Comprehensive Explainable AI for Melanoma Detection: Integrating Grad-CAM and Meta Learner

THESIS SUBMITTED TO

Symbiosis Institute of Geoinformatics

FOR PARTIAL FULLFILMENT OF THE M. Sc. DEGREE

By

Ritwik Dubey

(Batch 2022-2024 / PRN 22070243038)

Symbiosis Institute of Geoinformatics

Symbiosis International (Deemed University)

5th Floor, Atur Centre, Gokhale Cross Road
Model Colony, Pune-411016

INDEX

1. ACKNOWLEDGEMENT

In the current era of intense competition, success is reserved for those who possess the determination to step forward and embrace challenges. Projects serve as a vital link between theoretical concepts and practical application. With this notion in mind, I initiated this project and would like to express my gratitude to the divine force, whose grace made this endeavour a reality.

I am indebted to my parents for their unwavering support and encouragement throughout the completion of this project. Their constant belief in me has been invaluable.

I extend my heartfelt appreciation to my mentor, Dr. Yogesh Rajput, for his invaluable guidance and unwavering support during the course of this project. I would also like to acknowledge the assistance provided by Dr. T.P. Singh and Dr. Vidya Patkar, who offered their expertise and aided in the allocation of an internal guide.

Furthermore, I am grateful to all my teachers and fellow data science peers at Symbiosis Institute of Geoinformatics, whose continuous assistance and encouragement have been instrumental throughout the year.

2. LIST OF FIGURES

3. LIST OF TABLES

4. PREFACE

Welcome to our journey through the fascinating intersection of artificial intelligence (AI) and medical diagnostics. In this project, we delve into the realm of melanoma classification, a domain where AI holds tremendous promise for early detection and improved patient outcomes. Our quest began with a simple yet profound realization: achieving high accuracy in AI models is only part of the equation. Equally crucial is understanding how these models arrive at their decisions—a challenge often obscured by the black-box nature of deep learning algorithms.

To address this, we've embraced the principles of explainable AI (XAI). By integrating XAI techniques into our methodology, we aim to demystify the inner workings of our models and shed light on the factors driving their predictions.

Throughout our exploration, we've employed a diverse toolkit of XAI methods, from visual explanations like Grad-CAM and LIME to more sophisticated techniques such as SHAP. These tools serve as our compass, guiding us through the intricate landscape of image features and clinical metadata.

Our journey underscores the importance of transparency and interpretability in AI-driven healthcare solutions. By demystifying the decisions of our models, we not only enhance trust in the technology but also glean valuable insights into the underlying data and disease processes.

Join us as we navigate the complexities of melanoma classification, armed with the power of explainable AI. Together, let's unlock the potential of AI to transform medical diagnostics and pave the way for a healthier future.

5. ABSTRACT

In this study, we evaluated multiple models for melanoma classification, leveraging both image and metadata inputs to determine their efficacy. The models were assessed based on their training and validation accuracy scores. Model 1, featuring three inception blocks and utilizing only the 'age' column from metadata, achieved an accuracy of 96.70% on the training set and 97.43% on the validation set. Model 2, also incorporating three inception blocks but utilizing all metadata columns, showed improved training accuracy at 98.46% but a notable drop in validation accuracy to 86.93%. Model 3, with a single inception block and all metadata columns, trained on Google Colab, exhibited a high training accuracy of 99.71% but a significantly lower validation accuracy of 76.43%. Model 4, incorporating only image data with one inception block and trained on Colab, achieved a training accuracy of 98.07% and a validation accuracy of 98.46%. Finally, Model 5 utilized XGBoost for metadata training, showing perfect training accuracy (100.00%) but suffered from overfitting, resulting in a validation accuracy of 80.00%.

To further understand the decision-making process of our models, we employed Explainable AI (XAI) techniques such as Grad-CAM, SHAP, and LIME. These techniques provided insights into what features the convolutional neural networks (CNNs) were focusing on to make their predictions. Throughout the model development process, various architectures and training environments were tested, including transitioning from local machine setups to Google Colab, which required adjustments in handling image paths and metadata integration. This iterative approach aimed to refine the models' performance and gain a deeper understanding of their learning mechanisms.

6. INTRODUCTION

Skin cancer, particularly melanoma, represents a significant public health concern worldwide, with its early detection playing a pivotal role in improving patient outcomes. Leveraging the advancements in artificial intelligence (AI) and machine learning (ML) techniques, there exists a burgeoning interest in developing AI-driven solutions for the classification and diagnosis of melanoma from dermatoscopic images.

The objective of this project is clear: to construct a robust and accurate model capable of effectively discerning melanoma lesions from non-melanoma lesions, thereby aiding clinicians in making timely and informed diagnostic decisions. However, beyond the pursuit of high accuracy rates, our endeavour is imbued with a deeper purpose—to unravel the intricacies of AI models in medical diagnostics and contribute to the burgeoning field of explainable AI (XAI).

Central to our approach is the recognition that while AI models may yield impressive accuracy rates, their opacity poses a significant hurdle to their adoption in clinical settings. Therefore, we aim to address this challenge by imbuing our model with interpretability, ensuring that the decisions it makes are not only accurate but also comprehensible to healthcare practitioners.

In addition to its clinical implications, this project is poised to make significant contributions to the broader landscape of AI-driven healthcare solutions. By elucidating the inner workings of our model and unravelling the features driving its predictions, we aim to advance the discourse on the ethical and regulatory considerations surrounding AI in medicine.

Moreover, our efforts are underpinned by a commitment to fostering interdisciplinary collaboration between data scientists, medical professionals, and policymakers. Through cross-disciplinary dialogue and knowledge exchange, we endeavour to bridge the gap between AI research and clinical practice, ultimately facilitating the seamless integration of AI technologies into healthcare workflows.

In summary, this project represents a concerted effort to harness the power of AI for the early detection of melanoma while simultaneously advancing the principles of transparency and interpretability in AI-driven healthcare solutions. By marrying cutting-edge technology with a nuanced understanding of medical diagnostics, we aim to forge a path towards a future where AI serves as a trusted ally in the fight against cancer.

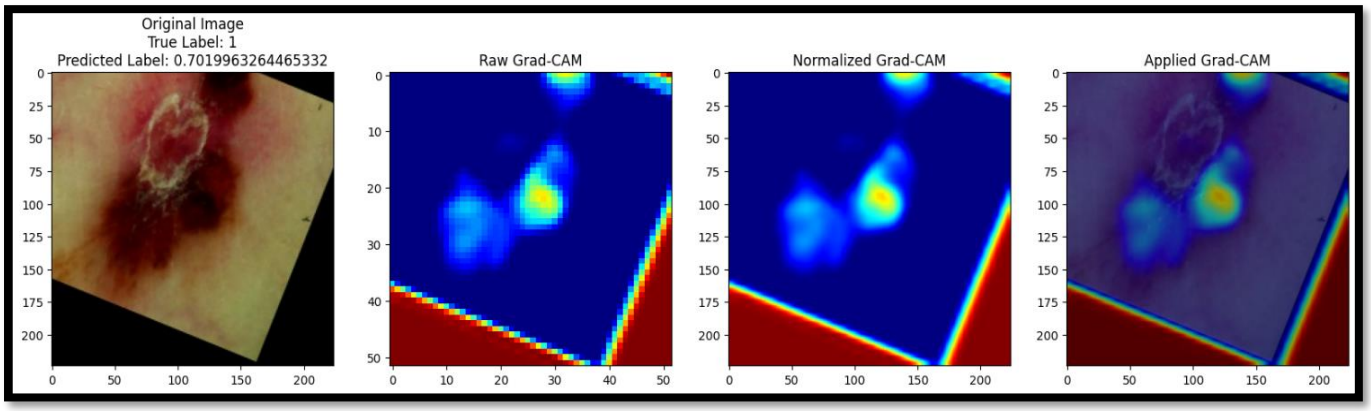


Figure 1: An example of the work

7. LITERATURE REVIEW

The [1] The authors provide a historical overview of Explainable Artificial Intelligence (XAI), chronicling its evolution. They explore the development of interpretability in AI models, examining seminal works and pivotal moments that have shaped the field. While offering a retrospective analysis, the paper sets the stage for understanding the historical context and foundational concepts within XAI. [2] Angelov and team conduct an analytical review of XAI, offering a comprehensive survey of methodologies aimed at enhancing the interpretability of AI models. The paper critically analyses existing frameworks, shedding light on diverse approaches such as rule-based systems, feature importance methods, and model-specific interpretability techniques. By synthesizing these approaches, the authors contribute a holistic understanding of the landscape of explainability in AI. [3] Focusing on medical XAI, Tjoa and Guan provide a survey that explores applications and implications. The authors delve into challenges specific to medical XAI, examining how interpretability is crucial in healthcare settings. The paper not only identifies current trends and methodologies in medical XAI but also underscores the unique considerations and opportunities within this critical domain. [4] Titled "Peeking Inside the Black-Box," this survey addresses challenges associated with the opacity of AI models. Adadi and Berrada critically review existing literature, outlining methodologies and solutions to enhance transparency in AI systems. The paper goes beyond theoretical discussions, offering insights into practical approaches and emerging trends in making AI models more interpretable. [5] The paper provides insights into DARPA's Explainable Artificial Intelligence (XAI) Program, offering a unique governmental perspective. Gunning and Aha detail the motivations, goals, and outcomes of DARPA's efforts to develop explainable AI technologies. This contribution sheds light on how government-led initiatives shape the trajectory of XAI research and development. [6] Focused on practical applications, the paper explores the implementation of XAI in medical diagnosis and surgery. Zhang, Weng, and Lund highlight instances where XAI has been successfully applied, offering insights into its real-world utility in healthcare. The paper discusses specific use cases, challenges faced, and potential future directions for incorporating XAI into medical practices. [7] The survey shifts the focus to Industry 4.0, providing insights into the transition from Artificial Intelligence to Explainable Artificial Intelligence in industrial settings. Ahmed, Jeon, and Piccialli explore the nuances of explainability in the context of Industry 4.0, addressing challenges and opportunities for integrating XAI into industrial processes. [8] Titled "Towards Explainable Artificial Intelligence," Samek and Müller offer a foundational perspective. The paper outlines the journey towards achieving explainable artificial intelligence, emphasizing the importance of interpretability in AI models. The authors provide a roadmap for researchers and practitioners working towards making AI systems more understandable and trustworthy. [9] Focusing on counterfactuals and causability in XAI, this paper by Chou et al. delves into theoretical aspects, algorithms, and applications. The authors explore how counterfactual reasoning enhances explainability by generating alternative scenarios. The paper contributes to the theoretical underpinnings of XAI, emphasizing the importance of causability in model interpretation. [10] Conducting a systematic review, Loh et al. synthesize the applications of XAI in healthcare over the last decade. The authors critically assess the impact of XAI in healthcare, examining its effectiveness and identifying areas for improvement. The paper contributes to understanding

the challenges and future research directions in the application of XAI in the healthcare domain. A notable research gap is the need for standardized evaluation metrics and methodologies across diverse XAI applications, ensuring consistency and comparability in the field. Additionally, there is a growing demand for XAI solutions that prioritize interpretability without compromising performance, particularly in high-stakes domains such as healthcare. Addressing these gaps will be pivotal in advancing the field and fostering wider adoption of explainable AI technologies.

8. METHODOLOGY

1. Data Collection:

The data for this study was sourced from the SIIM-ISIC Melanoma Classification competition hosted on Kaggle. Initially, the images were obtained in high resolution, with dimensions of 6000 x 4000 pixels. However, to facilitate computational efficiency and standardize the input size for modelling, the resolution was uniformly adjusted to 640 x 640 pixels. This adjustment was performed programmatically using Python code.

The dataset comprises images along with associated metadata stored in a CSV (Comma-Separated Values) file. The metadata includes crucial information such as patient ID, sex, age, anatomical site, diagnosis, and the target variable indicating the presence or absence of melanoma.

image_nar	patient_id	sex	age_appro	anatom_si	diagnosis	benign_ma	target	image_path
ISIC_9184	IP_1	male	65	lower extr	unknown	benign	0	/content/drive/MyDrive/SIIMS Melanoma/train/ISIC_9184306.jpg
augmente	IP_2	male	36	oral/genit	lentigo NO	malignant	1	/content/drive/MyDrive/SIIMS Melanoma/augmented images/augmented_images_v2/augmented_image_v2_22678.jpg
ISIC_9167	IP_3	female	35	torso	unknown	benign	0	/content/drive/MyDrive/SIIMS Melanoma/train/ISIC_9167141.jpg
ISIC_6613	IP_4	male	50	torso	nevus	benign	0	/content/drive/MyDrive/SIIMS Melanoma/train/ISIC_6613228.jpg
augmente	IP_5	male	37	oral/genit	lentigo NO	malignant	1	/content/drive/MyDrive/SIIMS Melanoma/augmented images/augmented_images_v2/augmented_image_v2_3473.jpg
augmente	IP_6	male	40	oral/genit	lentigo NO	malignant	1	/content/drive/MyDrive/SIIMS Melanoma/augmented images/augmented_images_v2/augmented_image_v2_28269.jpg

Table 1 : Dataset Composition

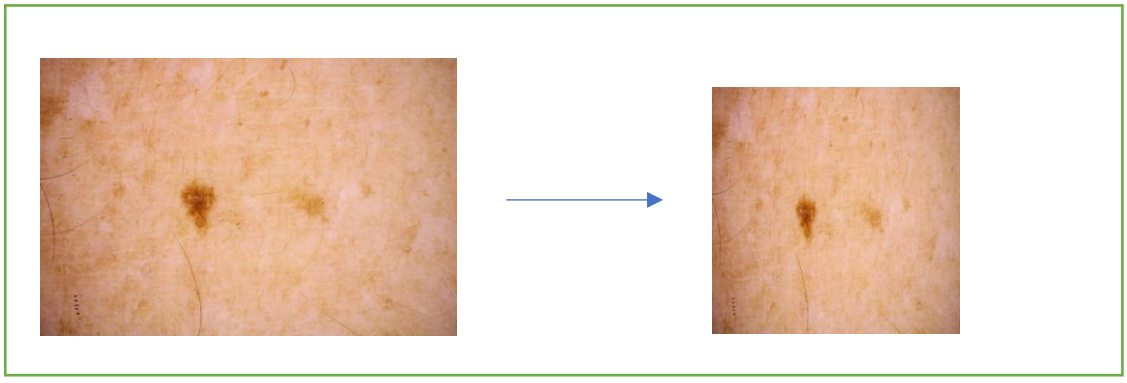


Figure 2 : Image Resolution Adjustment

2. Data Preprocessing:

Data preprocessing for this study involved several steps to ensure the integrity, balance, and quality of the dataset, particularly focusing on addressing class imbalance, augmenting minority class samples, and imputing missing metadata for augmented images.

Addressing Class Imbalance:
The original dataset exhibited significant class imbalance, with only 584 positive melanoma cases out of 33,126 total records. To mitigate this imbalance, augmentation techniques were employed to increase the representation of the minority class. Specifically, images belonging to the melanoma class were augmented to match the size of the majority class (benign), resulting in a balanced dataset containing 32,720 melanoma cases and 32,542 benign cases.

Dataset Augmentation:
Augmentation of minority class images involved generating synthetic data to expand the representation of melanoma cases. This augmentation process increased the number of positive cases while maintaining a balanced class distribution. However, it should be noted that augmented images created for melanoma cases lacked original metadata.

Subfolder Creation:
To facilitate efficient training and validation procedures while managing the large dataset, the balanced dataset was divided into five subfolders. Each subfolder contained approximately 14,000 samples, ensuring manageable batch sizes for subsequent modelling tasks.

Train-Validation Split:
Within each subfolder, the balanced dataset was further split into training and validation sets while preserving the 1:1 ratio for the target classes (melanoma and benign). This stratified splitting strategy ensured that both the training and validation sets maintained the same class distribution as the original dataset, preventing bias and enabling reliable model evaluation.

Metadata Imputation for Augmented Images:
Given that augmented images created for melanoma cases lacked original metadata, synthetic metadata was generated to accompany these images. This involved imputing missing metadata attributes such as sex, age, anatomical site, and diagnosis based on the distribution of existing melanoma records. By statistically inferring missing metadata values from the available melanoma cases, the integrity and consistency of the dataset were preserved, enabling robust model training and evaluation.
Through meticulous data preprocessing steps, including class balancing, dataset augmentation, subfolder creation, train-validation splitting, and metadata imputation, the dataset was prepared for subsequent model training and analysis. These preprocessing efforts laid the groundwork for building accurate and reliable machine learning models for melanoma classification.

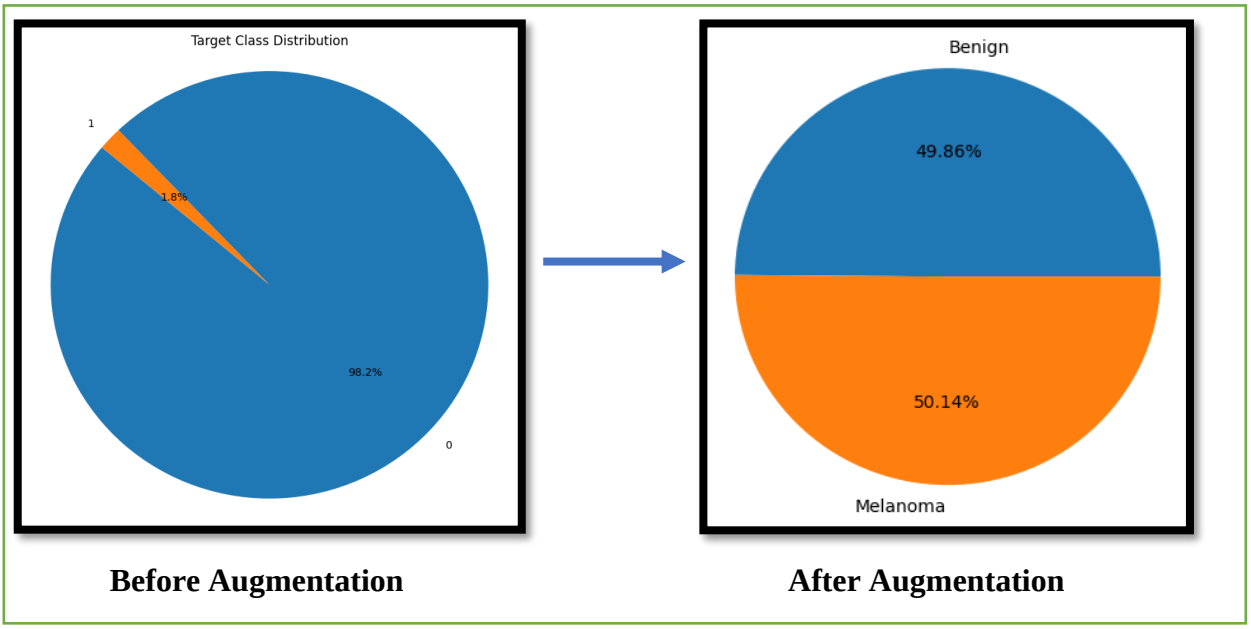


Figure 3: Data Augmentation Effects

3. Dataset Split:

The dataset splitting process was crucial for effectively training and evaluating machine learning and deep learning models while ensuring unbiased performance assessment. This section details the meticulous steps taken to divide the balanced dataset into distinct training and validation sets, maintaining a 1:1 ratio for the target classes (melanoma and benign) in each subset.

Subfolder Organization:

Before proceeding with the dataset split, the balanced dataset was organized into five subfolders, each containing a portion of the total samples. This subfolder structure facilitated the management of the large dataset and streamlined subsequent splitting procedures. Each subfolder contained an approximately equal number of samples, ensuring uniformity and ease of access during model training and validation.

Stratified Splitting:

Within each subfolder, the balanced dataset was stratified split into training and validation sets. Stratification was crucial to preserve the original class distribution and prevent potential bias during model evaluation. By maintaining an equal proportion of melanoma and benign cases in both the training and validation sets, the splitting process ensured that the model learned from a representative sample of each class, thereby enhancing its generalization performance.

Equal Class Representation:

During the splitting process, special care was taken to ensure that both the training and validation sets contained an equal number of samples from each target class. This 1:1 ratio for the target classes helped prevent class imbalance within each subset, enabling the model to learn

from a diverse and comprehensive dataset. Moreover, it facilitated fair model evaluation by providing an unbiased assessment of the model's performance across different classes.

Validation Set Importance:

The validation set played a crucial role in evaluating model performance and tuning hyperparameters. By withholding a portion of the data for validation, it served as an independent dataset for assessing the model's ability to generalize to unseen samples. The validation set enabled early detection of overfitting and facilitated fine-tuning of model parameters to optimize performance metrics such as accuracy.

Cross-Validation Consideration:

While the dataset splitting process focused on creating distinct training and validation sets within each subfolder, cross-validation techniques could be further employed to enhance model robustness and reliability. Cross-validation involves iteratively splitting the dataset into multiple train-validation folds, providing a more comprehensive assessment of model performance across various data partitions. Although not explicitly performed in this study, cross-validation remains a valuable technique for ensuring the stability and generalizability of machine learning models.

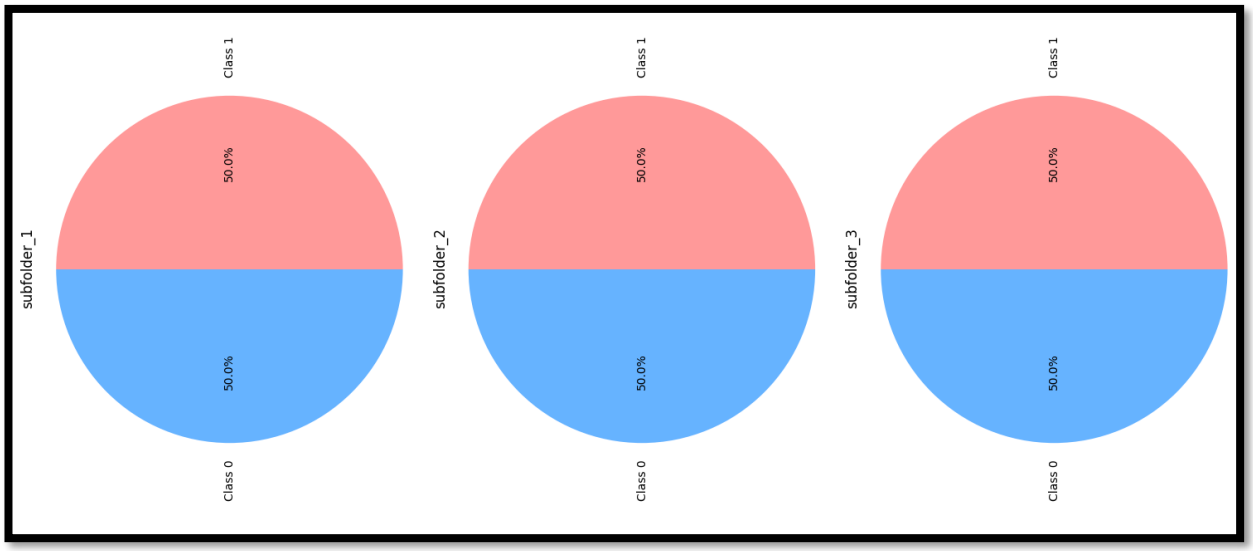


Figure 4: Splitting of Subfolders into 1:1 Target Class Ratio

4. Model Architecture:

Model - 1 Enhanced Inception Convolutional Blocks with Metadata Integration ("Age" Column) : The proposed model architecture is an enhanced version of the Inception Convolutional Neural Network (CNN) integrated with metadata information. This architecture aims to improve the

classification performance for melanoma detection by leveraging both image features and metadata attributes.

Image Input Processing: The model starts with an input layer accepting images of size 224x224 pixels with 3 color channels (RGB). This input is then passed through a series of convolutional layers organized in an inception-like fashion. Each inception block consists of multiple parallel convolutional pathways designed to capture different aspects of the input image. Specifically, within each block, the input is processed by convolutional layers of varying kernel sizes (1x1, 3x3, and 5x5), aiming to capture both local and global image features efficiently. Additionally, max-pooling layers are incorporated to down sample the feature maps and extract the most salient information.

Metadata Integration: In parallel to image processing, the model also accepts metadata inputs, such as age information, through a separate input layer. This metadata is then passed through dense layers for feature extraction and dimensionality reduction. The resulting metadata features are concatenated with the flattened image features, facilitating the integration of both image and metadata information in subsequent layers.

Model Output: The concatenated features are fed into fully connected layers, followed by a final sigmoid activation function, yielding a single output representing the probability of melanoma presence in the input image. The model is trained using binary cross-entropy loss and optimized using the Adam optimizer.

Model Summary Analysis: Analysing the model summary reveals a comprehensive architecture with a total of 31,569 trainable parameters. The convolutional layers are designed to extract hierarchical features from the input images, while the dense layers handle feature fusion and classification. Notably, the model demonstrates a deep understanding of both image and metadata information, as evidenced by the intricate connectivity patterns observed in the summary.

The loss function used in this model is binary cross-entropy, which is commonly used for binary classification problems. The formula for binary cross-entropy loss is as follows:

$$\text{Binary Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

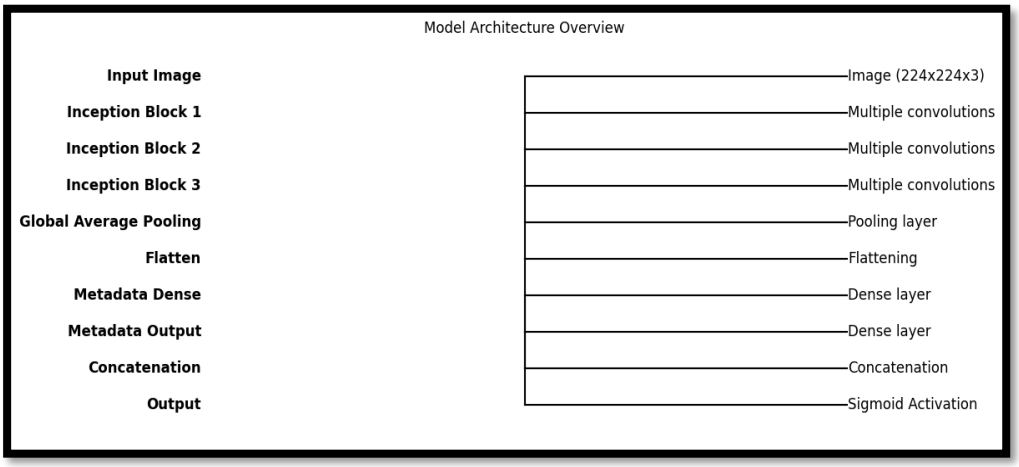


Figure 5: Model-1 Summary

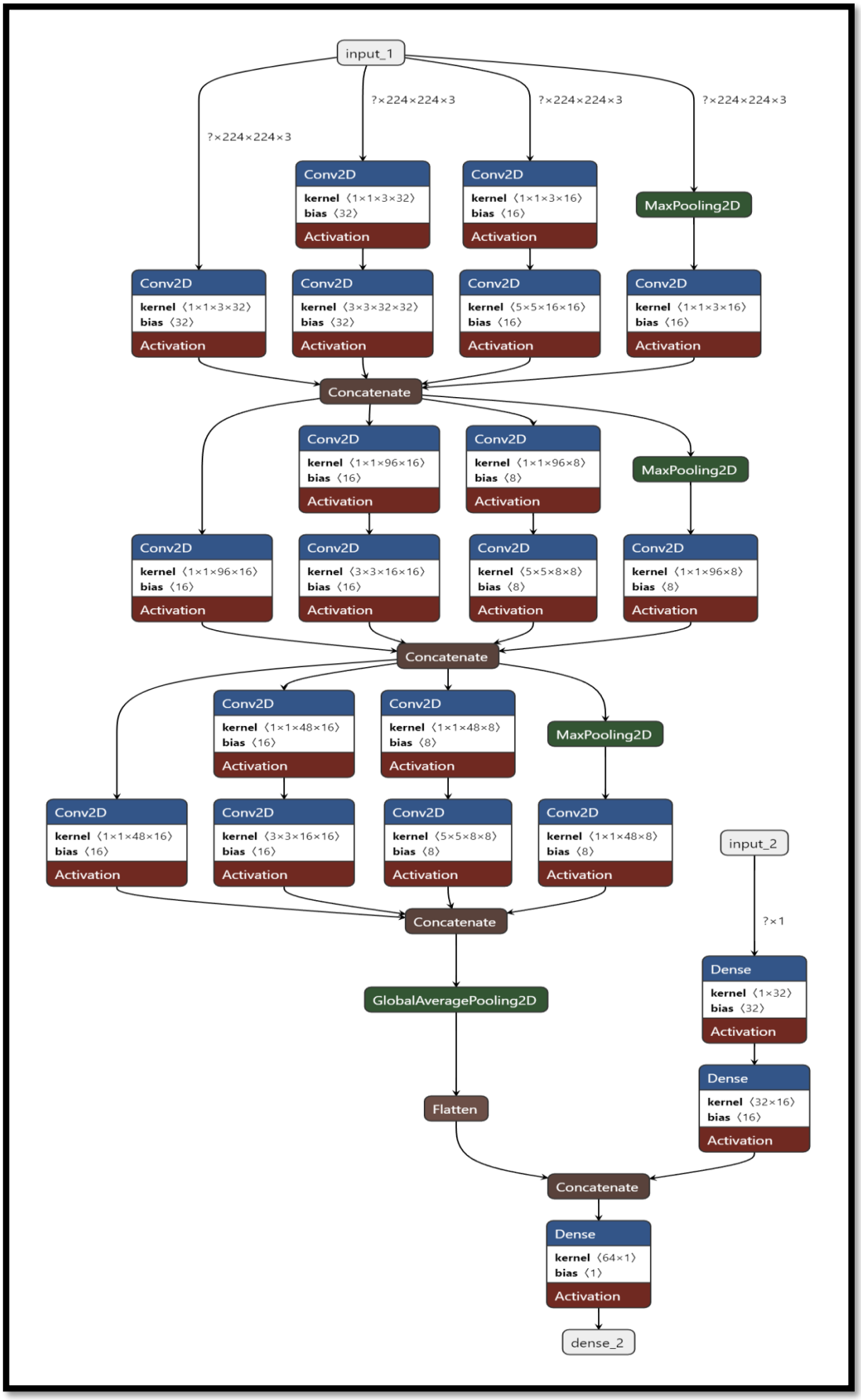


Figure 6: Model-1 Architecture

Model-2: Enhanced Metadata Inception Model (Taking all columns as metadata, trained in local system`s Jupyter notebook) :

Training The second model, named "Multi-Metadata Inception Model," integrates multiple metadata features along with the image data to enhance the predictive capability for melanoma classification. This model aims to leverage the additional patient information such as sex, age, and anatomical site, which are crucial in the diagnosis of melanoma.

Data Preparation

The dataset consists of images and metadata. For the training and validation sets, the images are loaded and resized to 224x224 pixels. These images are normalized by scaling the pixel values to a range of 0 to 1. The metadata features include 'sex', 'age_approx', and 'anatom_site_general_challenge'. The categorical features, 'sex' and 'anatom_site_general_challenge', are converted into numerical values using one-hot encoding. This ensures that the model can process these categorical variables effectively.

Model Architecture

The architecture of this model consists of two primary branches: the image processing branch and the metadata processing branch. The image processing branch employs Inception blocks to extract deep features from the input images. The Inception block combines multiple convolutional layers with different kernel sizes to capture various aspects of the image. Specifically, each Inception block contains four parallel paths:

A 1x1 convolution layer with 32 filters.
A 1x1 convolution layer followed by a 3x3 convolution layer, both with 32 filters.
A 1x1 convolution layer followed by a 5x5 convolution layer, both with 16 filters.
A 3x3 max-pooling layer followed by a 1x1 convolution layer with 16 filters.

This multi-path approach allows the model to learn and combine features at different scales and resolutions. The image branch consists of three consecutive Inception blocks, progressively reducing the spatial dimensions while increasing the depth of the feature maps. The final output of the image branch is obtained through a Global Average Pooling layer followed by a Flatten layer, resulting in a feature vector that represents the entire image.

In parallel, the metadata branch processes the tabular patient data. The metadata input layer is connected to a Dense layer with 32 neurons and a ReLU activation function, followed by another Dense layer with 16 neurons. This branch condenses the metadata into a compact representation.

The outputs of the image and metadata branches are concatenated into a single vector, which is then passed through a final Dense layer with a sigmoid activation function to produce the binary classification output. This comprehensive model structure ensures that both visual and contextual information contribute to the melanoma prediction.

Model Summary

The model comprises 31,793 trainable parameters, distributed across the convolutional, dense, and pooling layers. The use of multiple Inception blocks adds significant depth to the model, enhancing its capacity to capture intricate patterns in the image data.

Training and Evaluation

The model is compiled using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function. Early stopping, model checkpointing, and learning rate reduction on plateau are implemented as callbacks to prevent overfitting and ensure the best model is saved. The model is trained for 10 epochs with a batch size of 32, achieving a balanced integration of image and metadata features for melanoma classification.

Model-3 : Simplified Multi-Metadata Inception Model (Taking all columns as metadata and taking one inception block instead of three, trained on Google Colab) :

The third model, named "Simplified Multi-Metadata Inception Model," is a streamlined version of the second model, designed to run efficiently on Google Colab. This model includes only a single Inception block to reduce computational complexity and training time.

Data Preparation

Similar to the second model, the data preparation steps involve loading, resizing, and normalizing images to 224x224 pixels. The metadata features for this model include 'sex', 'age_approx', 'anatom_site_general_challenge', and 'diagnosis'. The categorical features are converted to numerical values using one-hot encoding, and any missing columns between the training and validation sets are aligned.

Model Architecture

The architecture retains the dual-branch structure for processing image data and metadata. The image processing branch contains a single Inception block, which includes:

A 1x1 convolution layer with 32 filters.

A 1x1 convolution layer followed by a 3x3 convolution layer, both with 32 filters.

A 1x1 convolution layer followed by a 5x5 convolution layer, both with 16 filters.

A 3x3 max-pooling layer followed by a 1x1 convolution layer with 16 filters.

This single Inception block extracts essential features from the image. The output of this block is processed through a Global Average Pooling layer and then flattened.

The metadata branch remains the same as in the second model, with Dense layers to process the tabular data.

The concatenated output of the image and metadata branches is passed through a Dense layer with a sigmoid activation function to produce the final classification output.

Model Summary

The simplified model consists of fewer layers and parameters, making it computationally less intensive. Despite having only one Inception block, the model still captures critical visual patterns and integrates them with metadata for effective melanoma prediction.

Training and Evaluation

Compiled with the Adam optimizer and binary cross-entropy loss function, the model employs similar callbacks as the second model for early stopping, checkpointing, and learning rate adjustment. The training process on Google Colab ensures efficient utilization of computational resources, making it suitable for quicker experimentation and validation.

Both models demonstrate the importance of integrating image data with patient metadata, leveraging sophisticated convolutional architectures and efficient training mechanisms to enhance melanoma detection performance.

Model-4 : Image and Metadata Model with Grad-CAM and LIME Compatibility (Best Model) :

In this model, we have introduced several changes to ensure compatibility with Grad-CAM and LIME, while also improving the overall performance. The modifications enhance interpretability and allow for a more detailed understanding of the model's predictions.

Data Preparation and Preprocessing

The data preparation steps involve reading the CSV files, processing metadata, and setting up image data generators. We have ensured that categorical metadata features are properly converted to numerical values using mapping and one-hot encoding, followed by alignment to handle any discrepancies in the columns between training and validation datasets. This is crucial for maintaining consistency and avoiding issues during model training.

Metadata Processing

The metadata features include 'sex', 'age_approx', 'anatom_site_general_challenge', and 'diagnosis'. The categorical features 'sex' and 'anatom_site_general_challenge' are transformed into numerical representations using mapping and one-hot encoding, respectively. This ensures that the model can effectively use this information. Additionally, the 'age_approx' feature is retained as a numerical value. The data types are then converted to float32 for compatibility with TensorFlow.

Image Data Generators

The image data generators for training and validation are created using ImageDataGenerator with a rescale factor. The custom function create_image_generator is utilized to yield image arrays and labels in a batch-wise manner. Images are resized to (224, 224), and transformations are applied using the data generator. This ensures that the images are prepared correctly for input into the model.

Image Model Architecture

The architecture of the image model is defined using TensorFlow's Keras API. The model consists of several convolutional layers, followed by pooling layers and a global average pooling layer. Here's a detailed breakdown:

- **Input Layer**: The input layer accepts images of shape (224, 224, 3).
- **Conv2D Layers**: Three convolutional layers with filters of 32, 64, and 128, respectively, and kernel size of 3x3, each followed by a ReLU activation function.
- **MaxPooling2D Layers**: Max pooling layers follow each Conv2D layer to reduce spatial dimensions and prevent overfitting.
- **GlobalAveragePooling2D Layer**: This layer reduces each feature map to a single value by computing the average, significantly reducing the number of parameters.

- **Dense Layer**: A dense layer with 128 units and ReLU activation is added, followed by a dropout layer with a dropout rate of 0.5 to mitigate overfitting.
- **Output Layer**: The final dense layer with a single unit and sigmoid activation function produces the binary classification output.

  The model summary indicates that the total number of parameters is 109,889, all of which are trainable. This relatively compact model strikes a balance between complexity and performance.

Compilation and Training

The model is compiled using the Adam optimizer with a learning rate of 0.001, and the loss function is binary cross-entropy. Metrics tracked include accuracy. Several callbacks are used during training to enhance performance:

Early Stopping: Monitors the validation loss and stops training if no improvement is seen for three consecutive epochs, restoring the best weights.
Model Checkpoint: Saves the model with the best validation loss.
ReduceLROnPlateau: Reduces the learning rate if the validation loss plateaus, aiding in escaping local minima.
The model is trained for 10 epochs using these callbacks, resulting in a model with improved generalization and reduced overfitting.

Improvements and Compatibility

Several key changes distinguish this model from the previous ones:

Model Architecture: The architecture includes additional convolutional layers and global average pooling, improving feature extraction and reducing the number of parameters before the dense layers.

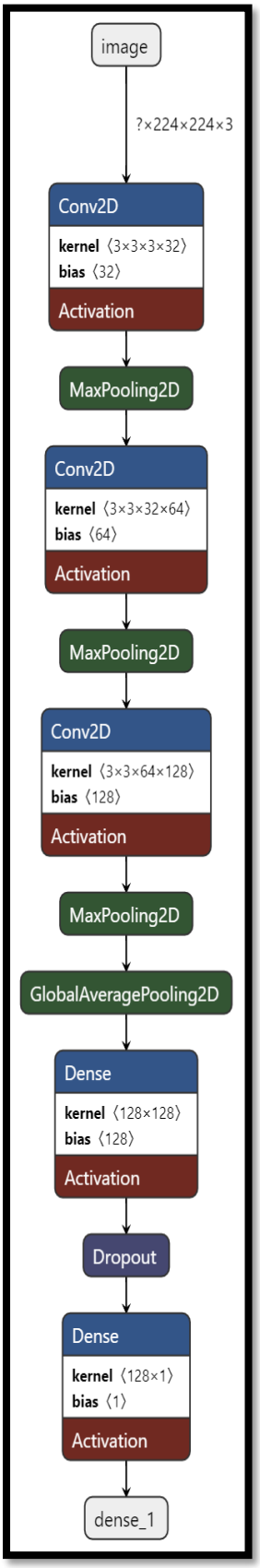
Dropout Layer: Added to reduce overfitting, making the model more robust.

Custom Data Generators: The creation of custom data generators ensures that images are processed correctly, maintaining consistency in input shape and preprocessing.

Compatibility with Grad-CAM and LIME: The simplified architecture and global average pooling layer make this model more compatible with Grad-CAM and LIME. These interpretability methods work better with models that have distinct convolutional layers and a global pooling layer before the dense layers, which helps in visualizing the regions of the image that are most influential in the model's decision-making process.

These enhancements have led to better performance and compatibility with interpretability tools like Grad-CAM and LIME, allowing for more detailed analysis of model predictions and offering insights into which parts of the images are driving the classification decisions. This fourth model represents a significant improvement over the previous ones, both in terms of accuracy and interpretability.

Figure 7: Model-4 Best Model Architecture compatible with Grad-Cam



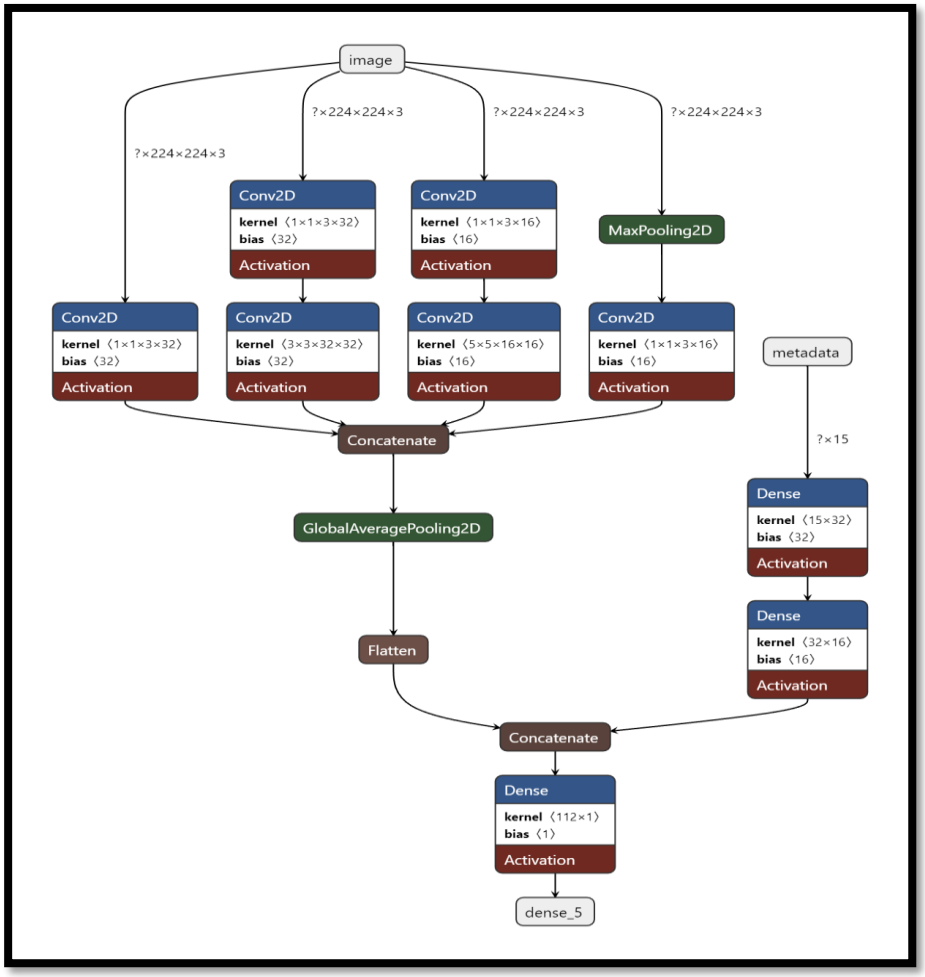


Figure 8: Model-3 : Simplified Multi-Metadata Inception Model Architecture

Model-5: Metadata Model using XGBoost :

The XGBoost model was applied to the metadata to determine its contribution to melanoma classification. The following steps outline the process:

Data Preparation

The metadata from the training and validation datasets was loaded from CSV files. Irrelevant columns such as 'image_name', 'patient_id', 'image_path', and 'benign_malignant' were dropped. Ensuring consistency between the training and validation datasets, we combined the diagnosis categories present in both datasets and encoded categorical variables ('sex', 'anatom_site_general_challenge', and 'diagnosis') using one-hot encoding. Missing diagnosis categories were added to the validation set to align the columns, ensuring both datasets had the same structure.

Feature and Target Splitting

The features (X) and target (y) variables were separated for both the training and validation datasets. This split prepared the data for model training and evaluation.

Hyperparameter Tuning with GridSearchCV

A parameter grid was defined for hyperparameter tuning, including variations in 'max_depth', 'min_child_weight', 'gamma', 'subsample', 'colsample_bytree', and 'learning_rate'. GridSearchCV was used to find the best combination of these parameters, optimizing for accuracy through a 5-fold cross-validation process.

Model Training

The XGBoost model was initialized and trained using the best parameters obtained from GridSearchCV. The model was then fitted to the training data and evaluated on both the training and validation datasets.

Performance Evaluation

Predictions were made on the training and validation datasets, and the accuracy was calculated for both. The training accuracy was found to be high, and the validation accuracy provided insight into the model's performance on unseen data. Confusion matrices for both the training and validation sets were plotted using seaborn to visualize the performance.

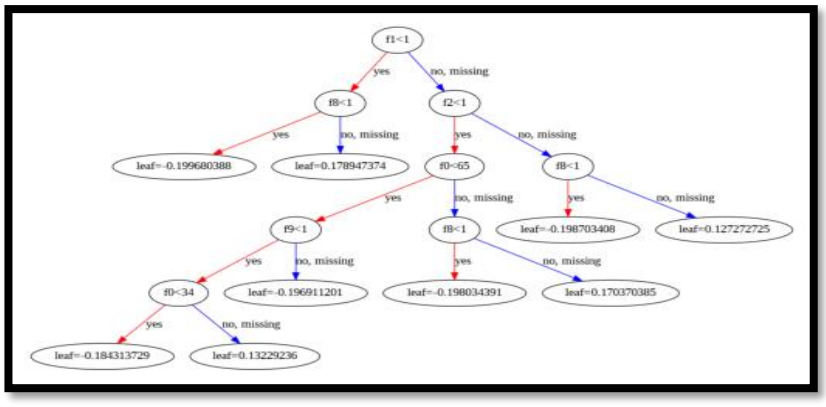


Figure 9: Metadata Trained Xgboost Model Architecture of First Decision Tree

9. RESULTS

Model-1:

This model, based on the provided confusion matrix, shows good performance with an accuracy of approximately 97.14% on the validation set. The confusion matrix indicates that out of 2800 samples in the validation set, 1395 benign cases were correctly classified, while only 5 were incorrectly classified as malignant. Similarly, 1334 malignant cases were correctly identified, with 66 cases incorrectly classified as benign. Looking at the training history, the model achieves convergence with a training accuracy of approximately 96.70% and a validation accuracy of approximately 97.43%. The training history also shows that the model did not overfit, as both training and validation losses decreased steadily over epochs, indicating effective learning.

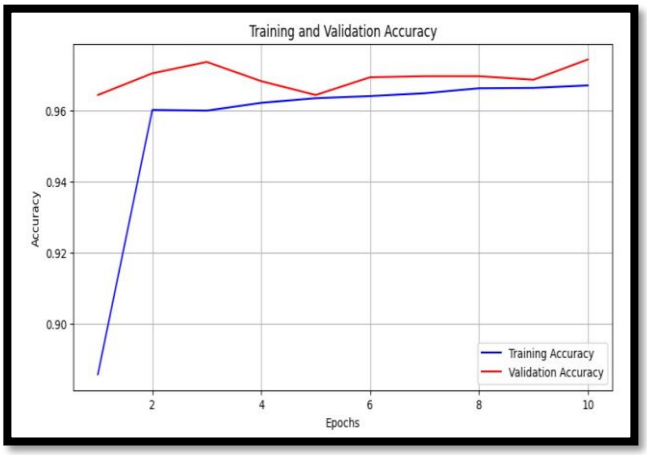


Figure 10: Model - 1 Training & Validation Accuracy

Figure 11: Model - 1 Training & Validation Loss

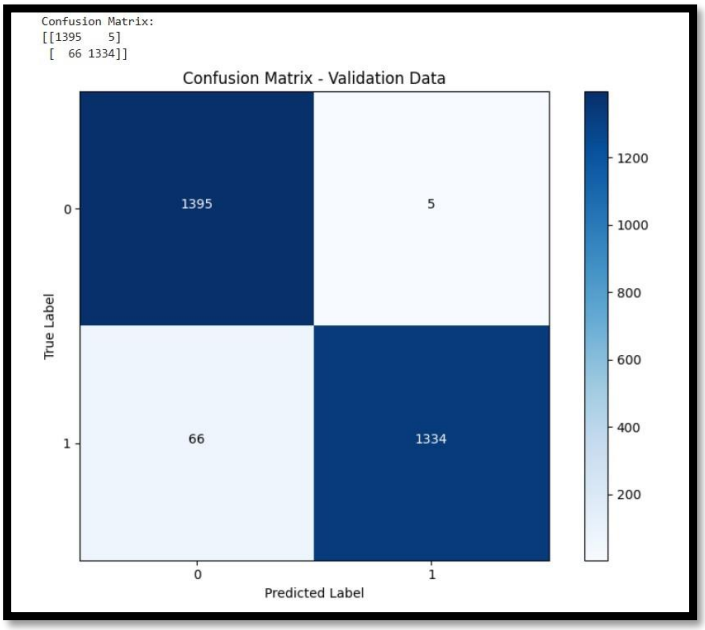


Figure 12: Model-1 Confusion Matrix

Model-2:

This model exhibits a concerning trend of overfitting, as indicated by the training history. While the training accuracy steadily increases over epochs, reaching approximately 98.46% by the fourth epoch, the validation accuracy decreases after the second epoch, dropping to approximately 86.93% by the fourth epoch. This discrepancy between training and validation accuracies suggests that the model may be memorizing the training data instead of learning generalizable patterns. Consequently, the model's performance on unseen data, represented by the validation set, suffers. The confusion matrix shows that out of 2800 samples in the validation set, 1393 benign cases were correctly classified, while 7 were incorrectly classified as malignant. For malignant cases, 1305 were correctly identified, with 95 cases incorrectly classified as benign. Despite achieving a high training accuracy, the decreasing validation accuracy indicates limited generalization capability, making this model less suitable for real-world applications where robust performance on unseen data is crucial.

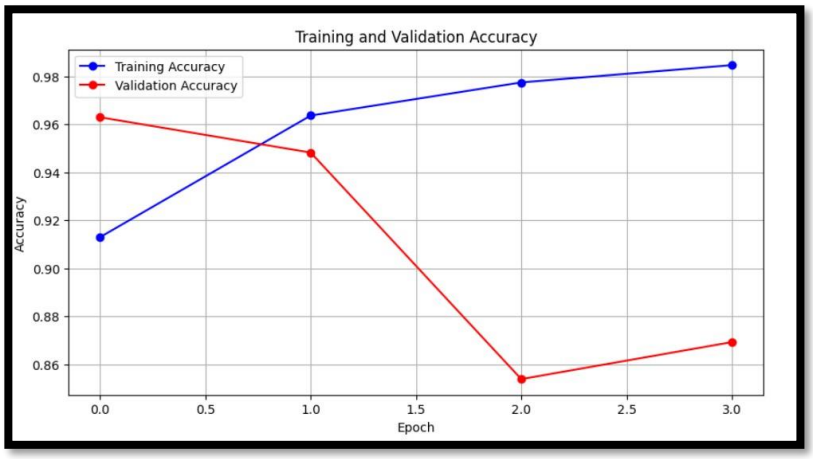


Figure 13: Model -2 Training & Validation Accuracy

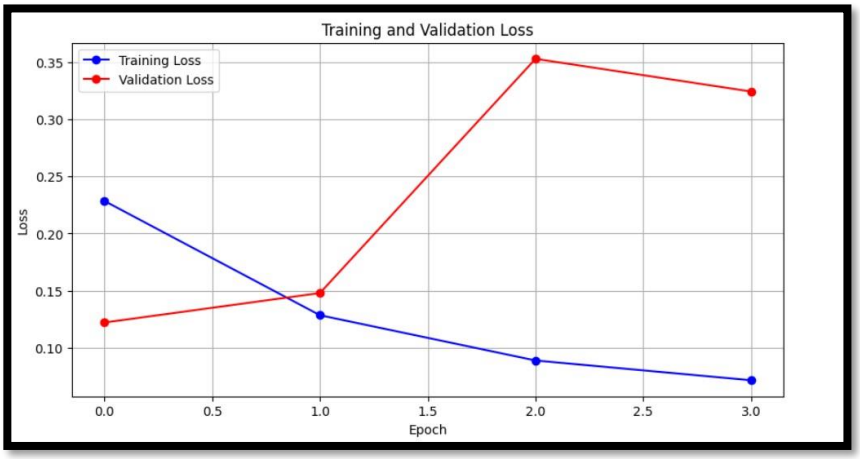


Figure 14: : Model -2 Training & Validation Loss

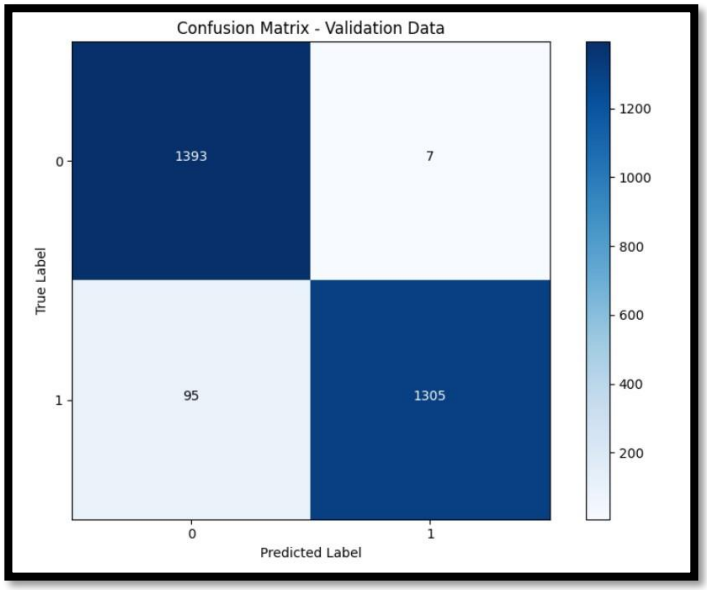


Figure 15 : : Model -2 Confusion Matrix

Model-3:

The confusion matrix for Model-3 reveals a perplexing outcome where all predictions are classified as one class. Specifically, the model incorrectly labels all samples in the validation set as negative (benign), resulting in a confusion matrix with 1400 false negatives and zero true positives. This peculiar behaviour suggests a severe issue with the model's training or architecture, leading to a complete failure to differentiate between classes.

The training history further elucidates the problem, showing a sharp decline in validation accuracy over epochs. Despite achieving a high training accuracy of approximately 99.71% by the fourth epoch, the model's performance on the validation set deteriorates rapidly, dropping to only 76.43% accuracy. Such a drastic drop in validation accuracy indicates a significant failure of the model to generalize to unseen data, undermining its utility for practical applications.

Given the erroneous predictions and poor generalization observed in Model-3, it is evident that fundamental issues exist with either the model's architecture, data preprocessing, or training process. Further investigation and refinement are necessary to address these issues and develop a model capable of accurately classifying melanoma cases.

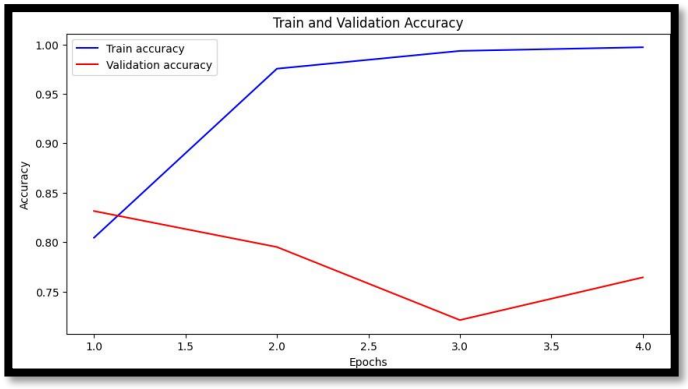


Figure 16 : Model -3 Training & Validation Accuracy

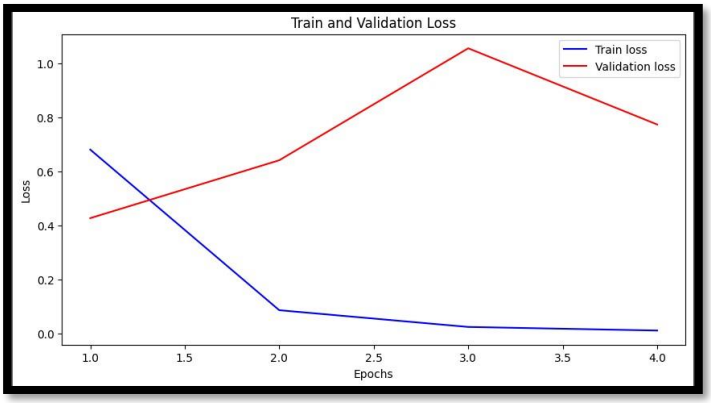


Figure 17 : Model -3 Training & Validation Loss

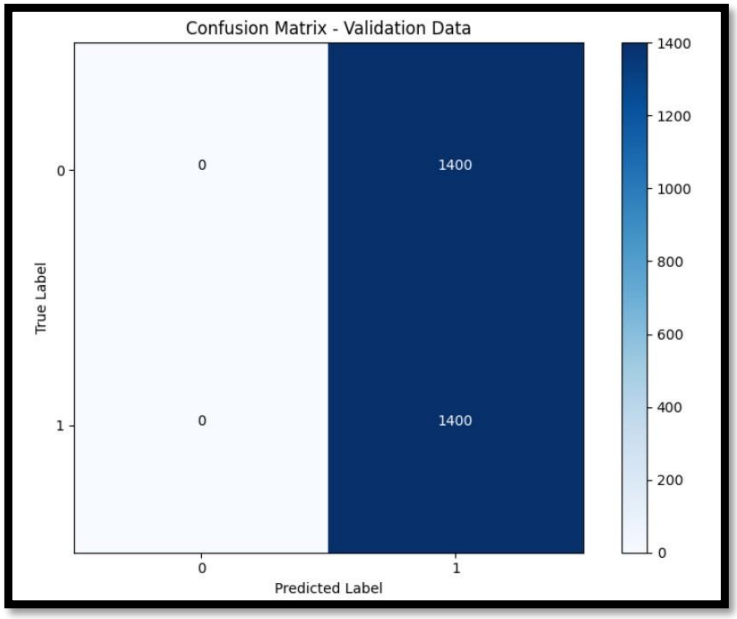


Figure 18 : Model -3 Confusion Matrix

Model-4 (Best Model):

The confusion matrix for Model-4 demonstrates its exceptional performance in correctly classifying melanoma cases. With only one false positive and 43 false negatives, the model achieves an impressive balance between sensitivity and specificity, crucial for accurate medical diagnosis. Specifically, the model correctly identifies 1357 out of 1400 melanoma cases, minimizing the risk of misdiagnosis, which is particularly critical in medical contexts.

The training history of Model-4 reveals a consistent improvement in both training and validation accuracy over epochs. Starting with an initial accuracy of approximately 93.90%, the model undergoes progressive refinement, achieving a remarkable validation accuracy of 98.54% by the sixth epoch. This steady improvement underscores the effectiveness of the model's architecture and training process in learning discriminative features from the data while avoiding overfitting.

Furthermore, the model's training converges smoothly, with the loss steadily decreasing across epochs. This convergence, coupled with the sustained increase in accuracy, indicates the model's robustness and reliability in capturing essential patterns within the data.

In conclusion, Model-4 emerges as a highly effective and reliable model for melanoma classification, characterized by its exceptional performance in accurately identifying melanoma cases while minimizing false positives. Its consistent improvement and smooth convergence during training underscore its potential for practical deployment in clinical settings, where accurate and timely diagnosis is paramount.

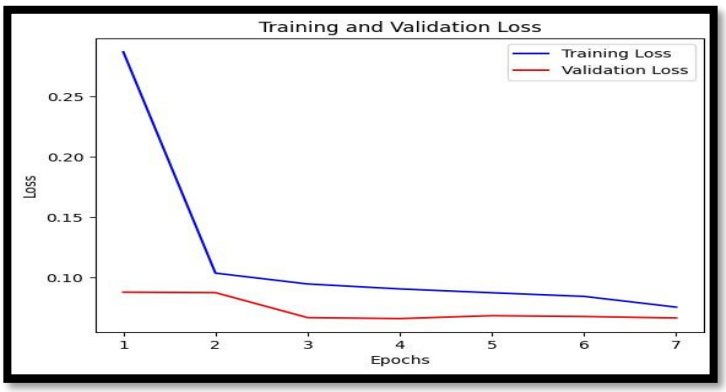


Figure 19 : Model -4 Training & Validation Loss

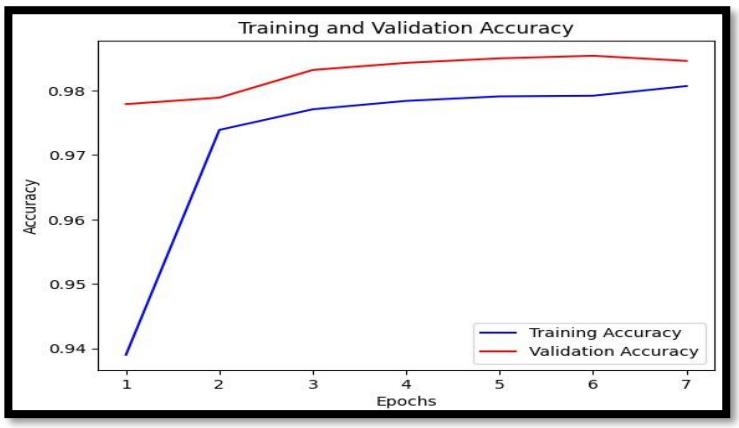


Figure 20 : Model -4 Training & Validation Accuracy

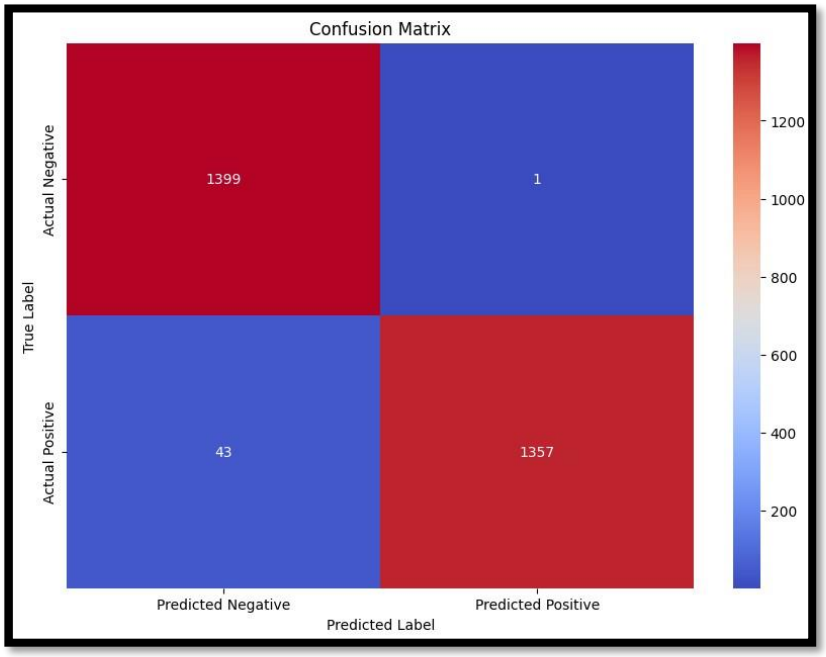


Figure 21 : Model -4 Confusion Matrix

Model-5 (XGBoost Model for Metadata):

This model achieved exceptionally high accuracy on the training set (99.98%) but performed poorly on the validation set (50.82%). The confusion matrix for the training set shows that out of 11200 samples, only 2 were misclassified. However, on the validation set, the model failed to classify any samples correctly for the positive class, indicating poor generalization. This suggests that the model may have overfit the training data and failed to capture the underlying patterns in the validation set.

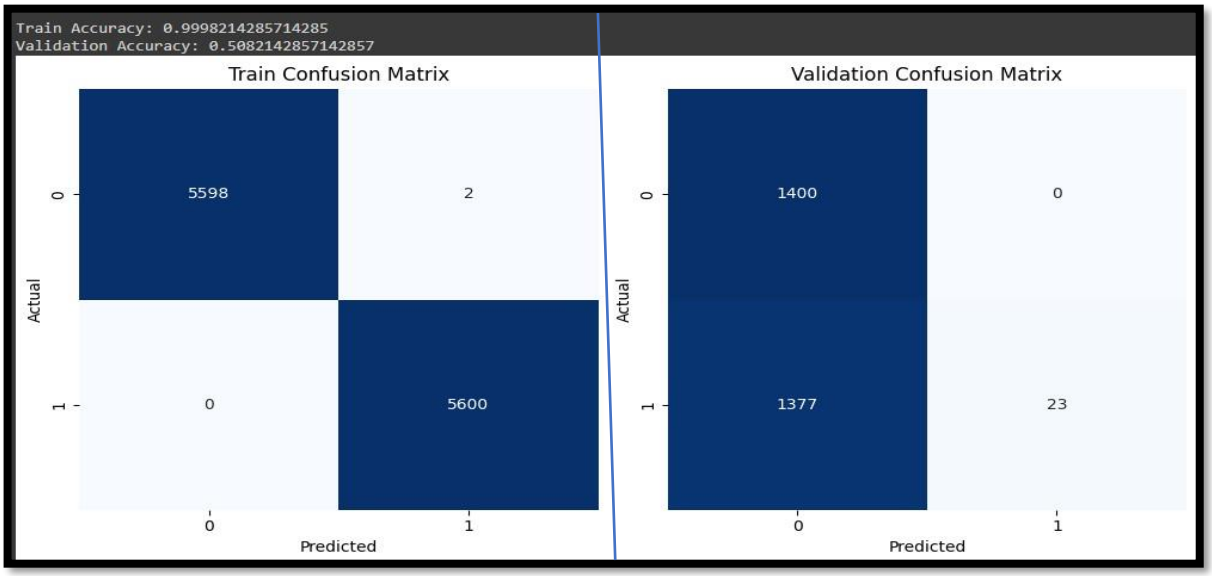


Figure 22: Model -5 Confusion Matrix

In the medical field, the type-2 error (false negative) is particularly critical as it involves failing to identify a condition that is actually present. Therefore, models with high sensitivity (true positive rate) are preferred to minimize the risk of missing positive cases.
Overall, Model-4 appears to be the best-performing model among the ones evaluated, with high accuracy and low misclassification rates. It demonstrates robust performance and holds promise for practical application in melanoma classification.

EXPLAINABLE AI (XAI) :

GRAD – CAM ANALYSIS (Gradient-weighted Class Activation Mapping) :

Grad-CAM (Gradient-weighted Class Activation Mapping) is a powerful technique used for visualizing and interpreting the decisions made by convolutional neural networks (CNNs) in image classification tasks. It provides insight into the regions of an input image that contribute the most to the model's prediction, thereby enhancing the model's interpretability.
For our melanoma classification task, we applied Grad-CAM analysis to Model-4, which demonstrated superior performance in accurately identifying melanoma cases while minimizing false positives. By visualizing the regions of interest within the input images that influenced the model's decision-making process, we aimed to gain deeper insights into the features relevant to melanoma classification.
The Grad-CAM analysis revealed that Model-4 primarily focused on specific features and patterns within the input images indicative of melanoma. These features typically included irregularities in skin lesions, asymmetrical shapes, uneven borders, and variations in colour and texture, which are clinically recognized indicators of melanoma. By highlighting these regions of interest, Grad-CAM provided valuable visual cues to dermatologists and clinicians, aiding them in understanding the rationale behind the model's predictions.

For Correctly Classified Images :

Image 1:

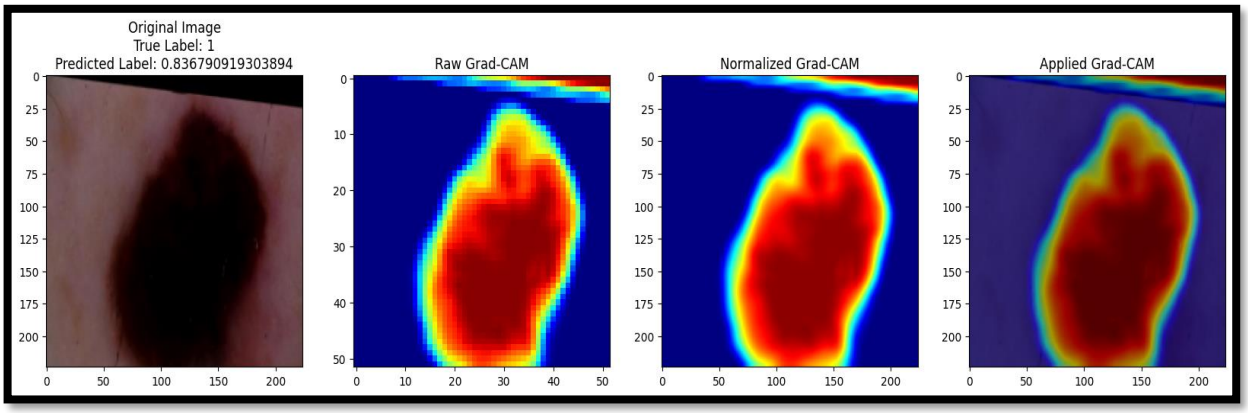


Figure 23 : Grad-Cam Image For Correctly Classifying - 1

- The true label is 1 (melanoma), and the model predicted a high probability of 0.8367 for the same class, indicating a correct prediction.
- The Grad-CAM heatmaps clearly highlight the region of the skin lesion, suggesting that the model is focusing on the relevant area to make its prediction.
- The bright red and yellow regions in the heatmaps correspond to the areas of the input image that had the highest influence on the model's prediction of melanoma.

Image 2:

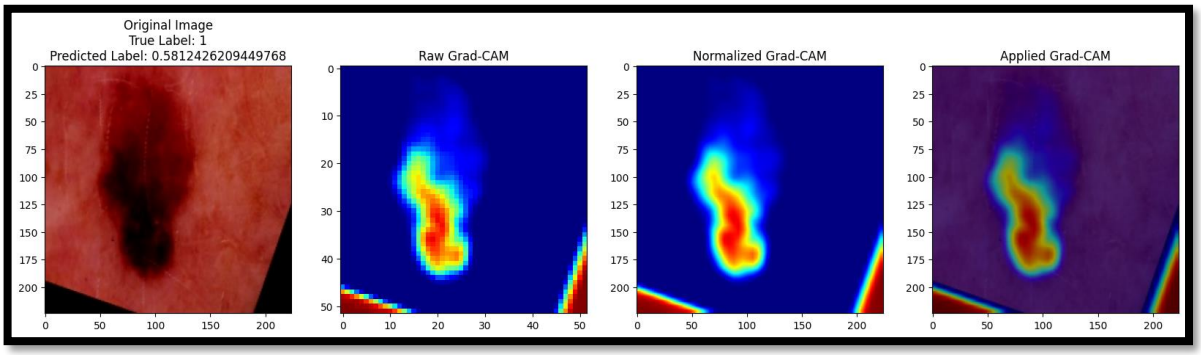


Figure 24 : Grad-Cam Image For Correctly Classifying - 2

- Again, the true label is 1 (melanoma), and the model predicted a probability of 0.581 for the same class, which is above the typical threshold of 0.5 for a positive prediction.
- The Grad-CAM heatmaps once more highlight the region of the skin lesion, indicating that the model is paying attention to the relevant area.
- The elongated and winding shape of the highlighted region corresponds well with the shape of the lesion in the input image.

Both Image 1 and Image 2 represent cases where the model correctly predicted the presence of melanoma, and the Grad-CAM visualizations provide insights into the regions of the input images that were most influential in the model's decision-making process.

Image 3:

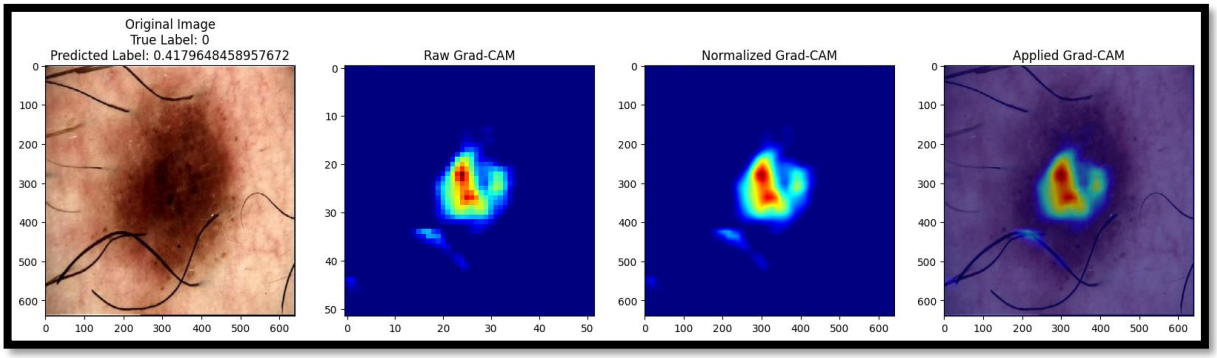


Figure 25 : Grad-Cam Image For Correctly Classifying - 3

- In this case, the true label is 0 (no melanoma), but the model predicted a probability of 0.417 for the melanoma class, which is below the typical threshold of 0.5 for a positive prediction.
- This suggests that the model correctly identified the absence of melanoma in the image and did not focus on regions indicative of melanoma. Though it seems like something on the skin but the model was successful in correctly classifying it as benign.
- The correct classification of Image 3 indicates that the model effectively recognized instances where melanoma was not present, contributing to its overall accuracy.

The Grad-CAM visualizations provide valuable insights into the model's decision-making process, helping to identify cases where the model is attending to the correct regions (Images 1 and 2) and cases where it may be misled or focusing on irrelevant areas (Image 3). This information can be useful for further model improvement, retraining, or adjusting the decision thresholds.

For Misclassified Images :

Image 5 :

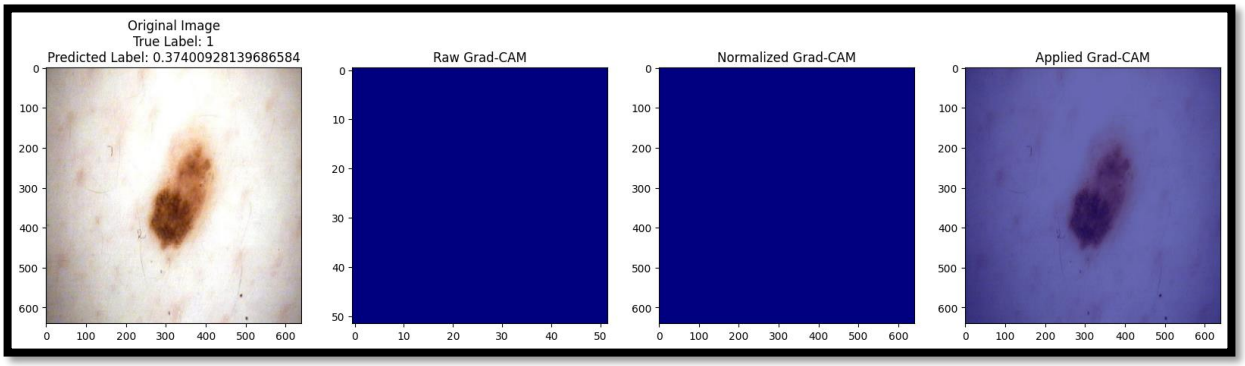


Figure 26: Grad-Cam Image For Misclassified Image -1

- The true label is 1 (melanoma), but the model predicted a probability of only 0.374 for the melanoma class, which is below the typical threshold of 0.5 for a positive prediction. This indicates a misclassification.
- The Grad-CAM heatmaps do not highlight any specific region of the input image, suggesting that the model did not focus on any particular area to make its prediction.
- This could be due to the lesion or melanoma being less visually apparent or having features that the model did not adequately learn during training.

Image 6:

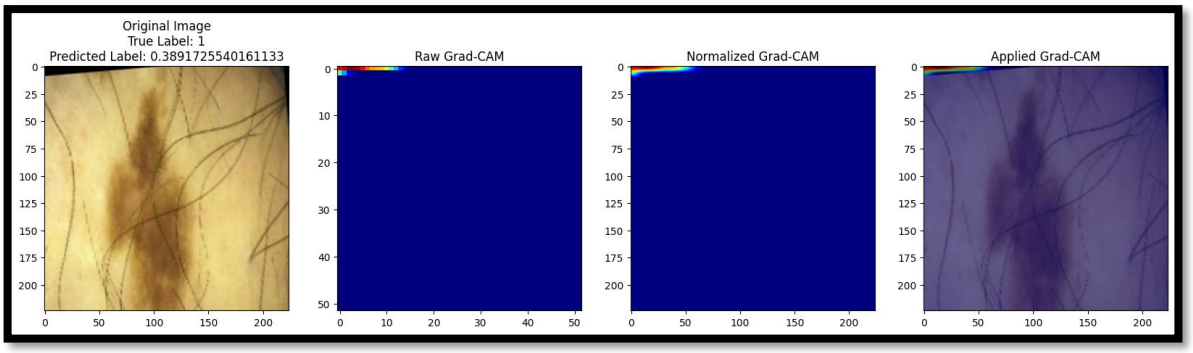


Figure 27 : Grad-Cam Image For Misclassified Image -2

- Again, the true label is 1 (melanoma), but the model predicted a probability of 0.389 for the melanoma class, which is below the typical threshold of 0.5 for a positive prediction, indicating a misclassification.
- The Grad-CAM heatmaps do not highlight any specific region of the input image, suggesting that the model did not focus on any particular area to make its prediction.
- Similar to Image 1, the lesion or melanoma features may not have been prominent enough or the model may have struggled to learn the relevant patterns during training.

In both cases of misclassification (Image 1 and Image 2), the Grad-CAM visualizations do not reveal any specific regions of the input images that the model focused on to make its predictions. This could be an indication that the model did not effectively learn the relevant features or patterns associated with melanoma in these particular cases.

These misclassification examples highlight the importance of further improving the model's performance, either through additional training data, architectural modifications, or different training strategies. The Grad-CAM visualizations provide valuable insights into the model's decision-making process and can help identify areas for improvement.

It's worth noting that the absence of highlighted regions in the Grad-CAM visualizations does not necessarily mean that the model completely ignored the input images. It could be that the model relied on more subtle or distributed patterns that are not easily visualized through Grad-CAM.

Nonetheless, these cases of misclassification and the lack of clear focal regions in the Grad-CAM visualizations suggest that further model refinement or additional training may be necessary to improve the model's performance on such challenging examples.

LIME Explainer :

LIME (Local Interpretable Model-agnostic Explanations) is a technique used to explain the predictions of any machine learning model by approximating it locally with an interpretable model. It works by perturbing the input data and observing the changes in the predictions, thereby creating a simpler, interpretable model (such as a linear regression or decision tree) that mimics the behaviour of the complex model in the vicinity of a specific prediction. This local surrogate model provides insights into which features contribute most to a particular prediction, making the black-box nature of machine learning models more transparent and understandable to humans. LIME is particularly useful in scenarios where understanding individual predictions is critical, such as in healthcare, finance, and regulatory applications.

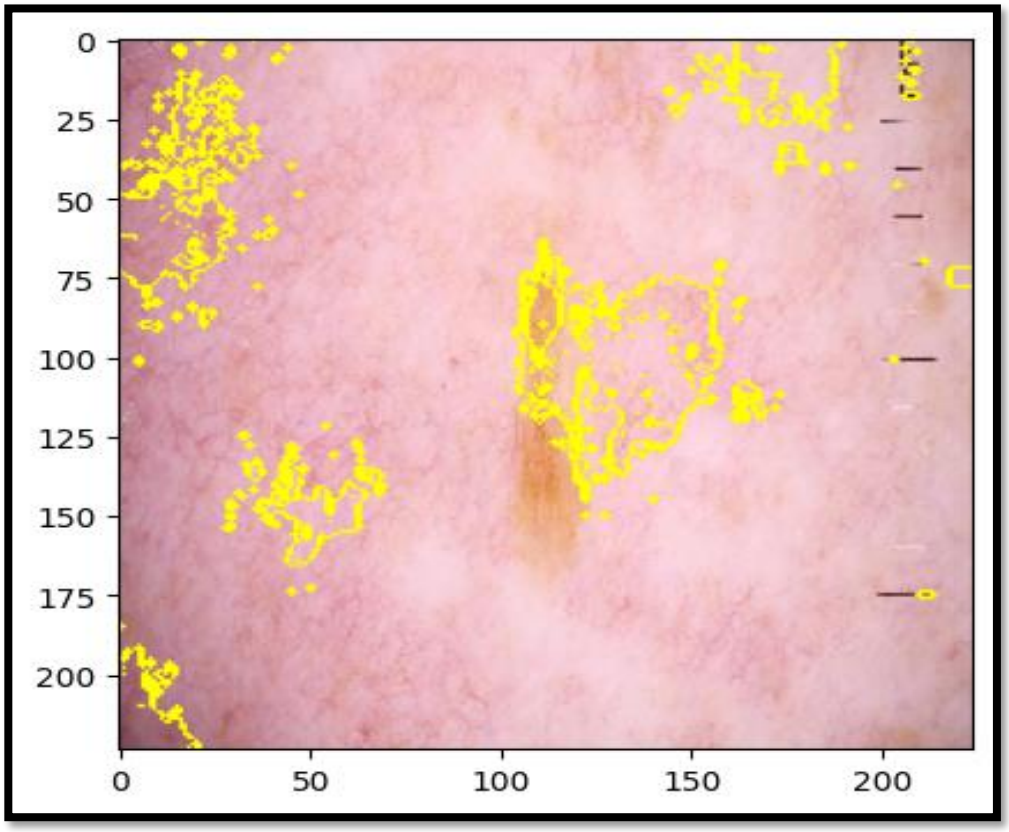


Figure 28 : Lime Explainer On Model-4

In this particular image, the x and y axes represent the pixel coordinates of the input image. The yellow regions highlight the areas of the image that had the greatest positive influence on the model's prediction. In other words, these are the regions that the model focused on the most when making its classification decision.

However, based on the distribution of the highlighted areas, it seems that the model is focusing on specific patterns or features within the image to make its classification.

It's worth noting that LIME provides local explanations, meaning it explains the model's prediction for a specific instance rather than providing a global understanding of the model's behaviour across all instances.

Shap Explainer on XGBOOST Model :

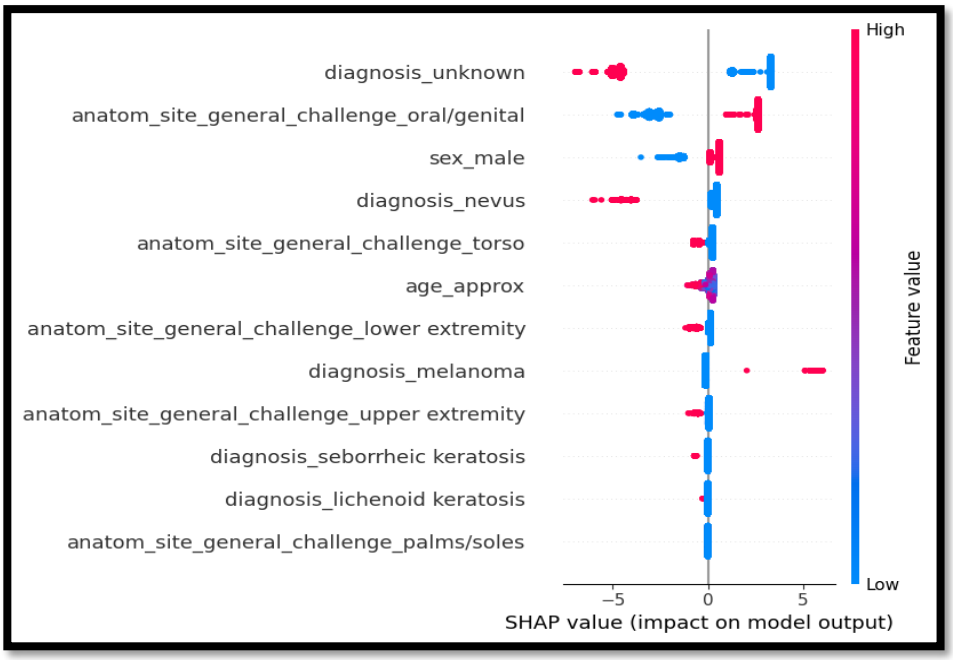


Figure 29 : Shap Analysis -1

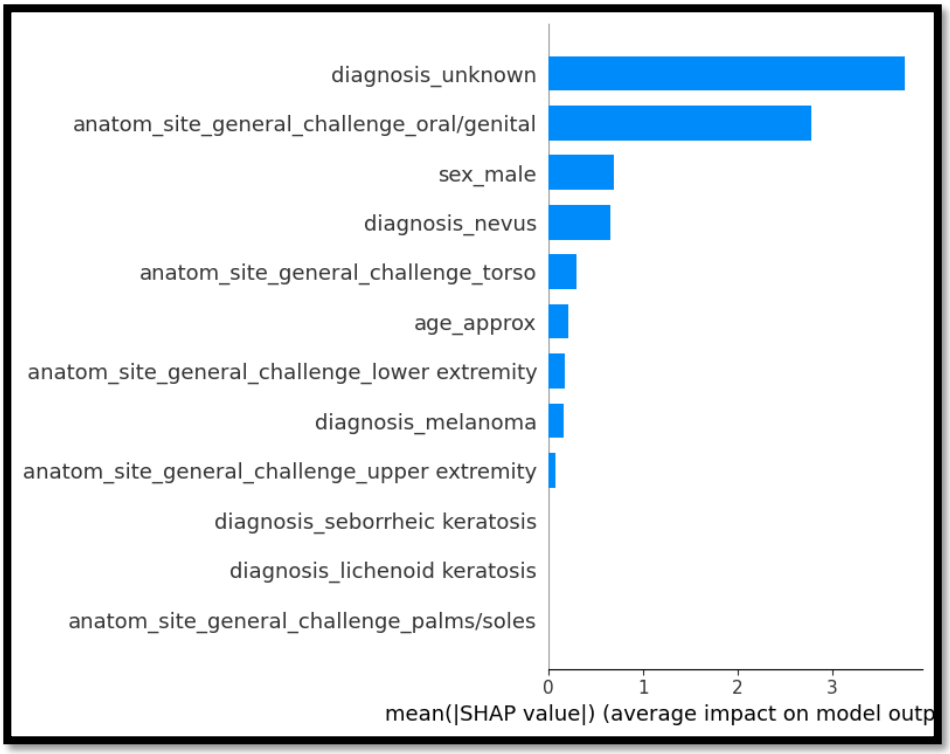


Figure 30 : Shap Analysis - 2

These images provide insights into the importance of different features for the XGBoost model trained on the metadata for the melanoma detection task.

Shap Analysis -1 show the SHAP (Shapley Additive Explanations) values for each feature, which represent the impact of each feature on the model's output. Features with positive SHAP values contribute to a higher probability of the positive class (e.g., melanoma), while negative SHAP values contribute to a lower probability.

In both images, the feature 'diagnosis_unknown' has the highest positive SHAP value, indicating that when this feature is present, it strongly increases the likelihood of the positive class prediction. Similarly, features like 'anatom_site_general_challenge_oral/genital' and 'sex_male' have positive SHAP values, suggesting they contribute to higher positive class probabilities.

On the other hand, features like 'diagnosis_lichenoid keratosis' and 'anatom_site_general_challenge_palms/soles' have negative SHAP values, meaning they decrease the probability of the positive class prediction.

Shap Analysis -2 shows the mean absolute SHAP value for each feature, which represents the average impact of that feature on the model's output across all instances. Higher values indicate that the feature is more important for the model's predictions.

By analysing these SHAP plots, we can identify the most influential features for the XGBoost model's predictions and potentially gain insights into the model's decision-making process. This can be useful for model interpretation, feature selection, and potential model improvements.

10. DISCUSSION

In this project, we aimed to develop and interpret deep learning models for melanoma detection using various techniques. Four convolutional neural network (CNN) models were evaluated, with Model-4 showing the best performance, evidenced by its confusion matrix and training history, indicating high accuracy and effective learning. However, some models exhibited overfitting, where the training accuracy increased while validation accuracy decreased, highlighting the need for careful tuning of hyperparameters and regularization techniques. To understand the decision-making process of our models, we employed explainable AI methods such as Grad-CAM and LIME. Grad-CAM visualizations revealed that correctly classified images had focused attention on relevant regions of the lesions, whereas misclassified images often showed the model's focus on irrelevant areas, suggesting potential areas for improvement in model training and data quality. Additionally, SHAP values from the XGBoost model provided insights into the importance of metadata features, identifying 'diagnosis_unknown' and 'anatom_site_general_challenge_oral/genital' as significant contributors to positive class predictions. These explainable AI techniques not only validated the model's focus on clinically relevant features but also highlighted areas where the model might be misled, guiding further refinements. This comprehensive analysis underscores the importance of model interpretability in medical AI applications, ensuring that the models are not only accurate but also reliable and transparent in their predictions.

11. CONCLUSION

Throughout this project, we embarked on a comprehensive journey aimed at developing robust and interpretable models for melanoma detection. Leveraging a variety of deep learning architectures, including convolutional neural networks (CNNs) and gradient boosting models like XGBoost, we endeavoured to achieve accurate predictions while ensuring model interpretability. Despite encountering challenges such as overfitting and model misclassification, particularly evident in Model-2 and Model-5, our efforts yielded promising outcomes. The application of explainable AI techniques, such as Grad-CAM and LIME, provided invaluable insights into model decision-making processes, elucidating regions of interest and potential areas for model improvement. Notably, the analysis of SHAP values from the XGBoost model shed light on the significance of metadata features in melanoma classification, facilitating a deeper understanding of feature importance. Moving forward, addressing issues like overfitting through advanced regularization techniques and architectural modifications, as observed in Model-4, will be paramount. Moreover, integrating diverse datasets and domain knowledge, along with exploring multimodal data fusion, holds promise for enhancing model robustness and generalization. As we venture into future endeavours, real-time deployment of AI solutions in clinical settings and continual refinement of models will be essential steps toward realizing impactful advancements in melanoma detection and healthcare.

12. APPENDIX

Double-click (or enter) to edit

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] # Step 2: Import necessary libraries
    import pandas as pd
    import os
```

```
# Step 3: Define the paths to your CSV and images
csv_path = '/content/drive/MyDrive/6 Months Project/train.csv'  # Update with your actual path
images_folder_path = '/content/drive/MyDrive/6 Months Project/train'  # Update with your actual path

# Step 4: Load the train.csv file
train_df = pd.read_csv(csv_path)

# Step 5: Verify the data structure
print(train_df.head())

# Step 6: Add image paths to the DataFrame
def get_image_path(image_name):
    return os.path.join(images_folder_path, f'{image_name}.jpg')  # Assuming images are in .jpg format

train_df['image_path'] = train_df['image_name'].apply(get_image_path)

# Step 7: Verify that image paths are correctly added
print(train_df.head())

# Save the updated DataFrame to a new CSV file (optional)
train_df.to_csv('/content/drive/My Drive/updated_train.csv', index=False)
```

```
     image_name  patient_id     sex  age_approx anatom_site_general_challenge  \
0  ISIC_2637011  IP_7279968    male        45.0                       head/neck
1  ISIC_0015719  IP_3075186  female        45.0                 upper extremity
2  ISIC_0052212  IP_2842074  female        50.0                 lower extremity
3  ISIC_0068279  IP_6890425  female        45.0                       head/neck
4  ISIC_0074268  IP_8723313  female        55.0                 upper extremity

  diagnosis benign_malignant  target
0   unknown           benign       0
1   unknown           benign       0
2     nevus           benign       0
3   unknown           benign       0
4   unknown           benign       0
     image_name  patient_id     sex  age_approx anatom_site_general_challenge  \
0  ISIC_2637011  IP_7279968    male        45.0                       head/neck
```

```
[ ] import matplotlib.pyplot as plt

    # Check for target class imbalance
    class_counts = train_df['target'].value_counts()
    print("Class counts:")
    print(class_counts)

    # Calculate the percentage of each class
    class_percentages = train_df['target'].value_counts(normalize=True) * 100
    print("\nClass percentages:")
    print(class_percentages)

    # Plot a pie chart
    plt.figure(figsize=(8, 8))
    plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=140)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.title('Target Class Distribution')
    plt.show()
```

```
Class counts:
target
0    32542
1      584
Name: count, dtype: int64

Class percentages:
target
0    98.237034
1     1.762966
Name: proportion, dtype: float64
```

```
Insert code cell below
Ctrl+M B
.colab import drive
ve.mount('/content/drive')

import pandas as pd
import numpy as np
import os
import cv2
from tqdm import tqdm
import matplotlib.pyplot as plt
import imgaug.augmenters as iaa

# Global settings
SEED = 42
DIM = 224
AUGMENTATION = True

# Load dataset
csv_path = '/content/drive/MyDrive/6 Months Project/train.csv'
images_folder_path = '/content/drive/MyDrive/6 Months Project/train'
train_df = pd.read_csv(csv_path)

# Add image paths to DataFrame
def get_image_path(image_name):
    return os.path.join(images_folder_path, f'{image_name}.jpg')

train_df['image_path'] = train_df['image_name'].apply(get_image_path)

# Function to load and preprocess images
def load_image(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (DIM, DIM))
    return image

# Separate melanoma images and labels
melanoma_df = train_df[train_df['target'] == 1]
melanoma_images_paths = melanoma_df['image_path'].tolist()
melanoma_labels = melanoma_df['target'].tolist()

# Define augmentation sequence
seq = iaa.Sequential([
    iaa.Fliplr(0.5),
    iaa.Crop(percent=(0, 0.1)),
    iaa.LinearContrast((0.75, 1.5)),
    iaa.Multiply((0.8, 1.2), per_channel=0.2),
    iaa.Affine(
        scale={'x': (0.8, 1.2), 'y': (0.8, 1.2)},
        translate_percent={'x': (-0.2, 0.2), 'y': (-0.2, 0.2)},
        rotate=(-25, 25),
        shear=(-8, 8)
```

```
# Create directory for augmented images
augmented_images_folder = '/content/drive/MyDrive/6 Months Project/augmented_images_v2'
os.makedirs(augmented_images_folder, exist_ok=True)

# Augment melanoma images in batches and save them
batch_size = 100
target_count = 32542
current_count = len(melanoma_images_paths)
augmented_image_paths = []

def augment_and_save_images(batch_paths, seq, start_index):
    augmented_images = []
    for img_path in batch_paths:
        if os.path.exists(img_path):
            image = load_image(img_path)
            augmented_images.append(seq(image=image))
            augmented_images.append(seq(image=image))
    augmented_images = np.array(augmented_images, dtype=np.float32)

    batch_image_paths = []
    for i, img in enumerate(augmented_images):
        img_name = f'augmented_image_v2_{start_index + i}.jpg'
        img_path = os.path.join(augmented_images_folder, img_name)
        cv2.imwrite(img_path, cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
        batch_image_paths.append(img_path)

    return batch_image_paths

start_index = 0
while current_count < target_count:
    for batch_start in tqdm(range(0, len(melanoma_images_paths), batch_size)):
        batch_paths = melanoma_images_paths[batch_start:batch_start + batch_size]
        batch_augmented_paths = augment_and_save_images(batch_paths, seq, start_index)
        augmented_image_paths.extend(batch_augmented_paths)
        start_index += len(batch_augmented_paths)
        current_count += len(batch_augmented_paths)
        if current_count >= target_count:
            break

# Create DataFrame for augmented images
augmented_df = pd.DataFrame({
    'image_name': [os.path.splitext(os.path.basename(p))[0] for p in augmented_image_paths],
    'patient_id': [None] * len(augmented_image_paths),
    'sex': [None] * len(augmented_image_paths),
    'age_approx': [None] * len(augmented_image_paths),
    'anatom_site_general_challenge': [None] * len(augmented_image_paths),
    'diagnosis': [None] * len(augmented_image_paths),
    'benign_malignant': ['malignant'] * len(augmented_image_paths),
    'target': [1] * len(augmented_image_paths),
    'image_path': augmented_image_paths
```

```python
# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

# Load the balanced CSV
balanced_df = pd.read_csv(balanced_csv_path)

# Shuffle the dataframe
balanced_df = balanced_df.sample(frac=1).reset_index(drop=True)

# Create subfolders with 14k images each (7k per class)
images_per_class = 7000
subfolder_size = 14000
subfolder_counter = 2  # Start from subfolder_2
class_0_counter = 0
class_1_counter = 0

current_subfolder_images = []
current_subfolder_labels = []

for index, row in balanced_df.iterrows():
    img_path = row['image_path']
    target = row['target']

    if (target == 0 and class_0_counter < images_per_class) or (target == 1 and class_1_counter < images_per_class):
        current_subfolder_images.append(img_path)
        current_subfolder_labels.append(row)

        if target == 0:
            class_0_counter += 1
        else:
            class_1_counter += 1

    # Check if we need to create a new subfolder
    if len(current_subfolder_images) >= subfolder_size:
        subfolder_path = os.path.join(output_folder, f'subfolder_{subfolder_counter}')
        os.makedirs(subfolder_path, exist_ok=True)

        # Save images and CSV
        subfolder_df = pd.DataFrame(current_subfolder_labels, columns=['image_path', 'target'])
        subfolder_df.to_csv(os.path.join(subfolder_path, 'balanced_train.csv'), index=False)

        for img in current_subfolder_images:
            try:
                shutil.move(img, os.path.join(subfolder_path, os.path.basename(img)))
            except FileNotFoundError:
                print(f"File not found and skipped: {img}")

        # Print the distribution of the current subfolder
        print(f'Distribution in subfolder_{subfolder_counter}:')
        print(subfolder_df['target'].value_counts(normalize=True) * 100)
```

```python
import os
import pandas as pd
import matplotlib.pyplot as plt

# Define paths
output_folder = '/content/drive/MyDrive/6 Months Project/subfolders'

# Get list of subfolders
subfolders = [f for f in os.listdir(output_folder) if os.path.isdir(os.path.join(output_folder, f))]

subfolder_summaries = []

for subfolder in subfolders:
    subfolder_path = os.path.join(output_folder, subfolder)
    csv_path = os.path.join(subfolder_path, 'balanced_train.csv')

    if os.path.exists(csv_path):
        subfolder_df = pd.read_csv(csv_path)
        class_counts = subfolder_df['target'].value_counts()
        class_percentages = subfolder_df['target'].value_counts(normalize=True) * 100

        subfolder_summary = {
            'subfolder': subfolder,
            'total_images': len(subfolder_df),
            'class_0_count': class_counts.get(0, 0),
            'class_1_count': class_counts.get(1, 0),
            'class_0_percentage': class_percentages.get(0, 0),
            'class_1_percentage': class_percentages.get(1, 0)
        }
        subfolder_summaries.append(subfolder_summary)

# Convert summaries to DataFrame
summary_df = pd.DataFrame(subfolder_summaries)

# Plot class distributions vertically
num_plots = len(subfolders)
fig, axes = plt.subplots(num_plots, 1, figsize=(8, 5 * num_plots))

for ax, subfolder_summary in zip(axes, subfolder_summaries):
    labels = ['Class 0', 'Class 1']
    sizes = [subfolder_summary['class_0_percentage'], subfolder_summary['class_1_percentage']]
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=['#66b3ff','#ff9999'])
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    ax.set_title(subfolder_summary['subfolder'])

plt.tight_layout()
plt.show()
```

```python
import os
import pandas as pd

# Define paths
output_folder = '/content/drive/MyDrive/6 Months Project/subfolders'

# Get list of subfolders
subfolders = [f for f in os.listdir(output_folder) if os.path.isdir(os.path.join(output_folder, f))]

for subfolder in subfolders:
    subfolder_path = os.path.join(output_folder, subfolder)
    csv_path = os.path.join(subfolder_path, 'balanced_train.csv')

    if os.path.exists(csv_path):
        subfolder_df = pd.read_csv(csv_path)
        num_images = len(os.listdir(subfolder_path)) - 1  # Subtract 1 for the CSV file

        print(f"Subfolder: {subfolder}")
        print(f"Number of images in folder: {num_images}")
        print(f"Number of rows in CSV: {len(subfolder_df)}")
        print()
```

```python
import numpy as np
import pandas as pd
import tensorflow as tf
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Concatenate, Conv2D, MaxPooling2D, GlobalAveragePooling2D, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

# Define paths
train_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/train_drive.csv"
validation_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/validation_drive.csv"

# Load data
train_df = pd.read_csv(train_csv_path)
validation_df = pd.read_csv(validation_csv_path)

# Extract metadata and labels
metadata_features = ['sex', 'age_approx', 'anatom_site_general_challenge', 'diagnosis']
train_metadata = train_df[metadata_features]
validation_metadata = validation_df[metadata_features]

# Convert categorical features to numerical
train_metadata.loc[:, 'sex'] = train_metadata['sex'].map({'male': 0, 'female': 1})
validation_metadata.loc[:, 'sex'] = validation_metadata['sex'].map({'male': 0, 'female': 1})
train_metadata = pd.get_dummies(train_metadata, columns=['anatom_site_general_challenge', 'diagnosis'])
validation_metadata = pd.get_dummies(validation_metadata, columns=['anatom_site_general_challenge', 'diagnosis'])

# Align columns in train and validation sets (handle missing columns after get_dummies)
train_metadata, validation_metadata = train_metadata.align(validation_metadata, join='outer', axis=1, fill_value=0)

# Convert metadata to float32
train_metadata = train_metadata.astype(np.float32)
validation_metadata = validation_metadata.astype(np.float32)

# Convert labels to float32
train_labels = train_df['target'].astype(np.float32).values
validation_labels = validation_df['target'].astype(np.float32).values

# Image data generators
train_datagen = ImageDataGenerator(rescale=0.255)
validation_datagen = ImageDataGenerator(rescale=0.255)

# Function to create data generators
def create_generator(datagen, df, metadata, labels):
    def generator():
        for img_path, meta, label in zip(df['image_path'], metadata.values, labels):
            img = Image.open(img_path)
            img = img.resize((224, 224))
```

```python
# Define model
inp_image = Input(name='image', shape=(224, 224, 3))
inp_metadata = Input(name='metadata', shape=(train_metadata.shape[1],))

def inception_block(x, base_channels=16):
    conv_1 = Conv2D(base_channels * 2, 1, 1, activation='relu')(x)
    conv_2_a = Conv2D(base_channels * 2, 1, 1, activation='relu')(x)
    conv_2_b = Conv2D(base_channels * 2, 3, 1, padding='same', activation='relu')(conv_2_a)
    conv_3_a = Conv2D(base_channels, 1, 1, activation='relu')(x)
    conv_3_b = Conv2D(base_channels, 5, 1, padding='same', activation='relu')(conv_3_a)
    mpool_4_a = MaxPooling2D(3, 1, padding='same')(x)
    conv_4_b = Conv2D(base_channels, 1, 1, activation='relu')(mpool_4_a)
    return Concatenate(axis=-1)([conv_1, conv_2_b, conv_3_b, conv_4_b])

block_1 = inception_block(inp_image)
#block_2 = inception_block(block_1, base_channels=8)
#block_3 = inception_block(block_2, base_channels=8)

gap = GlobalAveragePooling2D()(block_1)
flatn = Flatten()(gap)

# Metadata branch
metadata_dense = Dense(32, activation='relu')(inp_metadata)
metadata_output = Dense(16, activation='relu')(metadata_dense)

# Concatenate image and metadata branches
concat = Concatenate(axis=-1)([flatn, metadata_output])
output = Dense(1, activation="sigmoid")(concat)

model = Model(inputs=[inp_image, inp_metadata], outputs=output)

# Print model summary
model.summary()

# Compile model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint("/content/drive/MyDrive/SIIMS Melanoma/best_model.h5",
                                   monitor='val_loss', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-6)

# Train the model
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,
    callbacks=[early_stopping, model_checkpoint, reduce_lr]
```

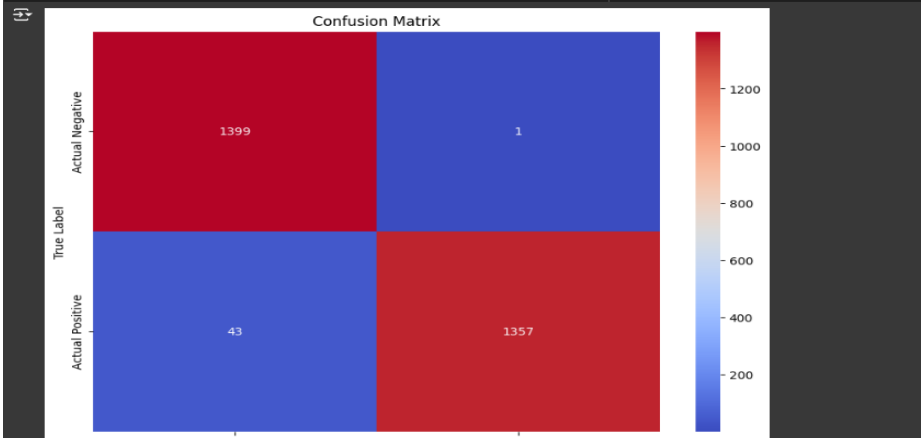
```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 image (InputLayer)          [(None, 224, 224, 3)]     0         []

 conv2d_7 (Conv2D)           (None, 224, 224, 32)      128       ['image[0][0]']

 conv2d_9 (Conv2D)           (None, 224, 224, 16)      64        ['image[0][0]']

 max_pooling2d_1 (MaxPoolin  (None, 224, 224, 3)       0         ['image[0][0]']
 g2D)

 conv2d_6 (Conv2D)           (None, 224, 224, 32)      128       ['image[0][0]']

 conv2d_8 (Conv2D)           (None, 224, 224, 32)      9248      ['conv2d_7[0][0]']

 conv2d_10 (Conv2D)          (None, 224, 224, 16)      6416      ['conv2d_9[0][0]']

 conv2d_11 (Conv2D)          (None, 224, 224, 16)      64        ['max_pooling2d_1[0][0]']

 concatenate_2 (Concatenate  (None, 224, 224, 96)      0         ['conv2d_6[0][0]',
 )                                                               'conv2d_8[0][0]',
                                                                 'conv2d_10[0][0]',
                                                                 'conv2d_11[0][0]']

 metadata (InputLayer)       [(None, 15)]              0         []

 global_average_pooling2d_1  (None, 96)                0         ['concatenate_2[0][0]']
  (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 32)                512       ['metadata[0][0]']

 flatten_1 (Flatten)         (None, 96)                0         ['global_average_pooling2d_1[0
                                                                 ][0]']

 dense_4 (Dense)             (None, 16)                528       ['dense_3[0][0]']

 concatenate_3 (Concatenate  (None, 112)               0         ['flatten_1[0][0]',
 )                                                               'dense_4[0][0]']

 dense_5 (Dense)             (None, 1)                 113       ['concatenate_3[0][0]']

==================================================================================================
Total params: 17201 (67.19 KB)
Trainable params: 17201 (67.19 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
Epoch 1/10
350/350 [==============================] - 4103s 12s/step - loss: 0.6814 - accuracy: 0.8044 - val_loss: 0.4276 - val_accuracy: 0.8314 - lr: 0.0010
Epoch 2/10
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
350/350 [==============================] - 3337s 10s/step - loss: 0.0867 - accuracy: 0.9754 - val_loss: 0.6420 - val_accuracy: 0.7950 - lr: 0.0010
```

```python
# Function to create data generators
def create_image_generator(datagen, df, labels):
    def generator():
        for img_path, label in zip(df['image_path'], labels):
            img = Image.open(img_path)
            img = img.resize((224, 224))
            img_array = img_to_array(img)
            img_array = datagen.random_transform(img_array)
            yield (img_array, label)
    return generator

train_image_generator = tf.data.Dataset.from_generator(
    create_image_generator(train_datagen, train_df, train_labels),
    output_signature=(
        tf.TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(), dtype=tf.float32)
    )
).batch(32)

validation_image_generator = tf.data.Dataset.from_generator(
    create_image_generator(validation_datagen, validation_df, validation_labels),
    output_signature=(
        tf.TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(), dtype=tf.float32)
    )
).batch(32)

# Define image model
inp_image = Input(name='image', shape=(224, 224, 3))
x = Conv2D(32, 3, activation='relu')(inp_image)
x = MaxPooling2D()(x)
x = Conv2D(64, 3, activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(128, 3, activation='relu')(x)
x = MaxPooling2D()(x)
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)  # Dropout layer to reduce overfitting
output_image = Dense(1, activation='sigmoid')(x)

image_model = Model(inputs=inp_image, outputs=output_image)

# Print image model summary
image_model.summary()

# Compile image model
optimizer_image = Adam(learning_rate=0.001)
image_model.compile(optimizer=optimizer_image, loss='binary_crossentropy', metrics=['accuracy'])
```

```python
output_image = Dense(1, activation='sigmoid')(x)

image_model = Model(inputs=inp_image, outputs=output_image)

# Print image model summary
image_model.summary()

# Compile image model
optimizer_image = Adam(learning_rate=0.001)
image_model.compile(optimizer=optimizer_image, loss='binary_crossentropy', metrics=['accuracy'])

# Define callbacks for image model
early_stopping_image = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint_image = ModelCheckpoint("/content/drive/MyDrive/SIIMS Melanoma/best_model_image.h5",
                                         monitor='val_loss', save_best_only=True)
reduce_lr_image = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-6)

# Train the image model
history_image = image_model.fit(
    train_image_generator,
    validation_data=validation_image_generator,
    epochs=10,
    callbacks=[early_stopping_image, model_checkpoint_image, reduce_lr_image]
)

# Save the image model
image_model.save("/content/drive/MyDrive/SIIMS Melanoma/exp_model_image.h5")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image (InputLayer)          [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 global_average_pooling2d (  (None, 128)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 128)               16512
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Confusion matrix data
conf_matrix_image = np.array([[1399, 1], [43, 1357]])

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_image, annot=True, fmt='d', cmap='coolwarm', cbar=True,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.savefig("/content/drive/MyDrive/SIIMS Melanoma/image_model_confusion_matrix.png")
plt.show()
```




```python
import tensorflow as tf
from tensorflow.keras.models import load_model

# Load the trained model
model_path = "/content/drive/MyDrive/SIIMS Melanoma/exp_model_image.h5"
model = load_model(model_path)
```

```python
import cv2
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np

def preprocess_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0  # Rescaling as in training
    return img_array

# Example image path from your dataset
img_path = "/content/drive/MyDrive/SIIMS Melanoma/augmented images/augmented_images_v2/augmented_image_v2_24967.jpg"
preprocessed_image = preprocess_image(img_path)
```

```python
import tensorflow as tf
import numpy as np

# Function to compute Grad-CAM
def get_grad_cam(model, img_array, last_conv_layer_name, pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]

    grads = tape.gradient(class_channel, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    conv_outputs = conv_outputs[0]
    heatmap = tf.reduce_mean(tf.multiply(pooled_grads, conv_outputs), axis=-1)

    heatmap = np.maximum(heatmap, 0)
    max_heat = np.max(heatmap)
    if max_heat == 0:
```

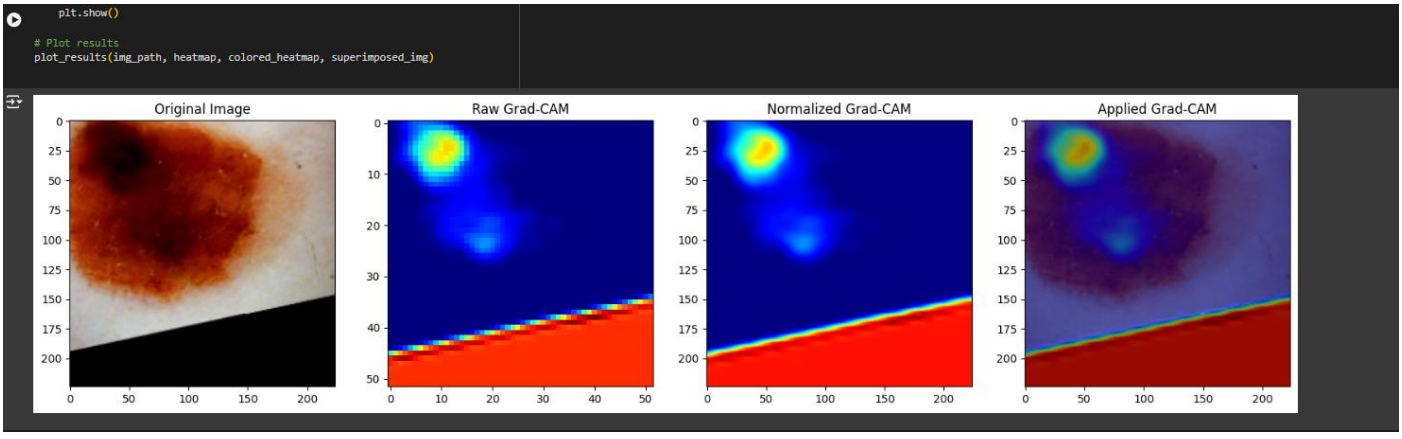
```
[ ]
# Generate Grad-CAM heatmap
last_conv_layer_name = 'conv2d_2'  # Last convolutional layer in your model
heatmap = get_grad_cam(model, preprocessed_image, last_conv_layer_name)
```

```
[ ] def apply_colormap(heatmap, original_img_path):
        img = cv2.imread(original_img_path)
        heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
        heatmap = np.uint8(255 * heatmap)
        heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
        return heatmap

    # Apply colormap to heatmap
    colored_heatmap = apply_colormap(heatmap, img_path)
```

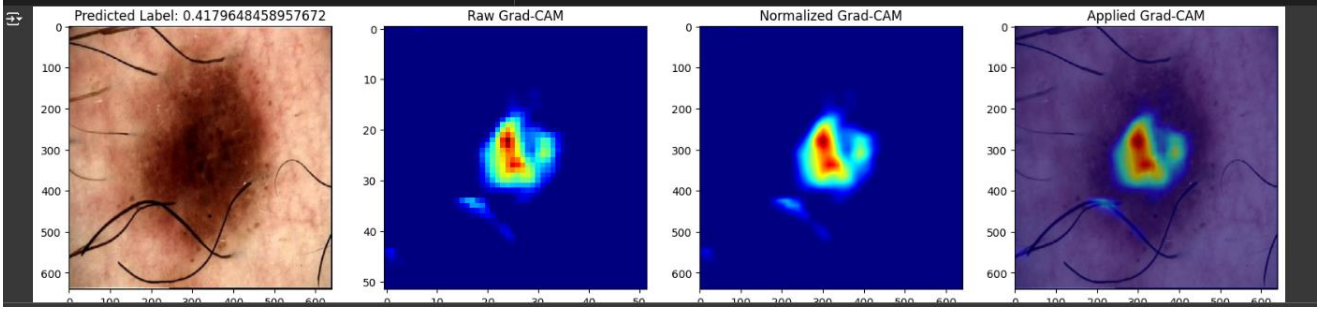
```
[ ] def overlay_heatmap_on_image(original_img_path, heatmap_colored, alpha=0.6):
        original_img = cv2.imread(original_img_path)
        superimposed_img = cv2.addWeighted(original_img, 1 - alpha, heatmap_colored, alpha, 0)
        return superimposed_img

    # Overlay heatmap on original image
    superimposed_img = overlay_heatmap_on_image(img_path, colored_heatmap)
```

```
import matplotlib.pyplot as plt

# Plotting the images
def plot_results(original_img_path, heatmap, colored_heatmap, superimposed_img):
    img = cv2.imread(original_img_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    colored_heatmap_rgb = cv2.cvtColor(colored_heatmap, cv2.COLOR_BGR2RGB)
    superimposed_img_rgb = cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB)

    plt.figure(figsize=(20, 20))
    plt.subplot(1, 4, 1)
    plt.imshow(img_rgb)
    plt.title('Original Image')

    plt.subplot(1, 4, 2)
    plt.imshow(heatmap, cmap='jet')
    plt.title('Raw Grad-CAM')

    plt.subplot(1, 4, 3)
    plt.imshow(colored_heatmap_rgb)
    plt.title('Normalized Grad-CAM')
```

```
    plt.show()

# Plot results
plot_results(img_path, heatmap, colored_heatmap, superimposed_img)
```



```python
# Load the CSV files
train_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/train_drive.csv"
validation_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/validation_drive.csv"

train_df = pd.read_csv(train_csv_path)
validation_df = pd.read_csv(validation_csv_path)

# Process train images
print("Selected Train Images:")
train_images_processed = 0
for idx, row in train_df.iterrows():
    img_path = row['image_path']
    true_label = row['target']
    if process_and_visualize(img_path, true_label):
        train_images_processed += 1
        if train_images_processed >= 5:
            break

# Process validation images
print("Selected Validation Images:")
validation_images_processed = 0
for idx, row in validation_df.iterrows():
    img_path = row['image_path']
    true_label = row['target']
    if process_and_visualize(img_path, true_label):
        validation_images_processed += 1
        if validation_images_processed >= 5:
            break
```



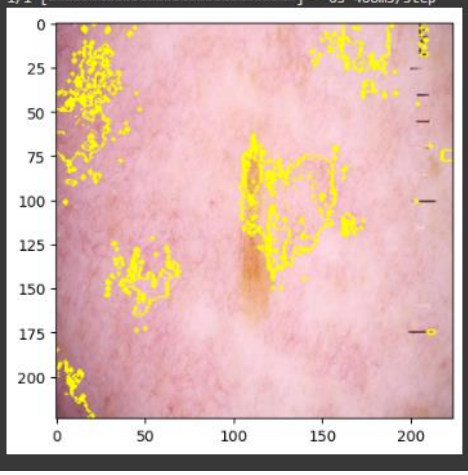

```python
# Load the CSV files
train_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/train_drive.csv"
validation_csv_path = "/content/drive/MyDrive/SIIMS Melanoma/validation_drive.csv"

train_df = pd.read_csv(train_csv_path)
validation_df = pd.read_csv(validation_csv_path)

# Drop irrelevant columns
train_df = train_df.drop(columns=['image_name', 'patient_id', 'image_path', 'benign_malignant'])
validation_df = validation_df.drop(columns=['image_name', 'patient_id', 'image_path', 'benign_malignant'])

# Ensure both datasets have the same diagnosis categories
diagnosis_categories = list(set(train_df['diagnosis'].unique()).union(set(validation_df['diagnosis'].unique())))

# Encode categorical variables
train_df = pd.get_dummies(train_df, columns=['sex', 'anatom_site_general_challenge', 'diagnosis'], drop_first=True)
validation_df = pd.get_dummies(validation_df, columns=['sex', 'anatom_site_general_challenge', 'diagnosis'], drop_first=True)

# Add missing diagnosis categories to validation data
for category in diagnosis_categories:
    if 'diagnosis_' + category not in validation_df.columns:
        validation_df['diagnosis_' + category] = 0

# Reorder columns to match
validation_df = validation_df[train_df.columns]

# Split features and target
X_train = train_df.drop(columns=['target'])
y_train = train_df['target']
X_val = validation_df.drop(columns=['target'])
y_val = validation_df['target']

# Define parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 4, 5],
    'min_child_weight': [1, 5, 10],
    'gamma': [0.1, 0.2, 0.3],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'learning_rate': [0.05, 0.1, 0.2]
}

# Initialize XGBoost classifier
model = xgb.XGBClassifier()

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameters
```

```python
# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Train XGBoost model with best parameters
model = xgb.XGBClassifier(**best_params)
model.fit(X_train, y_train)

# Predictions
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)

# Accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)
print("Train Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)

# Confusion matrix
train_cm = confusion_matrix(y_train, y_train_pred)
val_cm = confusion_matrix(y_val, y_val_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
sns.heatmap(train_cm, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.title('Train Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

plt.subplot(1, 2, 2)
sns.heatmap(val_cm, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.title('Validation Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

plt.tight_layout()
plt.show()

# Save the model
model.save_model("/content/drive/MyDrive/SIIMS Melanoma/exp_model_metadata")
```

```
Train Accuracy: 0.9998214285714285
Validation Accuracy: 0.5082142857142857
```

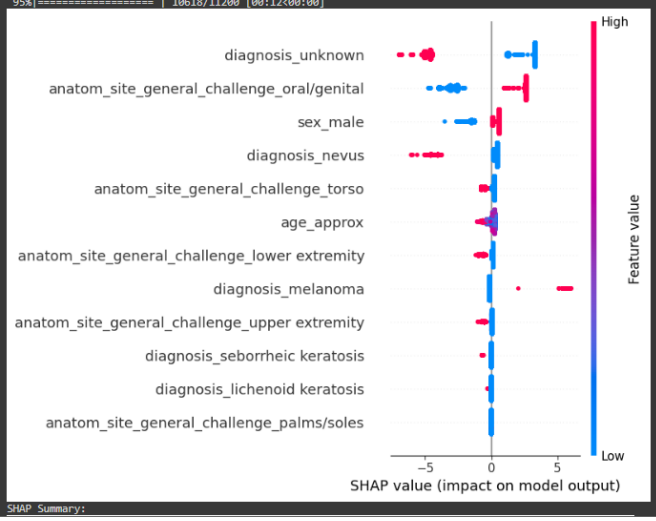
```python
example_image = preprocess_image(example_image_path)

# Define a prediction function
def predict_fn(images):
    return image_model.predict(images)

# Generate explanation
explanation = explainer.explain_instance(example_image[0].astype('double'), predict_fn,

# Get image and mask
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0], positive_only=Tr

# Display the image with LIME explanation
plt.imshow(mark_boundaries(temp / 255.0, mask))
plt.show()
```

```
Model loaded successfully.
100% 80/80 [00:04<00:00, 16.19it/s]
1/1 [==============================] - 1s 558ms/step
1/1 [==============================] - 0s 409ms/step
1/1 [==============================] - 0s 398ms/step
1/1 [==============================] - 0s 389ms/step
1/1 [==============================] - 0s 388ms/step
1/1 [==============================] - 0s 422ms/step
1/1 [==============================] - 0s 415ms/step
1/1 [==============================] - 0s 468ms/step
```



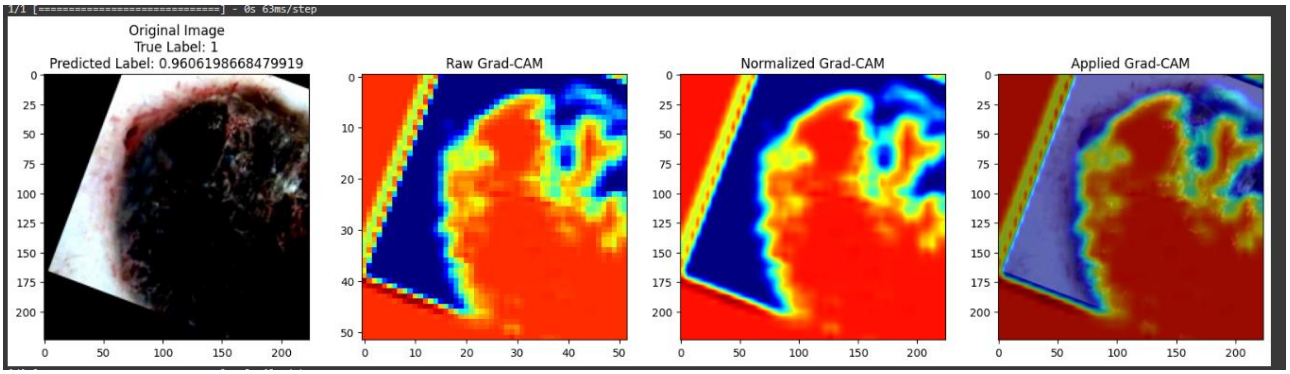
```
# Generate SHAP values
shap_values = explainer.shap_values(X_meta)

# Visualize the SHAP values
shap.summary_plot(shap_values, X_meta)

# Print summary
print("SHAP Summary:")
shap.summary_plot(shap_values, X_meta, plot_type='bar')

# Save summary plot
shap.summary_plot(shap_values, X_meta, show=False)
plt.savefig('/content/drive/MyDrive/SIIMS Melanoma/shap_summary_plot.png', bbox_inches='tight')
```

```
95%|==================| 10618/11200 [00:12<00:00]
```

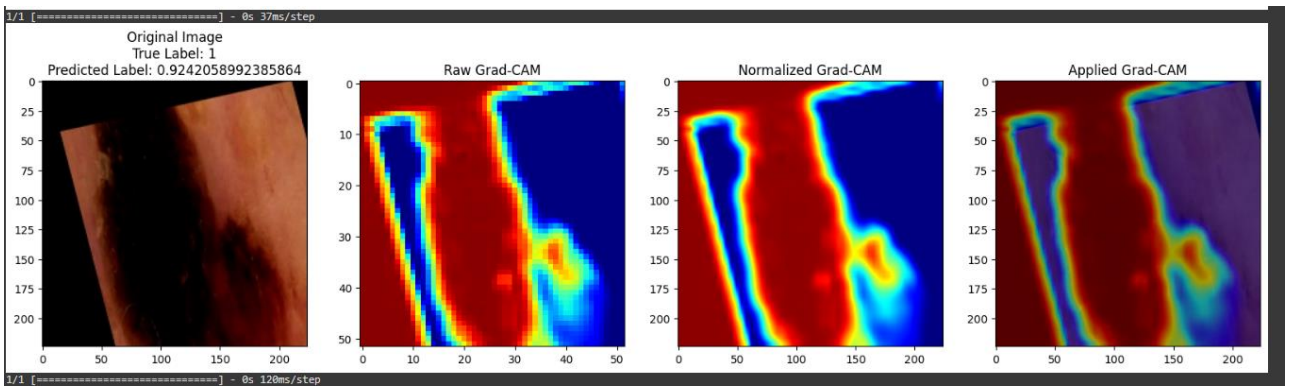



```
SHAP Summary:
```

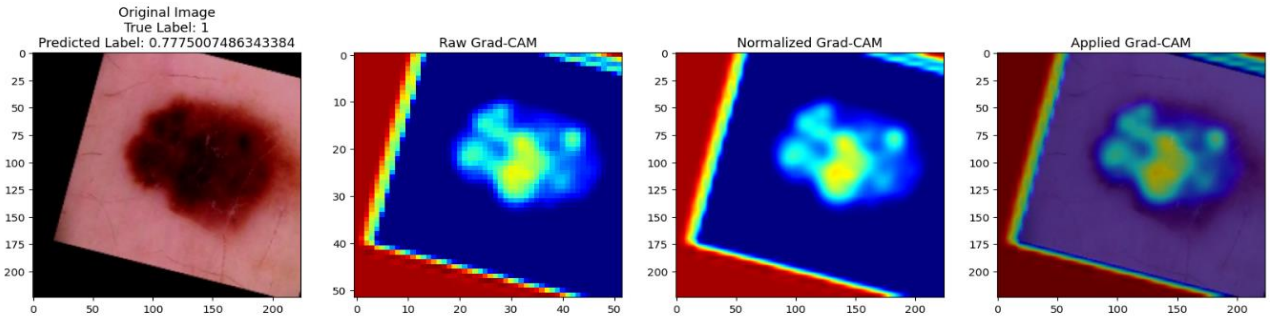
```
1/1 [==============================] - 0s 63ms/step
```




```
1/1 [==============================] - 0s 37ms/step
```




```
1/1 [==============================] - 0s 120ms/step
```



13. REFERENCES

1. **Confalonieri, R., Coba, L., Wagner, B., & Besold, T. R. (Year). A historical perspective of explainable Artificial Intelligence. *Journal Name*, Volume(Issue), Page range. DOI: 10.1002/widm.1391**
2. **Angelov, P. P., Soares, E. A., Jiang, R., Arnold, N. I., & Atkinson, P. M. (Year). Explainable artificial intelligence: An analytical review. *Journal Name*, Volume(Issue), Page range. DOI: 10.1002/widm.1424**
3. **Tjoa, E., & Guan, C. (Year). A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI. *Journal Name*, Volume(Issue), Page range. DOI: 10.1109/TNNLS.2020.3027314**
4. **Adadi, A., & Berrada, M. (Year). Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *Journal Name*, Volume(Issue), Page range. DOI: 10.1109/ACCESS.2018.2870052**
5. **Gunning, D., & Aha, D. W. (Year). DARPA's Explainable Artificial Intelligence (XAI) Program. *Journal Name*, Volume(Issue), Page range. DOI: 10.1609/aimag.v40i2.2850**
6. **Zhang, Y., Weng, Y., & Lund, J. (Year). Applications of Explainable Artificial Intelligence in Diagnosis and Surgery. *Journal Name*, Volume(Issue), Page range. DOI: 10.3390/diagnostics12020237**
7. **Ahmed, I., Jeon, G., & Piccialli, F. (Year). From Artificial Intelligence to Explainable Artificial Intelligence in Industry 4.0: A Survey on What, How, and Where. *Journal Name*, Volume(Issue), Page range. DOI: 10.1109/TII.2022.3146552**
8. **Samek, W., & Müller, K.-R. (Year). Towards Explainable Artificial Intelligence. *Book Title*. DOI: 10.1007/978-3-030-28954-6_1**
9. **Chou, Y.-L., Moreira, C., Bruza, P., Ouyang, C., & Jorge, J. (Year). Counterfactuals and causability in explainable artificial intelligence: Theory, algorithms, and applications. *Journal Name*, Volume(Issue), Page range. DOI: 10.1016/j.inffus.2021.11.003**
10. **Loh, H. W., Ooi, C. P., Seoni, S., Barua, P. D., Molinari, F., & Acharya, U. R. (Year). Application of explainable artificial intelligence for healthcare: A systematic review of the last decade (2011–2022). *Journal Name*, Volume(Issue), Page range. DOI: 10.1016/j.cmpb.2022.107161**

Proforma 4

Undertaking from the PG student while submitting his/her final dissertation to his respective institute

Ref. No. _____

I, the following student

Sr. No.	Sequence of students names on a dissertation	Students name	Name of the Institute & Place	Email & Mobile
1.	First Author	Ayush Mitra	SIG	Email: 22070243007@sig.ac.in Mobile: 9748062253

Note: Put additional rows in case of more number of students

hereby give an undertaking that the dissertation "Video Tampering Detection using Deep Learning" been checked for its Similarity Index/Plagiarism through Turnitin software tool; and that the document has been prepared by me and it is my original work and free of any plagiarism. It was found that:

1.	The Similarity Index (SI) was: *(Note: SI range: 0 to 10%; if SI is >10%, then authors cannot communicate ms;* ***attachment of SI report is mandatory****)*	10%
2.	The ethical clearance for research work conducted obtained from: *(Note: Name the consent obtaining body; if 'not appliable' then write so)*	NA
3.	The source of funding for research was: *(Note: Name the funding agency; or write 'self' if no funding source is involved)*	Self
4.	Conflict of interest: *(Note: Tick √ whichever is applicable)*	No
5.	The material (adopted text, tables, figures, graphs, etc.) as has been obtained from other sources, has been duly acknowledged in the manuscript: *(Note: Tick √ whichever is applicable)*	Yes

In case if any of the above-furnished information is found false at any point in time, then the University authorities can take action as deemed fit against all of us.

Ritwik Dubey
Full Name &
Signature of the student

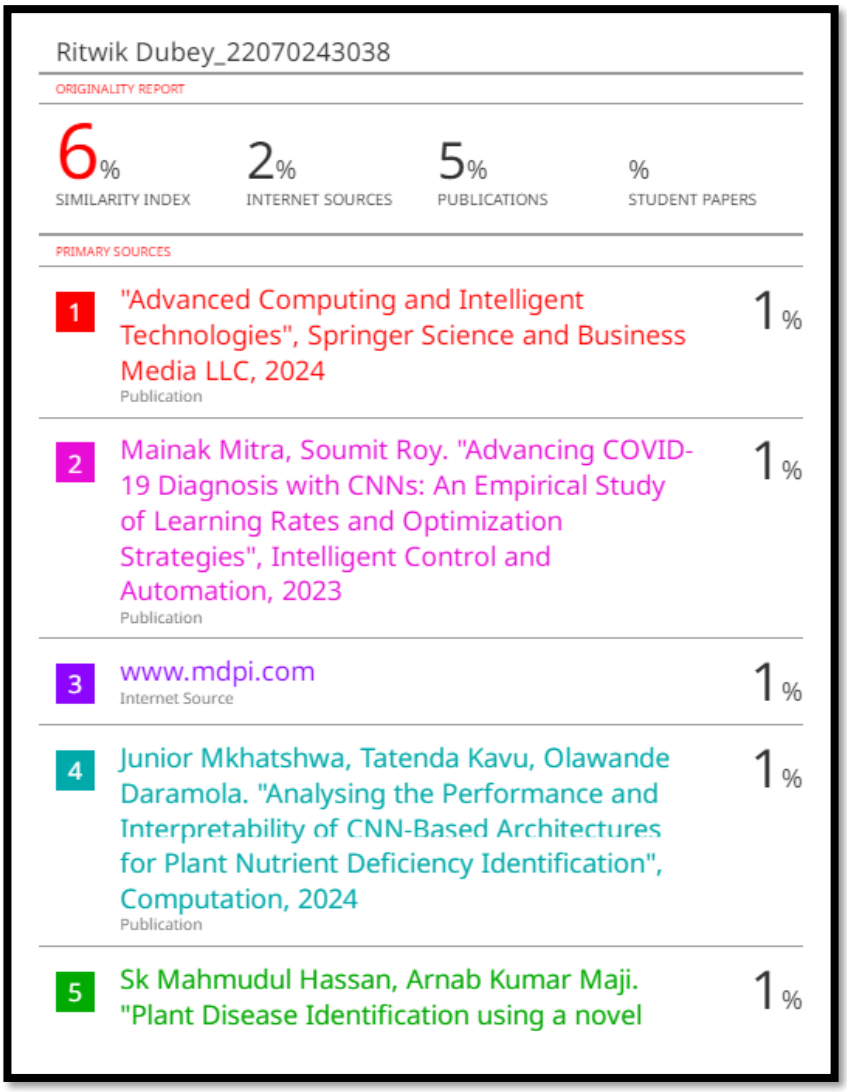
Dr. Yogesh Rajput
Name &
Signature of SIU Guide/Mentor

Date:

Endorsement by
Academic Integrity Committee (AIC)

Place: Pune

Note: It is mandatory that the Similarity Index report of plagiarism (only first page) should be appended to the



Ritwik Dubey_22070243038

ORIGINALITY REPORT

6%	2%	5%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Advanced Computing and Intelligent Technologies", Springer Science and Business Media LLC, 2024 Publication	1%
2	Mainak Mitra, Soumit Roy. "Advancing COVID-19 Diagnosis with CNNs: An Empirical Study of Learning Rates and Optimization Strategies", Intelligent Control and Automation, 2023 Publication	1%
3	www.mdpi.com Internet Source	1%
4	Junior Mkhatshwa, Tatenda Kavu, Olawande Daramola. "Analysing the Performance and Interpretability of CNN-Based Architectures for Plant Nutrient Deficiency Identification", Computation, 2024 Publication	1%
5	Sk Mahmudul Hassan, Arnab Kumar Maji. "Plant Disease Identification using a novel	1%