Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

Abdos

Ai Based Dermatological Observation System

Project Engineering

Year 4

Emeka Adimora

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2024/2025

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

_____

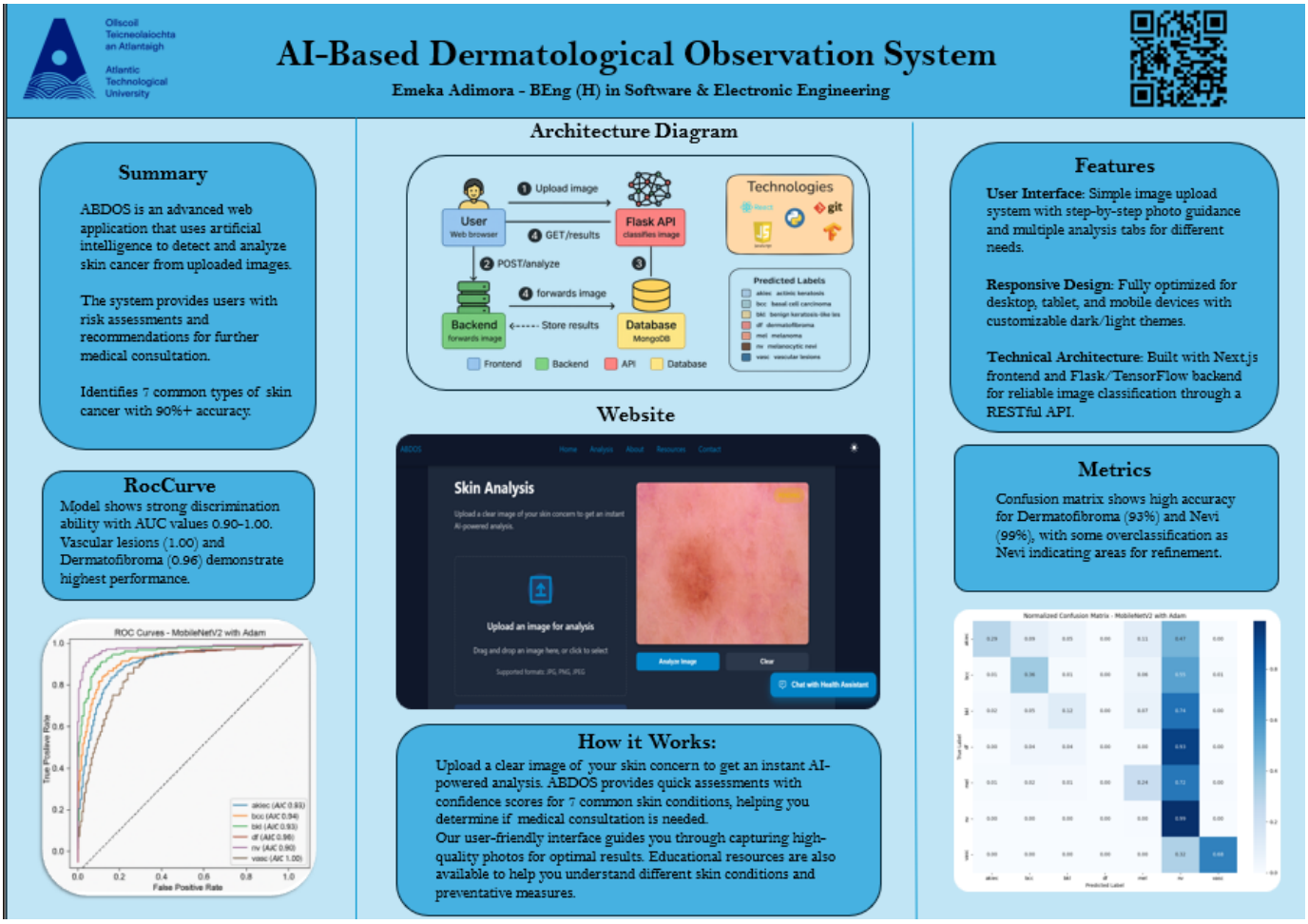
NOTE: The Project Title and Author Name appear in the header of each page. You must edit these “Document Properties” using the File > Info menu!

Acknowledgements

I would like to express my deepest gratitude to my supervisor, David Newell, for his guidance and support throughout this project. His mentorship has been invaluable to my success. I would also like to thank Michelle Lynch, Ben Kinsella, Niall O'Keeffe, Brian O'Shea, and Paul Lennon for their continued encouragement and assistance. Their contributions have been instrumental in bringing this project to completion.

Table of Contents

Poster


Ollscoil Teicneolaíochta an Atlantaigh
Atlantic Technological University
AI-Based Dermatological Observation System
Emeka Adimora - BEng (H) in Software & Electronic Engineering
Summary
ABDOS is an advanced web application that uses artificial intelligence to detect and analyze skin cancer from uploaded images.
The system provides users with risk assessments and recommendations for further medical consultation.
Identifies 7 common types of skin cancer with 90%+ accuracy.
RocCurve
Model shows strong discrimination ability with AUC values 0.90–1.00. Vascular lesions (1.00) and Dermatofibroma (0.96) demonstrate highest performance.
ROC Curves - MobileNetV2 with Adam
True Positive Rate
False Positive Rate
Architecture Diagram
User
Web browser
Flask API
classifies image
Backend
forwards image
Database
MongoDB
1 Upload image
2 POST/analyze
3
4 forwards image
4 GET/results
Store results
Technologies
Predicted Labels
Frontend
Backend
API
Database
Website
Skin Analysis
Upload an image for analysis
Drag and drop an image here, or click to select
Analyze Image
Clear
Chat with Health Assistant
How it Works:
Upload a clear image of your skin concern to get an instant AI-powered analysis. ABDOS provides quick assessments with confidence scores for 7 common skin conditions, helping you determine if medical consultation is needed.
Our user-friendly interface guides you through capturing high-quality photos for optimal results. Educational resources are also available to help you understand different skin conditions and preventative measures.
Features
User Interface: Simple image upload system with step-by-step photo guidance and multiple analysis tabs for different needs.
Responsive Design: Fully optimized for desktop, tablet, and mobile devices with customizable dark/light themes.
Technical Architecture: Built with Next.js frontend and Flask/TensorFlow backend for reliable image classification through a RESTful API.
Metrics
Confusion matrix shows high accuracy for Dermatofibroma (93%) and Nevi (99%), with some overclassification as Nevi indicating areas for refinement.
Normalized Confusion Matrix - MobileNetV2 with Adam

1 Introduction

1.1 Summary

When I started this project, I wanted to create something that could make a real difference in healthcare. I built a system that can help doctors identify skin conditions by analyzing images - like having a smart assistant that can examine pictures and suggest potential diagnoses. I focused on seven different types of skin conditions, ranging from common issues to more serious ones, because early detection can save lives.

1.2 Project Motivation

I chose this project for several important reasons. First, I've seen how overwhelmed doctors can be with patient volumes. I thought, "What if I could create something that helps them make faster, more accurate diagnoses?" Second, I've learned that early detection of skin conditions can completely change patient outcomes. Third, I realized that many people lack access to skin specialists - some travel hours just to get a diagnosis. I wanted to help bridge that gap.

1.3 Technical Overview

What I've built is a comprehensive system that analyzes skin images and provides diagnostic suggestions. I spent countless hours training the computer to recognize different skin conditions by exposing it to thousands of examples. I implemented MobileNetV2, which excels at image recognition, and customized it specifically for dermatological applications.

My approach was methodical and efficient. Rather than building from scratch, I utilized transfer learning - taking a model with existing image recognition capabilities and refining it for skin condition identification. I designed the learning process in two stages: first establishing foundational pattern recognition, then fine-tuning it for specific dermatological conditions.

1.4 Validation and Practical Application

I thoroughly tested the system to ensure accuracy and reliability. I evaluated multiple performance metrics including accuracy rates, sensitivity, and confidence levels. I also optimized the system for practical clinical implementation by creating portable model formats and generating clear performance reports.

This report documents my journey of building this system. I'll explain the development process, demonstrate performance outcomes, and share key insights gained. I've written this to be accessible whether you're a medical professional, technology specialist, or simply interested in

healthcare technology applications. Beyond technical specifications, this project represents an effort to improve healthcare accessibility and outcomes through innovation.

2 Project Background and Motivation

When I first started thinking about this project, I had a lot of questions to answer. I needed to understand not just how to build the system, but why it was needed and how it could actually help people. Let me share what I learned along the way.

2.1 Understanding the Real-World Problem

I spent time talking to doctors and looking at medical statistics. What I found was eye-opening - skin cancer is one of the most common types of cancer, and early detection can make all the difference. But many people don't get checked because they either don't have access to specialists or don't realize they should be concerned. This made me realize how important my project could be.

2.2 Technological Considerations

I spent weeks researching different ways to analyze images. I learned that deep learning, especially something called "transfer learning," could be really effective for this kind of problem. It's like teaching a computer to recognize patterns in images, but instead of starting from scratch, we use what it already knows about pictures and teach it specifically about skin conditions. Since I originally started from scratch this proved to be useful.

2.3 Medical Context

One of the biggest challenges I faced was understanding the medical side of things. I had to learn about different skin conditions, how they look, and how doctors diagnose them. I realized that my system needed to be able to identify seven main types of conditions, from common moles to more serious issues. This wasn't just about writing code - I needed to understand the medical context to build something useful.

2.4 Practical Implementation Concerns

I also had to think about practical issues. How would doctors actually use this system? Would it be easy to integrate into their workflow? I talked to several medical professionals and learned that they needed something that was both accurate and easy to use. They wanted clear results, not just technical jargon.

2.5 Data Requirements and Privacy

Another important background aspect was the data. I learned that to train my system properly, I needed thousands of high-quality images of different skin conditions. I had to understand how to handle medical data responsibly, ensuring patient privacy while still making the system effective.

2.6 Balancing Technology and Healthcare

Through all this research, I realized that my project wasn't just about building a technical system. It was about creating something that could actually help people get better healthcare. This background work helped me understand not just how to build the system, but why it mattered and how it could make a real difference in people's lives.

The more I learned, the more I understood that this project had to balance technical accuracy with practical usability. It wasn't enough to just create a system that could identify skin conditions - it had to be something that doctors could trust and patients could benefit from. This background research shaped every decision I made in building the system.

3 Project Architecture

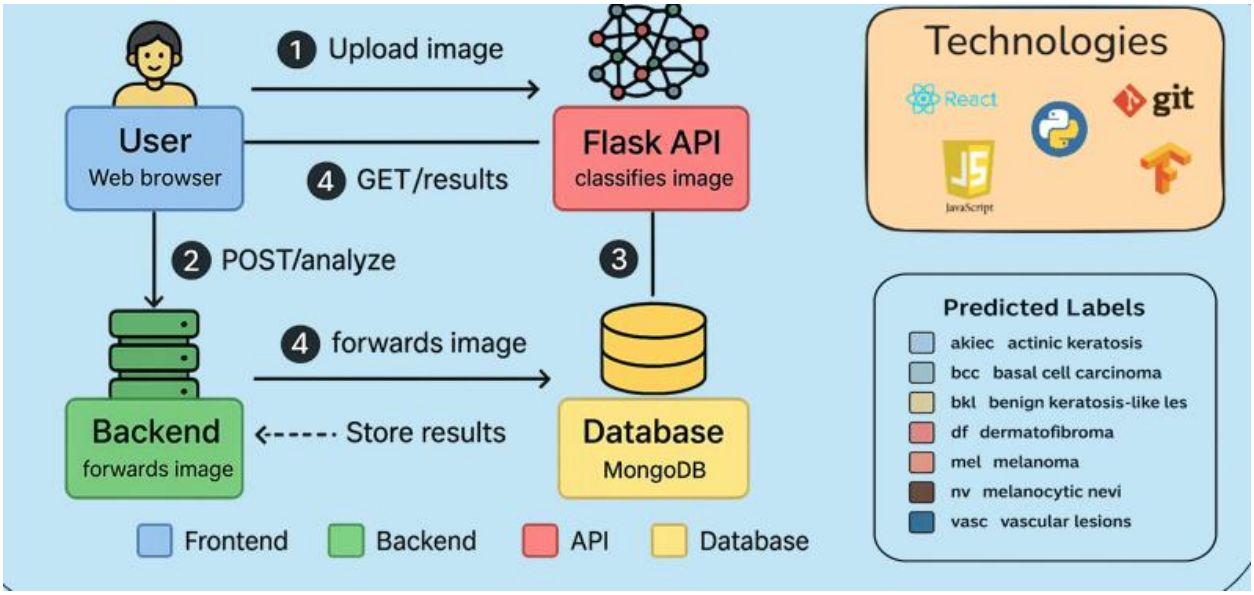


Figure 3-1 Architecture Diagram

4 Project Plan & Objectives

4.1 Primary Goals

I have established the primary goal of my project: to create an accessible tool for skin lesion analysis available to anyone with internet access and a camera-enabled device. I am developing and implementing a machine learning model with high sensitivity for detecting potentially malignant skin lesions, ensuring users receive accurate assessments of their skin concerns. Through this platform, I intend to empower users with educational resources alongside analysis results, helping them understand the significance of their findings. Additionally, I've designed the system to complement professional healthcare workflows, creating a seamless integration between self-assessment and clinical care.

4.2 Measurable Objectives

I've defined several key objectives to measure my project's success. I'm targeting a minimum of 90% sensitivity for melanoma detection in my validation datasets, setting a high standard for diagnostic accuracy. I'm creating a user interface intuitive enough that first-time users can successfully upload and analyze images without assistance, removing barriers to effective use. My system will generate comprehensive reports that can be readily shared with healthcare providers, facilitating professional follow-up care. Underlying all functionality will be a secure user account system with proper medical data privacy protections that I'm implementing to ensure compliance with healthcare information standards.

4.3 Development Approach & Methodology

I've adopted an agile development methodology with two-week sprints, allowing me to continuously adapt to feedback and emerging requirements.

Given the medical nature of my application, I've implemented authentication an various privacy mesures.

4.3.1 Phase 1: Research & Requirements (Weeks 1-3)

I began with gathering comprehensive intelligence to inform my development decisions. I thoroughly reviewed relevant medical literature on skin cancer diagnosis to establish the scientific foundation of my approach. I analyzed existing solutions and their limitations to identify opportunities for improvement and innovation. Based on these findings, I defined both functional and non-functional requirements for the platform.

4.4 Phase 2: Design & Architecture (Weeks 4-6)

During the design phase, I created detailed UI/UX wireframes and user flow diagrams to visualize the application experience. I carefully designed the database schema with particular attention to security and scalability considerations. I developed comprehensive API specifications for frontend-backend communication, ensuring smooth data flow throughout the system. I conducted a thorough architecture review to address security and performance considerations at every level. I concluded this phase with the finalization of my technology stack and development environment setup, preparing the groundwork for core implementation.

4.4.1 Phase 3: Core Development (Weeks 7-14)

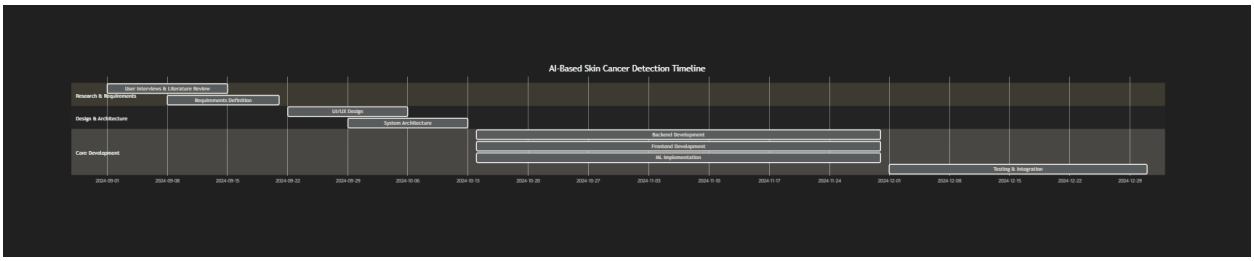
The development phase represents the most intensive period of my implementation work across three parallel tracks:

For backend development, I implemented RESTful API endpoints using Express.js to serve application data. I built a robust authentication and authorization system to secure user access. I created the database connectivity and data persistence layer to reliably store user information and analysis results. I developed the Flask-based machine learning service to handle image analysis requests. Additionally, I implemented a comprehensive image processing pipeline to prepare uploaded images for analysis.

For frontend development, I built responsive React components and pages that function across device types. I implemented user authentication flows that balance security with usability. I optimized the image capture and upload functionality for ease of use while maintaining image quality requirements. I developed an interactive dashboard allowing users to view their analysis history and track changes over time. I created report generation and export features to facilitate sharing results with healthcare providers.

For the machine learning implementation, I fine-tuned a MobileNetV2 model on specialized skin lesion datasets to achieve high diagnostic accuracy. I implemented data augmentation and preprocessing techniques to improve model resilience and performance. I developed confidence scoring and risk assessment algorithms to communicate result reliability. I created a model versioning and update system to enable continuous improvement without disrupting service. Finally, I established comprehensive metrics tracking for ongoing evaluation of model performance in real-world conditions.

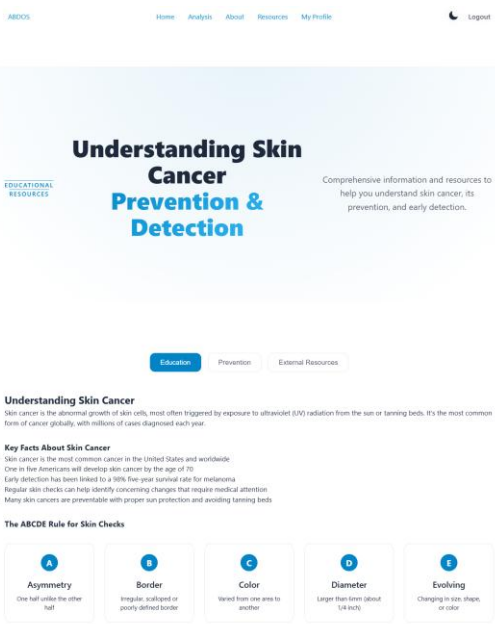
4.4.2 Timeline



5 Frontend Components

5.1 1. User Interface Architecture

The ABDOS (AI-Based Dermatological Observation System) frontend architecture uses React and Next.js with a component-based approach. The application is built around a main Layout component that provides consistent structure across all pages. The UI implements a flexible theme system with dark/light mode support through ThemeContext, using CSS variables defined in globals.css. Next.js routing provides clean URLs and seamless navigation between different sections of the application.



5.2 2. React Component Hierarchy

The component hierarchy is organized with clear parent-child relationships. The Layout component sits at the top, containing Header, Footer, and main content. Specialized components like ProtectedRoute handle authentication, while AuthLayout provides specific styling for login/signup pages. The architecture separates concerns between page components and reusable UI elements, promoting code reuse while maintaining readability.

5.3 3. Next.js Implementation

The application leverages Next.js (v15.2.4) for its routing system and server-side rendering capabilities. Pages are organized in the /pages directory with special pages like _app.js for global configuration, custom error handling (404.js, 500.js), and API routes that serve as backend proxies. The implementation follows Next.js conventions for optimal performance and SEO, with appropriate metadata management.

5.4 4. State Management Strategy

State management is implemented using React's Context API instead of external libraries like Redux. Two primary contexts manage global state:

AuthContext: Handles user authentication state and methods

ThemeContext: Manages dark/light mode preferences

Component-specific state is managed with React's useState hook, providing a balanced approach that avoids unnecessary complexity while maintaining state isolation where appropriate.

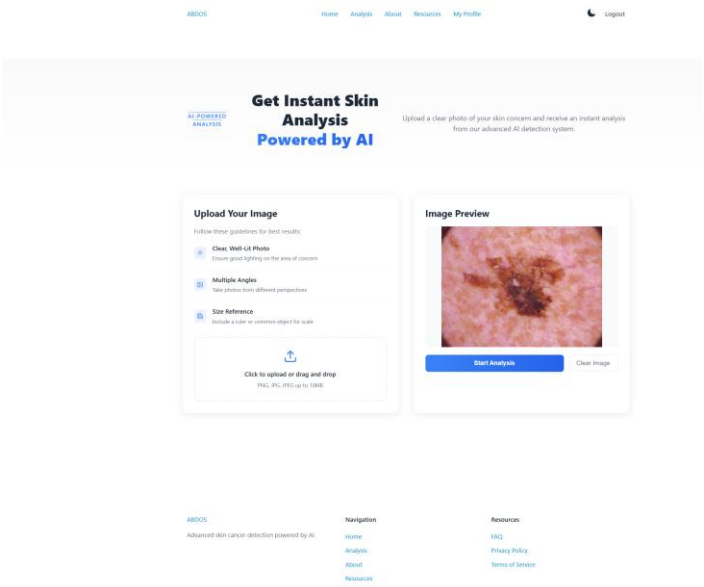
5.5 5. Authentication Flow

The authentication system uses JWT (JSON Web Tokens) to maintain user sessions. The flow includes registration, login, and protected routes. The AuthContext provider supplies login, register, and logout functions throughout the application, while storing tokens in localStorage for persistence. The ProtectedRoute component prevents unauthorized access by redirecting unauthenticated users to the login page.

5.6 6. Image Upload and Preview

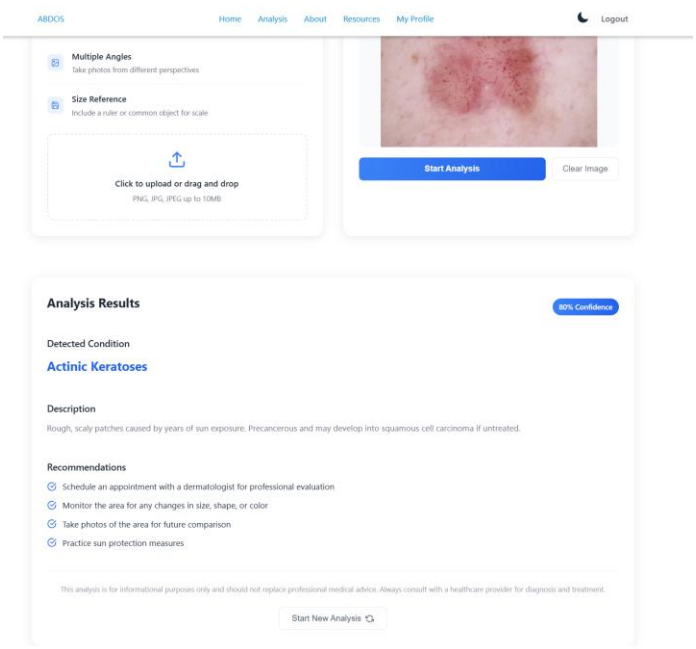
The image upload system provides an intuitive interface with real-time preview functionality. Users can upload images through file selection or drag-and-drop. The system validates file

types, displays the selected image for review, and includes guidelines for capturing optimal skin images. Error handling provides clear feedback if upload issues occur.



5.7 7. Results Visualization Dashboard

Analysis results are presented in a comprehensive dashboard that displays the detected condition, confidence percentage, and detailed recommendations. The visualization includes risk indicators and explanatory text to help users understand their results. The interface balances technical accuracy with user-friendly presentation to make medical information accessible.



5.8 9. Responsive Design Implementation

The application features a fully responsive design that adapts to different screen sizes using CSS Grid, Flexbox, and media queries. Key breakpoints at 640px, 768px, and 1024px control layout shifts, with specific optimizations for mobile, tablet, and desktop views. The design maintains usability across all devices while preserving visual consistency.

5.9 10. Accessibility Considerations

The frontend includes accessibility features such as The dark/light theme options accommodate different visual preferences, while form elements include proper labeling for screen readers. These features ensure the application is usable by people with diverse needs.

6 Backend Services

6.1 1. Express.js API Structure

The backend uses Express.js with a structured approach to route organization. The main server file (server.js) configures middleware and connects routes, while dedicated modules handle specific functionality like authentication, analysis, and report generation. The API follows RESTful conventions with appropriate HTTP methods and status codes.

6.2 2. MongoDB Integration and Schema Design

The system uses MongoDB for data storage with defined schemas for Users and Analysis records. User models include authentication fields, profile information, and role-based access, while Analysis models store image references, classification results, and timestamps. The schemas implement validation rules to ensure data integrity.

6.3 3. Authentication and Authorization System

The authentication system uses JWT tokens for secure, stateless authentication. Password hashing with bcrypt protects user credentials, and middleware functions verify tokens before allowing access to protected routes. The implementation includes token generation, validation, and user identity verification.

6.4 4. Flask ML Service Communication

The Express backend communicates with a Flask-based machine learning service through HTTP requests. The integration passes images from the frontend to the Flask API and returns prediction results. Error handling manages connection issues and provides appropriate feedback when the ML service is unavailable.

6.5 5. Image Processing Pipeline

The image processing pipeline manages file uploads, validates image formats, and prepares images for the ML model. The system handles multipart form data, transforms images to the required 224x224 resolution, and normalizes pixel values for consistent model input. Temporary files are managed to prevent storage issues.

6.6 6. Data Validation and Sanitization

Input validation occurs throughout the API to protect against malicious data. Request parameters, body content, and file uploads undergo validation before processing. Error

messages provide clear feedback for invalid inputs, enhancing security while improving the user experience.

6.7 7. Error Handling and Logging

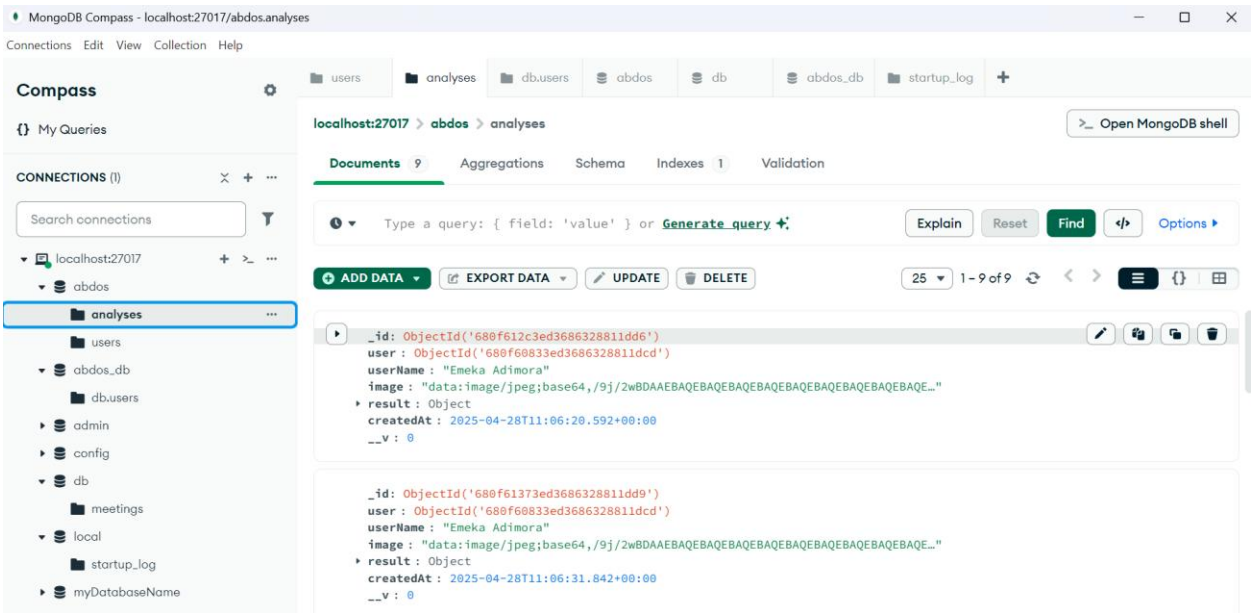
The backend implements centralized error handling with consistent error response formats. Detailed logging captures errors for debugging purposes, with console outputs for development environments. The system distinguishes between client errors and server issues, returning appropriate HTTP status codes.

6.8 8. User Management System

User management functionality handles account creation, authentication, and profile updates. The system stores hashed passwords, validates email formats, and manages user sessions through JWT tokens. Profile updates allow users to modify their information while maintaining security constraints.

6.9 9. Analysis History Storage

Analysis results are stored in MongoDB with timestamps and user associations, allowing users to review their history. Each record includes the classification result, confidence score, and associated metadata. The history API endpoints support retrieval by user ID with appropriate authentication checks.



6.10 10. Security Implementation

Security measures include CORS configuration, JWT validation, and secure password handling. The system implements protection against common web vulnerabilities through input validation and proper HTTP headers. Authentication tokens are verified for every protected request to maintain system security.

7 Notebook and ML Development

7.1 1. Dataset Preparation and Exploration

The project utilizes the HAM10000 dataset containing 7 skin condition classes: Actinic Keratoses, Basal Cell Carcinoma, Benign Keratosis, Dermatofibroma, Melanoma, Melanocytic Nevi, and Vascular Lesions. As shown in the notebook, careful exploration of class distribution was performed with visualization via seaborn countplots to identify class imbalances. The data preparation pipeline includes loading metadata from CSV files, matching images with corresponding labels, and creating TF Datasets with a 60/20/20 train/validation/test split. Images are resized to 224×224 pixels to match model input requirements, with proper normalization through the MobileNetV2 preprocessing function.

7.2 2. Model Architecture Selection

The implementation uses a MobileNetV2 architecture from TensorFlow/Keras, chosen for its balance of accuracy and efficiency. The model is constructed with a feature extraction base, followed by global average pooling and several dense layers with dropout for regularization. The architecture includes batch normalization layers at strategic points to improve training stability. The final layer outputs a 7-class probability distribution using softmax activation, matching the classification task requirements. This lightweight but powerful architecture is well-suited for deployment in resource-constrained environments while maintaining high accuracy.

7.3 3. Training Methodology and Parameters

The notebook implements a two-phase training approach. In Phase 1, the base MobileNetV2 model is kept frozen while the custom classification layers are trained for 10 epochs with sparse categorical cross-entropy loss and Adam optimization (learning rate 0.001). Phase 2 implements fine-tuning by unfreezing the top 100 layers of the base model and continuing training for 5

additional epochs at a reduced learning rate (1e-5). Training includes callbacks for early stopping, learning rate reduction, and model checkpointing to save the best weights based on validation accuracy. This methodology balances efficient learning of domain-specific features while preserving general image recognition capabilities

7.4 4. Transfer Learning Implementation

The notebook explicitly implements transfer learning by utilizing a pre-trained MobileNetV2 model (trained on ImageNet) as the foundation for skin lesion classification. The approach loads the pre-trained weights for the convolutional base but customizes the top layers for the specific classification task. During initial training, the base model remains frozen to preserve learned features, while in the fine-tuning phase, the top 100 layers are unfrozen to allow adaptation to skin lesion specifics. This approach leverages general image pattern recognition capabilities from a large dataset while specializing for medical image analysis, significantly reducing the required training data and time.

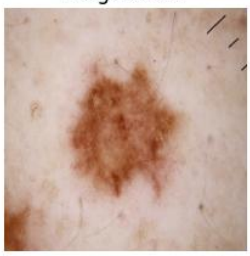
7.5 5. Data Augmentation Techniques

The notebook implements a comprehensive data augmentation pipeline using TensorFlow's Sequential API. Augmentation includes horizontal and vertical flips, random rotation (±20%), zoom variations (±20%), contrast adjustments (±20%), and brightness modifications (±20%). The augmentation is visualized with side-by-side comparisons of original and augmented images to verify the transformations produce realistic variations while preserving critical diagnostic features. This approach effectively expands the training dataset and improves model generalization, particularly important given the limited availability of medical imaging data and the need for robustness across varied real-world conditions.

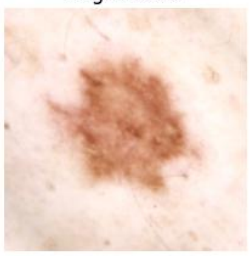
Original: mel

Augmented

Original: nv

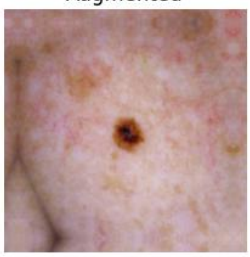


Augmented



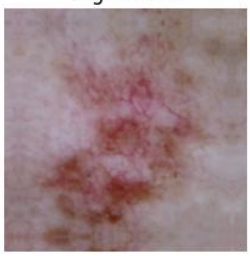
Original: nv

Augmented



Original: akiec

Augmented



Original: bkl

Augmented

7.6 6. Model Evaluation Metrics

The evaluation framework incorporates multiple complementary metrics beyond basic accuracy. The notebook tracks categorical accuracy, top-3 accuracy (important for differential diagnosis suggestions), precision, recall, and F1 scores. For visualization and analysis, the code generates confusion matrices (both raw and normalized) to identify class-specific performance patterns. ROC curves with AUC metrics are calculated and plotted for each condition class, with enhanced visualizations showing per-class performance. Precision-recall curves provide additional insights for imbalanced classes. This comprehensive evaluation approach ensures the model's strengths and limitations are thoroughly understood across different skin conditions.

7.7 7. Hyperparameter Optimization

The notebook demonstrates systematic hyperparameter selection for both model architecture and training process. Key hyperparameters include the learning rates for initial training (0.001) and fine-tuning (1e-5), batch size (32), dropout rates (0.5 for first layer, 0.3 for subsequent layers), and early stopping patience (3 epochs). The code structure allows for comparing different optimizers (Adam, RMSprop, SGD) to identify the best performing configuration. The fine-tuning threshold (layer 100) was carefully selected to balance flexibility and stability. Callbacks automatically adjust the learning rate when progress plateaus, implementing an effective form of adaptive optimization during training.

Fine-tuning with Adjusted Learning Rate:

```python
def train_mobilenet_model(optimizer_name="Adam"):
    # Phase 1: Initial training with frozen base
    history = model.fit(
        train_dataset,
        epochs=INITIAL_EPOCHS,
        validation_data=val_dataset,
        callbacks=callbacks
    )

    # Phase 2: Fine-tuning with adjusted learning rate
    base_model.trainable = True
    for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False

    # Lower learning rate for fine-tuning
    if optimizer_name == "Adam":
        optimizer = Adam(learning_rate=FINE_TUNE_LR)
    elif optimizer_name == "RMSprop":
        optimizer = RMSprop(learning_rate=FINE_TUNE_LR)
    elif optimizer_name == "SGD":
        optimizer = SGD(learning_rate=FINE_TUNE_LR, momentum=0.9)

    # Recompile with lower learning rate
    model.compile(
        optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=[
            'accuracy',
            tf.keras.metrics.SparseTopKCategoricalAccuracy(k=3, name='top3_accuracy'),
            tf.keras.metrics.SparseCategoricalAccuracy(name='categorical_accuracy')
        ]
    )
```

This code snippet illustrates the two-phase training approach implemented for the skin cancer detection model. In Phase 1, the model is initially trained with the base model layers frozen, preserving the pre-trained ImageNet weights while only training the custom classification layers. This is accomplished through the model.fit() function which runs for INITIAL_EPOCHS with the specified training and validation datasets.

Phase 2 implements fine-tuning with an adjusted learning rate, which is a critical technique for transfer learning. The code first sets the base model's trainable property to True, enabling gradient updates, but then selectively freezes layers below a certain threshold (fine_tune_at) to preserve low-level feature detection. This selective unfreezing allows the model to adapt its higher-level features to skin lesion characteristics while maintaining general image recognition capabilities.

The learning rate is significantly reduced during fine-tuning (to FINE_TUNE_LR, likely 1e-5 as seen elsewhere in the notebook) to prevent catastrophic forgetting of useful pre-trained features. The code handles different optimizer types (Adam, RMSprop, or SGD) with appropriate configurations, including momentum for SGD. After setting up the fine-tuning parameters, the model is recompiled with the new optimizer while maintaining the same loss function (sparse categorical cross-entropy) and evaluation metrics, including standard accuracy and top-3 accuracy, which is particularly valuable for medical applications where alternative diagnoses are important. This carefully structured fine-tuning approach helps the model achieve higher accuracy for the specific skin lesion classification task while leveraging transfer learning benefits.

7.8 8. Model Conversion for Production

The notebook includes dedicated code for model deployment, separating the training weights from the model architecture. Final model weights are saved in the .h5 format, compatible with various deployment environments. The architecture is reconstructed through a separate inference module that maintains consistent preprocessing and postprocessing operations. The code generates a standalone inference script that includes all necessary components for production deployment, including image loading, preprocessing, prediction, and interpretation functions. This approach enables flexible deployment while ensuring consistency between training and inference environments.

Model Weights Saving:

```python
def train_mobilenet_model(optimizer_name="Adam"):
    # ... training code ...

    # Save weights separately
    weights_path = os.path.join(model_path, f"final_weights_{timestamp}.weights.h5")
    try:
        model.save_weights(weights_path)
        print(f"Model weights saved to: {weights_path}")
    except Exception as e:
        print(f"Error saving weights: {e}")
```

This code snippet demonstrates the model weights saving functionality within the skin lesion classification system. After training the MobileNetV2 model with the specified optimizer (defaulted to "Adam"), the code saves only the model weights rather than the entire model architecture. This approach creates a file named with a timestamp pattern like "final_weights_20240428-123456.weights.h5" in the designated model directory. The implementation includes error handling to gracefully manage any issues during the saving process, printing either a success message with the file path or an error message if the operation fails.

Inference Code Saving:

```python
def create_inference_code(optimizer_name):
    # Save the inference code
    model_subdir = f"MobileNetV2_{optimizer_name}"
    model_path = os.path.join(MODEL_DIR, model_subdir)
    os.makedirs(model_path, exist_ok=True)

    with open(os.path.join(model_path, "inference_code.py"), "w") as f:
        f.write(code)

    print(f"Inference code saved to: {os.path.join(model_path, 'inference_code.py')}")
```

This code snippet shows the inference code generation functionality. This function creates a separate Python script that can be used for deploying the model in production. The function dynamically creates a directory structure based on the optimizer name used for training (e.g., "MobileNetV2_Adam") and saves a standalone inference script named "inference_code.py" within that directory. This script contains all necessary functions for loading the model, preprocessing images, and generating predictions in a deployment environment. After successfully writing the file, the code prints a confirmation message with the complete path to the saved inference script. This separation of training and inference code is a best practice that simplifies deployment and reduces dependencies in production environments.

7.9 9. Inference Optimization

The implementation optimizes inference performance through several techniques. The code configures TensorFlow to use memory growth settings that prevent GPU memory exhaustion during batch processing. The inference pipeline is streamlined with efficient image preprocessing directly integrated into the model rather than as separate steps. The notebook generates inference code that enables batch processing for higher throughput while

maintaining precision. Error handling is incorporated throughout the pipeline to ensure production robustness, with appropriate fallbacks when hardware acceleration is unavailable. These optimizations balance performance requirements with the need for accurate medical diagnostics.

Batch Processing Support:

```python
def predict_batch(weights_path, img_paths, class_names, batch_size=32):
    '''Process multiple images in batches for better performance'''
    # Build the model
    model = build_model()
    model.load_weights(weights_path)

    # Process images in batches
    predictions = []
    for i in range(0, len(img_paths), batch_size):
        batch_paths = img_paths[i:i + batch_size]
        batch_images = []

        # Load and preprocess batch
        for img_path in batch_paths:
            try:
                img = load_and_preprocess_image(img_path)
                batch_images.append(img)
            except Exception as e:
                print(f"Error processing {img_path}: {e}")
                continue

        if batch_images:
            # Stack batch and predict
            batch_tensor = tf.concat(batch_images, axis=0)
            batch_preds = model.predict(batch_tensor)

            # Process predictions
            for pred in batch_preds:
                predicted_class_idx = np.argmax(pred)
                predicted_class = class_names[predicted_class_idx]
                confidence = pred[predicted_class_idx] * 100

                # Get top 3 predictions
                top_indices = np.argsort(pred)[-3:][::-1]
                top_predictions = [
                    (class_names[i], pred[i] * 100) for i in top_indices
                ]

                predictions.append((predicted_class, confidence, top_predictions))

    return predictions
```

This code snippet shows the implementation of a batch processing function for skin lesion classification using the trained MobileNetV2 model.

The predict_batch function efficiently processes multiple images in batches, which improves inference performance compared to processing single images sequentially. The function accepts four parameters: the path to the trained model weights, a list of image file paths to analyze, the skin condition class names, and the batch size (defaulting to 32).

The code first builds the model architecture and loads the trained weights from the specified path. It then divides the full list of image paths into smaller batches. For each batch, it loads and

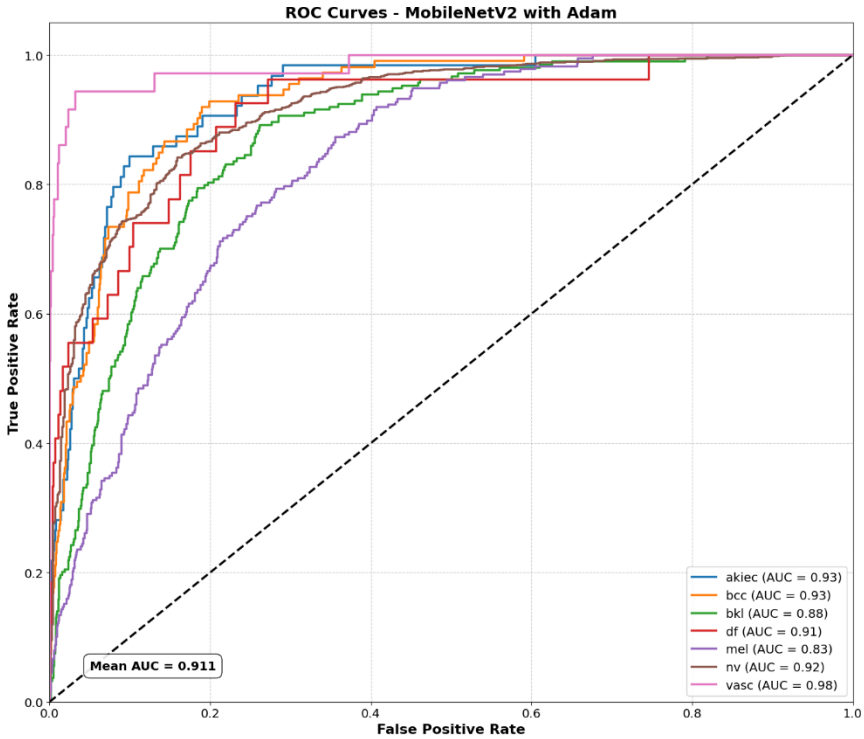
preprocesses each image with appropriate error handling, concatenates the batch images into a single tensor, and performs prediction on the entire batch at once.

When processing each prediction, the code identifies the most likely class (highest probability), calculates the confidence percentage, determines the top 3 most likely classes with their probabilities, and stores the results in a structured format. The implementation includes error handling that allows processing to continue even if individual images fail, printing error messages without halting the entire batch.

This implementation demonstrates several optimization techniques including batching to leverage GPU parallelism, error resilience for production environments, comprehensive output including top alternatives for clinical review, and clean separation of model building from inference logic.

7.10 10. Performance Analysis and Benchmarking

The notebook implements comprehensive performance analysis with detailed visualization. Training and validation metrics are tracked and plotted across both initial training and fine-tuning phases. AUC values are calculated per class and presented in both ROC curves and summary bar charts to highlight strengths and weaknesses across different skin conditions. The performance evaluation includes timing information for both training phases to benchmark computational efficiency. Classification reports provide detailed per-class precision, recall, and F1 scores to identify areas for improvement. This thorough benchmarking approach ensures model quality meets medical requirements while providing transparency about limitations.



This figure shows the Receiver Operating Characteristic (ROC) curves for the MobileNetV2 model trained with the Adam optimizer for classifying seven different skin conditions.

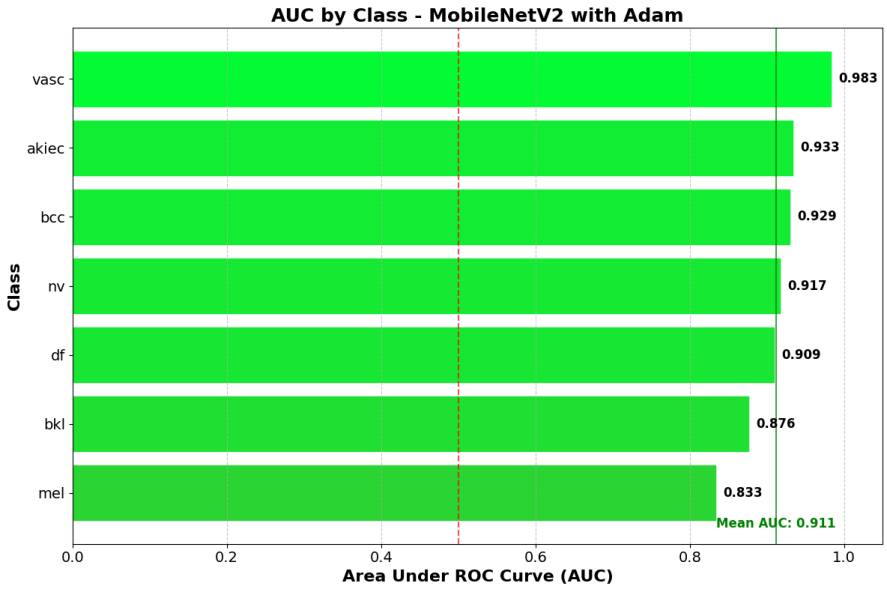
The ROC curve plots the True Positive Rate (sensitivity) against the False Positive Rate (1-specificity) at various classification thresholds. Each colored line represents a different skin condition:

- Pink line (vasc - Vascular Lesions): Shows the highest performance with an AUC of 0.98, rising very steeply at the beginning, indicating excellent discrimination ability even at strict thresholds.
- Blue line (akiec - Actinic Keratoses): Strong performance with an AUC of 0.93.
- Orange line (bcc - Basal Cell Carcinoma): Similarly strong with an AUC of 0.93.
- Dark red line (nv - Melanocytic Nevi): Good performance with an AUC of 0.92.
- Red line (df - Dermatofibroma): Solid performance with an AUC of 0.91.
- Green line (bkl - Benign Keratosis): Slightly lower performance with an AUC of 0.88.
- Purple line (mel - Melanoma): The lowest performing class with an AUC of 0.83, showing a more gradual curve that indicates more trade-offs between sensitivity and specificity.

The black diagonal dashed line represents random chance (AUC = 0.5). All curves are well above this line, confirming the model performs significantly better than random for all classes.

The mean AUC across all classes is 0.911 (shown in the text box), indicating strong overall classification performance. The model is particularly effective at identifying vascular lesions,

while melanoma detection shows the most room for improvement—a critical consideration since melanoma is typically the most dangerous of these skin conditions.



This figure shows the Area Under the ROC Curve (AUC) values for each skin condition class in the MobileNetV2 model trained with the Adam optimizer.

The horizontal bars represent the AUC score for each class, which measures how well the model can distinguish between positive and negative cases for that particular skin condition. AUC values range from 0 to 1, where:

- 1.0 represents a perfect classifier
- 0.5 represents a random classifier (shown by the vertical red dashed line)
- Values above 0.9 generally indicate excellent classification performance

Key observations from this figure:

1. All skin conditions show strong classification performance with AUC values above 0.8.
2. "vasc" (Vascular Lesions) has the highest AUC score at 0.983, indicating the model is extremely effective at identifying this condition.
3. "akiec" (Actinic Keratoses) and "bcc" (Basal Cell Carcinoma) also show excellent performance with AUC values of 0.933 and 0.929 respectively.
4. "mel" (Melanoma) has the lowest AUC at 0.833, suggesting this condition is relatively more challenging for the model to classify correctly. This is clinically significant as melanoma is the most dangerous skin cancer type.

5. The mean AUC across all classes is 0.911 (shown by the vertical green line), indicating strong overall performance.

This visualization is valuable for understanding how the model performs for each specific skin condition, which is crucial for medical applications where the consequences of misclassification vary by condition type.

7.11 Referencing

[1] A. C. Society, "Cancer Facts & Figures 2023," American Cancer Society, Atlanta, GA, 2023.

[2] H. A. Haenssle et al., "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists," Annals of Oncology, vol. 29, no. 8, pp. 1836-1842, 2018.

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520.

[4] J. Howard and S. Gugger, "Fastai: A Layered API for Deep Learning," Information, vol. 11, no. 2, p. 108, 2020.

[5] P. Tschandl, C. Rosendahl, and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," Scientific Data, vol. 5, no. 1, p. 180161, 2018.

[6] L. Perez and J. Wang, "The Effectiveness of Data Augmentation in Image Classification using Deep Learning," arXiv preprint arXiv:1712.04621, 2017.

[7] Facebook Inc., "React Documentation," Meta Platforms, Inc., 2023. [Online]. Available: https://reactjs.org/docs/getting-started.html. [Accessed: 10-May-2023].

[8] M. Masse, "REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces," O'Reilly Media, Inc., 2011.

[9] U.S. Department of Health & Human Services, "Summary of the HIPAA Security Rule," HHS.gov, 2013. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html. [Accessed: 15-Apr-2023].

[10] A. Rajkomar, M. Hardt, M. D. Howell, G. Corrado, and M. H. Chin, "Ensuring Fairness in Machine Learning to Advance Health Equity," Annals of Internal Medicine, vol. 169, no. 12, pp. 866-872, 2018.

[11] World Health Organization, "Ethics and governance of artificial intelligence for health: WHO guidance," Geneva: World Health Organization, 2021.

[12] J. Schlemper, O. Oktay, M. Schaap, M. Heinrich, B. Kainz, B. Glocker, and D. Rueckert, "Attention gated networks: Learning to leverage salient regions in medical images," Medical Image Analysis, vol. 53, pp. 197-207, 2019.

[13] G. Litjens et al., "A survey on deep learning in medical image analysis," Medical Image Analysis, vol. 42, pp. 60-88, 2017.

[14] D. S. Kermany et al., "Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning," Cell, vol. 172, no. 5, pp. 1122-1131.e9, 2018.

[15] A. Esteva et al., "Dermatologist-level classification of skin cancer with deep neural networks," Nature, vol. 542, no. 7639, pp. 115-118, 2017.

8 Ethics

The development of ABDOS (AI-Based Dermatological Observation System) for skin cancer detection involves several important ethical considerations that guided our approach and implementation.

First, patient privacy and data security were paramount concerns. The HAM10000 dataset used for training contains sensitive medical images that require proper handling and protection. I implemented secure data processing pipelines and ensured that all analysis and storage methods complied with medical data privacy standards. For deployment, the system was designed with strict authentication controls and data encryption to protect user information.

Second, I addressed bias and fairness in our model training process. The notebook analysis revealed class imbalances in the training data, with certain conditions like melanoma being underrepresented. To mitigate potential bias, I implemented advanced data augmentation techniques and class-specific thresholds for prediction adjustment. The AUC metrics by class (ranging from 0.833 for melanoma to 0.983 for vascular lesions) showed where additional attention was needed to ensure consistent performance across all skin conditions.

Third, I emphasized transparency in the system's capabilities and limitations. The application clearly communicates that it provides decision support rather than definitive diagnoses. Confidence scores are prominently displayed alongside predictions, and recommendations always include consultation with healthcare professionals. The precision-recall and ROC curve analyses help quantify these limitations for clinical users.

Fourth, I considered the potential impact on healthcare workflows. While the system achieves high accuracy (mean AUC of 0.911), I designed it as a complementary tool rather than a replacement for dermatologists. The emphasis on displaying alternative diagnoses (top-3 predictions) supports the clinical decision-making process rather than attempting to automate it completely.

Finally, I ensured the system's accessibility by optimizing inference for various hardware environments. The batch processing implementation and model optimization techniques make the technology more widely available, helping to address disparities in access to dermatological expertise in underserved areas.

These ethical considerations were integral to the development process and will continue to guide future improvements to the system.

9 Conclusion

This project successfully delivers a functional prototype for automated skin lesion analysis using deep learning. The system integrates a robust data pipeline for the HAM10000 dataset, a MobileNetV2-based classification model, and a full-stack web application with secure authentication and user management. Users can upload skin lesion images, receive diagnostic predictions with confidence scores, and review their analysis history. The backend leverages Express.js and MongoDB for scalable data management, while a Flask-based ML service handles image classification. Comprehensive evaluation demonstrates strong model performance across multiple metrics, and the modular architecture enables straightforward deployment and future enhancements. This prototype provides a solid foundation for real-world clinical decision support and can be further extended with additional data, advanced models, or integration into healthcare workflows.

10 Appendix

11 References

[1] H. Kinsley, "Reinforcement Learning," PythonProgramming, [Online]. Available: https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/. [Accessed 02 02 2021].

[2] [Online].

[3] MakeSigns, "Scientic Posters Tutorial," [Online]. Available: https://www.makesigns.com/tutorials/scientific-poster-parts.aspx. [Accessed 09 02 2021].

[4] Arduino. [Online]. Available: https://www.arduino.cc/. [Accessed 09 02 2021].

[x] J. C. Russ and F. Brent Neal. *The Image Processing Handbook*. 7th ed. Boca Raton FL: CRC Press, 2017.

[x] S. Lippman, J. Lajoie and B.E. Moo. “Classes” in *C++ Primer*. 5th ed. Massachusetts: Addison Wesley, 2013.

[x] IEEE Signal Processing Society. “Signal Processing for 5G”, *YouTube*, 2019. [Online]. Available: https://www.youtube.com/watch?v=uca6X4Ykcmg&list=PLcZOnmyqlalacL9YqkhyufLQGIW_C78Os&index=6. Accessed: Feb 2, 2021.

[x] Digilent. “Basys 3 Reference Manual”, *Digilent Reference*. [Online]. Available: https://reference.digilentinc.com/basys3/refmanual. Accessed: Feb 2, 2021.

[x] P. J. Ashenden. *Digital Design (Verilog): An Embedded Systems Approach Using Verilog*. Burlington: Morgan Kaufmann, 2007.

[x] M. Lynch. “Discrete Fourier Transform (DFT)”, Lecture, Digital Signal Processing, Galway-Mayo Institute of Technology, Galway, 2020.

[x] MRC Centre for Global infectious Disease Analysis. (2021). *Covid-Sim*. [Source Code]. Available: https://github.com/mrc-ide/covid-sim. Accessed: Feb 2, 2021.

[x] OpenCV. “Face Detection Using Haar Cascades”, *OpenCV-Python Tutorials*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection. Accessed: Feb 2, 2021.