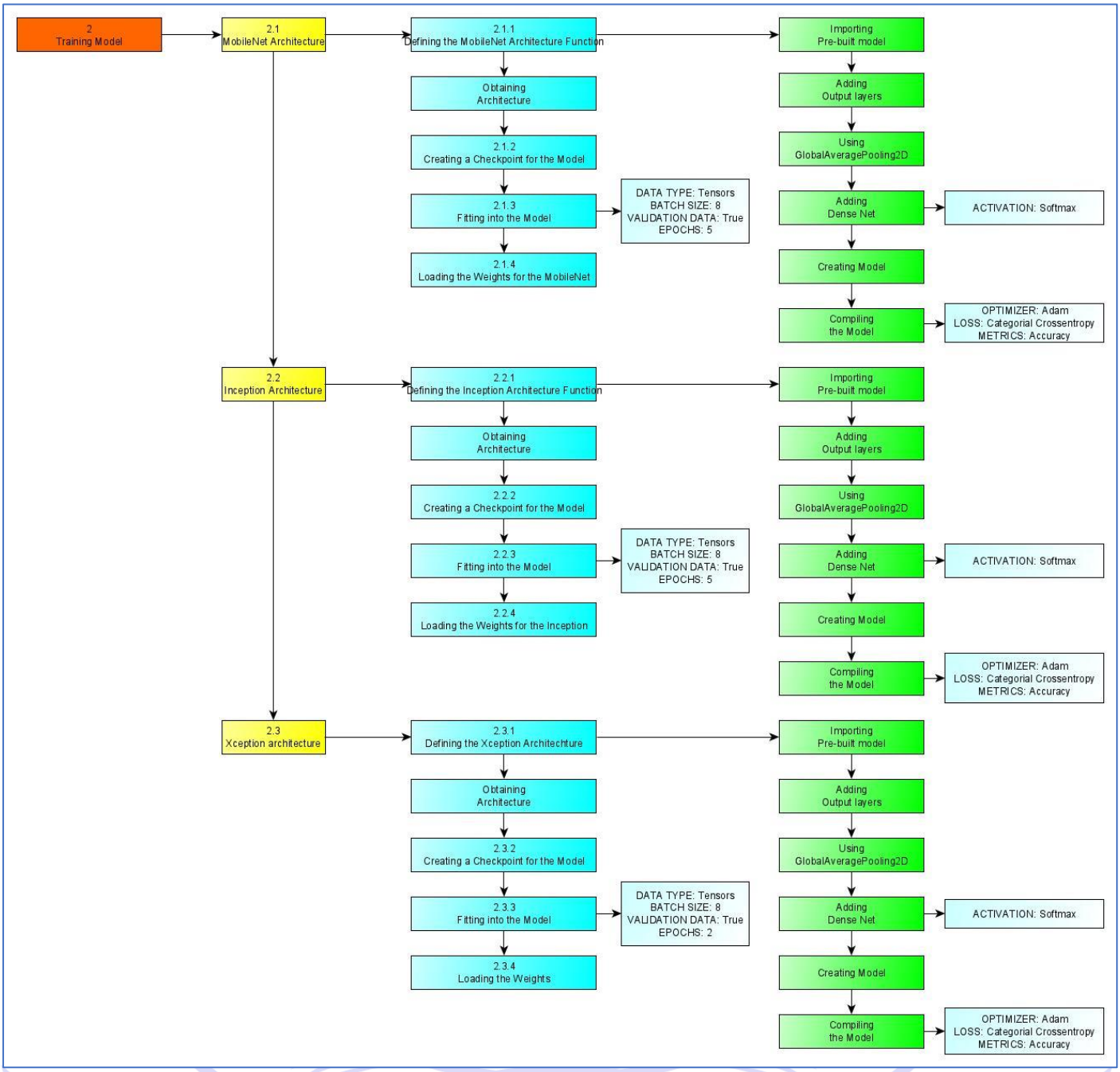


Figure 6: Proposed Architecture - Training Model

For this, we have used Transfer Learning and Ensemble Modelling. For the Transfer Learning, we have used the following pre-trained models:

- MobileNet Architecture
- InceptionV3 Architecture
- Xception Architecture

For the training, the pre-trained models are first loaded in as a function and then a since we have a checkpoint of these models, so we can start training these models from that checkpoint again. We start training it further by fitting it on the data which we have using the tensors.

PREDICTION

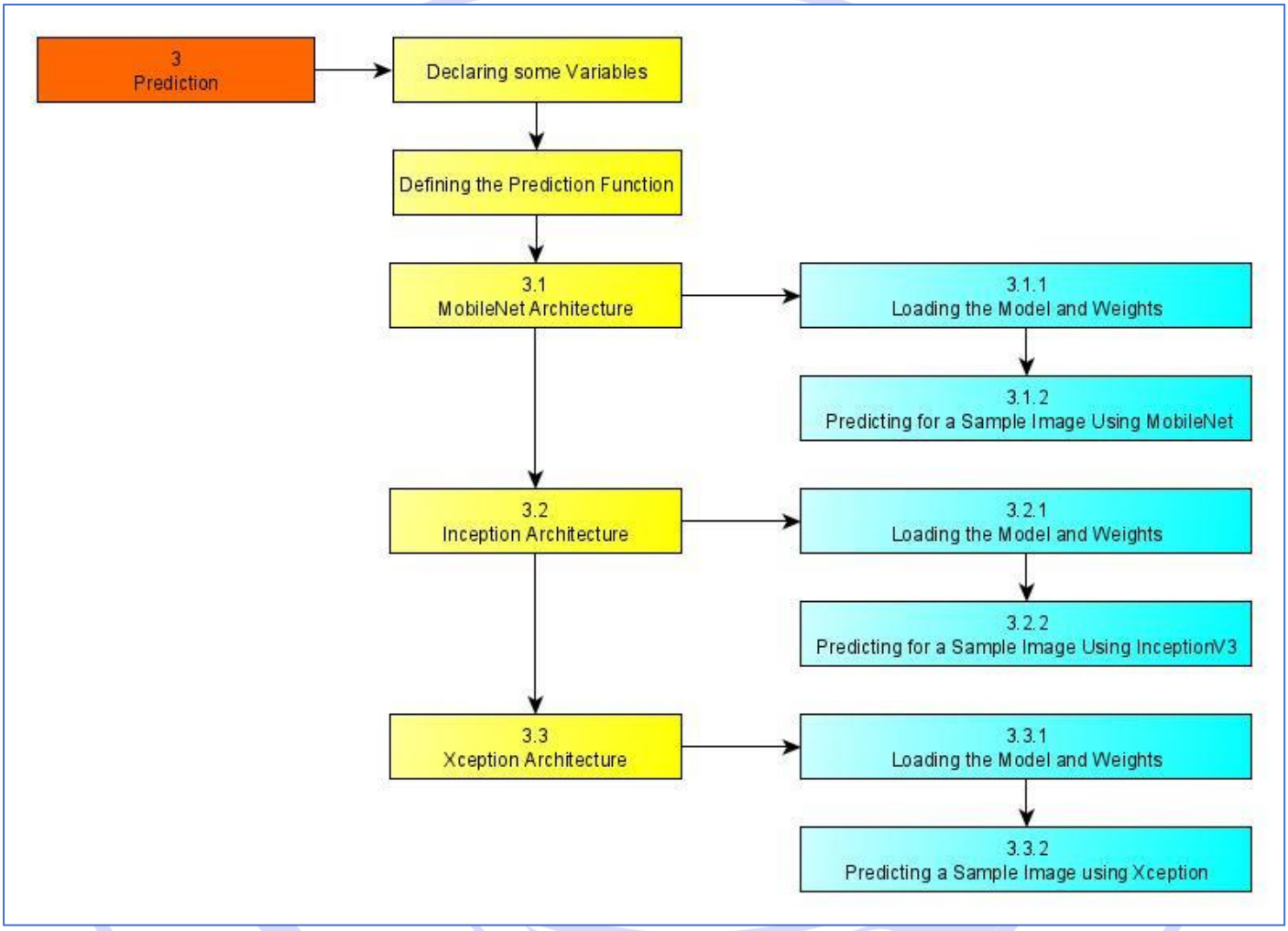


Figure 7: Proposed Architecture - Prediction Function

This prediction is done on a sample image just to check that all the models are working and are predicting. Working of this section denotes that the model can be taken to the further stage of evaluation of the validation data and the testing data.

For this part, we take the sample image and convert that image into a tensor. After that, the Individual Models and their respective weights are loaded so that the prediction can be made using the predictive function which we have defined. This is the function which will also be used for the Interface Application for the prediction on the image which will be uploaded by the user.

EVALUATING THE MODELS INDIVIDUALLY ON VALIDATION DATA

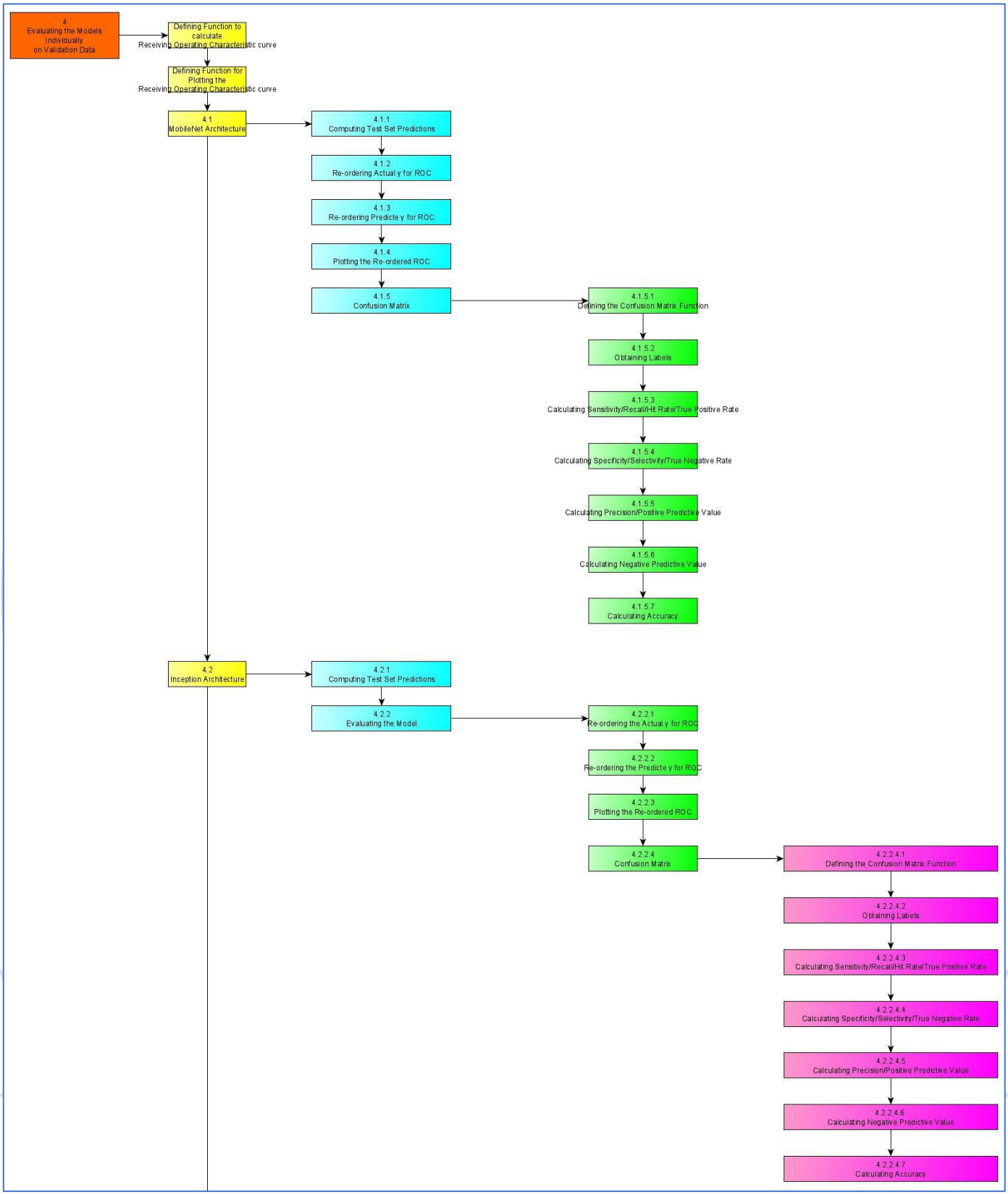


Figure 8: Proposed Architecture - Evaluating Models Individually on Validation Data (i)

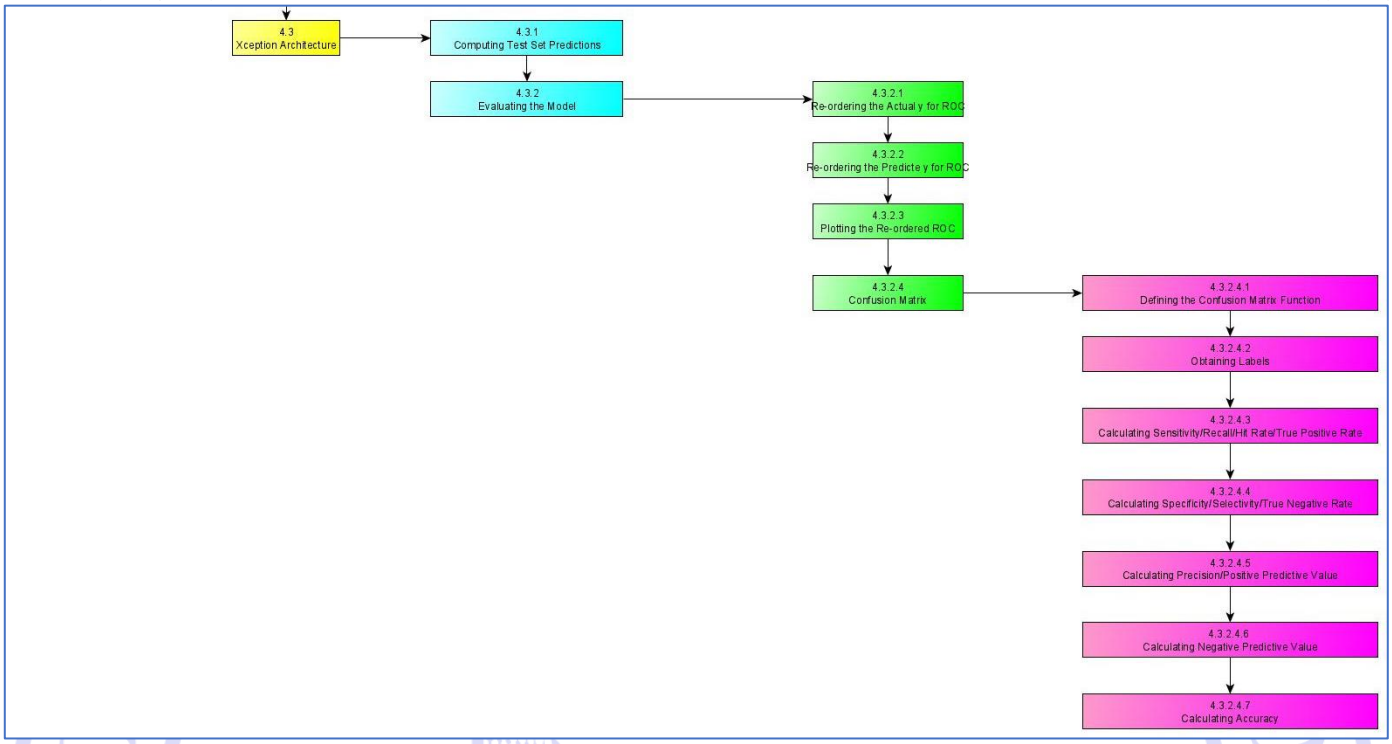


Figure 9: Proposed Architecture - Evaluating the Models Individually on Validation Data (ii)

For the evaluation of the models on the Validation Data, we have used the Receiving Operating Characteristic Curve. A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The method was originally developed for operators of military radar receivers starting in 1941, which led to its name.

For this, the we will be computing the test set predictions and then the model will be evaluated based on the Receiving Operating Characteristic Curve whose values will be obtained using the Confusion Matrix. Then, we will plot the values obtained for this in order to obtain the graph and check for the best possible values which can be used for prediction.

Further, after obtaining all these values, we will be calculating the Precision, Recall, Specificity, Sensitivity, Negative Predictive Value and Accuracy for further evaluation and after this, we can check the parameters, and if possible, we can tune the parameters to obtain better results.

At the end, we can compare the test results for the pre-trained architecture, as we have noted that the loss had reduced to the most efficient values possible and if we will train the model further with more epochs on the architectures, then we might make the model overfit on the data which can decrease the results and might lead to a higher number of wrong predictions, furthermore reducing the accuracy of the model and its efficiency.

EVALUATING THE MODELS TOGETHER ON VALIDATION DATA – ENSEMBLING THE MODELS

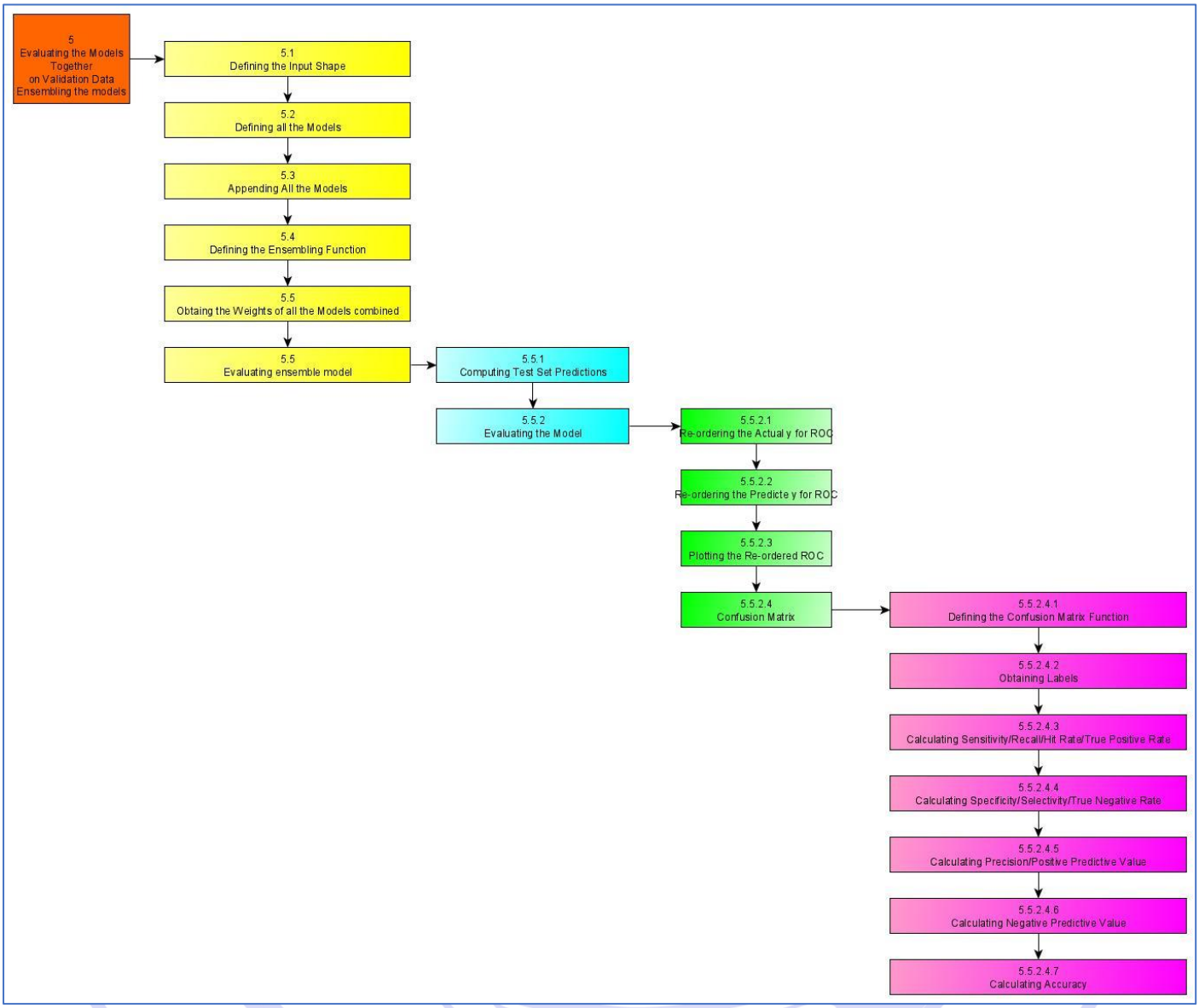


Figure 10: Proposed Architecture - Evaluating the Models Together on Validation Data

Defining a specific input shape for all the input images. The models will be brought together and appended so that the image passes through all the models one-by-one. The ensemble model will be defined which will contain all the models and the then the weights of the model will be combined and a file will be generated so that the weights are together into one and then they can be used together for the for the complete architecture.

For evaluating the models, we will be using the test set predictions and the ROC curve will be used whose computational values will be obtained using the Confusion Matrix. The other evaluation metrics are Precision, Recall, Sensitivity, Specificity, Negative Predictive Value and Accuracy. The results obtained for the Ensemble Model are better than each architecture individually.

EVALUATING THE MODELS INDIVIDUALLY ON TESTING DATA

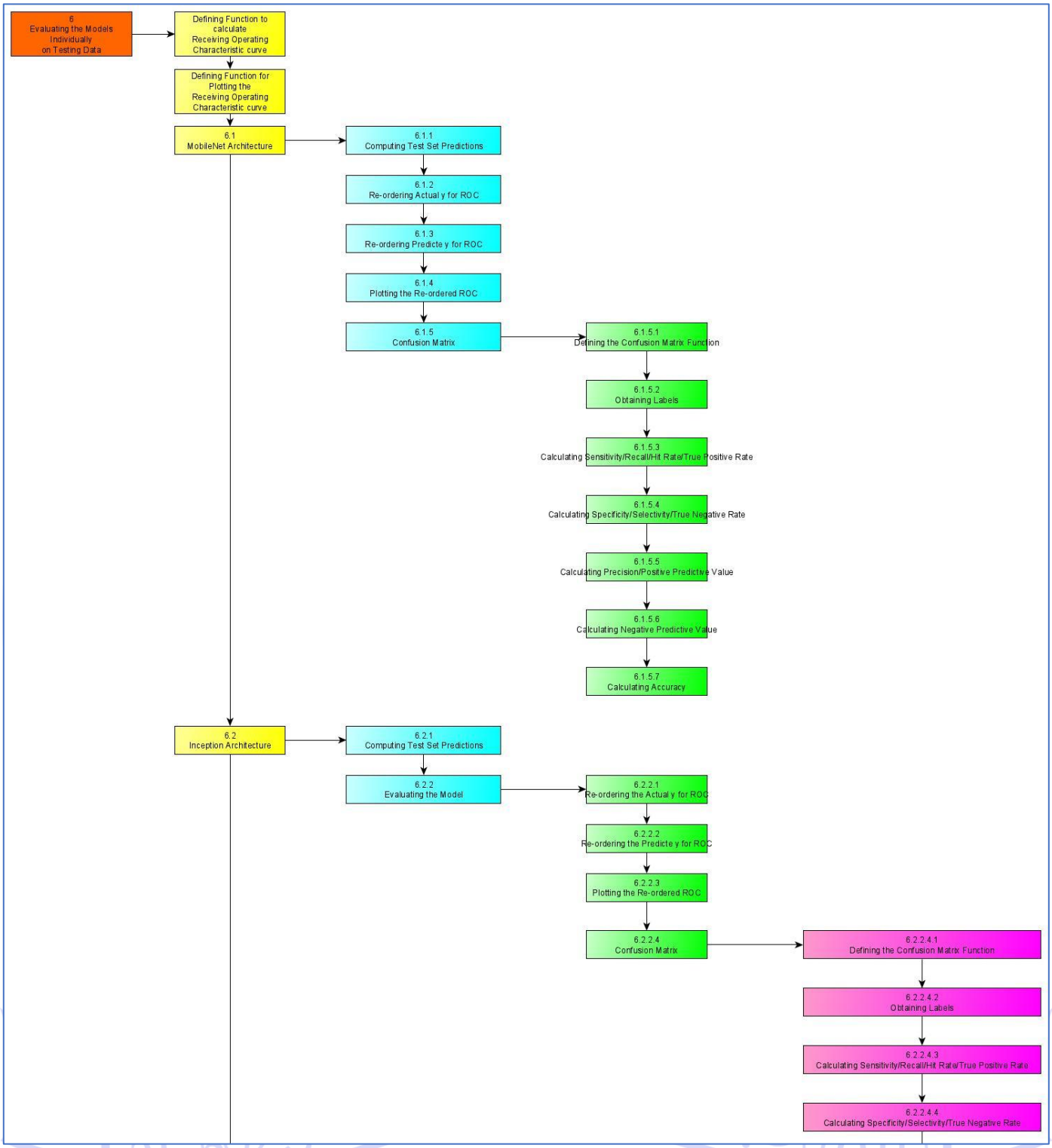


Figure 11: Proposed Architecture - Evaluating the Models Individually on Testing Data (i)

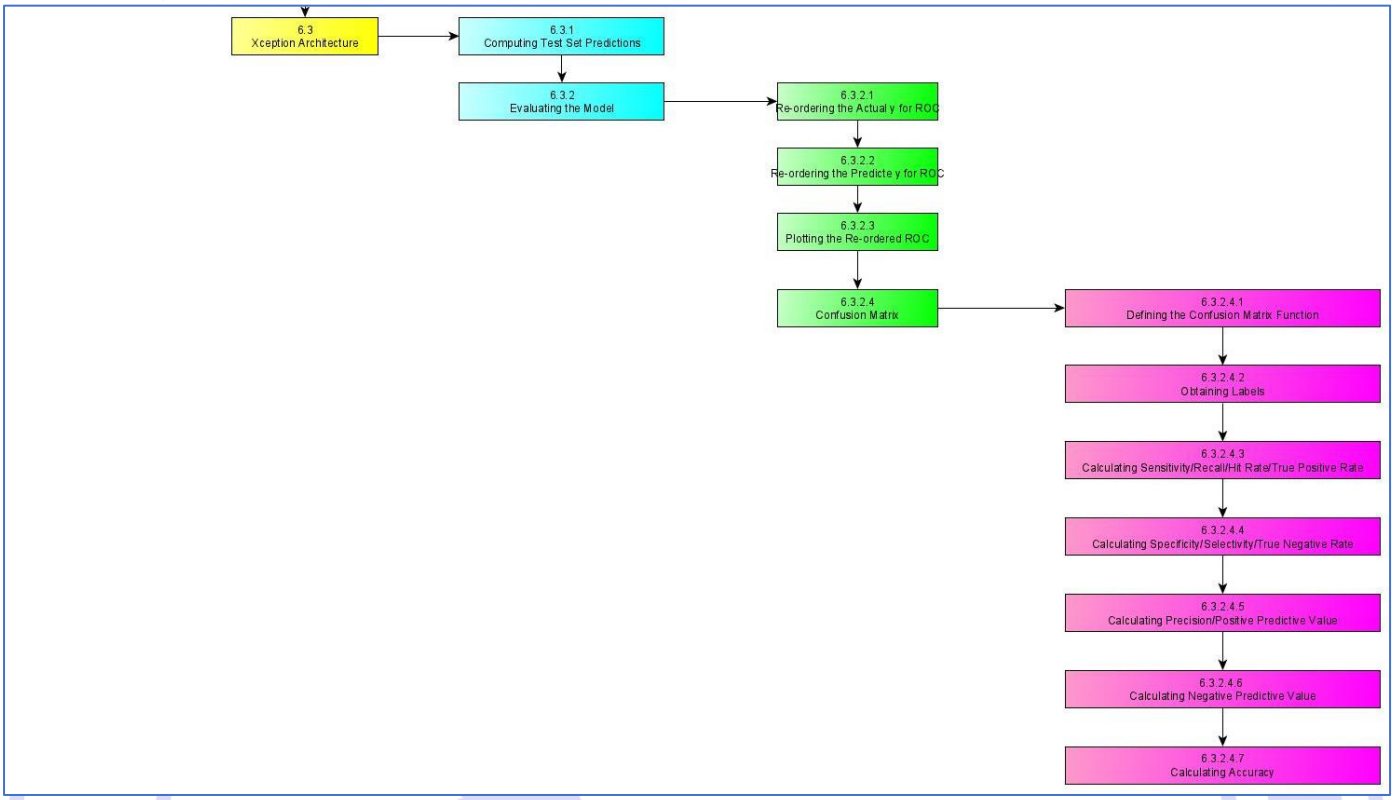


Figure 12: Proposed Architecture - Evaluating the Models Individually on Testing Data (ii)

For the evaluation of the models on the Testing Data, we have used the Receiving Operating Characteristic Curve. A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The method was originally developed for operators of military radar receivers starting in 1941, which led to its name.

For this, the we will be computing the test set predictions and then the model will be evaluated based on the Receiving Operating Characteristic Curve whose values will be obtained using the Confusion Matrix. Then, we will plot the values obtained for this in order to obtain the graph and check for the best possible values which can be used for prediction.

Further, after obtaining all these values, we will be calculating the Precision, Recall, Specificity, Sensitivity, Negative Predictive Value and Accuracy for further evaluation and after this, we can check the parameters, and if possible, we can tune the parameters to obtain better results.

At the end, we can compare the test results for the pre-trained architecture, as we have noted that the loss had reduced to the most efficient values possible and if we will train the model further with more epochs on the architectures, then we might make the model overfit on the data which can decrease the results and might lead to a higher number of wrong predictions, furthermore reducing the accuracy of the model and its efficiency.

The results obtained on the Testing data are very similar to the results obtained on the Validation data.

EVALUATING THE MODELS TOGETHER ON TESTING DATA – ENSEMBLING THE MODELS

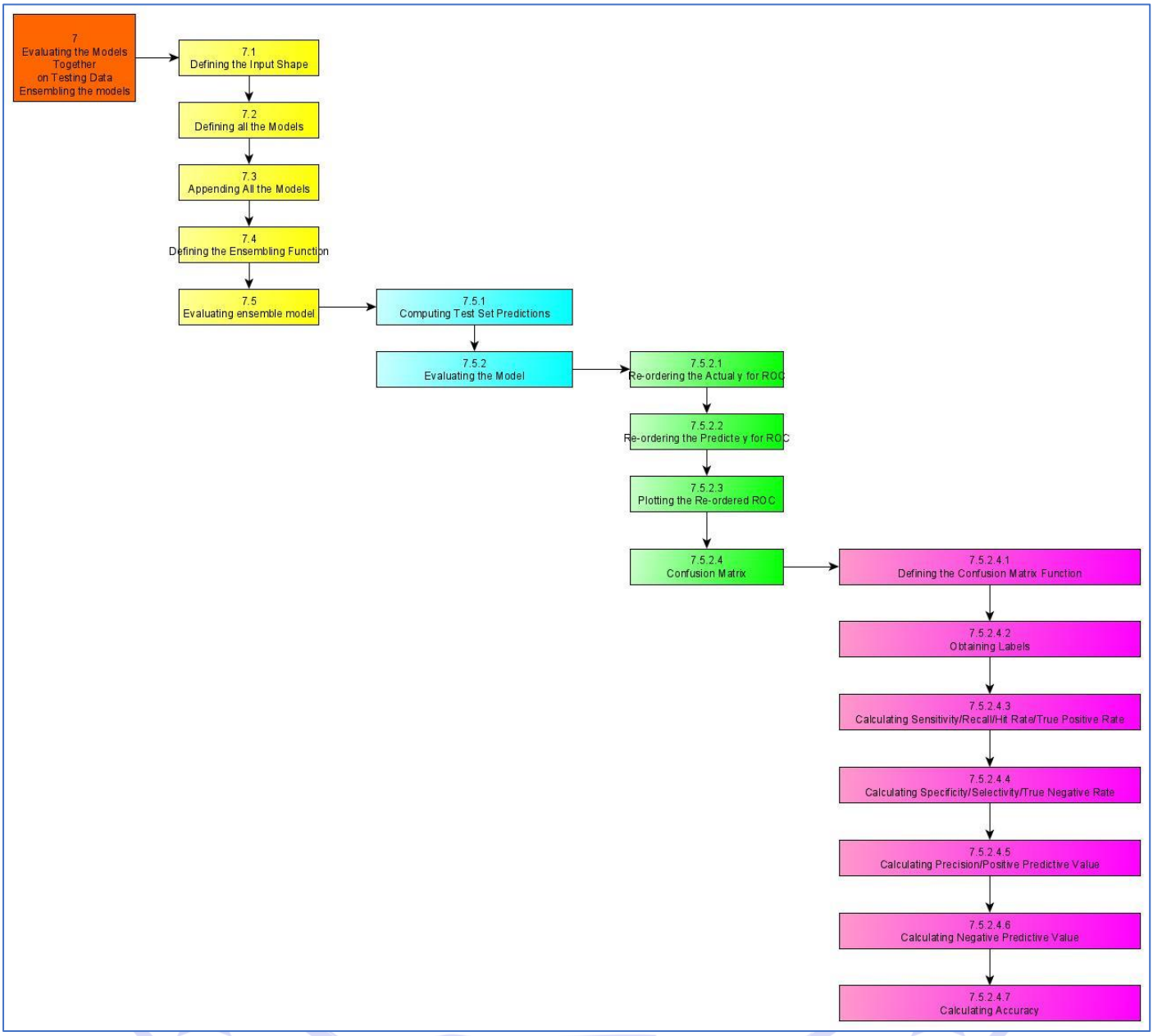


Figure 13: Proposed Model - Evaluating the Models Together on Testing Data

Defining a specific input shape for all the input images. The models will be brought together and appended so that the image passes through all the models one-by-one. The ensemble model will be defined which will contain all the models and the then the weights of the model will be combined and a file will be generated so that the weights are together into one and then they can be used together for the for the complete architecture.

For evaluating the models, we will be using the test set predictions and the ROC curve will be used whose computational values will be obtained using the Confusion Matrix. The other evaluation metrics are Precision, Recall, Sensitivity, Specificity, Negative Predictive Value and Accuracy. The results obtained for the Ensemble Model are better than each architecture individually.

LOCALIZATION

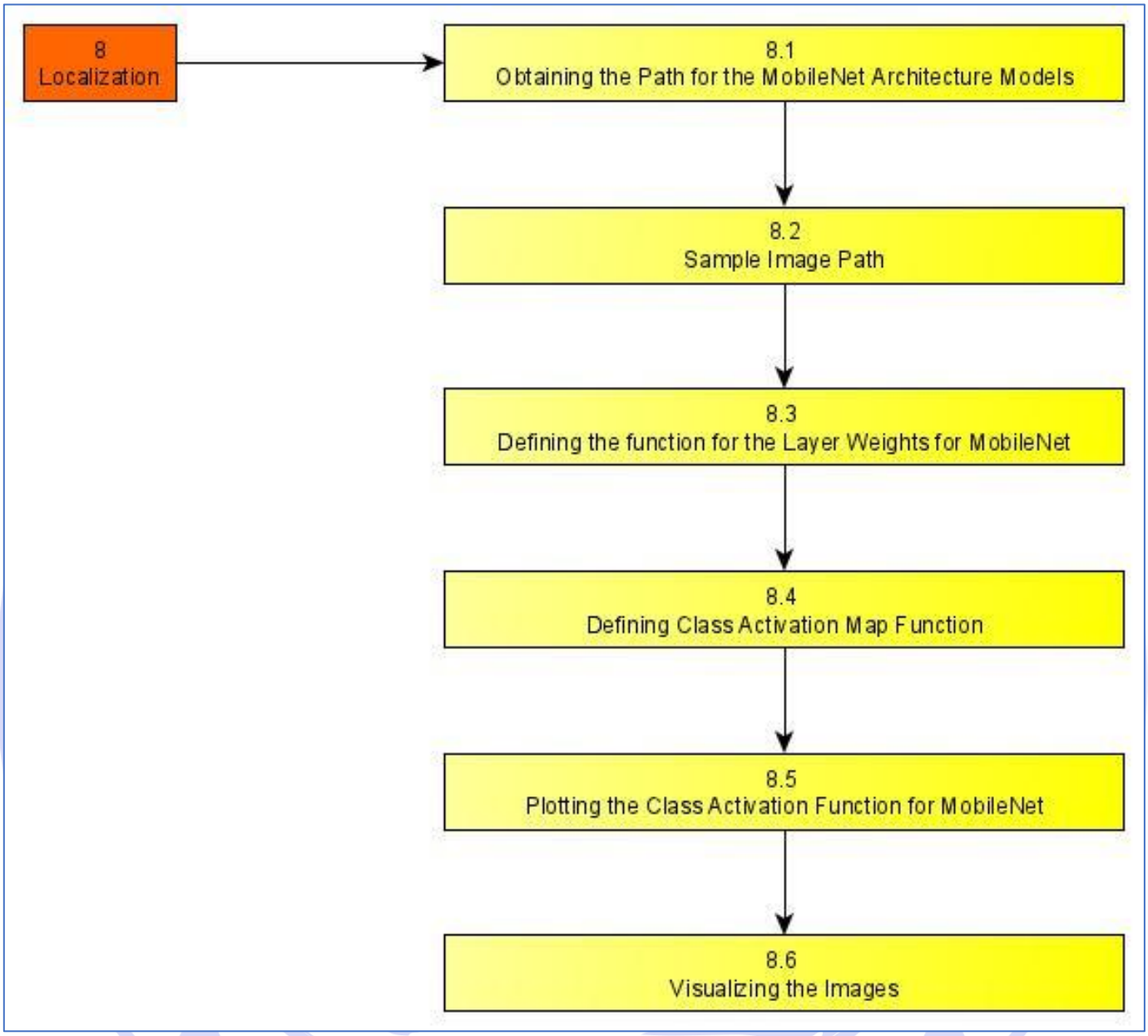


Figure 14: Proposed Architecture - Localization

For this, we define the Class Activation Map and then we create another function using which we can display the results obtained on a particular file, as we can use the address of the that image directly and then plot to even check the results obtained for the model which we have created. The results are displayed along with the plot which is created containing the Image and another heatmap image displayed along.

SAVING THE COMPLETE MODEL FOR THE PYTHON INTERFACE APPLICATION

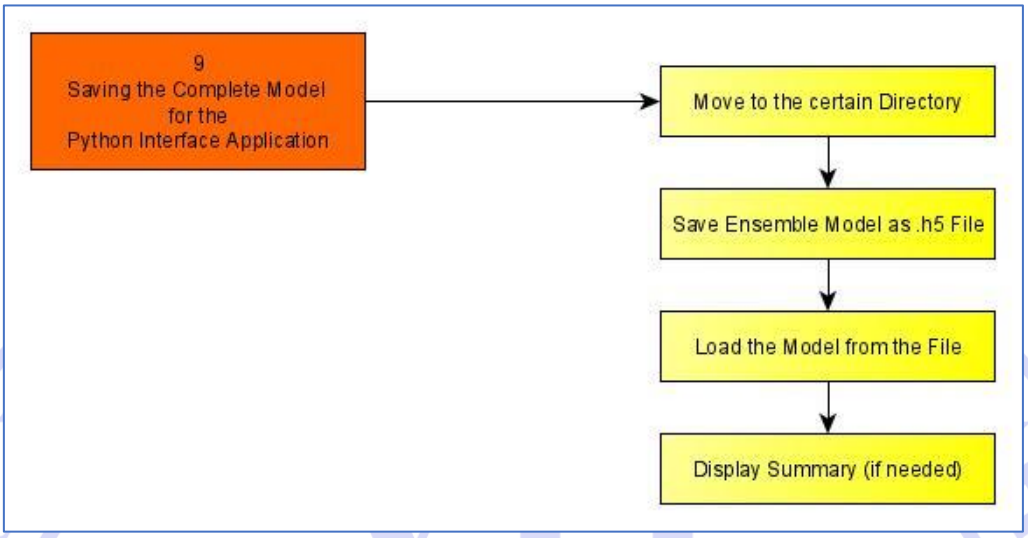


Figure 15: Proposed Architecture - Saving the Complete Model for the Application Interface

First, we will move to the required directory where we can save the file for the model. Then, we will use the complete model and use the save() function.

CONVERTING THE .H5 FILE TO .TFLITE FILE FOR THE PYTHON INTERFACE APPLICATION

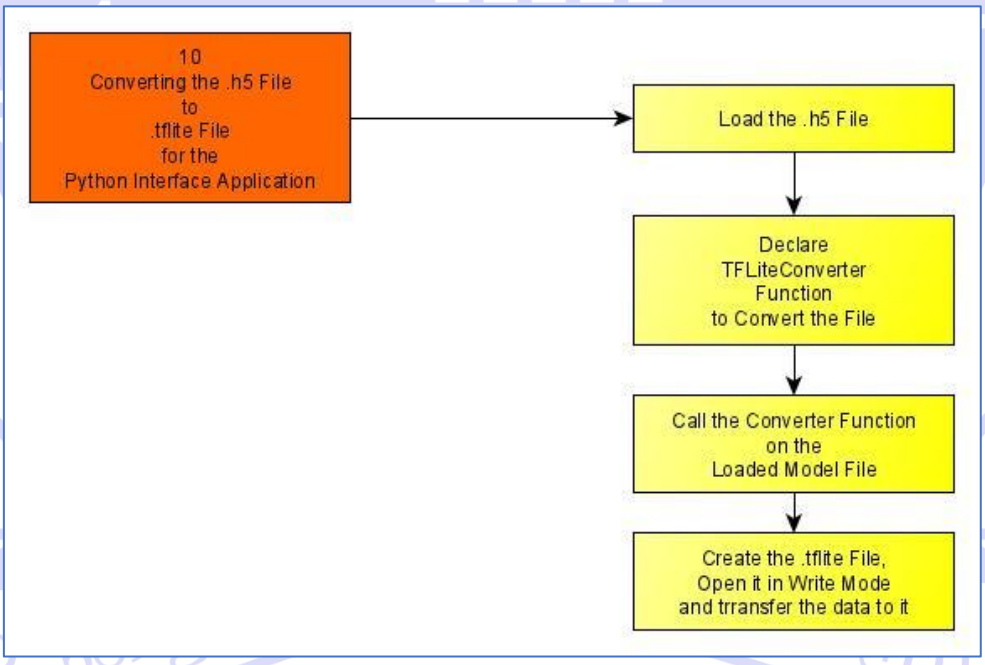


Figure 16: Converting the H5 File to TFLITE File for Application Interface

We will first load the saved model file and then we will use the converter function to convert the file into the into the required format which we can use for the Interface Application.

PROPOSED ARCHITECTURE SPECIFICATIONS

DATA SELECTION

The goal of the challenge is to help participants develop image analysis tools to enable the automated diagnosis of melanoma from dermoscopic images. Image analysis of skin lesions is composed of 3 parts:

- Part 1: Lesion Segmentation
- Part 2: Detection and Localization of Visual Dermoscopic Features/Patterns
- Part 3: Disease Classification

This challenge provides training data (150 images) and blind held-out test dataset (~600 images) will be provided for participants to generate and submit automated results.

BACKGRAOUND

MELANOMA

Skin cancer is a major public health problem, with over 5 million newly diagnosed cases in the United States each year. Melanoma is the deadliest form of skin cancer, responsible for over 9,000 deaths each year.

DERMOSCOPY

As pigmented lesions occurring on the surface of the skin, melanoma is amenable to early detection by expert visual inspection. It is also amenable to automated detection with image analysis. Given the widespread availability of high-resolution cameras, algorithms that can improve our ability to screen and detect troublesome lesions can be of great value. As a result, many centres have begun their own research efforts on automated analysis. However, a centralized, coordinated, and comparative effort across institutions has yet to be implemented.

Dermoscopy is an imaging technique that eliminates the surface reflection of skin. By removing surface reflection, visualization of deeper levels of skin is enhanced. Prior research has shown that when used by expert dermatologists, dermoscopy provides improved diagnostic accuracy, in comparison to standard photography. As inexpensive consumer dermatoscope attachments for smart phones are beginning to reach the market, the opportunity for automated dermoscopic assessment algorithms to positively influence patient care increases.

Figure 17: Sample Images of the Dataset

EXPLORATORY DATA ANALYSIS

CHECKING THE TYPES OF DATA

There are four types of EDA in all:

- **Univariate non-graphical**: This is the simplest form of data analysis among the four options. In this type of analysis, the data that is being analyzed consists of just a single variable. The main purpose of this analysis is to describe the data and to find patterns.
- **Univariate graphical**: Unlike the non-graphical method, the graphical method provides the full picture of the data. The three main methods of analysis under this type are histogram, stem and leaf plot, and box plots. The histogram represents the total count of cases for a range of values. Along with the data values, the stem and leaf plot show the shape of the distribution. The box plots graphically depict a summary of minimum, first quartile median, third quartile, and maximum.
- **Multivariate non-graphical**: The multivariate non-graphical type of EDA generally depicts the relationship between multiple variables of data through cross-tabulation or statistics.
- **Multivariate graphical**: This type of EDA displays the relationship between two or more set of data. A bar chart, where each group represents a level of one of the variables and each bar within the group represents levels of other variables.

FINDING THE OUTLIERS

There exist three different options on how to treat non-error outliers:

1. Keep
2. Delete
3. Recode

KEEP

When most of the detected outliers are non-error outliers and rightfully belong to the population of interest.

DELETE

The most straightforward option is to delete any outlying observation. However, this strategy bears a high risk of losing information. Especially if you find many outlying data points, try to avoid this. Also, deleting interesting and influential outliers (points that belong to the population of interest) can falsely impact any output, e.g., prediction or test result, you aim to achieve.

RECODE

Recoding outliers is a good option to treat outliers and keep as much information as possible simultaneously. This option should always be accompanied by sound reasoning and explanation.

DATA VISUALIZATION

- Univariate data analysis
- Bivariate data analysis
- Line plot
- Box plot
- Scatter matrix

DATA PRE-PROCESSING

SPLITTING THE DATA

Divide the dataset into three parts to avoid overfitting or underfitting and model selection bias called -

1. Training set (Has to be the largest set)
2. Cross-Validation set or Development set or Dev set
3. Testing Set

UNDERFITTING:

Underfitting mainly occurs when a machine learning algorithm is not able to capture the lower trend of data which is mainly when data is nor well fitted inside the model.

OVERFITTING?

When machine learning algorithm is trained on very well data and very closely on a dataset which can lead to a negative impact on the performance of the system leading to the wrong system and prediction model.

CHECKING FOR MISSING VALUES

There are three categories of missing data:

- **MCAR** (Missing Completely At Random) where the pattern of missingness is statistically independent of the data record. Example: you have a data set on a piece of paper and you spill coffee on the paper destroying part of the data.

- **MAR** (Missing At Random) where the probability distribution of the pattern of missingness is functionally dependent upon the observable component in the record. MCAR is a special case of MAR. Example: if a child does not attend an educational assessment because the child is (genuinely) ill, this might be predictable from other data we have about the child's health, but it would not be related to what we would have measured had the child not been ill.
- **MNAR** (Missing Not at Random) which is defined as the case which is NOT MAR, or when the missingness is specifically related to what is missing. Example: a person does not attend a drug test because the person took drugs the night before.

CHECKING CATEGORICAL FEATURES

Two major types of categorical features are

- **Nominal** – These are variables which are not related to each other in any order such as colour (black, blue, green).
- **Ordinal** – These are variables where a certain order can be found between them such as student grades (A, B, C, D, Fail).

NORMALIZING DATASET

The goal of normalization is to change values to a common scale without distorting the difference between the range of values.

Using this technique, we are going to have a mean of 0 and a standard deviation of 1 in our dataset. We can either do it normally by combining different functions in numpy, i.e.,

$$z = \frac{x.values - np.mean(x.values)}{np.std(x.values)}$$

where *x* is a data frame with all numerical indices. If we want to retain the values in a data frame, then we can simply remove. values in front of it.

FEATURE TRANSFORMATION

WHY DO WE NEED FEATURE TRANSFORMATION AND SCALING?

Oftentimes, we have datasets in which different columns have different units – like one column can be in kilograms, while another column can be in centimetres. Furthermore, we can have columns like income which can range from 20,000 to 100,000, and even more; while a benign column which can range from 0 to 100(at the most). Thus, Malignant is about 1,000 times larger than age. When we feed these features to the model as is, there is every chance that the Malignant values will influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. So, to give importance to both Benign, and Malignant, we need feature scaling.

FEATURE TRANSFORMATIONS USED IN THE MODELS

MODEL SELECTION

Tensors often offer more natural representations of data, e.g., consider video, which consists of obviously correlated images over time. You *can* turn this into a matrix, but it's just not natural or intuitive (what does a factorization of some matrix-representation of video mean?).

Tensors are trending for several reasons:

- our understanding of multilinear algebra is improving rapidly, specifically in various types of factorizations, which in turn helps us to identify new potential applications (e.g., multiway component analysis)
- software tools are emerging (e.g., Tensorlab) and are being welcomed
- Big Data applications can often be solved using tensors, for example recommender systems, and Big Data itself is hot
- increases in computational power, as some tensor operations can be hefty (this is also one of the major reasons why deep learning is so popular now)

MODEL TRAINING

Most machines cannot learn until and unless they have a proper dataset or at least any kind of **data**. And modern data is often multi-dimensional (having more than one dimensions). Tensors can play an important role in Machine Learning/Deep Learning by encoding multi-dimensional data. For example (a simple example to explain the advantages of using them), a picture is generally represented by three fields: *width, height* and *depth (color)*. It makes total sense to encode it as a 3D tensor. However, more than often we are dealing with tens of thousands of pictures. Hence this is where the fourth field, *sample size* comes into play. A series of images, such as the famous MNIST dataset, can be easily stored in a 4D tensor in Tensorflow. This representation allows problems involving big data to be solved easily.

(ML Concepts: Tensors, 2018)

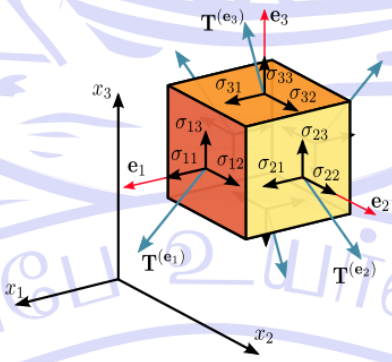


Figure 18: Visualizing a Tensor

A real-life example would be in most recommender systems, model-based Collaborative Filtering approaches such as Matrix Factorization do not the flexibility of integrating context information into the models. By using a **N-dimensional tensor** instead of the traditional 2D User-Item matrix, researchers have developed a more

robust model that provides context-aware recommendations. Increase in computing power in the recent years also allows these heavy tensor operations to be realized.

At this point you may wonder: why do most machine learning algorithms rely on vectors and matrices, while deep learning algorithms and neural networks mostly rely on tensors? A simple answer is that deep learning usually involves hundreds, if not thousands, of dimensions and fields. As we discussed previously, this is best represented by tensors since they can represent anything ranging from *zero* to *N* dimensions. In computer vision problems such as the Merck Molecular Activity Challenge, images can easily be broken down into a few hundred features. Therefore, tensors can be best viewed as containers that wrap and store data features in the context of machine learning and deep learning.

PROJECT DEMONSTRATION

APPLICATION INTERFACE

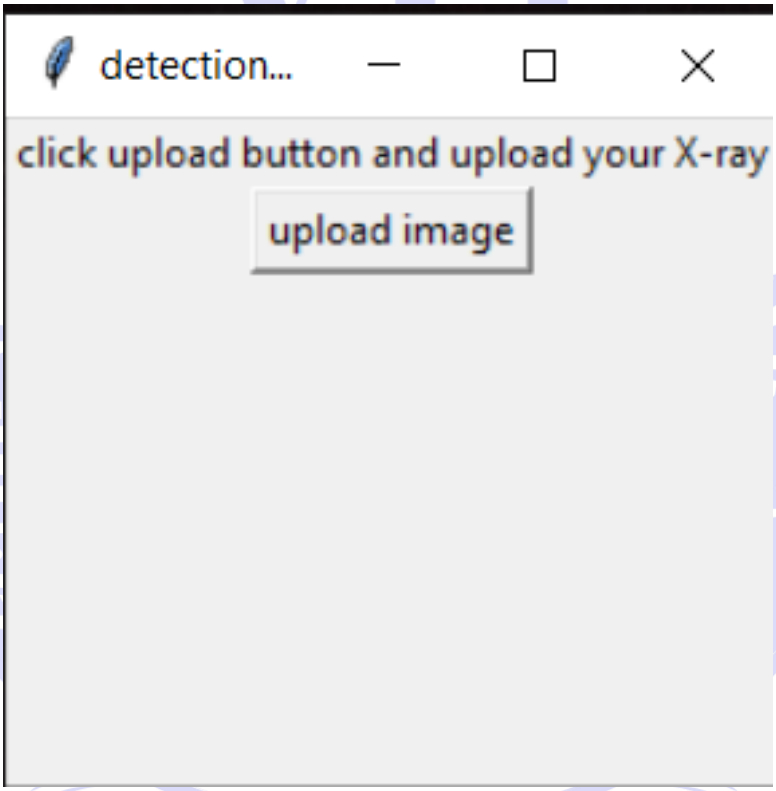


Figure 19: First View of Interface

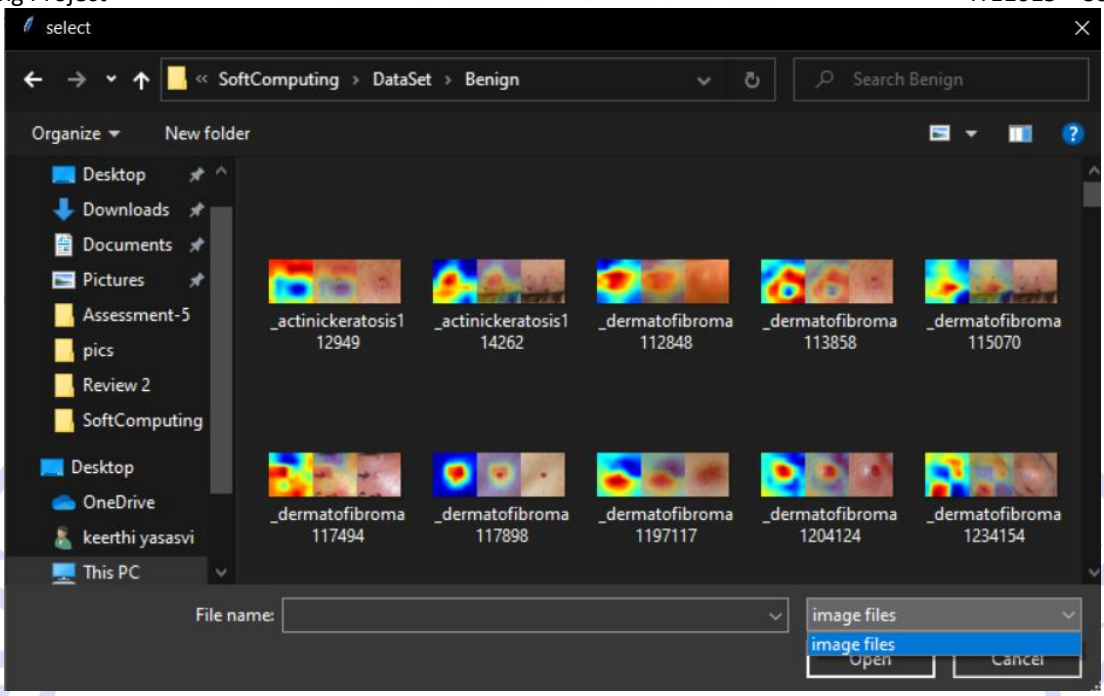


Figure 20: Selecting a File for Uploading

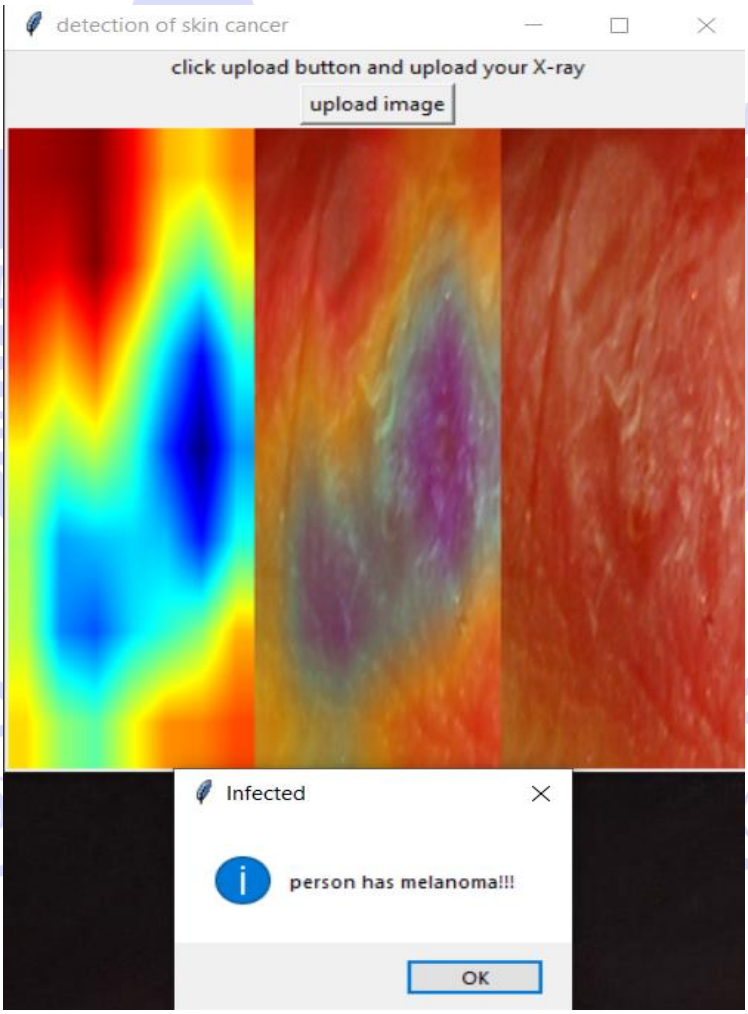


Figure 21: Display of Output

MOUNTING GOOGLE DRIVE

1.2.1. Mounting Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 22: Project Demonstration - Mounting Google Drive for Authorization

LOADING AND DISPLAY THE CSV DATASET

TRAINING CSV FILE

1.2.2.2 Display the Training Data of the CSV File

```
training_data
```

	image_id	melanoma	seborrheic_keratosis
0	ISIC_0000000	0.0	0.0
1	ISIC_0000001	0.0	0.0
2	ISIC_0000002	1.0	0.0
3	ISIC_0000003	0.0	0.0
4	ISIC_0000004	1.0	0.0
...	...	...	...
1995	ISIC_0015220	0.0	1.0
1996	ISIC_0015233	0.0	1.0
1997	ISIC_0015260	0.0	1.0
1998	ISIC_0015284	1.0	0.0
1999	ISIC_0015295	0.0	1.0

2000 rows × 3 columns

Figure 23: Project Demonstration - Data in CSV File for Training Data

VALIDATION CSV FILE

1.2.3.2 Display the Validation Data of the CSV File

```
validation_data
```

	image_id	melanoma	seborrheic_keratosis
0	ISIC_0001769	0.0	0.0
1	ISIC_0001852	0.0	0.0
2	ISIC_0001871	0.0	0.0
3	ISIC_0003462	0.0	0.0
4	ISIC_0003539	0.0	0.0
...	...	...	...
145	ISIC_0015443	0.0	0.0
146	ISIC_0015445	0.0	0.0
147	ISIC_0015483	0.0	0.0
148	ISIC_0015496	0.0	0.0
149	ISIC_0015627	0.0	0.0

150 rows × 3 columns

Figure 24: Project Demonstration - Data in CSV File for Validation Data

TESTING CSV FILE

1.2.4.2. Display the Testing Data of the CSV File

```
testing_data
```

	image_id	melanoma	seborrheic_keratosis
0	ISIC_0012086	0.0	1.0
1	ISIC_0012092	0.0	0.0
2	ISIC_0012095	0.0	0.0
3	ISIC_0012134	0.0	1.0
4	ISIC_0012136	0.0	1.0
...	...	...	...
595	ISIC_0016068	0.0	0.0
596	ISIC_0016069	0.0	0.0
597	ISIC_0016070	0.0	0.0
598	ISIC_0016071	0.0	0.0
599	ISIC_0016072	0.0	0.0

600 rows × 3 columns

Figure 25: Project Demonstration - Data in CSV File for Testing Data

CONVERTING THE DATA INTO TENSORS

TRAINING TENSOR

1.3.1.4. Converting the Complete Filenames to Tensors

```
train_tensors = paths_to_tensor(files).astype('float32')/255
```

```
100%|██████████| 2000/2000 [19:09<00:00,  1.74it/s]
```

Figure 26: Project Demonstration - Converting Training Files to Tensors

VALIDATION TENSOR

1.3.2.4. Converting the Complete Filenames to Tensors

```
valid_tensors = paths_to_tensor(validation_files).astype('float32')/255
```

```
100%|██████████| 150/150 [01:53<00:00,  1.32it/s]
```

Figure 27: Project Demonstration - Converting Validation Files to Tensors

TESTING TENSOR

1.3.3.4. Converting the Complete Filenames to Tensors

```
test_tensors = paths_to_tensor(test_files).astype('float32')/255
```

```
100%|██████████| 600/600 [10:38<00:00,  1.06s/it]
```

Figure 28: Project Demonstration - Converting Testing Files to Tensors

FITTING INTO THE MODELS

2.1.3. Fitting into the Model

```
mobilenet_model.fit(train_tensors,
                    train_targets,
                    batch_size = 8,
                    validation_data = (valid_tensors, valid_targets),
                    epochs = 5,
                    callbacks=[checkpointer],
                    verbose=1)
```

```
Epoch 1/5
250/250 [==============================] - 84s 208ms/step - loss: 0.5928 - accuracy: 0.7956 - val_loss: 1.5225 - val_accuracy:
0.2000

Epoch 00001: val_loss improved from inf to 1.52253, saving model to /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organize
d/Saved Models/weights.best.mobilenet.hdf5
Epoch 2/5
250/250 [==============================] - 50s 200ms/step - loss: 0.5103 - accuracy: 0.8002 - val_loss: 0.5847 - val_accuracy:
0.8000

Epoch 00002: val_loss improved from 1.52253 to 0.58473, saving model to /content/drive/MyDrive/dataset/ISIC Challenge 2017 Orga
nized/Saved Models/weights.best.mobilenet.hdf5
Epoch 3/5
250/250 [==============================] - 50s 201ms/step - loss: 0.5342 - accuracy: 0.7867 - val_loss: 0.6205 - val_accuracy:
0.8000

Epoch 00003: val_loss did not improve from 0.58473
Epoch 4/5
250/250 [==============================] - 50s 200ms/step - loss: 0.5311 - accuracy: 0.7961 - val_loss: 0.5127 - val_accuracy:
0.8000

Epoch 00004: val_loss improved from 0.58473 to 0.51270, saving model to /content/drive/MyDrive/dataset/ISIC Challenge 2017 Orga
nized/Saved Models/weights.best.mobilenet.hdf5
Epoch 5/5
250/250 [==============================] - 50s 200ms/step - loss: 0.5047 - accuracy: 0.8116 - val_loss: 0.6225 - val_accuracy:
0.8000

Epoch 00005: val_loss did not improve from 0.51270

<tensorflow.python.keras.callbacks.History at 0x7fa500592d90>
```

Figure 29: Project Demonstration - Training the MobileNet Architecture

2.2.3 Fitting into the Model

```
inception_model.fit(train_tensors,
                    train_targets,
                    batch_size = 8,
                    validation_data = (valid_tensors, valid_targets),
                    epochs = 5,
                    callbacks=[checkpointer],
                    verbose=1)
```

```
Epoch 1/5
250/250 [==============================] - 60s 213ms/step - loss: 0.5970 - accuracy: 0.7834 - val_loss: 0.5039 - val_accuracy:
0.8000

Epoch 00001: val_loss improved from inf to 0.50388, saving model to /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organize
d/Saved Models/weights.best.InceptionV3.hdf5
Epoch 2/5
250/250 [==============================] - 52s 207ms/step - loss: 0.4849 - accuracy: 0.8230 - val_loss: 0.8663 - val_accuracy:
0.7800

Epoch 00002: val_loss did not improve from 0.50388
Epoch 3/5
250/250 [==============================] - 52s 207ms/step - loss: 0.5306 - accuracy: 0.7890 - val_loss: 0.5700 - val_accuracy:
0.8000

Epoch 00003: val_loss did not improve from 0.50388
Epoch 4/5
250/250 [==============================] - 52s 207ms/step - loss: 0.5063 - accuracy: 0.8151 - val_loss: 0.5268 - val_accuracy:
0.8000

Epoch 00004: val_loss did not improve from 0.50388
Epoch 5/5
250/250 [==============================] - 52s 207ms/step - loss: 0.4957 - accuracy: 0.8166 - val_loss: 0.5712 - val_accuracy:
0.8000

Epoch 00005: val_loss did not improve from 0.50388

<tensorflow.python.keras.callbacks.History at 0x7fa4ac069690>
```

Figure 30: Project Demonstration - Training the Inception Architecture

2.3.3 Fitting into the Model

```
xception_model.fit(train_tensors,
                   train_targets,
                   batch_size = 8,
                   validation_data = (valid_tensors, valid_targets),
                   epochs = 2,
                   callbacks=[checkpointer, tensor_board],
                   verbose=1)
```

```
Epoch 1/2
250/250 [==============================] - 145s 565ms/step - loss: 0.7223 - accuracy: 0.7660 - val_loss: 0.5035 - val_accuracy:
0.8000

Epoch 00001: val_loss improved from inf to 0.50352, saving model to /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organize
d/Saved Models/xception_weights.hdf5
Epoch 2/2
250/250 [==============================] - 140s 558ms/step - loss: 0.5142 - accuracy: 0.8096 - val_loss: 8.2784 - val_accuracy:
0.6800

Epoch 00002: val_loss did not improve from 0.50352

<tensorflow.python.keras.callbacks.History at 0x7fa276ed2f10>
```

Figure 31: Project Demonstration - Training the Xception Architecture

PREDICTIONS ON A SAMPLE IMAGE

3.1.3 Predicting for a Sample Image Using MobileNet

```
predict("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.jpg",
```

```
Image Path: /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.
jpg
Arhitecture Used:
<tensorflow.python.keras.engine.functional.Functional object at 0x7fa2740562d0>
Path for Model Weights:
/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/weights.best.mobilenet.hdf5
Model Weights:
None
Prediction... Melanoma :  0.86639947  | Other :  0.13360052

[1.0, 0.0]
```

Figure 32: Project Demonstration - Predicting on a Sample Image using MobileNet

3.2.2 Predicting for a Sample Image Using InceptionV3

```
predict("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.jpg",
```

```
Image Path: /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.
jpg
Arhitecture Used:
<tensorflow.python.keras.engine.functional.Functional object at 0x7fa173ac0950>
Path for Model Weights:
/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/weights.best.InceptionV3.hdf5
Model Weights:
None
Prediction... Melanoma :  0.8329839  | Other :  0.1670161

[1.0, 0.0]
```

Figure 33: Project Demonstration - Predicting on a Sample Image using Inception

3.3.2 Predicting a Sample Image using Xception

```
predict("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.jpg",
```

```
Image Path: /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/ISIC-2017_Validation_Data/Data Image JPG/ISIC_0001769.
jpg
Arhitecture Used:
<tensorflow.python.keras.engine.functional.Functional object at 0x7fa2741a2650>
Path for Model Weights:
/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/xception_weights.hdf5
Model Weights:
None
Prediction... Melanoma :  0.7678717  | Other :  0.23212835

[1.0, 0.0]
```

Figure 34: Project Demonstration - Predicting on a Sample Image using Xception

EVALUATION OF MOBILENET ON VALIDATION DATA

4.1.5.1 Defining the Confusion Matrix Function

```
def positive_negative_measurement(y_true, y_score):
    # Initialization
    TRUE_POSITIVE = 0
    FALSE_POSITIVE = 0
    TRUE_NEGATIVE = 0
    FALSE_NEGATIVE = 0

    # Calculating the model
    for i in range(len(y_score)):
        if y_true[i] == y_score[i] == 1:
            TRUE_POSITIVE += 1
        if (y_score[i] == 1) and (y_true[i] != y_score[i]):
            FALSE_POSITIVE += 1
        if y_true[i] == y_score[i] == 0:
            TRUE_NEGATIVE += 1
        if (y_score[i] == 0) and (y_true[i] != y_score[i]):
            FALSE_NEGATIVE += 1

    return(TRUE_POSITIVE, FALSE_POSITIVE, TRUE_NEGATIVE, FALSE_NEGATIVE)
```

```
TRUE_POSITIVE_MobileNet, FALSE_POSITIVE_MobileNet, TRUE_NEGATIVE_MobileNet, FALSE_NEGATIVE_MobileNet = positive_negative_measurer
postives_negatives_MobileNet = [[TRUE_POSITIVE_MobileNet, FALSE_POSITIVE_MobileNet],
                                [FALSE_NEGATIVE_MobileNet, TRUE_NEGATIVE_MobileNet]]
```

```
postives_negatives_MobileNet
```

```
[[120, 30], [0, 0]]
```

Figure 35: Project Demonstration - Evaluation of MobileNet on Training Data - Confusion Matrix

4.1.5.2 Obtaining Labels

```
import seaborn as sns
sns.set()
labels_MobileNet =  np.array([['True positive: ' + str(TRUE_POSITIVE_MobileNet),
                               'False positive: ' + str(FALSE_POSITIVE_MobileNet)],
                              ['False negative: ' + str(FALSE_NEGATIVE_MobileNet),
                               'True negative: ' + str(TRUE_NEGATIVE_MobileNet)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_MobileNet, annot = labels_MobileNet, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa17161d410>
```

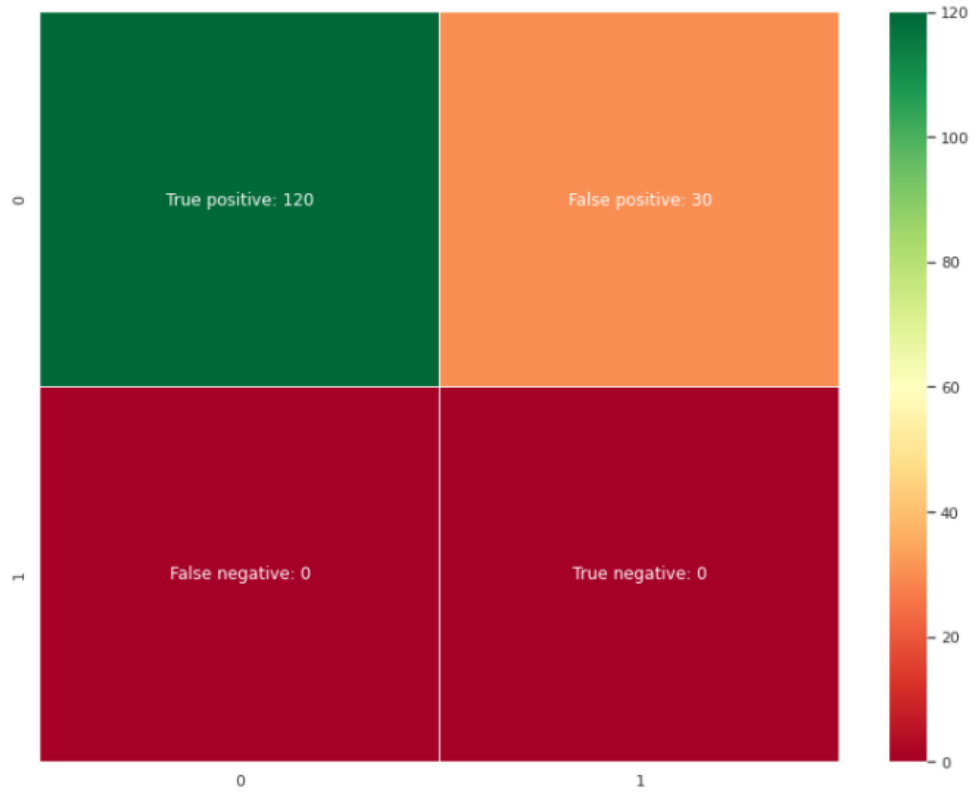


Figure 36: Project Demonstration - Evaluation of MobileNet on Training Data - Obtaining Labels

```
labels_MobileNet
```

```
array([['True positive: 120', 'False positive: 30'],
       ['False negative: 0', 'True negative: 0']], dtype='<U18')
```

4.1.5.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_MobileNet = TRUE_POSITIVE_MobileNet / (TRUE_POSITIVE_MobileNet + FALSE_NEGATIVE_MobileNet)
print("Sensitivity: ", sensitivity_MobileNet)
```

```
Sensitivity:  1.0
```

4.1.5.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_MobileNet = TRUE_NEGATIVE_MobileNet / (TRUE_NEGATIVE_MobileNet + FALSE_NEGATIVE_MobileNet)
    print("Specifity: ", specifity_MobileNet)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

4.1.5.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_MobileNet = TRUE_POSITIVE_MobileNet / (TRUE_POSITIVE_MobileNet + FALSE_POSITIVE_MobileNet)
print("Precision: ", predcision_MobileNet)
```

```
Precision:  0.8
```

Figure 37: Project Demonstration - Evaluation of MobileNet on Training Data(i)

4.1.5.6 Calculating Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_MobileNet = TRUE_NEGATIVE_MobileNet / (TRUE_NEGATIVE_MobileNet + FALSE_NEGATIVE_MobileNet)
    print("Negative predictive value: ", npv_MobileNet)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

4.1.5.7 Calculating Accuracy

```
# Accuracy
accuracy_MobileNet = (TRUE_POSITIVE_MobileNet + TRUE_NEGATIVE_MobileNet) / (TRUE_POSITIVE_MobileNet + FALSE_POSITIVE_MobileNet +
print("Accuracy: ", accuracy_MobileNet)
```

```
Accuracy:  0.8
```

Figure 38: Project Demonstration - Evaluation of MobileNet on Training Data(ii)

EVALUATION OF INCEPTION ON VALIDATION DATA

```
import seaborn as sns
sns.set()
labels_Inception =  np.array([['True positive: ' + str(TRUE_POSITIVE_Inception),
                     'False positive: ' + str(FALSE_POSITIVE_Inception)],
                    ['False negative: ' + str(FALSE_NEGATIVE_Inception),
                     'True negative: ' + str(TRUE_POSITIVE_Inception)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Inception, annot = labels_Inception, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa16660ec10>
```

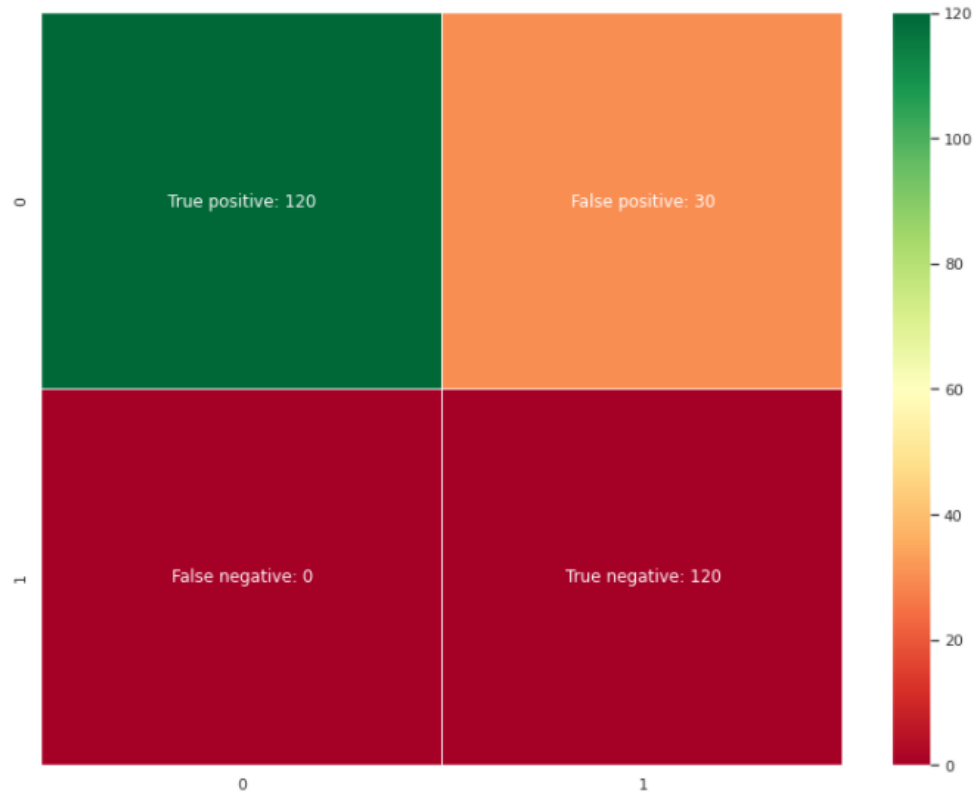


Figure 39: Project Demonstration - Evaluation of Inception on Training Data - Confusion Matrix

4.2.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Inception = TRUE_POSITIVE_Inception / (TRUE_POSITIVE_Inception + FALSE_NEGATIVE_Inception)
print("Sensitivity: ", sensitivity_Inception)
```

```
Sensitivity:  1.0
```

4.2.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Inception = TRUE_NEGATIVE_Inception / (TRUE_NEGATIVE_Inception + FALSE_NEGATIVE_Inception)
    print("Specifity: ", specifity_Inception)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

4.2.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Inception = TRUE_POSITIVE_Inception / (TRUE_POSITIVE_Inception + FALSE_POSITIVE_Inception)
print("Precision: ", predcision_Inception)
```

```
Precision:  0.8
```

Figure 40: Project Demonstration - Evaluation of Inception on Training Data(i)

4.2.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Inception = TRUE_NEGATIVE_Inception / (TRUE_NEGATIVE_Inception + FALSE_NEGATIVE_Inception)
    print("Negative predictive value: ", npv_Inception)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

4.2.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Inception = (TRUE_POSITIVE_Inception + TRUE_NEGATIVE_Inception) / (TRUE_POSITIVE_Inception + FALSE_POSITIVE_Inception +
print("Accuracy: ", accuracy_Inception)
```

```
Accuracy:  0.8
```

Figure 41: Project Demonstration - Evaluation of Inception on Training Data(ii)

EVALUATION OF XCEPTION ON VALIDATION DATA

4.3.2.4.2 Obtaining the Labels

```
import seaborn as sns
sns.set()
labels_Xception =  np.array([['True positive: ' + str(TRUE_POSITIVE_Xception),
                      'False positive: ' + str(FALSE_POSITIVE_Xception)],
                     ['False negative: ' + str(FALSE_NEGATIVE_Xception),
                      'True negative: ' + str(TRUE_POSITIVE_Xception)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Xception, annot = labels_Xception, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa1665d8610>
```

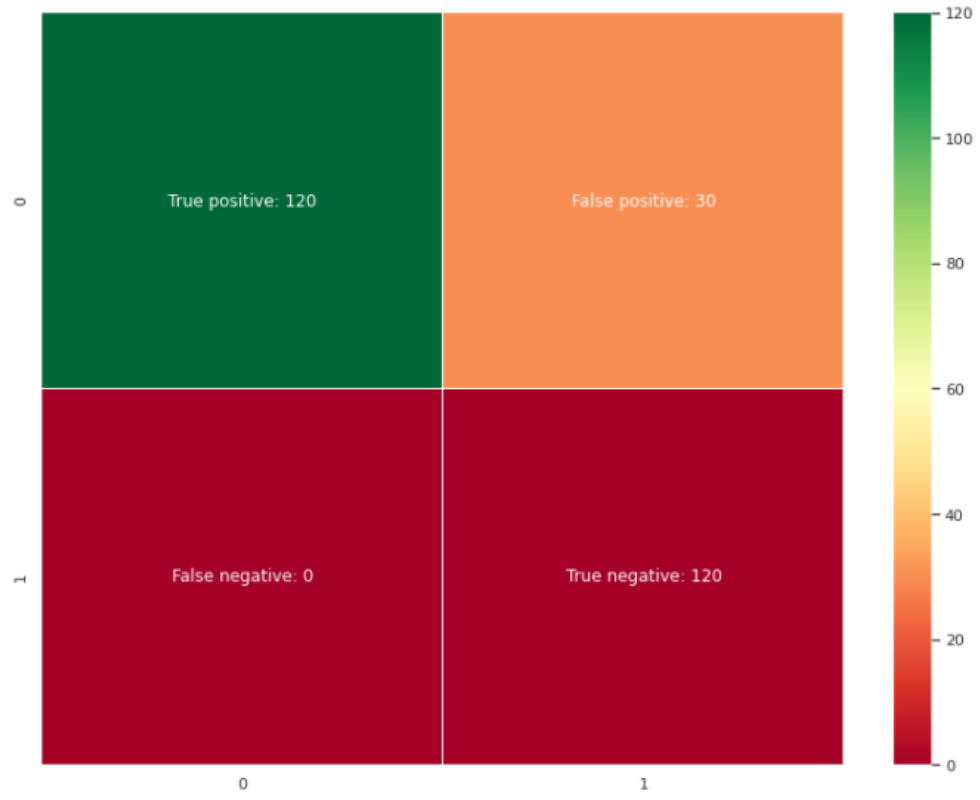


Figure 42: Project Demonstration - Evaluation of Xception on Training Data - Confusion Matrix

4.3.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Xception = TRUE_POSITIVE_Xception / (TRUE_POSITIVE_Xception + FALSE_NEGATIVE_Xception)
print("Sensitivity: ", sensitivity_Xception)
```

```
Sensitivity:  1.0
```

4.3.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Xception = TRUE_NEGATIVE_Xception / (TRUE_NEGATIVE_Xception + FALSE_NEGATIVE_Xception)
    print("Specifity: ", specifity_Xception)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

4.3.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Xception = TRUE_POSITIVE_Xception / (TRUE_POSITIVE_Xception + FALSE_POSITIVE_Xception)
print("Precision: ", predcision_Xception)
```

```
Precision:  0.8
```

Figure 43: Project Demonstration - Evaluation of Xception on Training Data(i)

4.3.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Xception = TRUE_NEGATIVE_Xception / (TRUE_NEGATIVE_Xception + FALSE_NEGATIVE_Xception)
    print("Negative predictive value: ", npv_Xception)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

4.3.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Xception = (TRUE_POSITIVE_Xception + TRUE_NEGATIVE_Xception) / (TRUE_POSITIVE_Xception + FALSE_POSITIVE_Xception + TRUE_
print("Accuracy: ", accuracy_Xception)
```

```
Accuracy:  0.8
```

Figure 44: Project Demonstration - Evaluation of Xception on Training Data (ii)

5.4 Defining the Ensembling Function

```
def ensemble(models, model_input):
    outputs = [model.outputs[0] for model in models]
    print("Outputs: ")
    print(outputs)
    y = keras.layers.Average()(outputs)
    print("y: ")
    print(y)
    model = Model(model_input, y, name='ensemble')
    print("Model: ")
    print(model)
    return model
```

```
# Getting ensemble model
ensemble_model = ensemble(models, model_input)
```

```
Outputs:
[<KerasTensor: shape=(None, 2) dtype=float32 (created by layer 'dense_9')>, <KerasTensor: shape=(None, 2) dtype=float32 (create
d by layer 'dense_10')>, <KerasTensor: shape=(None, 2) dtype=float32 (created by layer 'dense_11')>]
y:
KerasTensor(type_spec=TensorSpec(shape=(None, 2), dtype=tf.float32, name=None), name='average/truediv:0', description="created
by layer 'average'")
Model:
<tensorflow.python.keras.engine.functional.Functional object at 0x7fa1660b9910>
```

Figure 45: Project Demonstration - Evaluation of Models Together on Validation Data - Defining Ensemble Function

```
# load all models into memory
members = load_all_models(weight_file_names_with_path)
print('Loaded %d models' % len(members))
```

```
>loaded /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/weights.best.mobilenet.hdf5
>loaded /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/weights.best.InceptionV3.hdf5
>loaded /content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/xception_weights.hdf5
Loaded 3 models
```

Figure 46: Project Demonstration - Evaluation of Models Together on Validation Data - Loading all Model Weights

```
weights_of_MobileNet_Inception_and_Xception = c = np.concatenate((weights_of_MobileNet_and_Inception, weights_of_Xception[0])) #
print(type(weights_of_MobileNet_Inception_and_Xception))
print(weights_of_MobileNet_Inception_and_Xception.shape)
```

```
<class 'numpy.ndarray'>
(9, 3, 3, 32)
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
```

```
# serialize model to JSON
model_weights_for_json = ensemble_model.to_json()
with open("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/Ensemble Model/ensemble_model.json", "w") as
    json_file.write(model_weights_for_json)
# serialize weights to HDF5
ensemble_model.save_weights("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Saved Models/Ensemble Model/ensemble_mo
print("Saved model to disk")
```

```
Saved model to disk
```

Figure 47: Project Demonstration - Evaluation of Models Together on Validation Data - Serializing Weights into one

In [ ]:
```
# Compute test set predictions
NUMBER_TEST_SAMPLES_Ensemble = 150

y_true_Ensemble = valid_targets[:NUMBER_TEST_SAMPLES_Ensemble]
y_score_Ensemble = []
for index in range(NUMBER_TEST_SAMPLES_Ensemble): #compute one at a time due to memory constraints
    image_to_predict_Ensemble = path_to_tensor(validation_files[index]).astype("float32")/255.
    probs_Ensemble = ensemble_model.predict(image_to_predict_Ensemble,)
    if np.argmax(probs_Ensemble) == 0:
        y_score_Ensemble.append([1., 0.])
    elif np.argmax(probs_Ensemble) == 1:
        y_score_Ensemble.append([0., 1.])
    print("Predicted value {}... ".format(index+1) + " Melanoma : ", probs_Ensemble[0][0],  " | Other : ", probs_Ensemble[0][1])
    print("Real values {}...".format(index+1) + "      Melanoma : ", valid_targets[index][0], "      | Other : ", valid_targets[i
    print("-------------------------------------------------------------------------------")


correct_Ensemble = np.array(y_true_Ensemble) == np.array(y_score_Ensemble)
```

```
Predicted value 1...  Melanoma :  0.8269514  | Other :  0.17304868
Real values 1...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 2...  Melanoma :  0.8224184  | Other :  0.17758167
Real values 2...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 3...  Melanoma :  0.83412284  | Other :  0.16587715
Real values 3...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 4...  Melanoma :  0.82447284  | Other :  0.17552719
Real values 4...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 5...  Melanoma :  0.8221907  | Other :  0.1778094
Real values 5...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 6...  Melanoma :  0.83770597  | Other :  0.16229409
Real values 6...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
Predicted value 7...  Melanoma :  0.82638013  | Other :  0.17361985
Real values 7...      Melanoma :  1.0       | Other :  0.0
-------------------------------------------------------------------------------
```

In [ ]:
```
print("Accuracy = %2.2f%%" % (np.mean(correct_Ensemble)*100))
```

```
Accuracy = 80.00%
```

In [ ]:
```
image_to_predict_Ensemble = path_to_tensor("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Output_melanoma/ISIC_006
ensemble_model.predict(image_to_predict_Ensemble)
```

Out[162]:
```
array([[0.8248186 , 0.17518133]], dtype=float32)
```

Figure 48: Project Demonstration - Evaluation of Models Together on Validation Data - Computing Test Set Predictions

5.5.2.4.2 Obtaining the Labels

```
import seaborn as sns
sns.set()
labels_Ensemble =  np.array([['True positive: ' + str(TRUE_POSITIVE_Ensemble),
                      'False positive: ' + str(FALSE_POSITIVE_Ensemble)],
                     ['False negative: ' + str(FALSE_NEGATIVE_Ensemble),
                      'True negative: ' + str(TRUE_POSITIVE_Ensemble)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Ensemble, annot = labels_Ensemble, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa164688950>
```

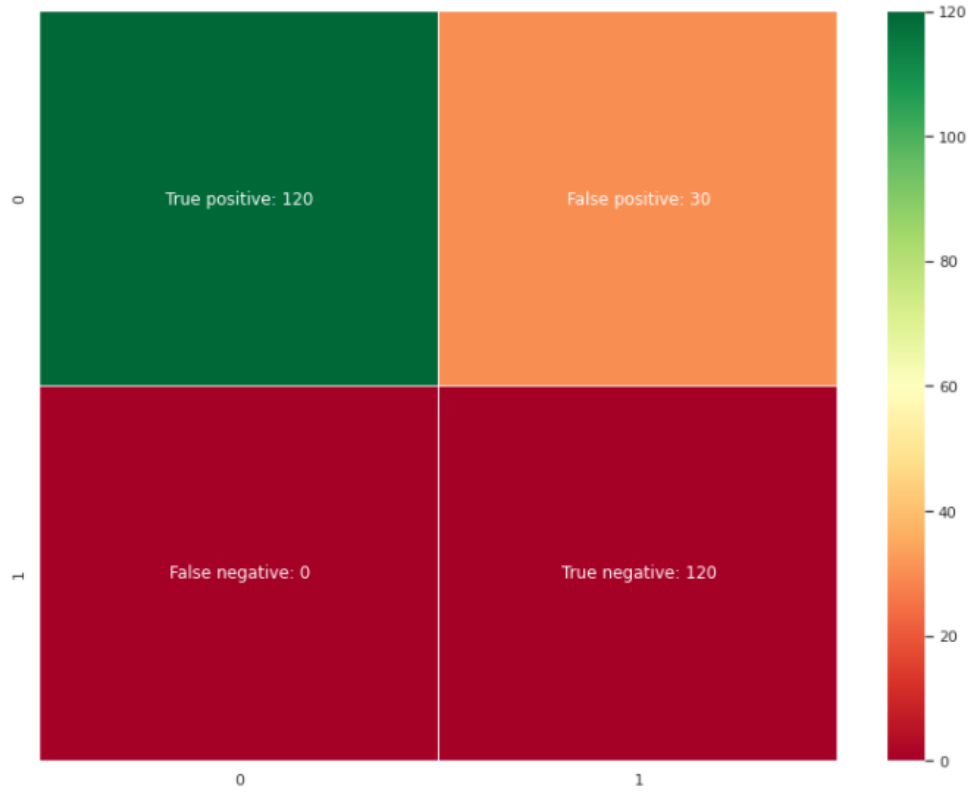


Figure 49: Project Demonstration - Evaluation of Models Together on Validation Data - Obtaining Labels

5.5.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Ensemble = TRUE_POSITIVE_Ensemble / (TRUE_POSITIVE_Ensemble + FALSE_NEGATIVE_Ensemble)
print("Sensitivity: ", sensitivity_Ensemble)
```

```
Sensitivity:  1.0
```

5.5.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Ensemble = TRUE_NEGATIVE_Ensemble / (TRUE_NEGATIVE_Ensemble + FALSE_NEGATIVE_Ensemble)
    print("Specifity: ", specifity_Ensemble)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

5.5.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Ensemble = TRUE_POSITIVE_Ensemble / (TRUE_POSITIVE_Ensemble + FALSE_POSITIVE_Ensemble)
print("Precision: ", predcision_Ensemble)
```

```
Precision:  0.8
```

Figure 50: Project Demonstration - Evaluation of Models Together on Validation Data(i)

5.5.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Ensemble = TRUE_NEGATIVE_Ensemble / (TRUE_NEGATIVE_Ensemble + FALSE_NEGATIVE_Ensemble)
    print("Negative predictive value: ", npv_Ensemble)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

5.5.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Ensemble = (TRUE_POSITIVE_Ensemble + TRUE_NEGATIVE_Ensemble) / (TRUE_POSITIVE_Ensemble + FALSE_POSITIVE_Ensemble + TRUE_
print("Accuracy: ", accuracy_Ensemble)
```

```
Accuracy:  0.8
```

Figure 51: Project Demonstration - Evaluation of Models Together on Validation Data(ii)

EVALUATING THE MODELS INDIVIDUALLY ON TESTING DATA

MOBILENET

6.1.5 Confusion Matrix

6.1.5.1 Defining the Confusion Matrix Function

```
def positive_negative_measurement(y_true, y_score):
    # Initialization
    TRUE_POSITIVE = 0
    FALSE_POSITIVE = 0
    TRUE_NEGATIVE = 0
    FALSE_NEGATIVE = 0

    # Calculating the model
    for i in range(len(y_score)):
        if y_true[i] == y_score[i] == 1:
            TRUE_POSITIVE += 1
        if (y_score[i] == 1) and (y_true[i] != y_score[i]):
            FALSE_POSITIVE += 1
        if y_true[i] == y_score[i] == 0:
            TRUE_NEGATIVE += 1
        if (y_score[i] == 0) and (y_true[i] != y_score[i]):
            FALSE_NEGATIVE += 1

    return(TRUE_POSITIVE, FALSE_POSITIVE, TRUE_NEGATIVE, FALSE_NEGATIVE)
```

```
TRUE_POSITIVE_MobileNet_Test, FALSE_POSITIVE_MobileNet_Test, TRUE_NEGATIVE_MobileNet_Test, FALSE_NEGATIVE_MobileNet_Test = positi
postives_negatives_MobileNet_Test = [[TRUE_POSITIVE_MobileNet_Test, FALSE_POSITIVE_MobileNet_Test],
                                     [FALSE_NEGATIVE_MobileNet_Test, TRUE_NEGATIVE_MobileNet_Test]]
```

```
postives_negatives_MobileNet_Test
```

```
[[483, 117], [0, 0]]
```

Figure 52: Project Demonstration - Evaluation of MobileNet Model Individually on Testing Data - Confusion Matrix

6.1.5.2 Obtaining Labels

```
import seaborn as sns
sns.set()
labels_MobileNet_Test =  np.array([['True positive: ' + str(TRUE_POSITIVE_MobileNet_Test),
                                    'False positive: ' + str(FALSE_POSITIVE_MobileNet_Test)],
                                   ['False negative: ' + str(FALSE_NEGATIVE_MobileNet_Test),
                                    'True negative: ' + str(TRUE_NEGATIVE_MobileNet_Test)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_MobileNet_Test, annot = labels_MobileNet_Test, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa2749d8fd0>
```

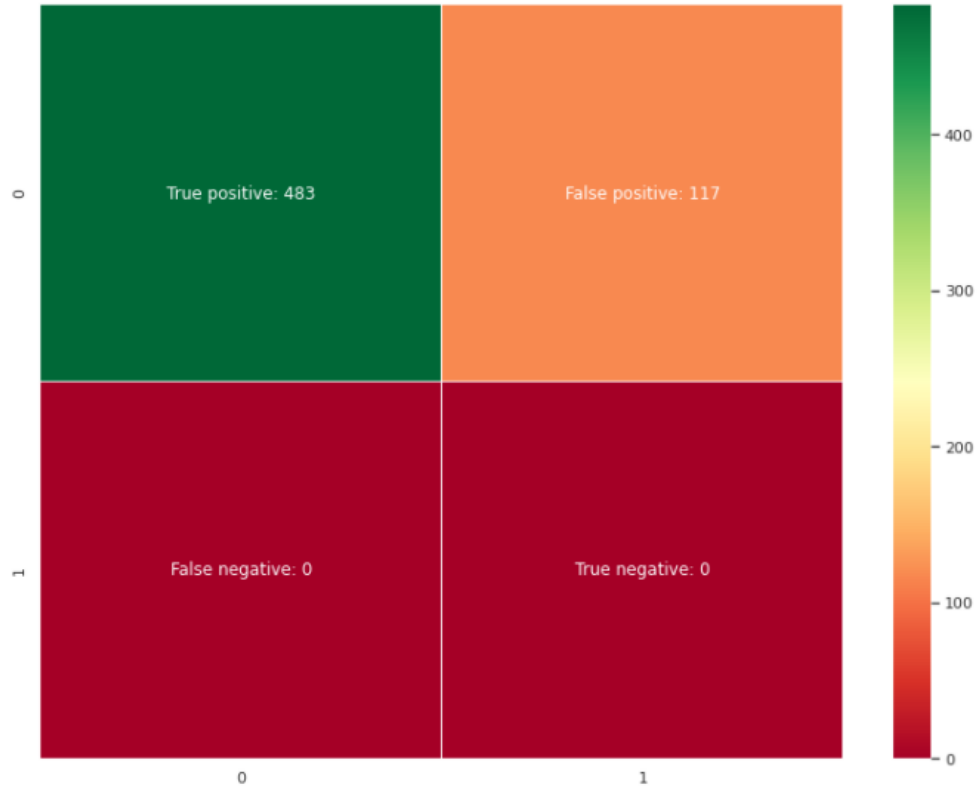


Figure 53: Project Demonstration - Evaluation of MobileNet Model Individually on Testing Data - Obtaining Labels

```
labels_MobileNet_Test
```

```
array([['True positive: 483', 'False positive: 117'],
       ['False negative: 0', 'True negative: 0']], dtype='<U19')
```

6.1.5.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_MobileNet_Test = TRUE_POSITIVE_MobileNet_Test / (TRUE_POSITIVE_MobileNet_Test + FALSE_NEGATIVE_MobileNet_Test)
print("Sensitivity: ", sensitivity_MobileNet_Test)
```

```
Sensitivity:  1.0
```

6.1.5.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_MobileNet_Test = TRUE_NEGATIVE_MobileNet_Test / (TRUE_NEGATIVE_MobileNet_Test + FALSE_NEGATIVE_MobileNet_Test)
    print("Specifity: ", specifity_MobileNet_Test)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

6.1.5.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_MobileNet_Test = TRUE_POSITIVE_MobileNet_Test / (TRUE_POSITIVE_MobileNet_Test + FALSE_POSITIVE_MobileNet_Test)
print("Precision: ", predcision_MobileNet_Test)
```

```
Precision:  0.805
```

Figure 54: Project Demonstration - Evaluation of MobileNet Model Individually on Testing Data (i)

6.1.5.6 Calculating Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_MobileNet_Test = TRUE_NEGATIVE_MobileNet_Test / (TRUE_NEGATIVE_MobileNet_Test + FALSE_NEGATIVE_MobileNet_Test)
    print("Negative predictive value: ", npv_MobileNet_Test)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

6.1.5.7 Calculating Accuracy

```
# Accuracy
accuracy_MobileNet_Test = (TRUE_POSITIVE_MobileNet_Test + TRUE_NEGATIVE_MobileNet_Test) / (TRUE_POSITIVE_MobileNet_Test + FALSE_P
print("Accuracy: ", accuracy_MobileNet_Test)
```

```
Accuracy:  0.805
```

Figure 55: Project Demonstration - Evaluation of MobileNet Model Individually on Testing Data (ii)

INCEPTION

```
import seaborn as sns
sns.set()
labels_Inception_Test =  np.array([['True positive: ' + str(TRUE_POSITIVE_Inception_Test),
                                    'False positive: ' + str(FALSE_POSITIVE_Inception_Test)],
                                   ['False negative: ' + str(FALSE_NEGATIVE_Inception_Test),
                                    'True negative: ' + str(TRUE_POSITIVE_Inception_Test)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Inception_Test, annot = labels_Inception_Test, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa166505390>
```

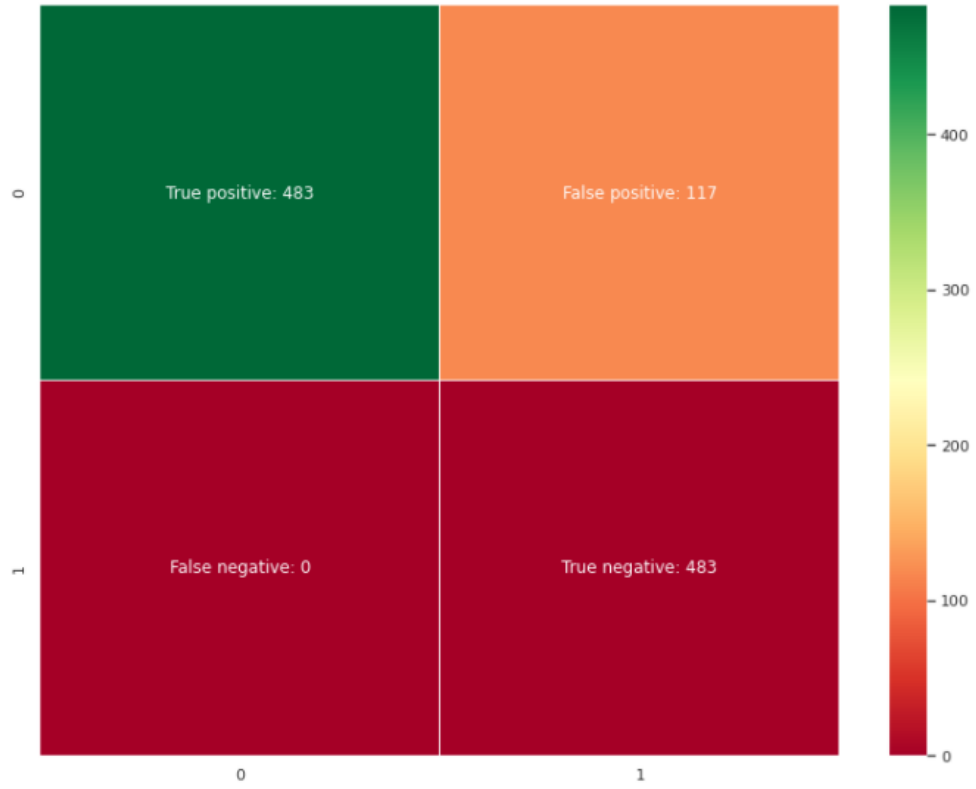


Figure 56: Project Demonstration - Evaluation of Inception Model Individually on Testing Data - Confusion Matrix

6.2.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Inception_Test = TRUE_POSITIVE_Inception_Test / (TRUE_POSITIVE_Inception_Test + FALSE_NEGATIVE_Inception_Test)
print("Sensitivity: ", sensitivity_Inception_Test)
```

```
Sensitivity:  1.0
```

6.2.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Inception_Test = TRUE_NEGATIVE_Inception_Test / (TRUE_NEGATIVE_Inception_Test + FALSE_NEGATIVE_Inception_Test)
    print("Specifity: ", specifity_Inception_Test)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

6.2.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Inception_Test = TRUE_POSITIVE_Inception_Test / (TRUE_POSITIVE_Inception_Test + FALSE_POSITIVE_Inception_Test)
print("Precision: ", predcision_Inception_Test)
```

```
Precision:  0.805
```

Figure 57: Project Demonstration - Evaluation of Inception Model Individually on Testing Data(i)

6.2.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Inception_Test = TRUE_NEGATIVE_Inception_Test / (TRUE_NEGATIVE_Inception_Test + FALSE_NEGATIVE_Inception_Test)
    print("Negative predictive value: ", npv_Inception_Test)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

6.2.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Inception_Test = (TRUE_POSITIVE_Inception_Test + TRUE_NEGATIVE_Inception_Test) / (TRUE_POSITIVE_Inception_Test + FALSE_F
print("Accuracy: ", accuracy_Inception_Test)
```

```
Accuracy:  0.805
```

Figure 58: Project Demonstration - Evaluation of Inception Model Individually on Testing Data(ii)

XCEPTION

6.3.2.4.2 Obtaining the Labels

```
import seaborn as sns
sns.set()
labels_Xception_Test =  np.array([['True positive: ' + str(TRUE_POSITIVE_Xception_Test),
                                   'False positive: ' + str(FALSE_POSITIVE_Xception_Test)],
                                  ['False negative: ' + str(FALSE_NEGATIVE_Xception_Test),
                                   'True negative: ' + str(TRUE_POSITIVE_Xception_Test)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Xception_Test, annot = labels_Xception_Test, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa172937750>
```

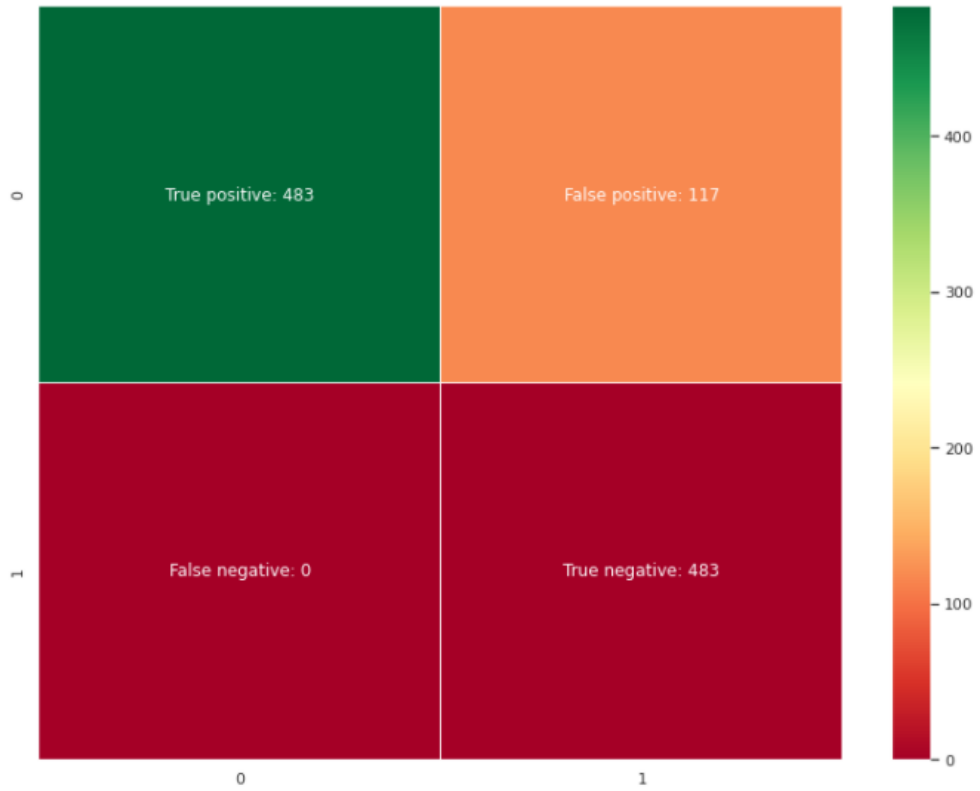


Figure 59: Project Demonstration - Evaluation of Xception Model Individually on Testing Data - Obtaining Labels

6.3.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Xception_Test = TRUE_POSITIVE_Xception_Test / (TRUE_POSITIVE_Xception_Test + FALSE_NEGATIVE_Xception_Test)
print("Sensitivity: ", sensitivity_Xception_Test)
```

```
Sensitivity:  1.0
```

6.3.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Xception_Test = TRUE_NEGATIVE_Xception_Test / (TRUE_NEGATIVE_Xception_Test + FALSE_NEGATIVE_Xception_Test)
    print("Specifity: ", specifity_Xception_Test)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

6.3.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Xception_Test = TRUE_POSITIVE_Xception_Test / (TRUE_POSITIVE_Xception_Test + FALSE_POSITIVE_Xception_Test)
print("Precision: ", predcision_Xception_Test)
```

```
Precision:  0.805
```

Figure 60: Project Demonstration - Evaluation of Xception Model Individually on Testing Data(i)

6.3.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Xception_Test = TRUE_NEGATIVE_Xception_Test / (TRUE_NEGATIVE_Xception_Test + FALSE_NEGATIVE_Xception_Test)
    print("Negative predictive value: ", npv_Xception_Test)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

6.3.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Xception_Test = (TRUE_POSITIVE_Xception_Test + TRUE_NEGATIVE_Xception_Test) / (TRUE_POSITIVE_Xception_Test + FALSE_POSIT
print("Accuracy: ", accuracy_Xception_Test)
```

```
Accuracy:  0.805
```

Figure 61: Project Demonstration - Evaluation of Xception Model Individually on Testing Data(ii)

EVALUATING THE MODELS TOGETHER ON TESTING DATA - ENSEMBLING THE MODELS

```
print("Accuracy = %2.2f%%" % (np.mean(correct_Ensemble_Test)*100))
```

```
Accuracy = 80.50%
```

```
image_to_predict_Ensemble_Test = path_to_tensor("/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/Output_melanoma/ISI
ensemble_model.predict(image_to_predict_Ensemble_Test)
```

```
array([[0.8248186 , 0.17518133]], dtype=float32)
```

Figure 62: Project Demonstration - Evaluation of Models Together on a Sample Image

7.5.2.4.2 Obtaining the Labels

```
import seaborn as sns
sns.set()
labels_Ensemble_Test =  np.array([['True positive: ' + str(TRUE_POSITIVE_Ensemble_Test),
                                   'False positive: ' + str(FALSE_POSITIVE_Ensemble_Test)],
                                  ['False negative: ' + str(FALSE_NEGATIVE_Ensemble_Test),
                                   'True negative: ' + str(TRUE_POSITIVE_Ensemble_Test)]])
plt.figure(figsize = (13, 10))
sns.heatmap(postives_negatives_Ensemble_Test, annot = labels_Ensemble_Test, linewidths = 0.1, fmt="", cmap = 'RdYlGn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa162f37dd0>
```

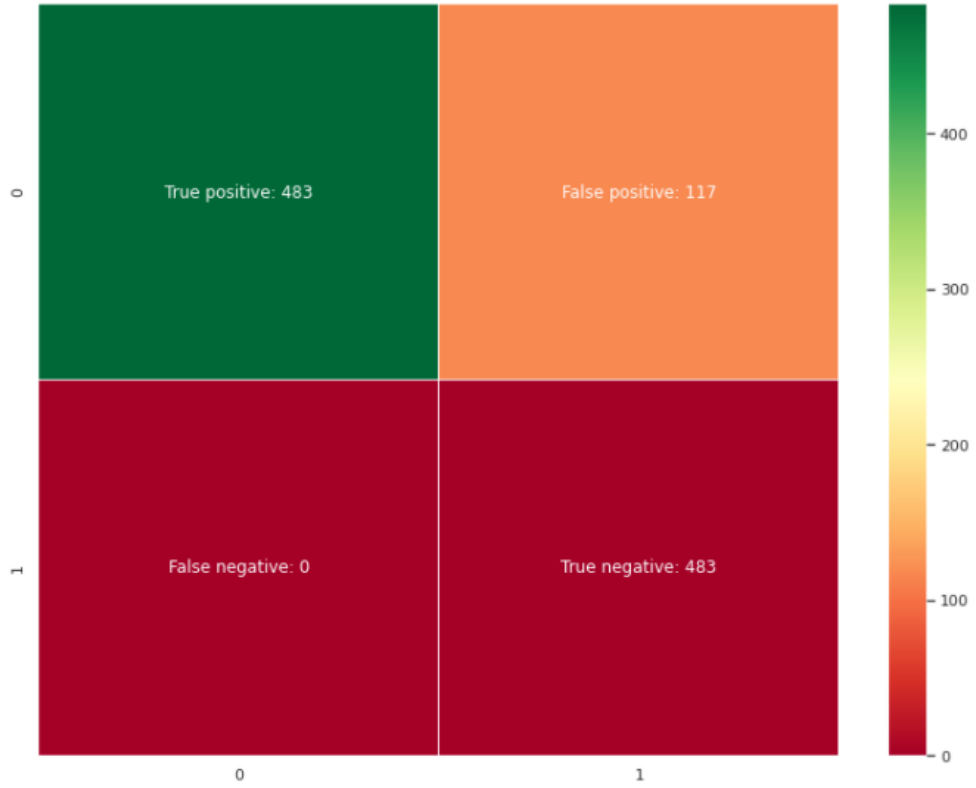


Figure 63: Project Demonstration - Evaluation of Models Together - Obtaining the Labels

7.5.2.4.3 Calculating Sensitivity/Recall/Hit Rate/True Positive Rate

```
# Sensitivity | Recall | hit rate | true positive rate (TPR)
sensitivity_Ensemble_Test = TRUE_POSITIVE_Ensemble_Test / (TRUE_POSITIVE_Ensemble_Test + FALSE_NEGATIVE_Ensemble_Test)
print("Sensitivity: ", sensitivity_Ensemble_Test)
```

```
Sensitivity:  1.0
```

7.5.2.4.4 Calculating Specificity/Selectivity/True Negative Rate

```
# Specificity | selectivity | true negative rate (TNR)
try:
    specifity_Ensemble_Test = TRUE_NEGATIVE_Ensemble_Test / (TRUE_NEGATIVE_Ensemble_Test + FALSE_NEGATIVE_Ensemble_Test)
    print("Specifity: ", specifity_Ensemble_Test)
except:
    print("No Specificity due to NO NEGATIVE results.")
```

```
No Specificity due to NO NEGATIVE results.
```

7.5.2.4.5 Calculating Precision/Positive Predictive Value

```
# Precision | positive predictive value (PPV)
predcision_Ensemble_Test = TRUE_POSITIVE_Ensemble_Test / (TRUE_POSITIVE_Ensemble_Test + FALSE_POSITIVE_Ensemble_Test)
print("Precision: ", predcision_Ensemble_Test)
```

```
Precision:  0.805
```

Figure 64: Project Demonstration - Evaluation of Models Together on Metrics(i)

7.5.2.4.6 Negative Predictive Value

```
# Negative predictive value (NPV)
try:
    npv_Ensemble_Test = TRUE_NEGATIVE_Ensemble_Test / (TRUE_NEGATIVE_Ensemble_Test + FALSE_NEGATIVE_Ensemble_Test)
    print("Negative predictive value: ", npv_Ensemble_Test)
except:
    print("0 Negative Predictions")
```

```
0 Negative Predictions
```

7.5.2.4.7 Calculating Accuracy

```
# Accuracy
accuracy_Ensemble_Test = (TRUE_POSITIVE_Ensemble_Test + TRUE_NEGATIVE_Ensemble_Test) / (TRUE_POSITIVE_Ensemble_Test + FALSE_POSIT
print("Accuracy: ", accuracy_Ensemble_Test)
```

```
Accuracy:  0.805
```

Figure 65: Project Demonstration - Evaluation of Models Together on Metrics(ii)

8.6 Visualizing the Images

```
# Visualizing images with and without localization
# Canvas
fig, ax = plt.subplots(nrows=1, ncols=2, figsize = (10, 10))
# Image without localization
ax[0].imshow((path_to_tensor(img_path).astype('float32')/255).squeeze())
# Image with localization
CAM = plot_CAM(img_path, ax[1], mobilenet_model, all_amp_layer_weights)
plt.show()
```

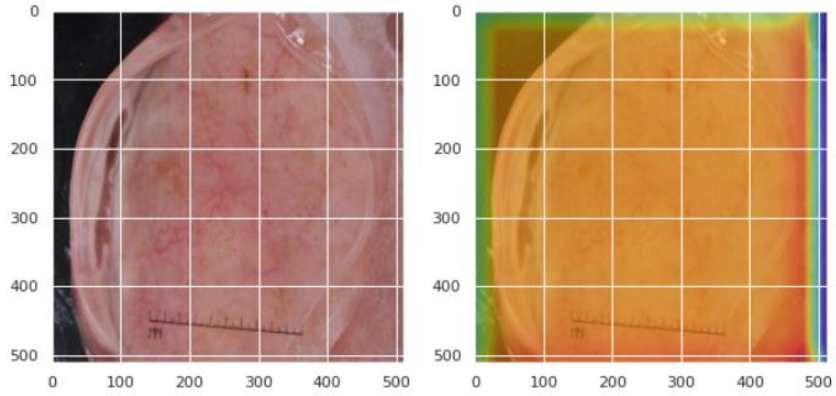


Figure 66: Project Demonstration - Localization - Visualizing Images

```
# Getting the image tensor
image_to_predict = path_to_tensor(img_path).astype('float32')/255

# Predicting the image
prediction = ensemble_model.predict(image_to_predict)
prediction_final = "Melanoma: " + str(np.round(predict_0_0*100, decimals = 4)) + "%" + \
                   " | Other illness: " + str(np.round(predict_0_1*100, decimals = 4)) + "%"

# Canvas initialization
fig = plt.figure(figsize = (10, 10))

# First image
ax = fig.add_subplot(121)
ax.imshow(image_to_predict.squeeze())
ax.text(0.3, 1.6, prediction_final)

# Second image
ax = fig.add_subplot(122)
CAM = plot_CAM(img_path, ax, mobilenet_model, all_amp_layer_weights)

plt.show()
```

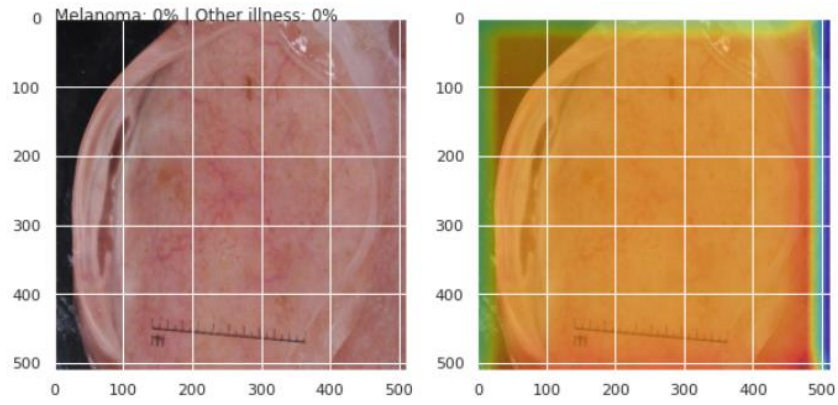


Figure 67: Project Demonstration - Localization - Display of Result

SAVING THE COMPLETE MODEL FOR PYTHON INTERFACE APPLICATION

9. Saving the Complete Model for the Python Interface Application

```
In [ ]: %cd "/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/FINAL SAVED OUTPUTS/"
ensemble_model.save('FINAL_FILE_for_Soft_Computing_Project_Skin_Cancer.h5')
from tensorflow.keras.models import load_model
h5_saved_ensemble_model = load_model('FINAL_FILE_for_Soft_Computing_Project_Skin_Cancer.h5')
h5_saved_ensemble_model.summary()
```

```
/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/FINAL SAVED OUTPUTS
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
Model: "ensemble"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_11 (InputLayer)        [(None, 512, 512, 3) 0
_________________________________________________________________________________________________
conv2d_490 (Conv2D)          (None, 255, 255, 32) 864         input_11[0][0]
_________________________________________________________________________________________________
batch_normalization_490 (BatchN (None, 255, 255, 32) 96          conv2d_490[0][0]
_________________________________________________________________________________________________
activation_470 (Activation)  (None, 255, 255, 32) 0           batch_normalization_490[0][0]
_________________________________________________________________________________________________
conv2d_491 (Conv2D)          (None, 253, 253, 32) 9216        activation_470[0][0]
_________________________________________________________________________________________________
batch_normalization_491 (BatchN (None, 253, 253, 32) 96          conv2d_491[0][0]
_________________________________________________________________________________________________
activation_471 (Activation)  (None, 253, 253, 32) 0           batch_normalization_491[0][0]
```

Figure 68: Project Demonstration - Saving the Model for Application Interface

CONVERTING THE .H5 FILE TO .TFLITE FILE FOR THE PYTHON INTERFACE APPLICATION

10. Converting the .h5 File to .tflite File for the Python Interface Application

```
import tensorflow as tf

saved_ensemble_model = tf.keras.models.load_model('/content/drive/MyDrive/dataset/ISIC Challenge 2017 Organized/FINAL SAVED OUTPU
converter = tf.lite.TFLiteConverter.from_keras_model(saved_ensemble_model)
tflite_model = converter.convert()
open("FINAL_FILE_for_Interface_Soft_Computing_Project_Skin_Cancer.tflite", "wb").write(tflite_model)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
INFO:tensorflow:Assets written to: /tmp/tmp_6qz3iyy/assets

183140620
```

Figure 69: Project Demonstration - Converting the H5 File to TFLITE File

COST ANALYSIS, RESULT AND DISCUSSION

The project required to be subscribed to the Pro version of Google Colab which is on a Recurring Billing system and charges with $9.99/month excluding the taxes. The subscription was needed for 3 months which is the time period required for the project to complete.

SUMMARY

With the help of the model created with the help of Transfer Learning and Ensemble Modelling, we were able to achieve an accuracy of 80.5%. we have the results as:

Evaluation Type	Architecture	Confusion Matrix				Sensitivity/ Recall	Specificity	Precision	Negative Predictive Value	Accuracy
		True Positive	False Positive	True Negative	False Negative					
	MobileNet	120	30	0	0	1	0	0.8	0	0.8

Evaluation of Individual Models on Validation Data	Inception	120	30	0	120	1	0	0.8	0	0.8
	Xception	120	30	0	120	1	0	0.8	0	0.8
Evaluation of Ensemble Model on Validation Data	Ensemble Model	120	30	0	120	1	0	0.8	0	0.8
Evaluation of Individual Models on Testing Data	MobileNet	483	117	0	0	1	0	0.805	0	0.805
	Inception	483	117	0	483	1	0	0.805	0	0.805
	Xception	483	117	0	483	1	0	0.805	0	0.805
Evaluation of Ensemble Model on Testing Data	Ensemble Model	483	117	0	483	1	0	0.805	0	0.805

REFERENCES

WEBSITES

- https://www.analyticsvidhya.com/blog/2020/07/types-of-feature-transformation-and-scaling/
- https://towardsdatascience.com/detecting-and-treating-outliers-in-python-part-3-dcb54abaf7b0
- https://towardsdatascience.com/data-visualization-for-eda-exploratory-data-analysis-f001a1bf0087
- https://analyticsindiamag.com/exploratory-data-analysis-functions-types-tools/
- https://medium.com/mljcunito/the-last-step-in-data-preprocessing-handling-missing-values-ae8f19dae309
- https://www.kdnuggets.com/2020/07/easy-guide-data-preprocessing-python.html

CONCEPTS & INFORMATION

- ResearchGate.com
- ScienceDirect.com
- GeeksforGeeks.com
- TutorialsPoint.com
- cs.stanford.edu
- Springer.com
- towardsdatascience.com
- academictorrents.com
- kaagle.com
- ncbi.nlm.nih.gov
- springml.com

- https://www.researchgate.net/publication/334751850_Skin_Cancer_Detection_Using_Convolutional_Neural_Network
- Peer-review under responsibility of the scientific committee of the 16th International Learning & Technology Conference 2019. 10.1016/j.procs.2019.12.090
- https://thesai.org/Publications/IJACSA
- https://www.thelancet.com/journals/ebiom/article/PIIS2352-3964(19)30294-4/fulltext
- https://www.sciencedirect.com/science/article/pii/S2215016120300832?via%3Dihub
- https://www.sciencedirect.com/science/article/pii/S1532046418301618?via%3Dihub
- https://www.ejcancer.com/article/S0959-8049(19)30381-8/fulltext
- https://www.researchgate.net/publication/335321267_Bio-Inspired_DeepCNN_Pipeline_for_Skin_Cancer_Early_Diagnosis
- https://www.researchgate.net/publication/347927500_Automated_Multiclass_Classification_of_Skin_Lesions_through_Deep_Convolutional_Neural_Network_with_Dermoscopic_Images
- https://www.researchgate.net/publication/339804392_Deep_Learning_Solutions_for_Skin_Cancer_Detection_and_Diagnosis?enrichId=rgreq-cc1ddabe639ba17298c595c27b48021c-XXX&enrichSource=Y292ZXJQYWdlOzMzOTgwNDM5MjtBUzo4ODM0MjE3NjUxNzMyNTFAMTU4NzYzNTU3MDg4Nw%3D%3D&el=1_x_3&_esc=publicationCoverPdf
- https://ieeexplore.ieee.org/document/9062473
- https://www.sciencedirect.com/science/article/pii/S0933365719301460
- https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3352407
- https://www.researchgate.net/publication/335337495_A_CNN_toolbox_for_skin_cancer_classification?enrichId=rgreq-91f014447a4c29cec1b8c22e6f8930fb-XXX&enrichSource=Y292ZXJQYWdlOzMzNTMzNzQ5NTtBUzo4MDExNDUyMDMyMjQ1NzdAMTU2ODAxOTMwOTE5Nw%3D%3D&el=1_x_3&_esc=publicationCoverPdf
- https://ieeexplore.ieee.org/document/8720210
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6231861/
- https://iopscience.iop.org/article/10.1088/1757-899X/982/1/012005
- https://www.researchgate.net/publication/327539277_Skin_Cancer_Classification_using_Convolutional_Neural_Networks_Systematic_Review_Preprint
- https://www.sciencedirect.com/science/article/pii/S2352914819302047
- https://www.researchgate.net/publication/334123580_Melanoma_Skin_Cancer_Detection_using_Image_Processing_and_Machine_Learning
- https://www.nature.com/articles/nature21056
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9113301
- https://pubmed.ncbi.nlm.nih.gov/28117445/
- https://pubmed.ncbi.nlm.nih.gov/30333097/
- https://ieeexplore.ieee.org/document/9198489