Skin Cancer Detection

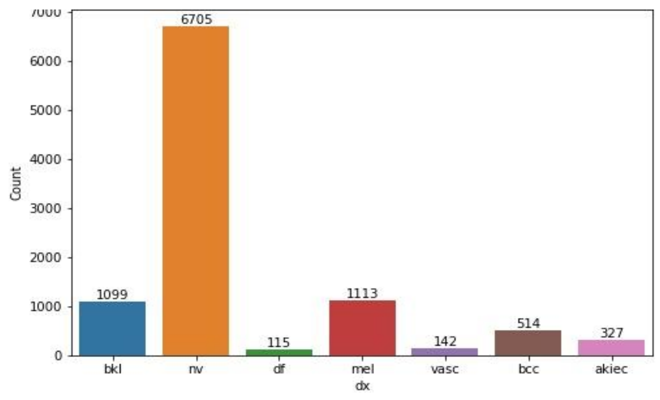
By: Ablokit Joshi
IIT Kanpur
joshiablokit2001@gmail.com
ablokitj20@iitk.ac.in
+919315800879

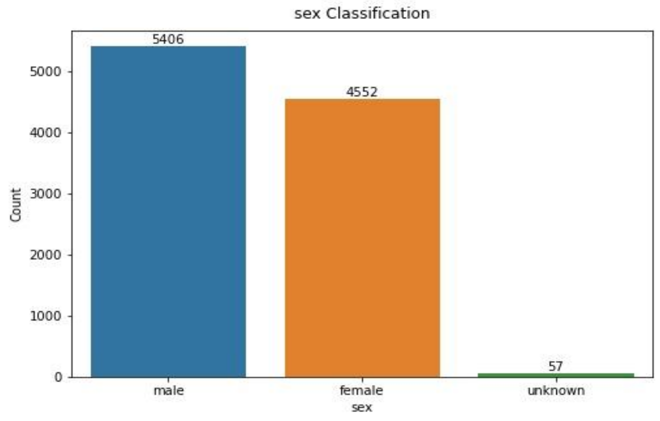
The approach used to develop the skin cancer detection model involved several steps:

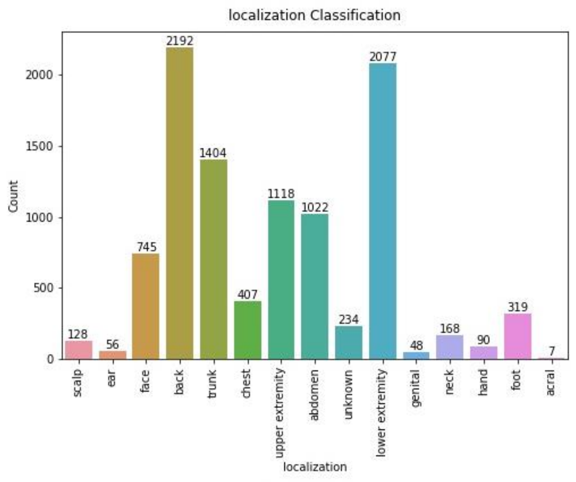
1) Data Analysis: The file had 10015 images spanning over two folders (HAM10000_images_part_1 and HAM10000_images_part_2) and five CSV files, out of which one was the HAM10000_metadata containing all the necessary information like Image ID, age, sex, localization, and label of cancer. The other four were the pixel coordinates for sizes 28x28 and 8x8, in both colored and greyscale.

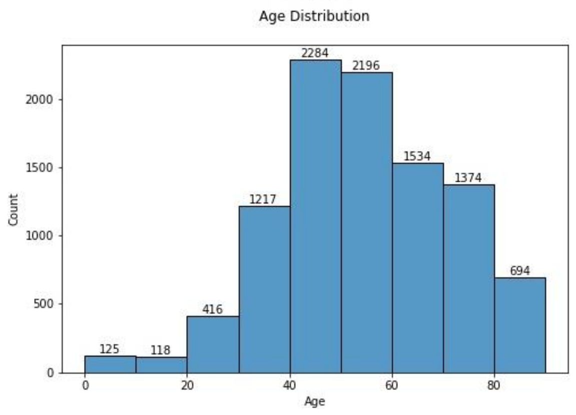
   To save memory and time, I will be using only the hmnist_28_28_RGB.csv file and the metadata to train the model.

   The initial step was to analyze the data and gain insights into the distribution of different attributes.
   Following are the plots for different attributes:





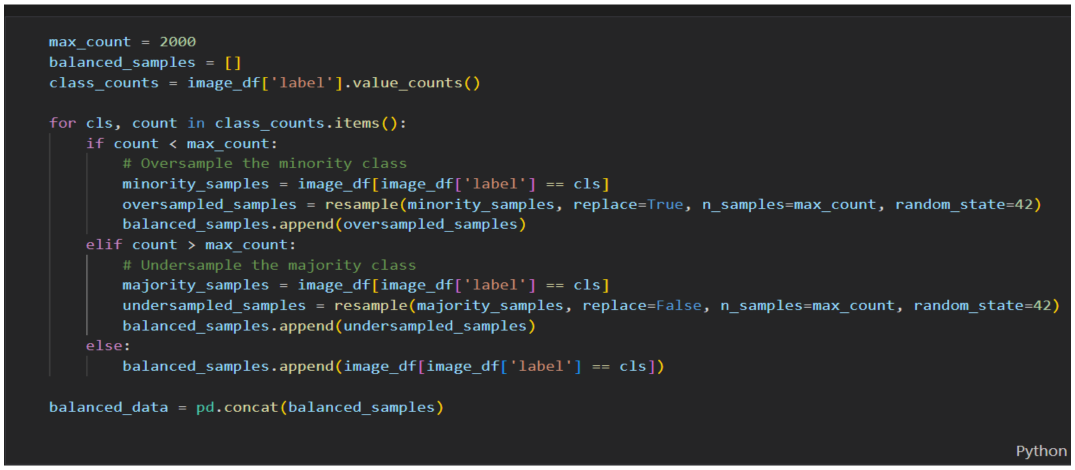




As it is clearly visible that there is a considerable imbalance in the labels section, and thus, there is a need for Data Preprocessing.

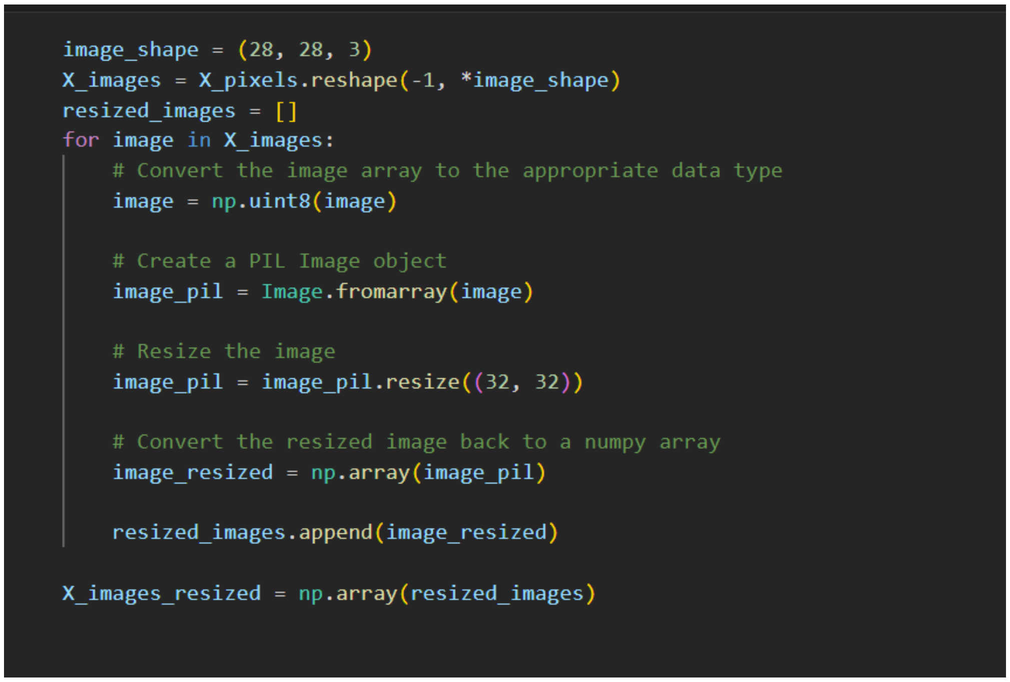
2) Data Preprocessing: There were several preprocessing techniques used in the model:

- **Trimming and Balancing**: The dataset initially had imbalanced classes, with some classes having significantly fewer samples than others. To address this issue, the dataset was balanced by oversampling the minority classes and undersampling the majority classes. This ensured that each class had an equal representation of 2000 samples, which helps prevent bias in the model's training.



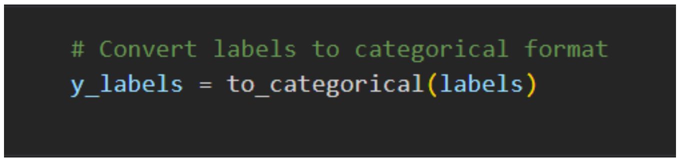
```
max_count = 2000
balanced_samples = []
class_counts = image_df['label'].value_counts()

for cls, count in class_counts.items():
    if count < max_count:
        # Oversample the minority class
        minority_samples = image_df[image_df['label'] == cls]
        oversampled_samples = resample(minority_samples, replace=True, n_samples=max_count, random_state=42)
        balanced_samples.append(oversampled_samples)
    elif count > max_count:
        # Undersample the majority class
        majority_samples = image_df[image_df['label'] == cls]
        undersampled_samples = resample(majority_samples, replace=False, n_samples=max_count, random_state=42)
        balanced_samples.append(undersampled_samples)
    else:
        balanced_samples.append(image_df[image_df['label'] == cls])

balanced_data = pd.concat(balanced_samples)
```

- **Resizing the images**: The original images in the dataset had a shape of (28, 28, 3). The images were resized to (32, 32, 3) to achieve a standardized input size for the model. Resizing the images to a larger size can also help retain more detailed information, which can benefit model performance.



```
image_shape = (28, 28, 3)
X_images = X_pixels.reshape(-1, *image_shape)
resized_images = []
for image in X_images:
    # Convert the image array to the appropriate data type
    image = np.uint8(image)

    # Create a PIL Image object
    image_pil = Image.fromarray(image)

    # Resize the image
    image_pil = image_pil.resize((32, 32))

    # Convert the resized image back to a numpy array
    image_resized = np.array(image_pil)

    resized_images.append(image_resized)

X_images_resized = np.array(resized_images)
```

- **Converting Labels to Categorical Format:** The labels in the dataset were in a numerical format representing different classes of skin cancer. To prepare the labels for multiclass classification, they were converted into a categorical format using one-hot encoding. This categorical format is suitable for training a model with multiple output classes. To do this, we imported to_categorical module from the tensorflow.keras.utils library

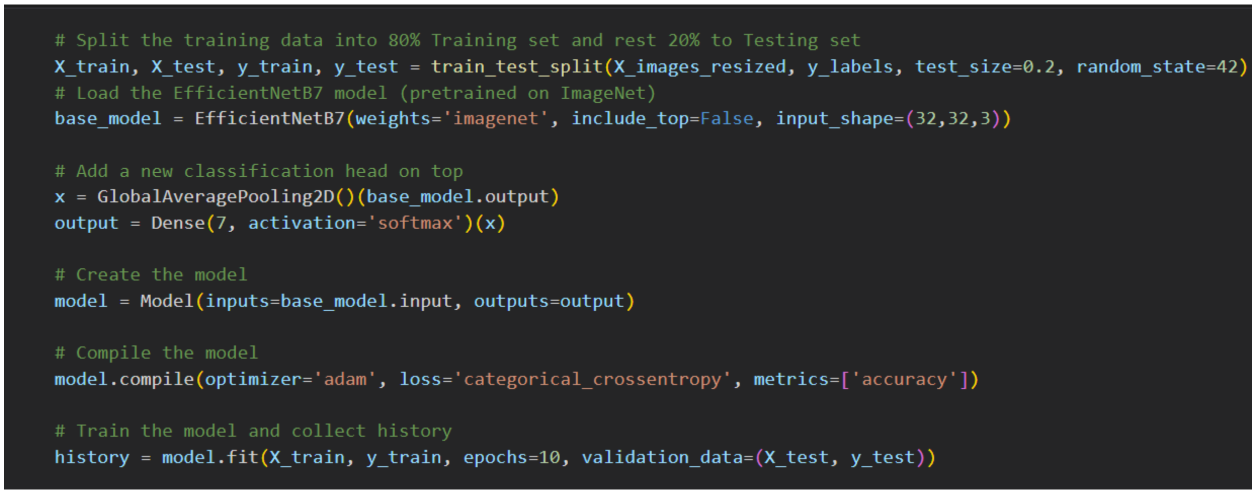


```
# Convert labels to categorical format
y_labels = to_categorical(labels)
```

3) **Model Selection:** The EfficientNetB7 model, pretrained on ImageNet, was chosen as the base model for skin cancer detection. EfficientNetB7 is a Convolutional Neural Network (CNN) model. It belongs to the family of CNN architectures specifically designed for image classification tasks. The choice of the EfficientNetB7 model was motivated by its strong performance in image classification tasks and its efficiency in terms of model size and computational requirements. By leveraging the pretraining and architectural efficiency of EfficientNetB7, the model can benefit from learned features, generalize well to skin cancer images, and potentially achieve high accuracy in classifying skin cancer types.

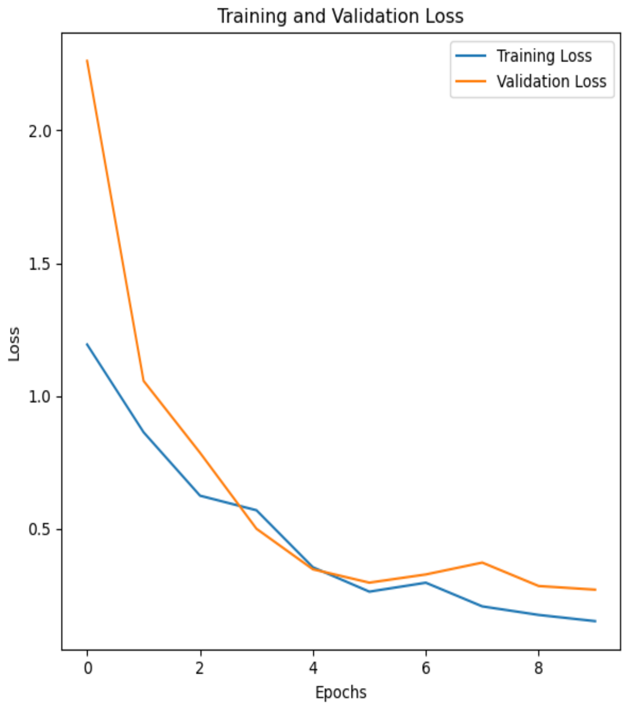
4) **Model Training:** The model was compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric. The Adam optimizer is a popular and widely used optimization algorithm for training neural networks. Adam adapts the learning rate for each parameter individually, which helps in faster convergence and better overall performance. The categorical cross-entropy loss function measures the dissimilarity between the predicted probability distribution and the true

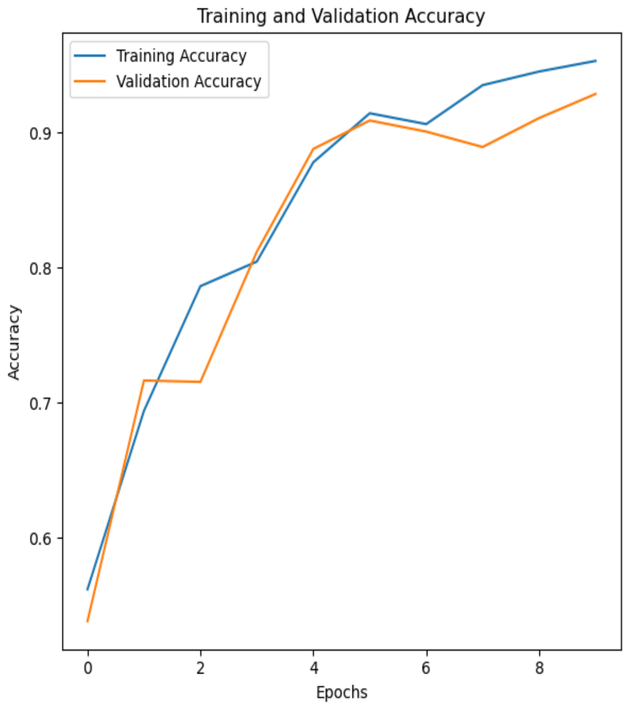
distribution of the target labels. By minimizing the cross-entropy loss, the model learns to assign high probabilities to the correct class and low probabilities to the incorrect classes. The model was trained using the training data and validated using the testing data for 10 epochs. We can use more epochs, but it can result in overfitting, so 10 epochs will serve our purpose.



```
# Split the training data into 80% Training set and rest 20% to Testing set
X_train, X_test, y_train, y_test = train_test_split(X_images_resized, y_labels, test_size=0.2, random_state=42)
# Load the EfficientNetB7 model (pretrained on ImageNet)
base_model = EfficientNetB7(weights='imagenet', include_top=False, input_shape=(32,32,3))

# Add a new classification head on top
x = GlobalAveragePooling2D()(base_model.output)
output = Dense(7, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=output)

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

# Train the model and collect history
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

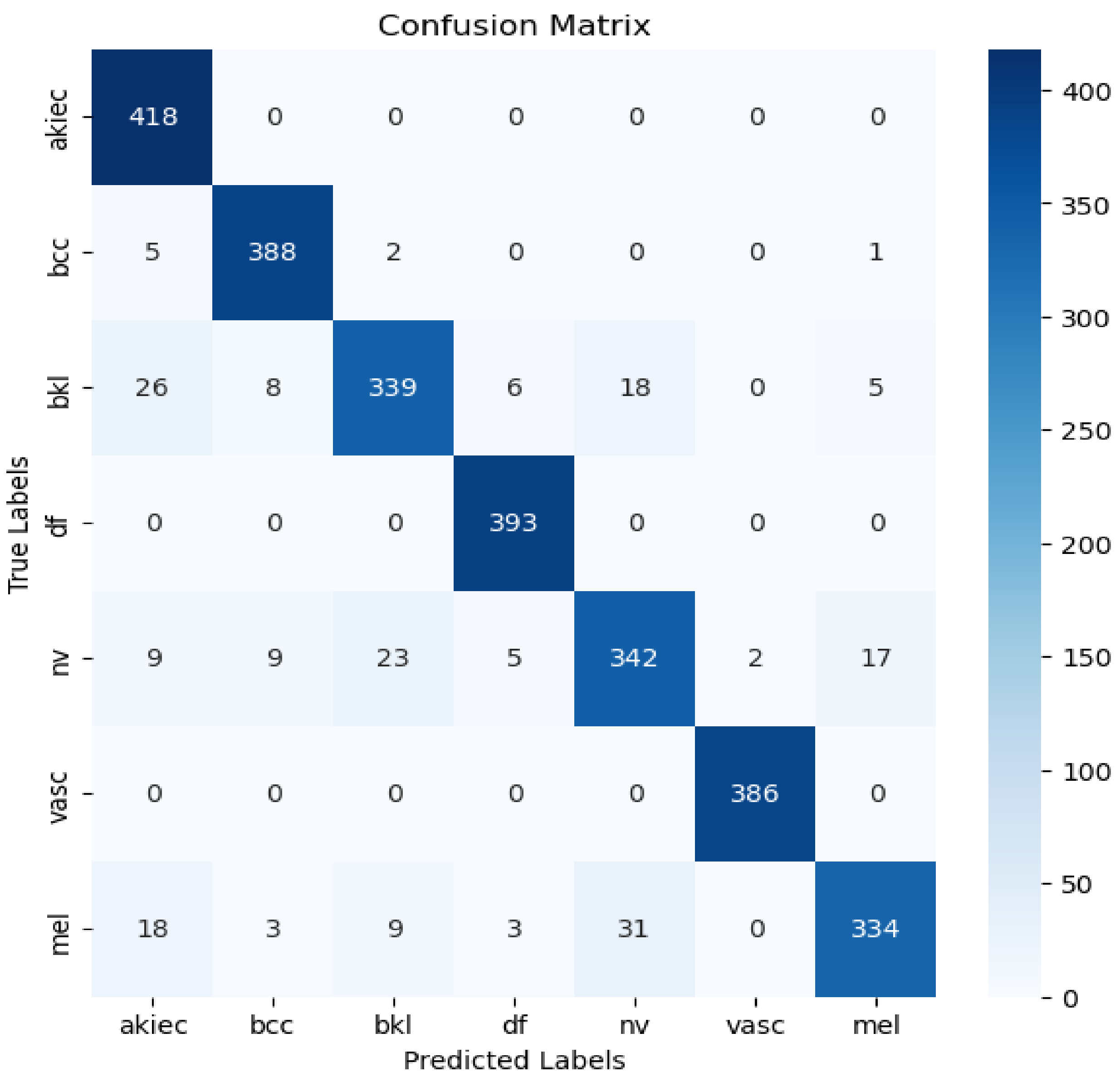
5) **Performance Evaluation:** The training and validation loss curves and the training and validation accuracy curves were plotted to assess the model's performance. Additionally, a confusion matrix was generated using the predicted and true labels of the testing set. Finally, the model accuracy is calculated.





If both the training and validation loss decrease steadily or plateau at a low value, it indicates that the model is learning the patterns in the data and is not overfitting or underfitting. This is desirable because it suggests that the model captures the data's underlying relationships and makes accurate predictions.

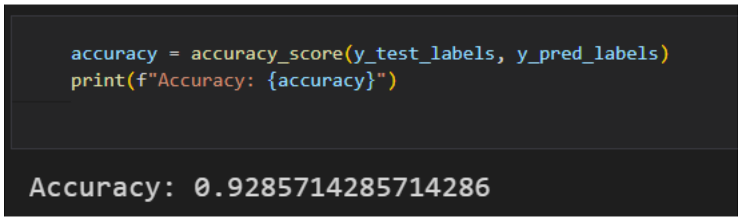
If the training and validation accuracy increase steadily or plateau at a high value, the model performs well and achieves accurate classifications on both the training and validation data. If the training accuracy continues to increase, but the validation accuracy stagnates or decreases, the model may be overfitting.



	akiec	bcc	bkl	df	nv	vasc	mel
akiec	418	0	0	0	0	0	0
bcc	5	388	2	0	0	0	1
bkl	26	8	339	6	18	0	5
df	0	0	0	393	0	0	0
nv	9	9	23	5	342	2	17
vasc	0	0	0	0	0	386	0
mel	18	3	9	3	31	0	334

In a confusion matrix, the rows represent the actual or true labels of the data, while the columns represent the predicted labels. Each cell in the matrix contains the count or percentage of samples that belong to a particular true class and were predicted as a particular predicted class. The main diagonal of the confusion matrix represents the correctly predicted samples, while the off-diagonal cells in the confusion matrix indicate misclassifications. By visualizing the confusion matrix, one can assess the model's overall accuracy and gain a deeper understanding of its strengths and weaknesses in classifying different categories.

Calculating the accuracy:



```
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9285714285714286
```

As evident from the confusion matrix, the model achieved a good accuracy of 92.86% on the testing set. This indicates that the model successfully learned and generalized patterns in the data, achieving a high level of accuracy in predicting the classes.

6) **Conclusion:** The model achieved an accuracy of 92.86% on the testing set, indicating its ability to learn and generalize patterns in the data. The high accuracy demonstrates the model's effectiveness in classifying skin cancer images. The project highlights the potential of utilizing state-of-the-art models like EfficientNetB7 for accurate image classification tasks and opens up avenues for further research and applications. Further improvements can be explored through different model architectures, hyperparameter tuning, and data augmentation techniques to enhance the model's performance further.