Configuration Manual

MSc Research Project
Data Analytics

Syed Munazir Shajahan
Student ID: x23103914

School of Computing
National College of Ireland

Supervisor: Prof. Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Syed Munazir Shajahan
Student ID:	x23103914
Programme:	MSc in Data Analytics — **Year:** 2023 - 2024
Module:	MSc Research Project
Lecturer:	Prof. Jaswinder Singh
Submission Due Date:	29/01/2025
Project Title:	Skin Cancer Detection: Image Classification Using CNN Architectures with CLAHE
Word Count:	786 — **Page Count:** 13

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: Syed Munazir Shajahan

Date: 29/01/2025

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

Syed Munazir Shajahan
Student ID: x23103914

1 Introduction

This configuration manual provides step-by-step instructions for replicating the experimental setup and results of the research project "Skin Cancer Detection: Image Classification Using CNN Architectures with CLAHE". The study evaluates the performance of the CNN architectures ResNet50, DenseNet121, and EfficientNetB0 on raw and CLAHE-enhanced datasets. Following this manual, one can reproduce the specific environment, datasets, methodologies, and analyses to reproduce results and contribute to a further understanding of the automated skin cancer detection process.

2 Development Environment

2.1 Hardware Specifications

- **Processor:** Intel Quad-Core i7- 2.3 GHz
- **RAM:** 16 GB
- **GPU:** Intel Irish Plus 1.5GB

2.2 Software Specifications

- **Operating System:** iOS
- **Programming Language:** Python
- **Integrated Development Environment (IDE):** Jupyter Notebook

2.3 Python Libraries Required

The following libraries and their versions were utilized. They can be installed using pip install:

- NumPy

- Pandas
- TensorFlow
- Keras
- OpenCV
- Matplotlib
- Seaborn
- scikit-learn

Importing Libraries

```
import pandas as pd
import numpy as np
import os
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.applications import ResNet50, DenseNet121, EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 1: Libraries Used

3 Data Source

One dataset is used in this project below is the dataset link and it’s took from the Kaggle.

Skin Cancer MNIST: HAM10000 -
https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000

```
<bound method NDFrame.head of          lesion_id      image_id     dx dx_type   age     sex localization
0      HAM_0000118  ISIC_0027419    bkl   histo  80.0    male        scalp
1      HAM_0000118  ISIC_0025030    bkl   histo  80.0    male        scalp
2      HAM_0002730  ISIC_0026769    bkl   histo  80.0    male        scalp
3      HAM_0002730  ISIC_0025661    bkl   histo  80.0    male        scalp
4      HAM_0001466  ISIC_0031633    bkl   histo  75.0    male          ear
...            ...           ...    ...     ...   ...     ...          ...
10010  HAM_0002867  ISIC_0033084  akiec   histo  40.0    male      abdomen
10011  HAM_0002867  ISIC_0033550  akiec   histo  40.0    male      abdomen
10012  HAM_0002867  ISIC_0033536  akiec   histo  40.0    male      abdomen
10013  HAM_0000239  ISIC_0032854  akiec   histo  80.0    male         face
10014  HAM_0003521  ISIC_0032258    mel   histo  70.0  female         back

[10015 rows x 7 columns]>
```

Figure 2: Dataset Head

4 Project Code Files

Two Jupyter Notebook Code Files are used in this project, and it's named as Raw_x23103914 and CLAHE_x23103914.

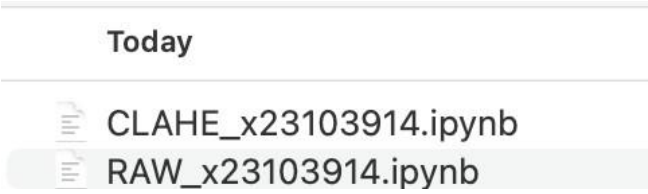

Figure 3: Code File

5 Data Preparation

5.1 Data Directory Set-up

Downloaded dataset from Kaggle has been extracted and loaded on the Jupyter notebook.

SETUP DIRECTORIES AND LOAD METADATA

```
metadata_path = "/Users/apple/Documents/PROJECT RESEARCH/Dataset/HAMM 2/HAM10000_metadata.csv"
img_dir_part1 = "/Users/apple/Documents/PROJECT RESEARCH/Dataset/HAMM 2/HAM10000_images_part_1"
img_dir_part2 = "/Users/apple/Documents/PROJECT RESEARCH/Dataset/HAMM 2/HAM10000_images_part_2"

# Load metadata
metadata = pd.read_csv(metadata_path)
metadata['image_id'] = metadata['image_id'].astype(str)

# Encode labels
label_encoder = LabelEncoder()
metadata['label'] = label_encoder.fit_transform(metadata['dx'])
```

Figure 4: Directory Set-up

5.2 Data Cleaning

Data Cleaning is implemented to remove all the unnecessary data from the dataset and to provide the implementation progress in a right way.

```
Initial Missing Values:
lesion_id        0
image_id         0
dx               0
dx_type          0
age              0
sex              0
localization     0
dtype: int64

Number of duplicate rows: 0
Removed 0 duplicate rows.
Dropped 0 rows with missing critical values.

Standardized 'sex' column values.
Removed rows with invalid age values (<= 0).
Added 'is_malignant' column.
Removed 0 rows with missing or invalid image paths.
Normalized column names.
Dropped irrelevant columns.

Final Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9919 entries, 0 to 10014
Data columns (total 7 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   image_id       9919 non-null    object
 1   dx             9919 non-null    object
 2   age            9919 non-null    float64
 3   sex            9919 non-null    object
 4   localization   9919 non-null    object
 5   is_malignant   9919 non-null    int64
 6   image_path     9919 non-null    object
dtypes: float64(1), int64(1), object(5)
```

Figure 5: Data Cleaning

5.3 Exploratory Data Analysis

EDA is implemented for demonstrating more about the dataset in multiple aspects by fetching information on feature basis. In this project, metadata file has been used to fetch the data from Image directories, by using image_id it fetches images from the directories.

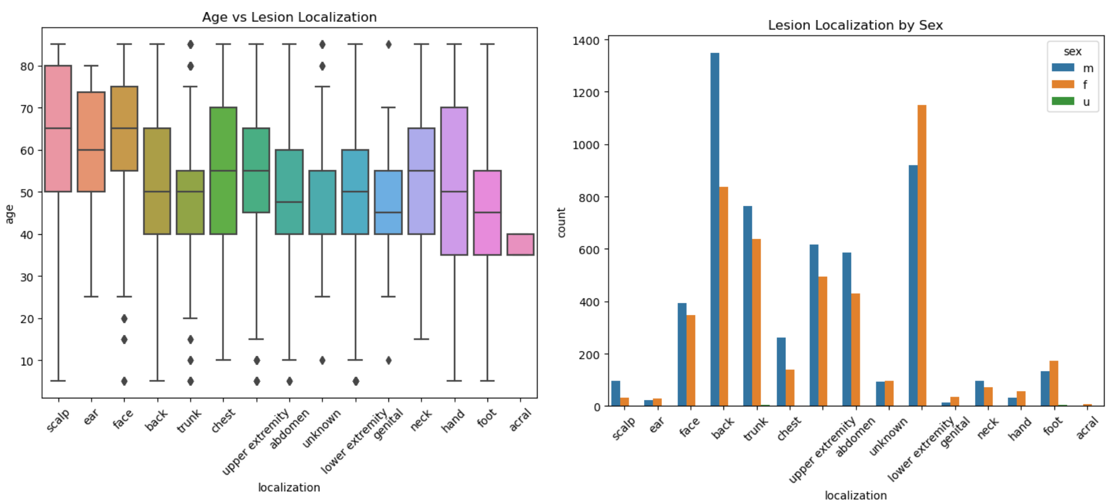

Figure 6: EDA

5.4 CLAHE Technique

CLAHE is an image enhancement technique, the ideology behind for incorporating with CNN architectures – ResNet, DenseNet and EfficientNet is to acquire the better model evaluations.

CLAHE - Image Enhancment

```python
def apply_clahe(img):
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    img = clahe.apply(img)
    return cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)

def preprocess_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (128, 128))
    img = apply_clahe(img)
    img = img / 255.0
    return img
```

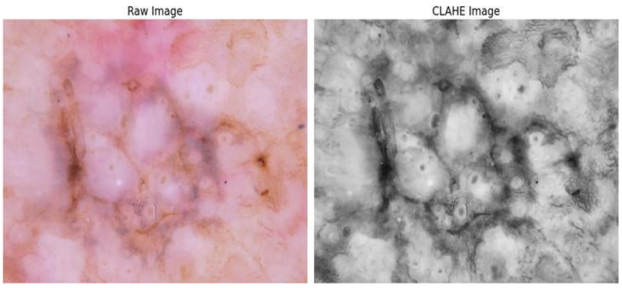

Figure 7: CLAHE

5.5 Data Generator

Data Generator is implemented in this project for loading & model building for the large/huge datasets. This project uses 10,000+ images (Medical Digital Images) for detect the skin cancer.

```python
# ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    x_col='image_path',
    y_col='class',
    target_size=target_size,
    batch_size=batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_dataframe(
    test_df,
    x_col='image_path',
    y_col='class',
    target_size=target_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 8012 validated image filenames belonging to 7 classes.
Found 2003 validated image filenames belonging to 7 classes.
```

Figure 8: Data Generator

6 Implementation Approach

In this research project, the goal is to compare two approaches – with CLAHE and without CLAHE using CNN architecture (ResNet, DenseNet and EfficientNet). The comparison is implemented after getting model evaluations, before that project go through with data preparation, model training and building. Below is the flowchart about the approaches,

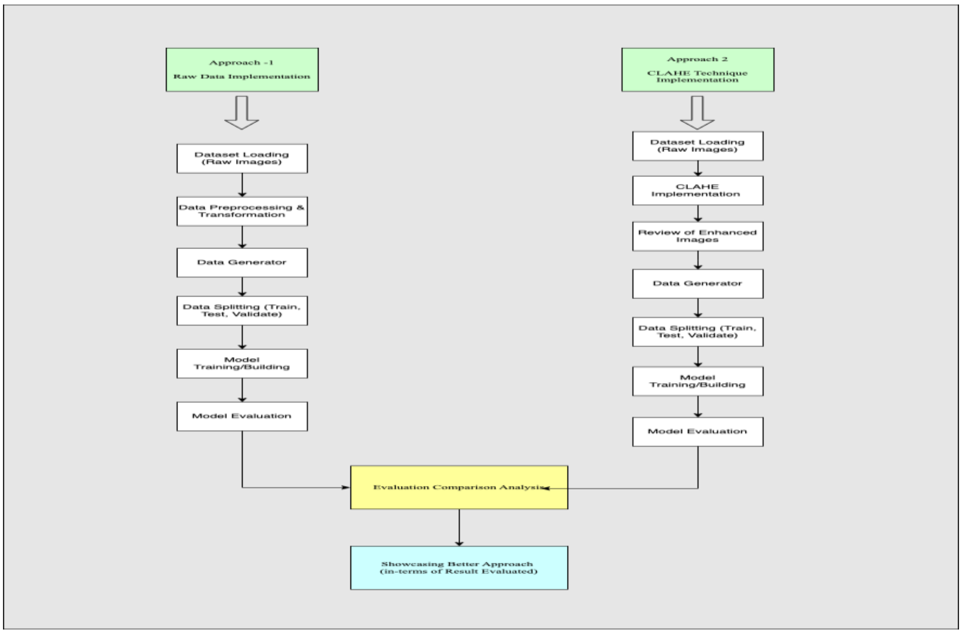

Figure 9: Implementation Approach

6.1 Approach 1: Raw Images Method

- Data Splitting: Dataset is Splitted into Train (80%) and Test (20%).

```
# Train-Test Split
train_df, test_df = train_test_split(metadata, test_size=0.2, stratify=metadata['class'], random_state=42)

# Convert class column to strings for ImageDataGenerator
train_df['class'] = train_df['class'].astype(str)
test_df['class'] = test_df['class'].astype(str)
```

Figure 10: Data Splitting

- CNN Model Building: Model training and building has done for this approach as below,

```
# Model Building
def build_model(base_model_class):
    base_model = base_model_class(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    base_model.trainable = False
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(len(label_mapping), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Train Models
models = {
    'ResNet50': ResNet50,
    'DenseNet121': DenseNet121,
    'EfficientNetB0': EfficientNetB0
}

results = {}
```

Figure 11: CNN Building

- CNN Model Evaluation: Model Evaluation has done for this approach as below,

```
for model_name, base_model_class in models.items():
    print(f"Training {model_name}...")
    model = build_model(base_model_class)
    history = model.fit(train_generator, validation_data=test_generator, epochs=5)

    # Evaluate model
    y_true = test_generator.classes
    y_pred = np.argmax(model.predict(test_generator), axis=1)

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    correct = np.trace(cm)
    misclassified = np.sum(cm) - correct
    accuracy = correct / np.sum(cm)
    precision = np.sum(cm.diagonal()) / np.sum(cm)
    recall = correct / (correct + misclassified)
    f1 = 2 * (precision * recall) / (precision + recall)

    print(f"{model_name} Confusion Matrix Summary:")
    print(f"Correctly Classified Samples: {correct}")
    print(f"Misclassified Samples: {misclassified}")
    print(f"{model_name} Metrics:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 Score: {f1:.2f}")

    results[model_name] = history
```

Figure 12: Model Evaluation

- Evaluation plots: Model's Evaluation has been illustrated in graphs reprenstation, below is the snippet for code, epochs and graph.

```
# Ploting Training Results
for model_name, history in results.items():
    plt.plot(history.history['accuracy'], label=f'{model_name} Train Accuracy')
    plt.plot(history.history['val_accuracy'], label=f'{model_name} Validation Accuracy')
plt.title('Model Training Performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figure 13: Code for Plotting

```
Training ResNet50...
Epoch 1/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 486s 2s/step - accuracy: 0.6383 - loss: 1.2753 - val_accuracy: 0.6695 - val_loss: 1.13
70
Epoch 2/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 468s 2s/step - accuracy: 0.6741 - loss: 1.1547 - val_accuracy: 0.6695 - val_loss: 1.12
25
Epoch 3/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 456s 2s/step - accuracy: 0.6692 - loss: 1.1569 - val_accuracy: 0.6695 - val_loss: 1.11
59
Epoch 4/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 466s 2s/step - accuracy: 0.6705 - loss: 1.1379 - val_accuracy: 0.6695 - val_loss: 1.12
50
Epoch 5/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 486s 2s/step - accuracy: 0.6669 - loss: 1.1505 - val_accuracy: 0.6695 - val_loss: 1.11
17
63/63 ━━━━━━━━━━━━━━━━━━━━ 102s 2s/step
ResNet50 Confusion Matrix Summary:
Correctly Classified Samples: 1341
Misclassified Samples: 662
ResNet50 Metrics:
Accuracy: 0.67
Precision: 0.67
Recall: 0.67
F1 Score: 0.67
```

Figure 14: Evaluation of ResNet

```
Training DenseNet121...
Epoch 1/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 678s 3s/step - accuracy: 0.6492 - loss: 1.0879 - val_accuracy: 0.7074 - val_loss: 0.78
23
Epoch 2/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 690s 3s/step - accuracy: 0.6989 - loss: 0.8214 - val_accuracy: 0.7294 - val_loss: 0.74
57
Epoch 3/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 731s 3s/step - accuracy: 0.7202 - loss: 0.7611 - val_accuracy: 0.7539 - val_loss: 0.69
05
Epoch 4/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 787s 3s/step - accuracy: 0.7384 - loss: 0.7069 - val_accuracy: 0.7549 - val_loss: 0.68
94
Epoch 5/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 733s 3s/step - accuracy: 0.7409 - loss: 0.7163 - val_accuracy: 0.7564 - val_loss: 0.68
16
63/63 ━━━━━━━━━━━━━━━━━━━━ 157s 2s/step
DenseNet121 Confusion Matrix Summary:
Correctly Classified Samples: 1515
Misclassified Samples: 488
DenseNet121 Metrics:
Accuracy: 0.76
Precision: 0.76
Recall: 0.76
F1 Score: 0.76
```

Figure 15: Evaluation of DenseNet

```
Training EfficientNetB0...
Epoch 1/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 352s 1s/step - accuracy: 0.6544 - loss: 1.2227 - val_accuracy: 0.6695 - val_loss: 1.13
62
Epoch 2/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 334s 1s/step - accuracy: 0.6754 - loss: 1.1553 - val_accuracy: 0.6695 - val_loss: 1.13
21
Epoch 3/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 310s 1s/step - accuracy: 0.6669 - loss: 1.1611 - val_accuracy: 0.6695 - val_loss: 1.13
66
Epoch 4/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 313s 1s/step - accuracy: 0.6747 - loss: 1.1605 - val_accuracy: 0.6695 - val_loss: 1.13
20
Epoch 5/5
251/251 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - accuracy: 0.6692 - loss: 1.1569 - val_accuracy: 0.6695 - val_loss: 1.13
29
63/63 ━━━━━━━━━━━━━━━━━━━━ 63s 971ms/step
EfficientNetB0 Confusion Matrix Summary:
Correctly Classified Samples: 1341
Misclassified Samples: 662
EfficientNetB0 Metrics:
Accuracy: 0.67
Precision: 0.67
Recall: 0.67
F1 Score: 0.67
```

Figure 16: Evaluation of EfficientNet

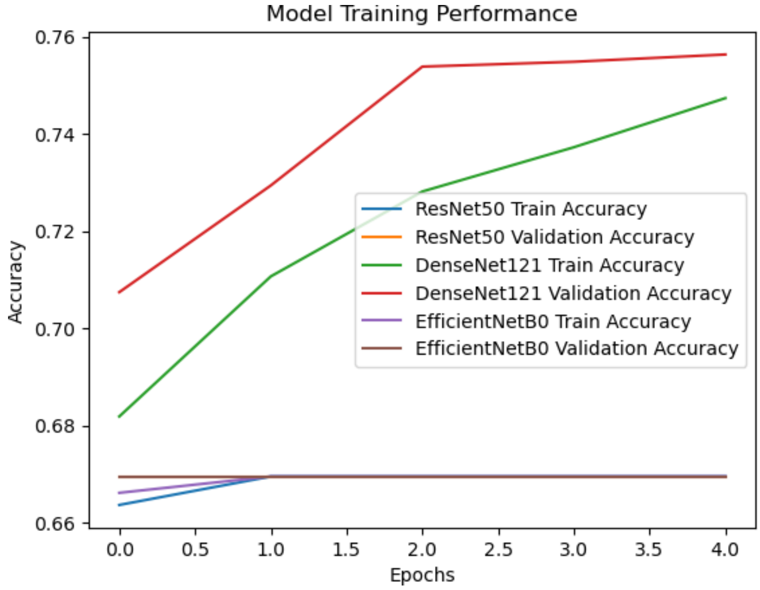

Figure 17: Training Performance Plot of Models

6.2 Approach 2: CLAHE Images Method

- Data Splitting and Loading Paths: Dataset is splitted into Train, Test and Validate as below ratio (80%)

```
: train_metadata, test_metadata = train_test_split(metadata, test_size=0.2, stratify=metadata['label'], random_state=4
  train_metadata, val_metadata = train_test_split(train_metadata, test_size=0.125, stratify=train_metadata['label'], r

  def load_image_paths(metadata, img_dir1, img_dir2):
      paths = []
      labels = []
      for idx, row in metadata.iterrows():
          img_name = row['image_id'] + '.jpg'
          label = row['label']
          img_path = os.path.join(img_dir1 if os.path.exists(os.path.join(img_dir1, img_name)) else img_dir2, img_name
          paths.append(img_path)
          labels.append(label)
      return np.array(paths), np.array(labels)

  # Load training, validation, and test data paths
  train_paths, train_labels = load_image_paths(train_metadata, img_dir_part1, img_dir_part2)
  val_paths, val_labels = load_image_paths(val_metadata, img_dir_part1, img_dir_part2)
  test_paths, test_labels = load_image_paths(test_metadata, img_dir_part1, img_dir_part2)
```

Figure 18: CLAHE Model Training & Image Path Mapping

- CLAHE: Contrast Limited Adaptive Histogram Equalization (CLAHE) an image enhancement technique, which helps to make the images quality better.

```python
def apply_clahe(img):
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    img = clahe.apply(img)
    return cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)

def preprocess_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (128, 128))
    img = apply_clahe(img)
    img = img / 255.0
    return img
```

\

Figure 19: Applying CLAHE

- Model Building: Model training and building has done for this approach as below,

```python
def build_model(base_model):
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.5),  # Dropout for regularization
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(len(np.unique(train_labels)), activation='softmax')
    ])
    model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

base_models = {
    'ResNet50': ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3)),
    'DenseNet121': DenseNet121(weights='imagenet', include_top=False, input_shape=(128, 128, 3)),
    'EfficientNetB0': EfficientNetB0(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
}

callbacks = [
    EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True),
    ModelCheckpoint('best_model.keras', monitor='val_accuracy', save_best_only=True)
]
```

Figure 20: Model Building

- Visualising Function: This set of code is written to display the sample images from the directory (skin lesion – image directory)

```python
# Showcase that data is loaded correctly
def showcase_data_loading():
    print(f"Training dataset size: {len(train_paths)} images")
    print(f"Validation dataset size: {len(val_paths)} images")
    print(f"Test dataset size: {len(test_paths)} images")

    # Display sample images from the dataset
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    axes[0].imshow(cv2.cvtColor(cv2.imread(train_paths[0]), cv2.COLOR_BGR2RGB))
    axes[0].set_title(f"Raw Image: {train_metadata.iloc[0]['image_id']}")
    axes[1].imshow(cv2.cvtColor(apply_clahe(cv2.imread(train_paths[0])), cv2.COLOR_BGR2RGB))
    axes[1].set_title(f"CLAHE Image: {train_metadata.iloc[0]['image_id']}")
    axes[2].imshow(cv2.cvtColor(cv2.imread(train_paths[1]), cv2.COLOR_BGR2RGB))
    axes[2].set_title(f"Raw Image: {train_metadata.iloc[1]['image_id']}")
    plt.show()
```

Figure 21: Visualising Function

- Preview Image Comparison: This set of code is written to display the comparison images of Raw and CLAHE

```
def preview_comparison():
    img_idx = 0  # You can change this to show different images
    raw_img = cv2.imread(train_paths[img_idx])
    clahe_img = apply_clahe(raw_img)

    fig, axes = plt.subplots(1, 2, figsize=(14, 7))
    axes[0].imshow(cv2.cvtColor(raw_img, cv2.COLOR_BGR2RGB))
    axes[0].set_title("Raw Image", fontsize=16)
    axes[0].axis('off')
    axes[1].imshow(cv2.cvtColor(clahe_img, cv2.COLOR_BGR2RGB))
    axes[1].set_title("CLAHE Image", fontsize=16)
    axes[1].axis('off')
    plt.tight_layout()
    plt.show()
```

Figure 22: CLAHE vs RAW Images

- Model Training and Evaluation: The below code is written to train and evaluate for CLAHE approach.

```
def train_and_evaluate_model(model_name, base_model, epochs=30):
    print(f"\nTraining {model_name} model for {epochs} epochs...")
    model = build_model(base_model)

    # Train the model
    history = model.fit(
        data_generator(train_paths, train_labels, batch_size=32),
        steps_per_epoch=len(train_paths) // 32,
        validation_data=data_generator(val_paths, val_labels, batch_size=32),
        validation_steps=len(val_paths) // 32,
        epochs=epochs,
        callbacks=callbacks
    )

    # Preprocess test data
    test_images = np.array([preprocess_image(path) for path in test_paths])
    test_labels_cat = tf.keras.utils.to_categorical(test_labels, num_classes=len(np.unique(train_labels)))

    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(test_images, test_labels_cat, verbose=0)
    print(f"{model_name} Test Accuracy: {test_accuracy * 100:.2f}%")

    # Generate predictions
    predictions = model.predict(test_images)
    predicted_labels = np.argmax(predictions, axis=1)

    # Compute confusion matrix
    cm = confusion_matrix(test_labels, predicted_labels)

    # Display confusion matrix with enhanced visualization
    plot_confusion_matrix(cm, model_name)

    # Print confusion matrix summary
    correct_predictions = np.trace(cm)  # Sum of diagonal elements
    total_samples = np.sum(cm)
    misclassified_samples = total_samples - correct_predictions

    print(f"{model_name} Confusion Matrix Summary:")
    print(f"Correctly Classified Samples: {correct_predictions}")
    print(f"Misclassified Samples: {misclassified_samples}")

    # Compute precision, recall, and F1 score
    precision = precision_score(test_labels, predicted_labels, average='weighted')
    recall = recall_score(test_labels, predicted_labels, average='weighted')
    f1 = f1_score(test_labels, predicted_labels, average='weighted')
```

Figure 23: CLAHE

- Main Execution Code: The below code is written for execute the all the set of code of this approach.

```
# Showcase that all the data is loaded
showcase_data_loading()

# Preview CLAHE vs Raw Images for the first few images
preview_comparison()

# Train and evaluate models with increased epochs
train_and_evaluate_model('ResNet50', base_models['ResNet50'], epochs=30)
train_and_evaluate_model('DenseNet121', base_models['DenseNet121'], epochs=30)
train_and_evaluate_model('EfficientNetB0', base_models['EfficientNetB0'], epochs=30)
```

```
Training dataset size: 7010 images
Validation dataset size: 1002 images
Test dataset size: 2003 images
```

Figure 24: Mian Execution Code

```
04
Epoch 9/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 617s 3s/step - accuracy: 0.9515 - loss: 0.1623 - val_accuracy: 0.7598 - val_loss: 1.01
29
Epoch 10/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 656s 3s/step - accuracy: 0.9628 - loss: 0.1219 - val_accuracy: 0.7320 - val_loss: 1.65
34
Epoch 11/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 657s 3s/step - accuracy: 0.9774 - loss: 0.0800 - val_accuracy: 0.7639 - val_loss: 1.68
47
Epoch 12/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 656s 3s/step - accuracy: 0.9687 - loss: 0.1019 - val_accuracy: 0.7567 - val_loss: 1.22
75
Epoch 13/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 657s 3s/step - accuracy: 0.9771 - loss: 0.0719 - val_accuracy: 0.7402 - val_loss: 1.37
74
Epoch 14/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 657s 3s/step - accuracy: 0.9676 - loss: 0.0991 - val_accuracy: 0.7515 - val_loss: 1.21
68
Epoch 15/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 609s 3s/step - accuracy: 0.9767 - loss: 0.0698 - val_accuracy: 0.7680 - val_loss: 1.32
61
Epoch 16/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 598s 3s/step - accuracy: 0.9708 - loss: 0.1111 - val_accuracy: 0.7278 - val_loss: 1.24
33
Epoch 17/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 601s 3s/step - accuracy: 0.9828 - loss: 0.0655 - val_accuracy: 0.7371 - val_loss: 1.48
62
Epoch 18/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 591s 3s/step - accuracy: 0.9831 - loss: 0.0493 - val_accuracy: 0.7113 - val_loss: 1.45
63
Epoch 19/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 622s 3s/step - accuracy: 0.9838 - loss: 0.0536 - val_accuracy: 0.7588 - val_loss: 1.36
19
Epoch 20/30
219/219 ━━━━━━━━━━━━━━━━━━━━ 588s 3s/step - accuracy: 0.9677 - loss: 0.1093 - val_accuracy: 0.7196 - val_loss: 1.48
44
```

Figure 25: Epoch

- Model Evaluation: Confusion Matrix is plotted for all the three models to establish the results.

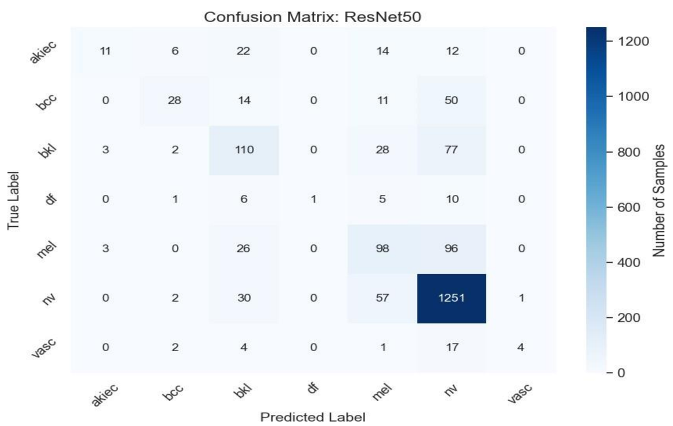

Figure 26: ResNet Confusion Matrix

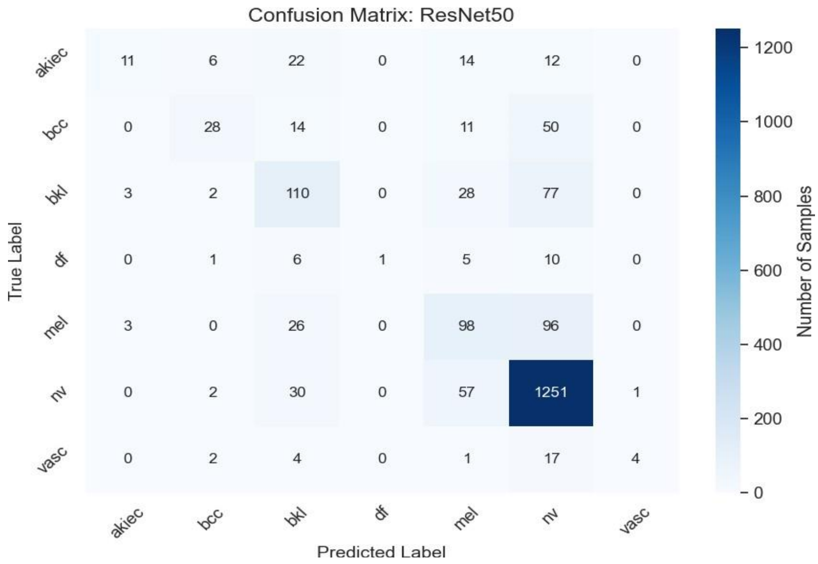

Figure 27: DenseNet Confusion Matrix

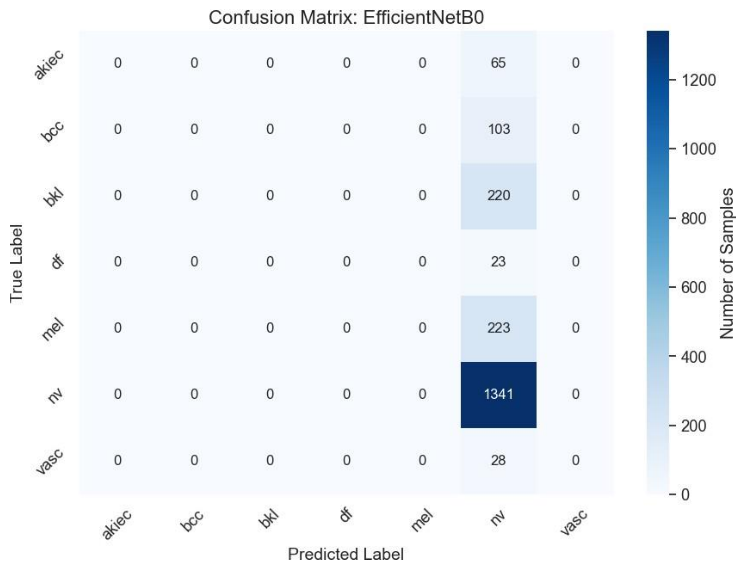

Figure 28: EfficientNet Confusion Matrix

- Result Summary: The below snippet is displayed is for accuracy, precision, recall and F1 score, correctly classified images and misclassified images.

```
DenseNet121 Confusion Matrix Summary:
Correctly Classified Samples: 1345
Misclassified Samples: 658
DenseNet121 Metrics:
Accuracy: 0.67
Precision: 0.53
Recall: 0.67
F1 Score: 0.55
```

```
ResNet50 Confusion Matrix Summary:
Correctly Classified Samples: 1503
Misclassified Samples: 500
ResNet50 Metrics:
Accuracy: 0.75
Precision: 0.74
Recall: 0.75
F1 Score: 0.73
```

```
EfficientNetB0 Confusion Matrix Summary:
Correctly Classified Samples: 1341
Misclassified Samples: 662
EfficientNetB0 Metrics:
Accuracy: 0.67
Precision: 0.45
Recall: 0.67
F1 Score: 0.54
```

Figure 29: Result Summary