Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Bhagyalakshmi Shridhar Bichchal
Student ID: x23109149

School of Computing
National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Bhagyalakshmi Shridhar Bichchal
Student ID:	x23109149
Programme:	MSc in Artificial Intelligence
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Victor del Rosal
Submission Due Date:	12/08/2024
Project Title:	Integrating Deep Learning Algorithms into a Web Application for Accurate Melanoma Skin Cancer Detection
Word Count:	451
Page Count:	12

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

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

Signature:	Bhagyalakshmi Shridhar Bichchal
Date:	12th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Configuration Manual

Bhagyalakshmi Shridhar Bichchal
x23109149

1 Introduction

The purpose of this configuration manual is to provide a detailed information about the steps that were kept in mind to complete the research project titled ”Integrating Deep Learning Algorithms into a Web Application for Accurate Melanoma Skin Cancer Detection”. The information contains System Configuration, Software and Hardware specifications, development and deployment process with the tasks which are required to run the code.

2 System and Software Requirements

The Project was developed and implemented on the below configuration:

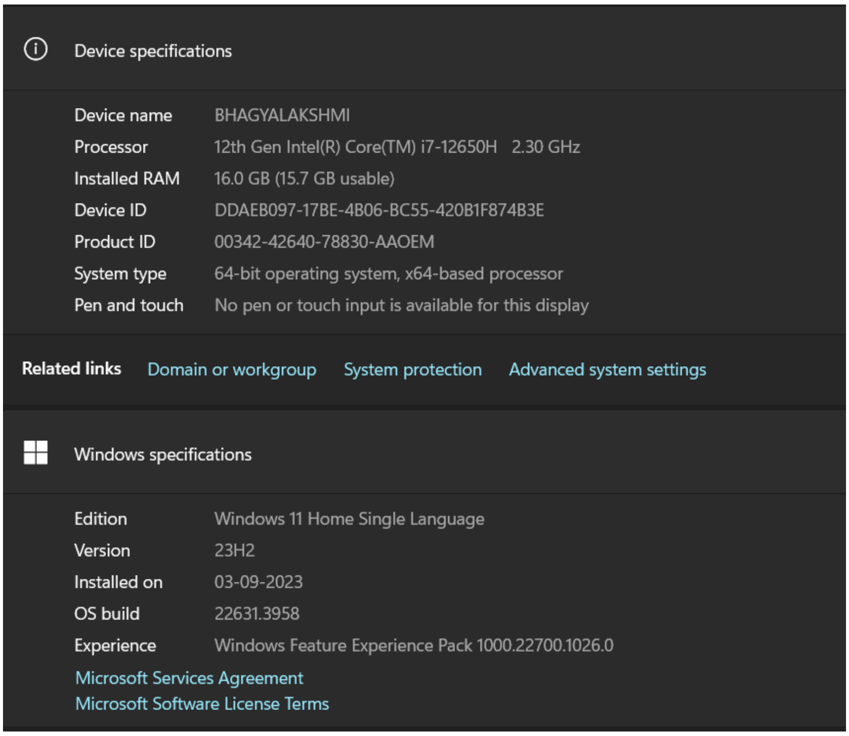


Figure 1: System Configuration

Table 1: Hardware Configuration

Operating System	Windows 11
RAM	56.9 GB (Google Colab)
Disk Space	201.23 GB (Google Colab)
Runtime Model Name	12th Gen Intel(R) Core(TM) i7-12650H @ 2.30 GHz

2.1 Software Requirements:

1. **Programming Language:** Python 3.10.12
2. **IDE:** Jupyter Notebook
3. **Tensorflow Version:** 2.15.0

3 Python Libraries:

I used the following python libraries to conduct my research project of predicting melanoma skin cancer:

1. Pandas
2. Numpy
3. OS
4. CV2
5. Matplotlib
6. Seaborn
7. Plotly
8. Tensorflow
9. Keras
10. Sklearn

4 Dataset Description:

- The dataset which is used for this research project is publicly available on kaggle hosted by user Hasnain Javed link:Melanoma Skin Cancer Dataset on Kaggle.

4.1 Description:

Dataset contain 10,000 dermatoscopic images which includes two classes: Malignant and Benign which provides a balanced framework for training models.

Size and Structure:

- **Total Images:** 10,000.
- **Training Set:** 9,600 Images.
- **Evaluation Set:** 1,000 Images.
- **Classes:** Malignant, Benign.
- **Image format:** JPEG.

5 Data Analysis and Visualisation:

5.1 Data Distribution:

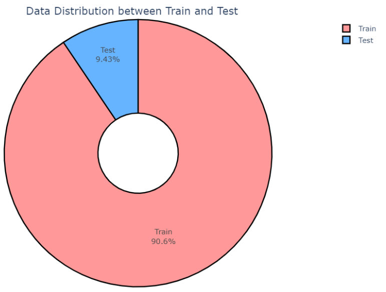


Figure 2: Data Distribution

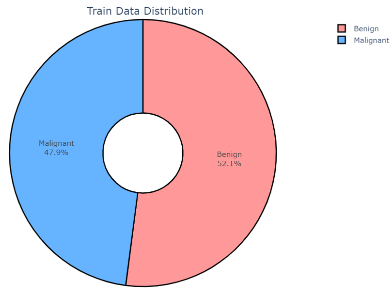


Figure 3: Train Data Distribution

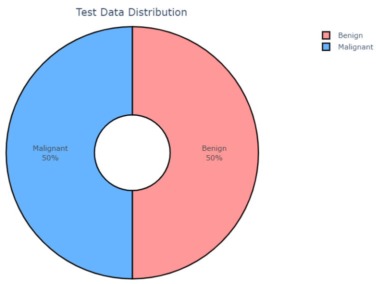


Figure 4: Test Data Distribution

5.2 Understanding Data:

Visualizing some random images from data from benign class and Malignant class to understand the data.

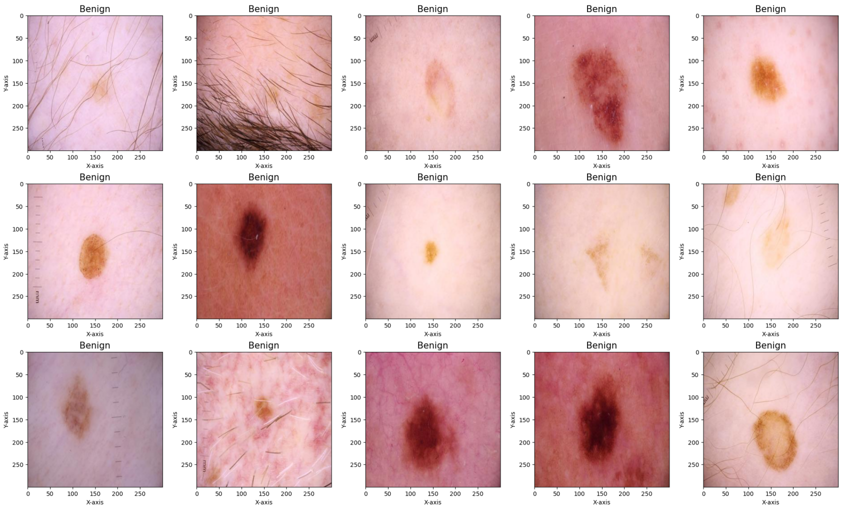


Figure 5: Random sample images (benign Class)

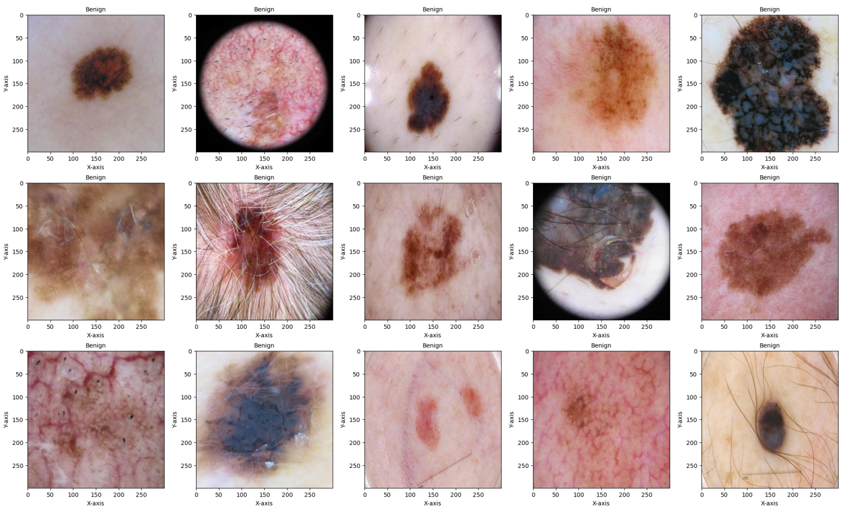


Figure 6: Random sample images (Malignant Class)

Creating Histograms to visualize the width, height and aspect ratio distribution of Images.

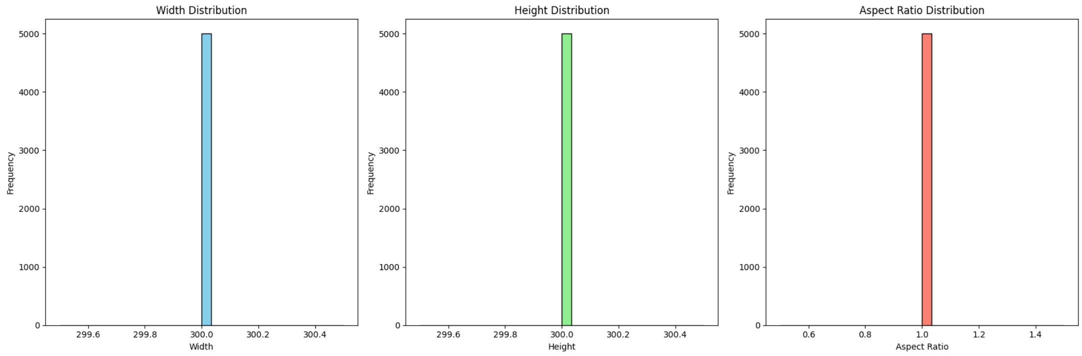


Figure 7: Histogram plots

6 Image Processing

Plotting 5 images and their RGB histograms from the train dataset.

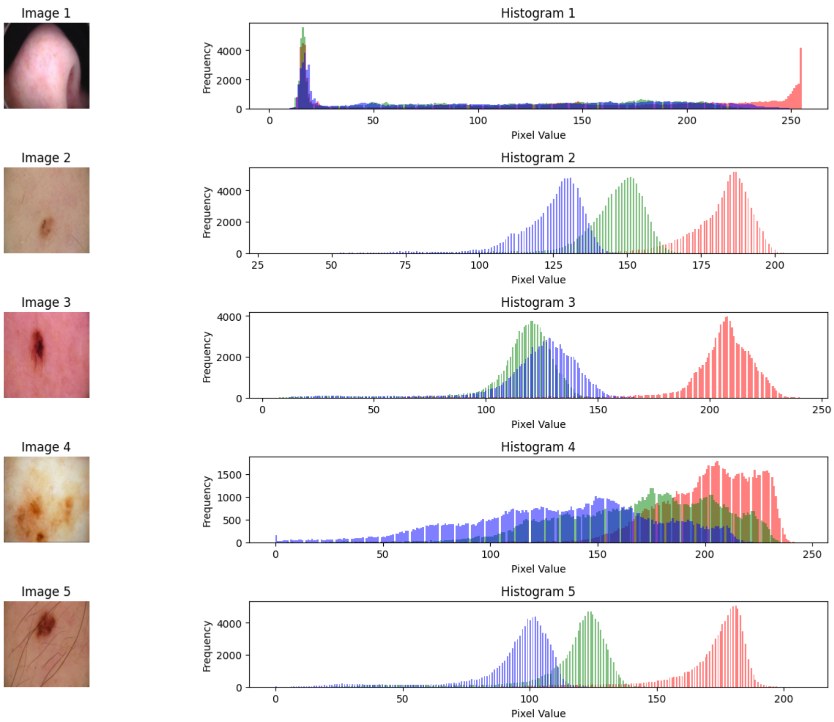


Figure 8: RGB Histogram plots

Applying Image Sharping Method of Image Processing.

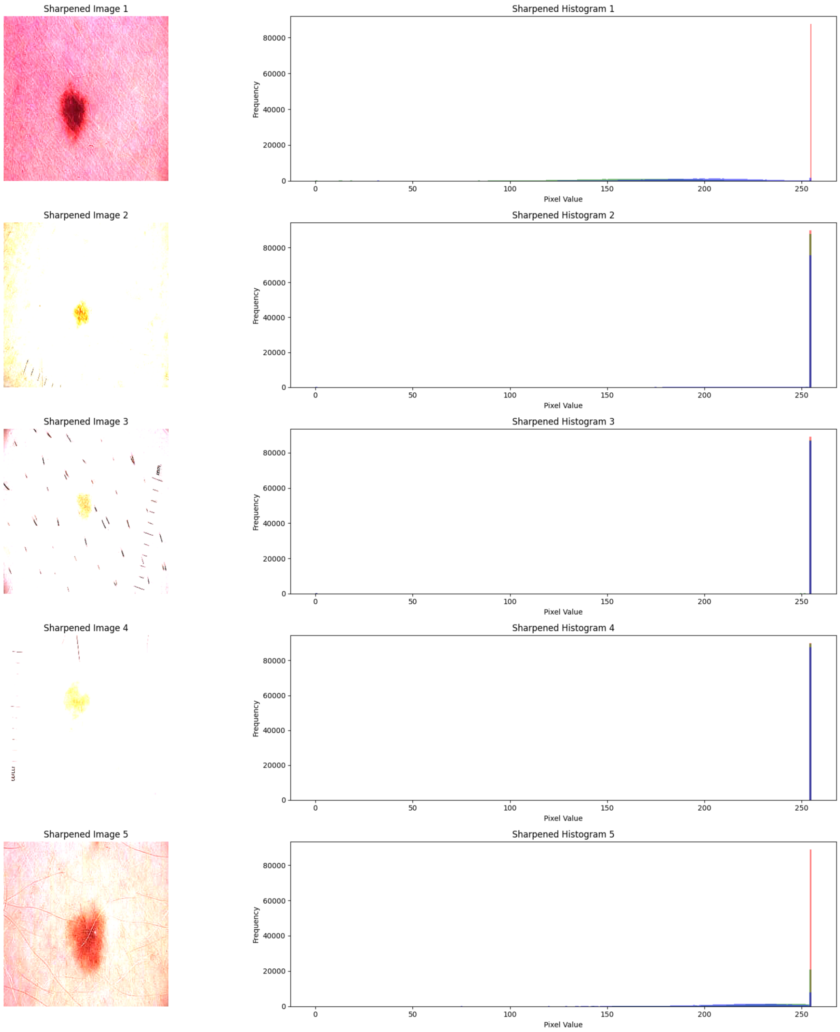


Figure 9: sharpened images plots

7 Model Implementation:

The models used for this research are vgg-19, Mobilenet, DenseNet-121, custom CNN.

VGG19 Model:

```python
tf.keras.backend.clear_session()
def train_vgg19_model(train_generator, test_generator, input_shape=(64, 64, 3),
                      epochs=10, batch_size=32, early_stop_patience=5,
                      lr_reduction_patience=2, lr_reduction_factor=0.1, min_lr=1e-8):

    # Loading VGG19 model without the top classification layer
    vgg19 = VGG19(weights="imagenet", input_shape=input_shape, include_top=False)

    # Freezing all layers in the VGG19 model
    for layer in vgg19.layers:
        layer.trainable = False

    # Building the new model on top of VGG19
    x = Flatten()(vgg19.output)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    prediction = Dense(1, activation='sigmoid')(x)

    # Creating a model object
    model = Model(inputs=vgg19.input, outputs=prediction)

    # Compiling the model
    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(),
                  metrics=["accuracy", Precision(), Recall()])

    # Defining callbacks
    early_stop = EarlyStopping(monitor='loss', patience=early_stop_patience)
    learning_rate_reduction = ReduceLROnPlateau(monitor='loss',
                                                patience=lr_reduction_patience,
                                                factor=lr_reduction_factor,
                                                min_lr=min_lr)

    # Training the model
    history = model.fit(train_generator,
                        epochs=epochs,
                        validation_data=test_generator,
                        callbacks=[early_stop, learning_rate_reduction])
```

Figure 10:

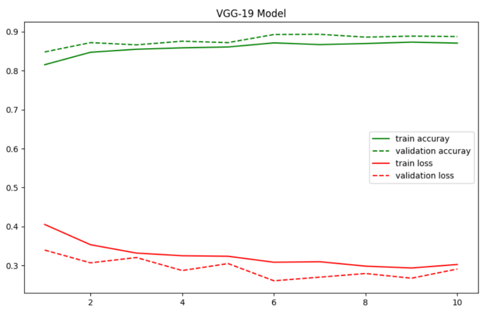


Figure 11: Training and Validation Accuracy and Loss.

Classification Report

	precision	recall	f1-score	support
benign	0.82	0.93	0.87	500
malignant	0.91	0.79	0.85	500
accuracy			0.86	1000
macro avg	0.87	0.86	0.86	1000
weighted avg	0.87	0.86	0.86	1000

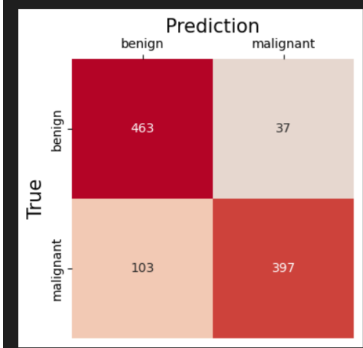


Figure 12: Confusion Matrix.

MobileNet Model:

```python
tf.keras.backend.clear_session()
def train_MobileNet_model(train_generator, test_generator, input_shape=(64, 64, 3),
                          epochs=10, batch_size=32, early_stop_patience=5,
                          lr_reduction_patience=2, lr_reduction_factor=0.1, min_lr=1e-8):

    # Load Mobilenet model without the top classification layer
    mbnet = MobileNet(weights="imagenet", input_shape=input_shape, include_top=False)

    # Freeze all layers in the Mobilenet model
    for layer in mbnet.layers:
        layer.trainable = False

    # Build the new model on top of Mobilenet
    x = Flatten()(mbnet.output)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    prediction = Dense(1, activation='sigmoid')(x)

    # Creating a model object
    model = Model(inputs=mbnet.input, outputs=prediction)

    # Compiling the model
    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(),
                  metrics=["accuracy", Precision(), Recall()])

    # Defining callbacks
    early_stop = EarlyStopping(monitor='loss', patience=early_stop_patience)
    learning_rate_reduction = ReduceLROnPlateau(monitor='loss',
                                                patience=lr_reduction_patience,
                                                factor=lr_reduction_factor,
                                                min_lr=min_lr)

    # Training the model
    history = model.fit(train_generator,
                        epochs=epochs,
                        validation_data=test_generator,
                        callbacks=[early_stop, learning_rate_reduction])
```

Figure 13:

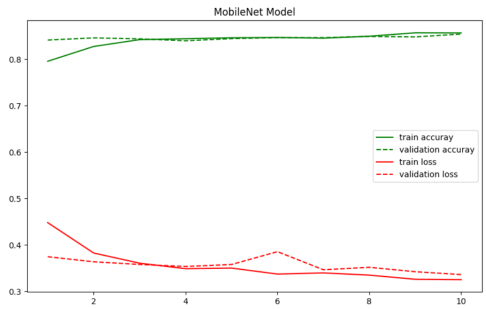


Figure 14: Training and Validation Accuracy and Loss.

```
Classification Report
              precision    recall  f1-score   support

      benign       0.86      0.81      0.83       500
   malignant       0.82      0.87      0.84       500

    accuracy                           0.84      1000
   macro avg       0.84      0.84      0.84      1000
weighted avg       0.84      0.84      0.84      1000

W0000 00:00:1723308251.643960     147 graph_launch.cc:671]
```

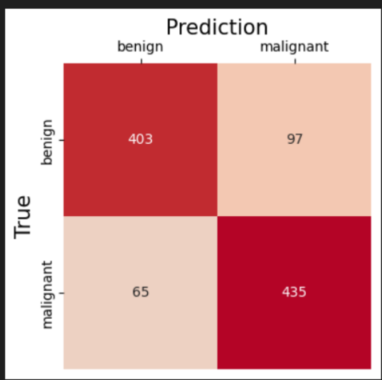


Figure 15: Confusion Matrix.

DenseNet Model:

```
tf.keras.backend.clear_session()
def train_densenet_model(train_generator, test_generator, input_shape=(64, 64, 3),
                         epochs=10, batch_size=32, early_stop_patience=5,
                         lr_reduction_patience=2, lr_reduction_factor=0.1, min_lr=1e-8):

    # Loading DenseNet121 model without the top classification layer
    dn121 = DenseNet121(weights="imagenet", input_shape=input_shape, include_top=False)

    # Freezing all layers in the DenseNet121 model
    for layer in dn121.layers:
        layer.trainable = False

    # Building the new model on top of DenseNet121
    x = Flatten()(dn121.output)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    prediction = Dense(1, activation='sigmoid')(x)

    # Creating a model object
    model = Model(inputs=dn121.input, outputs=prediction)

    # Compiling the model
    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(),
                  metrics=["accuracy", Precision(), Recall()])

    # Defining callbacks
    early_stop = EarlyStopping(monitor='loss', patience=early_stop_patience)
    learning_rate_reduction = ReduceLROnPlateau(monitor='loss',
                                                patience=lr_reduction_patience,
                                                factor=lr_reduction_factor,
                                                min_lr=min_lr)

    # Trainung the model
    history = model.fit(train_generator,
                        epochs=epochs,
                        validation_data=test_generator,
                        callbacks=[early_stop, learning_rate_reduction])
```

Figure 16:

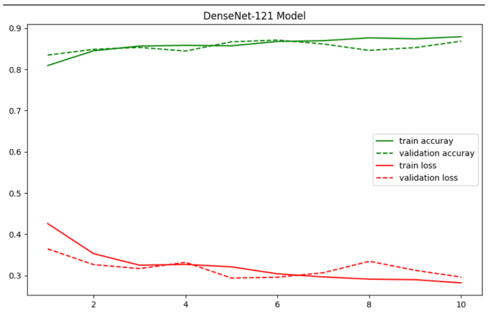


Figure 17: Training and Validation Accuracy and Loss.

Classification Report

	precision	recall	f1-score	support
benign	0.85	0.91	0.88	500
malignant	0.91	0.84	0.87	500
accuracy			0.88	1000
macro avg	0.88	0.88	0.88	1000
weighted avg	0.88	0.88	0.88	1000

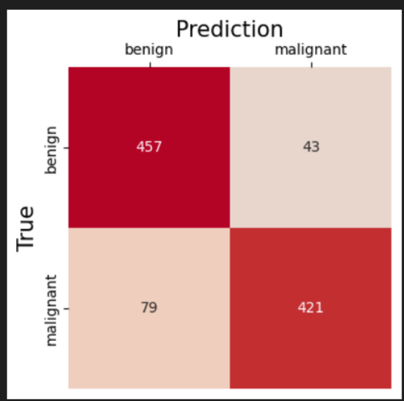


Figure 18: Confusion Matrix.

CNN Model:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.metrics import Precision, Recall
tf.keras.backend.clear_session()

def train_cnn_model(train_generator, test_generator, input_shape=(64, 64, 3),
                    epochs=10, batch_size=32, early_stop_patience=5,
                    lr_reduction_patience=2, lr_reduction_factor=0.1, min_lr=1e-8):

    # Initializing the model
    model = Sequential()

    # CNN Layers
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())


    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))

    # Flatten and Dense Layers
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))  # Assuming 2 classes

    # Compiling the model
    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(),
                  metrics=["accuracy", Precision(), Recall()])

    # Defining callbacks
    early_stop = EarlyStopping(monitor='loss', patience=early_stop_patience)
    learning_rate_reduction = ReduceLROnPlateau(monitor='loss',
                                                patience=lr_reduction_patience,
                                                factor=lr_reduction_factor,
                                                min_lr=min_lr)

    # Training the model
    history = model.fit(train_generator,
                        epochs=epochs,
                        validation_data=test_generator,
                        callbacks=[early_stop, learning_rate_reduction])

    return model, history

model4, history4 = train_cnn_model(training_set, val_set)
```

Figure 19:

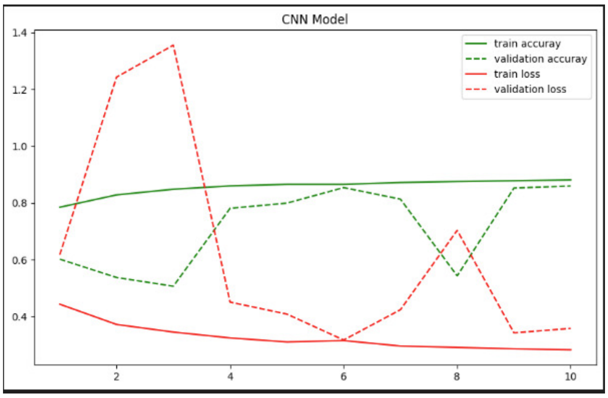
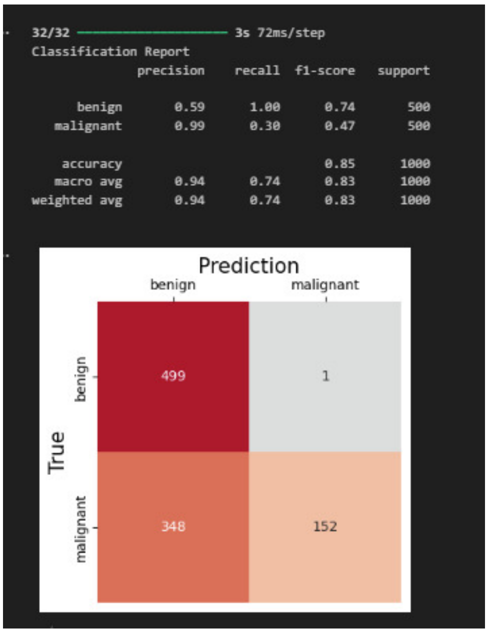


Figure 20: Training and Validation Accuracy and Loss.



```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step
Classification Report
              precision    recall  f1-score   support

      benign       0.59      1.00      0.74       500
   malignant       0.99      0.30      0.47       500

    accuracy                           0.85      1000
   macro avg       0.94      0.74      0.83      1000
weighted avg       0.94      0.74      0.83      1000
```


Figure 21: Confusion Matrix.

8 F1-Score Visualization:

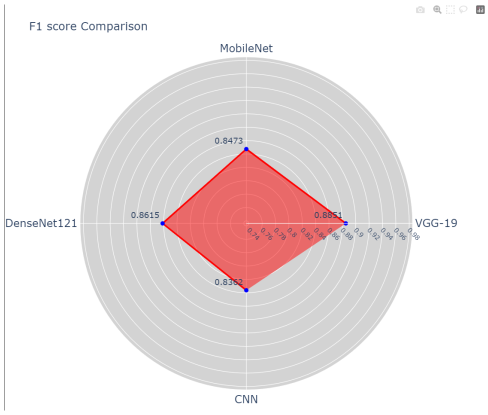


Figure 22: F1 Score comparison between models

Predictions:

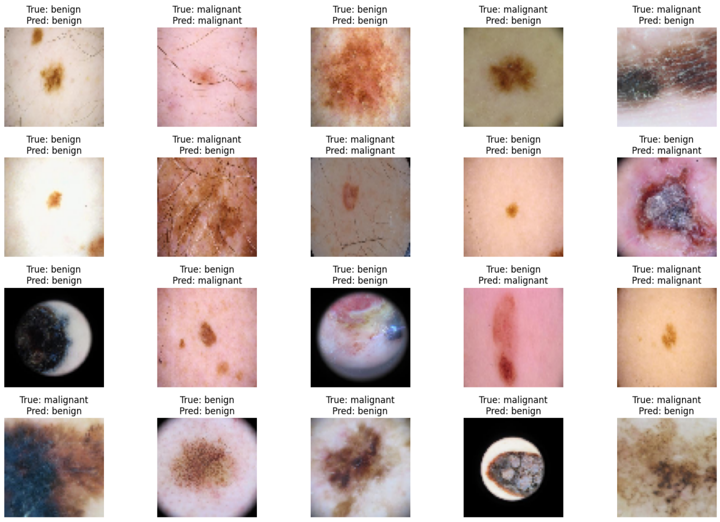


Figure 23: Predicted images

8.1 Interactive Web UI:

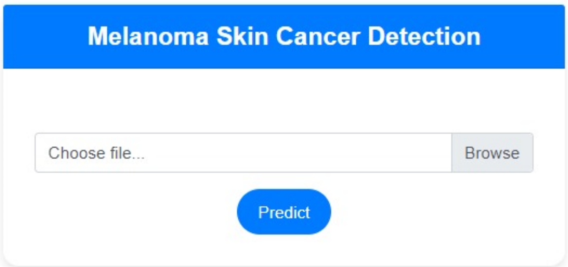


Figure 24: UI

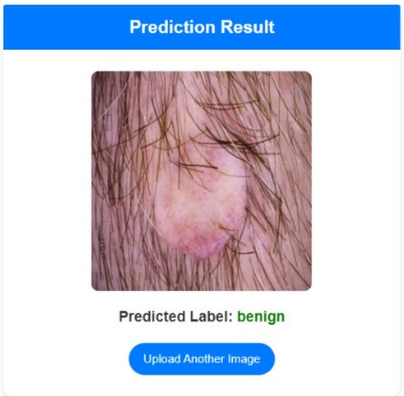


Figure 25: Predicted Image

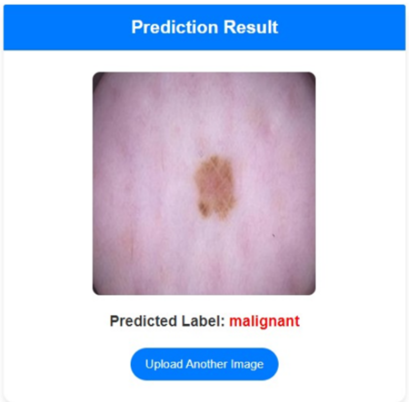


Figure 26: Predicted Image

References