*A project report on*

**SKIN CONDITION DIAGNOSIS USING CNN**

*Submitted in partial fulfillment for the award of the degree of*

## BTECH - Computer Science and Engineering (Artificial Intelligence and Machine Learning)

*by*

**CHERUKURI PAVAN SATISH-21BCE9357**

**VADLAPUDI KALYAN HANUMAN CHOWDARY-** **21BCE9180**

**MARISARLA YATHEESWAR-** **21BCE9094**

Text, logo

Description automatically generated

**SCHOOL OF COMPUTER SCIENCE & ENGINEERING**

May, 2025

**DECLARATION**

I here by declare that the thesis entitled **“Skin condition diagnosis using CNN”** submitted by me, for the award of the degree of Specify the name of the degree VIT is a record of Bonafide work carried out by me under the supervision of Dr. Koduru Hajarathaiah.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

**Signature of the Candidate**

**CERTIFICATE**

This is to certify that the Senior Design Project titled “**Fingerprint-Based Gender and Hand Classification with Voice Authenticity Detection**” that is being submitted by **CHERUKURI PAVAN SATISH(21BCE9357),VADLAPUDI KALYAN HANUAMNA CHOWDARY(21BCE9180),MARISARLA YATHEESWAR(21BCE9094)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. Koduru Hajarathaiah

Guide

**The thesis is satisfactory / unsatisfactory**

**Internal Examiner External Examiner**

**Approved by**

**PROGRAM CHAIR DEAN**

B. Tech. CSE School Of Computer Science & Engineering

**ABSTRACT**

Skin diseases affect a large portion of the global population and can range from benign conditions to life-threatening melanomas. Early detection and accurate diagnosis are essential to ensure timely treatment and reduce the risk of complications. However, access to dermatologists is limited in many parts of the world, which highlights the need for automated diagnostic tools.

Convolutional Neural Networks (CNNs) have shown great potential in image-based medical diagnostics due to their ability to automatically learn relevant features from raw image data. In this study, we explore the application of CNNs for diagnosing common skin conditions using both clinical and dermatoscopic images.

A custom CNN architecture was developed and trained on a labeled dataset comprising images of various skin conditions such as eczema, psoriasis, acne, benign nevi, and melanoma. Data augmentation and preprocessing techniques were applied to improve model performance and robustness across diverse skin tones and image qualities.

The model was evaluated using standard metrics including accuracy, precision, recall, and F1-score. Results indicate that the CNN performs comparably to dermatologists in recognizing certain conditions, particularly in distinguishing between benign and malignant lesions.

This research demonstrates the feasibility of using deep learning for automated skin disease diagnosis. It has the potential to support clinical decision-making, enhance teledermatology services, and increase access to dermatological care, especially in underserved areas.

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude to Dr. Koduru Hajarathaiah ,

Assistant Professor Senior Grade, School of Computer Science & Engineering,VIT-AP, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of machine learning and Deep learning

I would like to express my gratitude to Dr. G. Viswanathan, <VPs>, Dr. S. V. Kota Reddy, and Dr.S.Sudhakar Ilango, School of Computer Science & Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to <Program char- name>. <Program Chair designation>, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Cherukuri Pavan Satish

*ii*

## CONTENTS

[CONTENTS iii](#_TOC_250001)

[LIST OF FIGURES iv](#_TOC_250000)

LIST OF TABLES v

LIST OF ACRONYMS vi

CHAPTER 1 INTRODUCTION

* 1. INTRODUCTION 1
  2. OVERVIEW 3
  3. CHALLENGES 6
  4. PROJECT STATEMENT 10
  5. OBJECTIVES 13
  6. SCOPE OF THE PROJECT 16

**LIST OF FIGURES**

**LIST OF ACRONYMS**

CNN: Convolutional Neural Network

SVM: Support Vector Machine

Eczema Prediction

Psoriasis Prediction Kaggle Dataset

Machine Learning in Dermatology

KNN: K-Nearest Neighbours

*vi*

**Chapter 1**

**Introduction**

Skin diseases are a significant public health concern that can significantly impact a person’s quality of life. Eczema and psoriasis are two of the most prevalent chronic skin conditions that affect millions of individuals worldwide. Eczema, also known as atopic dermatitis, is a chronic skin condition characterized by red, itchy, and inflamed skin. It affects people of all ages, but it is more common in children. Genetics, environmental factors, stress, and allergies are just a few of the factors that can cause eczema. While there is no cure for eczema, it can be managed with proper treatment, including topical creams, oral medications, and lifestyle changes.

Psoriasis is another chronic skin condition that affects millions of individuals worldwide. It is characterized by red, scaly patches on the skin that can be itchy and painful. Psoriasis is believed to be an autoimmune disorder, in which the immune system attacks healthy skin cells. The exact cause of psoriasis is unknown, but genetics and environmental factors are believed to play a role. Like eczema, there is no cure for psoriasis, but it can be managed with proper treatment, including topical creams, oral medications, and light therapy. Both eczema and psoriasis can significantly impact a person’s quality of life, causing physical discomfort and emotional distress. Early detection and diagnosis of these conditions are crucial to improving treatment outcomes and reducing the associated healthcare costs. Dermatologists often rely on their clinical expertise to diagnose these diseases, which can be time-consuming and subject to human error. Machine learning techniques, such as deep learning, have shown great promise in aiding the early detection and diagnosis of skin diseases like eczema and psoriasis.

Despite the growing interest in using machine learning techniques, including deep learning, for skin disease detection, previous work in this field faces several challenges and limitations. One of the significant challenges is the lack of a large and diverse dataset of skin images for training and testing machine learning models. The limited availability of high-quality and annotated skin images can hinder the development and evaluation of accurate and robust machine learning models. Moreover, traditional machine learning algorithms can struggle to handle image-based data due to their high-dimensionality and complex nature. This can lead to suboptimal performance in skin disease detection tasks, especially when dealing with complex skin diseases such as eczema and psoriasis. Another significant limitation of previous work in this field is the lack of models that can detect multiple skin diseases simultaneously. Most existing methods focus on detecting a single skin disease, which can be time-consuming and inefficient, especially in clinical settings.

Automatic detection of skin diseases, such as eczema and psoriasis, is essential for assisting dermatologists in diagnosing these conditions at their initial stages. Early detection allows for timely intervention, leading to improved patient outcomes and a better quality of life for those affected.

In addition to Convolutional Neural Networks (CNNs), traditional machine learning algorithms like Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) have also proven effective for image-based classification tasks. CNNs are particularly adept at handling image data by automatically extracting relevant features through convolution operations, enabling the identification of significant patterns, such as skin texture and lesion appearance. SVM, on the other hand, is effective for binary and multi-class classification by finding the optimal hyperplane that separates different classes of skin diseases. KNN works by comparing new input images to the closest examples in the dataset, using proximity-based classification.

Each of these models—CNN, SVM, and KNN—contributes unique strengths to the task of skin disease detection. By leveraging these models, this research aims to provide a comprehensive approach to classifying eczema and psoriasis, improving diagnostic accuracy and offering valuable support to dermatologists.

**Overview**

Human skin disease prediction is a rapidly evolving field that merges the power of medical science with modern technologies such as artificial intelligence (AI), machine learning (ML), and image processing. Skin diseases are among the most common health problems worldwide and can range from temporary and mild conditions to chronic and life-threatening illnesses. Some of the most prevalent skin diseases include acne, eczema, psoriasis, fungal infections, and more serious diseases such as melanoma and non-melanoma skin cancers. Early and accurate diagnosis is critical for effective treatment, especially in conditions like skin cancer where delayed diagnosis can lead to severe complications or death. However, diagnosing skin diseases is often challenging because many conditions exhibit similar visual symptoms, and their appearance can vary depending on the individual’s age, skin tone, and environment. This complexity has led to the increasing interest in using technology to assist in predicting and diagnosing skin conditions with higher accuracy and efficiency.

The application of artificial intelligence in dermatology, particularly in skin disease prediction, has shown significant promise in recent years. By leveraging vast datasets and advanced computing techniques, AI systems can learn to recognize patterns and features in skin images that may not be easily noticeable to the human eye. Among the various AI methods, deep learning—specifically Convolutional Neural Networks (CNNs)—has been widely used due to its superior performance in image classification tasks. These models are trained on large collections of labeled skin images, where each image is annotated with the correct diagnosis. As the model is exposed to more data, it learns to identify key features such as lesion size, shape, color, texture, and distribution, which are critical in distinguishing between different skin diseases.

For the success of any AI-driven skin disease prediction system, access to high-quality and diverse datasets is essential. Well-known datasets such as **HAM10000**, **ISIC (International Skin Imaging Collaboration)**, and **DermNet** provide thousands of clinical and dermoscopic images covering a broad spectrum of skin conditions. These datasets also contain important metadata including patient age, sex, lesion location, and diagnostic confirmation from dermatologists. Before these images are used for training machine learning models, they undergo preprocessing steps such as resizing, normalization, contrast adjustment, and augmentation to enhance image quality and increase the dataset’s variability. Augmentation techniques, like flipping, rotating, or adding noise to images, help in reducing overfitting and making the models more robust to real-world variations.

One of the key benefits of AI-based skin disease prediction systems is their ability to make healthcare more accessible. With the integration of these models into mobile applications, individuals can now use their smartphones to take pictures of skin anomalies and receive immediate analysis. These tools are particularly useful in remote or rural areas where access to dermatologists is limited. Moreover, these systems can serve as decision-support tools for general practitioners, helping them to refer patients to specialists when needed or to rule out benign conditions that do not require urgent care.

Despite the remarkable progress in this field, several challenges remain. A major issue is **dataset bias**—many skin disease datasets are disproportionately composed of images from individuals with lighter skin tones. This lack of diversity can lead to significant drops in performance when models are used on patients with darker skin. Another problem is **symptom similarity**, as many skin conditions share overlapping visual features that can confuse both human doctors and AI systems. Additionally, most deep learning models are often considered "black boxes" because they do not explain the reasoning behind their predictions, which makes it difficult for medical professionals to fully trust and adopt these tools in clinical practice. There are also important **ethical and legal concerns**, especially regarding patient data privacy, consent, and the potential misuse of AI in diagnosis without proper human oversight.

Looking to the future, human skin disease prediction is expected to become more accurate and comprehensive. Researchers are working on combining image data with other clinical information, such as patient history, genetic profiles, lifestyle factors, and even environmental exposure to improve prediction models. New advancements such as **explainable AI (XAI)** aim to make machine learning models more transparent and interpretable, which can build trust among healthcare providers. There is also growing interest in developing **real-time diagnostic tools** using wearable devices and IoT (Internet of Things) technology that can monitor skin health continuously and alert users to early signs of skin problems.

In conclusion, human skin disease prediction stands at the intersection of medicine and technology, offering the potential to revolutionize dermatological care. With continued advancements in AI, access to more representative datasets, and collaboration between technologists and healthcare professionals, these tools can greatly enhance the speed, accuracy, and accessibility of skin disease diagnosis. While there are still obstacles to overcome, the integration of predictive technologies into dermatology marks a significant step toward personalized and preventive healthcare for all.

**1.3 Challenges**

**1.3.1 . Lack of Diversity in Datasets**

One of the most pressing challenges in human skin disease prediction is the lack of diversity in the datasets used to train AI models. Many widely used skin image datasets, such as HAM10000 and ISIC, primarily consist of images from individuals with lighter skin tones (Fitzpatrick skin types I–III). This imbalance causes AI models to perform poorly when applied to people with darker skin tones (Fitzpatrick types IV–VI), leading to inaccurate diagnoses and a higher risk of missed or misclassified conditions. Certain skin diseases manifest differently across skin tones—conditions like psoriasis, melanoma, and eczema may appear with varying colors, patterns, or inflammation levels, which the model may not recognize properly if not adequately trained.

This bias can contribute to healthcare disparities, especially in countries with ethnically diverse populations. When AI models fail to generalize across different skin tones, it undermines their reliability and clinical usefulness. Overcoming this challenge requires active efforts in collecting more inclusive and balanced datasets that represent the full spectrum of human skin diversity. Collaborative global data-sharing initiatives and contributions from underrepresented regions can help close this gap and ensure equitable healthcare support.

**1.3.2. Visual Similarity Between Different Skin Conditions**

Skin diseases often present with overlapping visual features, making it difficult even for trained dermatologists to differentiate between them—this challenge also extends to AI systems. For example, benign moles can sometimes resemble melanoma, and psoriasis can appear similar to eczema or fungal infections in early stages. These visual similarities can confuse AI models, especially when they rely heavily on external features such as color, shape, size, and texture without deeper clinical context.

Moreover, variations in lighting, camera quality, and angle at which images are taken can further distort how skin conditions appear in photographs, adding to the confusion. The AI model may also struggle with detecting diseases in images that include obstructions like hair, shadows, or partial views. To improve accuracy, AI systems must be trained with highly curated and labeled datasets that include a wide variety of presentations for each disease and incorporate techniques like multi-modal learning, which combines image analysis with patient history, symptoms, and medical records.

**1.3.3. Black-Box Nature and Lack of Interpretability**

Many deep learning models used for skin disease prediction, especially Convolutional Neural Networks (CNNs), function as "black boxes." This means that while they may produce accurate predictions, they do not provide explanations or reasoning that is easily interpretable by humans, including medical professionals. In a medical setting, trust and transparency are crucial—doctors need to understand how a conclusion was reached before relying on it to make critical healthcare decisions.

This lack of interpretability poses a significant barrier to clinical adoption. Medical practitioners are understandably hesitant to use AI-generated diagnoses without insights into how or why the system arrived at its conclusion. Explainable AI (XAI) is a growing area of research that aims to address this issue by creating models that offer visual maps, heatmaps (like Grad-CAM), or text-based explanations alongside predictions. However, creating highly accurate yet interpretable models remains a difficult trade-off. Until AI predictions can be transparently justified, their use in high-stakes medical environments will continue to face resistance.

**1.3.4. Privacy, Ethics, and Regulatory Barriers**

Privacy concerns represent a serious obstacle in deploying skin disease prediction technologies, especially when patient images are used. Skin images are considered sensitive personal data, and improper handling can lead to privacy violations and ethical concerns. If patient data is shared without proper anonymization or consent, it can result in legal issues, breach of trust, and reputational damage for developers and healthcare institutions alike.

Moreover, ethical considerations arise when AI is used for autonomous diagnosis, especially in consumer-facing applications. The risk of false positives (predicting a disease where there is none) or false negatives (failing to detect an actual condition) can have serious health consequences. Without regulatory oversight, there’s a danger that unreliable or unapproved apps could mislead users into either ignoring real issues or panicking over harmless skin conditions.

In addition, skin disease prediction systems must comply with healthcare regulations like HIPAA in the U.S. or GDPR in Europe, which impose strict controls on data collection, storage, and sharing. Gaining clinical approval (such as FDA clearance) is another complex and lengthy process that requires extensive validation studies and documentation. All of these factors slow down the deployment of AI in real-world healthcare environments and add layers of complexity to product development.

**1.4 Problem Statement**

Skin diseases constitute a significant portion of global health concerns, affecting millions of individuals regardless of age, gender, or ethnicity. These conditions can vary in severity from mild irritation to life-threatening illnesses like melanoma. Early detection and appropriate classification of skin disease severity are vital for effective treatment, better patient outcomes, and reduced healthcare burdens. However, manual diagnosis by dermatologists is often subjective, time-consuming, and prone to errors—especially in remote or resource-limited regions where access to specialists is limited.

To address these challenges, this project proposes an automated skin disease prediction system using **Convolutional Neural Networks (CNN)**, a powerful deep learning technique widely used in image classification tasks. The system aims to analyze dermatoscopic or clinical images of affected skin areas and accurately classify them into **six distinct severity categories**:

1. **Very Mild**
2. **Mild**
3. **Moderate**
4. **Severe**
5. **Very Severe**
6. **Critical**

The primary objective is to not only detect the presence of a skin disease but also assess the **severity level**, enabling healthcare professionals to prioritize treatment based on risk and urgency. The CNN model will be trained on a labeled dataset comprising skin images annotated with both disease type and severity level. By learning visual patterns and features from the data, the model can generalize to unseen cases and provide a fast, reliable prediction.

This automated approach is expected to reduce diagnostic delays, support dermatologists with second opinions, and make basic skin disease assessment accessible to users through mobile or web-based platforms. Moreover, severity-level classification can help in monitoring disease progression and guiding treatment strategies, thereby enhancing overall patient care.

**Objective and Scope:**

The main objective of this project is to develop an intelligent and automated skin disease prediction system using **Convolutional Neural Networks (CNN)** to classify skin conditions based on severity levels. The model aims to:

* Analyze clinical or dermatoscopic images of skin lesions with high accuracy.
* Classify the detected skin disease into **six predefined severity levels**: *Very Mild, Mild, Moderate, Severe, Very Severe,* and *Critical*.
* Assist medical professionals by providing a reliable second opinion to support diagnosis and treatment planning.
* Improve accessibility to basic dermatological assessment in remote or under-resourced areas via mobile or web-based platforms.
* Reduce diagnostic time and minimize human error in early-stage detection and severity evaluation.

This project focuses on the application of deep learning techniques, particularly CNNs, to predict the presence and **severity of skin diseases** from image data. The scope of the project includes:

* **Data Collection & Preprocessing**: Utilizing publicly available and/or custom datasets containing labeled images of skin conditions with annotated severity levels.
* **Model Development**: Designing and training a CNN model capable of learning visual features from the dataset to perform multi-class classification into six severity levels.
* **Performance Evaluation**: Assessing the model’s performance using standard metrics such as accuracy, precision, recall, F1-score, and confusion matrix analysis.
* **Deployment Potential**: Exploring the integration of the model into a prototype mobile or web application for real-time skin disease prediction.
* **Limitations**: The system is limited to image-based classification and may not incorporate additional clinical inputs such as patient history or genetic factors. Also, it does not replace professional medical diagnosis but serves as a decision-support tool.

**1.5 Objectives**

In this study, we aim to develop robust machine learning models for the detection of eczema and psoriasis using a dataset sourced from Kaggle. We experiment with three prominent models: Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Convolutional Neural Networks (CNN). These models are fine-tuned and optimized to achieve the best performance in identifying and classifying skin diseases, specifically focusing on eczema and psoriasis. By leveraging these diverse models, we aim to improve diagnostic accuracy and provide valuable support for early detection and clinical decision-making in dermatology. particularly in the field of dermatology. Numerous studies have focused on the detection of skin diseases using various algorithms, highlighting the potential of these methods to enhance diagnostic accuracy and efficiency.

1. Machine Learning in Dermatology  
A study by Esteva et al. (2017) demonstrated the efficacy of deep learning models in diagnosing skin cancer, showing that convolutional neural networks could match or even exceed the diagnostic performance of dermatologists. This pioneering work set the stage for further exploration of machine learning applications in dermatological conditions beyond cancer.

2. Eczema Detection  
Research by Karamizadeh et al. (2020) focused on the automatic detection of eczema using image processing techniques combined with machine learning classifiers such as SVM and KNN. The study illustrated that these algorithms could effectively classify eczema images based on texture and color features, although the need for larger datasets was emphasized to improve model robustness.

3. Psoriasis Classification  
Similarly, Wang et al. (2018) explored the use of CNNs for the classification of psoriasis lesions in dermoscopic images. Their findings indicated that deep learning models, particularly CNNs, achieved high accuracy rates in differentiating between various types of psoriasis, underlining the importance of feature extraction capabilities inherent in these networks.

4. Comparative Studies  
Recent studies have compared the performance of different machine learning models in dermatological applications. For instance, a study by Ali et al. (2021) evaluated SVM, KNN, and CNN models for various skin diseases, concluding that while CNNs generally provided superior performance due to their ability to learn complex patterns, traditional algorithms like SVM and KNN were effective in specific scenarios where computational resources were limited.

**1.6 Scope**

The scope of this project encompasses the research, development, and testing of an artificial intelligence-based system designed to predict human skin disease severity using Convolutional Neural Networks (CNNs). With the growing prevalence of skin disorders globally, and the limited access to specialized dermatological care in many regions, there is a pressing need for technological solutions that can assist in early detection and severity assessment. This system aims to bridge that gap by offering an automated tool capable of analyzing skin lesion images and classifying them into six defined severity categories: *Very Mild, Mild, Moderate, Severe, Very Severe,* and *Critical*. Each of these categories reflects a different stage in the progression of a skin condition, thereby enabling targeted and timely intervention.

At its core, this project focuses on leveraging deep learning for the purpose of image-based skin analysis. CNNs have proven to be exceptionally effective for tasks involving visual recognition, and in this context, they serve as the primary tool for learning patterns and features present in dermatoscopic or clinical skin images. The system will be trained using a large dataset of skin lesion images that are properly labeled with the corresponding severity level. These images will undergo preprocessing steps such as normalization, resizing, contrast enhancement, and augmentation to ensure that the model can generalize well across various skin tones, lighting conditions, and image qualities.

A significant part of the project’s scope includes the design and optimization of the CNN architecture. This involves experimenting with different layers, filter sizes, activation functions, and optimization algorithms to achieve the best possible performance. The final model will be expected to classify input images into one of the six severity levels with a high degree of accuracy and minimal false positives or negatives. Alongside the development of the predictive model, the scope also includes performance evaluation using statistical measures such as accuracy, precision, recall, F1-score, and confusion matrix. These metrics will be used to validate the reliability and robustness of the system under various test conditions.

Another crucial aspect covered within the scope of this project is the system's potential for real-world application. While the core focus remains on model development and classification performance, the broader vision includes implementing the trained model into a user-friendly interface—either through a mobile app or a web platform. Such integration will allow users, especially in remote or underserved regions, to take photos of skin lesions and receive instant feedback on the probable severity of the condition. This could prove invaluable for triage purposes, guiding patients on whether to seek immediate medical attention or monitor the condition.

Furthermore, the scope addresses the ethical, clinical, and technical limitations of the proposed system. The model is designed strictly for educational and supportive use, not as a replacement for professional dermatological evaluation. It must be clearly communicated that the tool offers a prediction based on image data alone, without considering other clinical factors such as family history, previous conditions, allergies, or comorbidities. Additionally, there is an inherent limitation in relying solely on image datasets, especially if those datasets are not fully representative of global skin tone diversity or include mislabeled data. Measures such as dataset augmentation, quality filtering, and future dataset expansion are proposed to mitigate these issues.

Finally, the scope includes future enhancements and directions for expansion. This includes integrating patient metadata (age, gender, symptoms, history) to enable multimodal prediction, applying transfer learning techniques to reduce training time while improving model accuracy, and exploring the use of explainable AI to provide interpretability to the predictions made by the CNN model. These future advancements, though outside the initial implementation phase, are aligned with the broader goal of creating a comprehensive, scalable, and ethically sound AI-based skin disease prediction tool.

**Results**

1. **Hand (Left or Right) Classification Using Fingerprints**

**VGG16:**

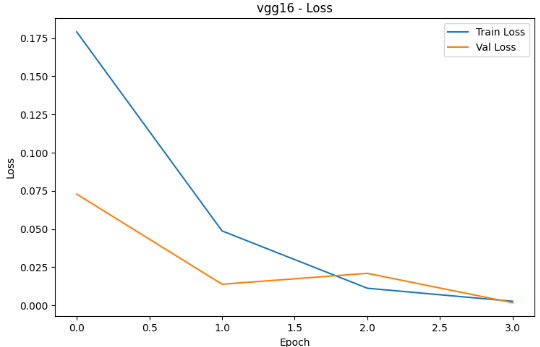
****

Fig 1: vgg16 loss curve

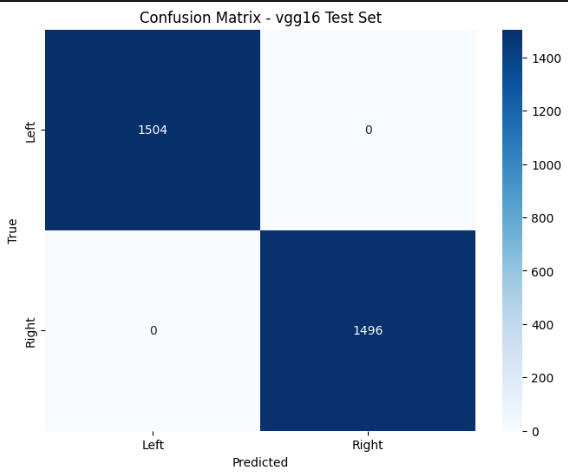
****

Fig 2: VGG16 confusion matrix test set

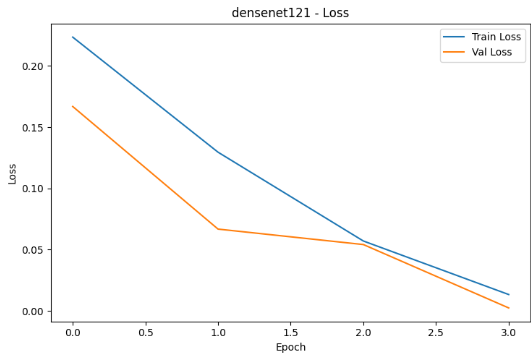
**Densenet121:  
**

Fig 3: Densenet121 loss curve

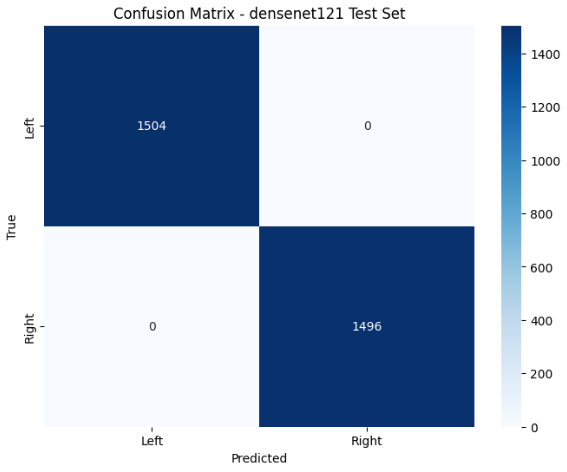
****

Fig 4: DenseNet121 confusion matrix

**EfficientNetB0**

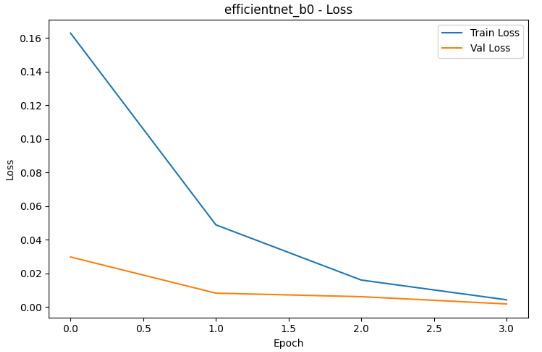
****

Fig 5: efficientnet\_b0 loss curve

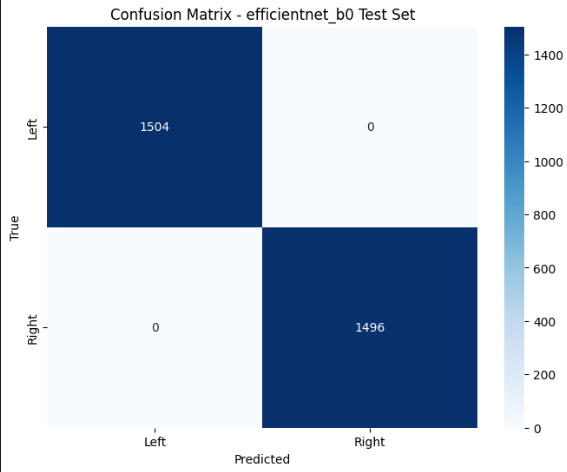
****

Fig 6: efficientnet\_b0 confusion matrix

**Resnet50**

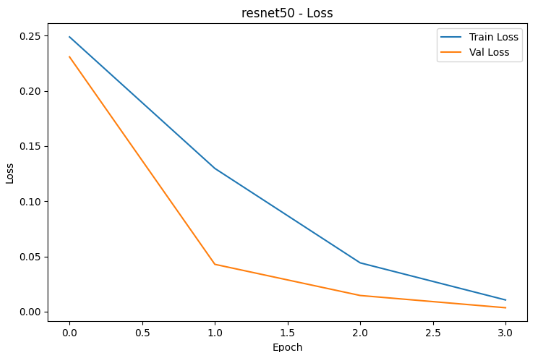


Fig 7: Resnet50 loss curve

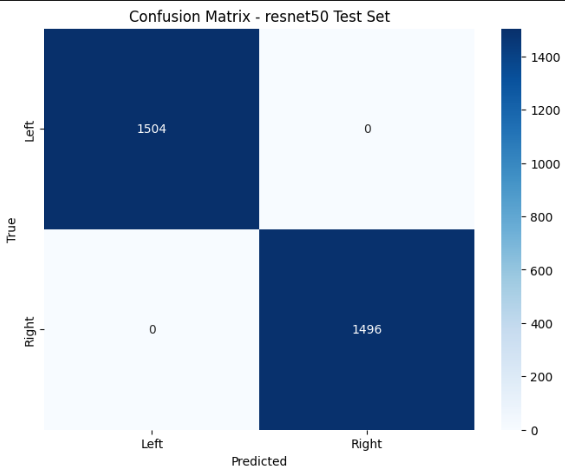


Fig 8: Resnet50 confusion matrix

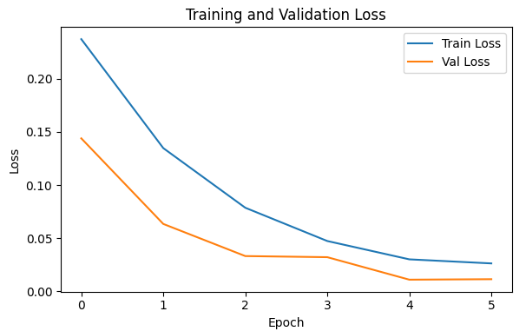
**Best model : Resnet50  
  
**

Fig 9: Resnet50(best model) training and validation loss curve

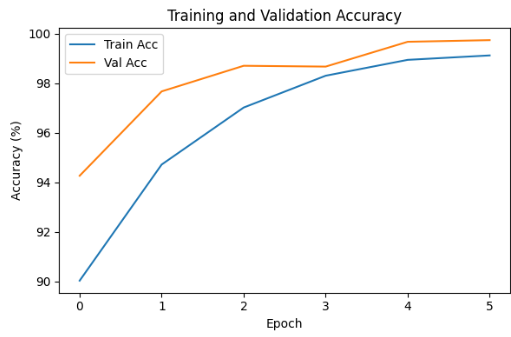
****

Fig 10: Resnet50 training and validation accuracy curve

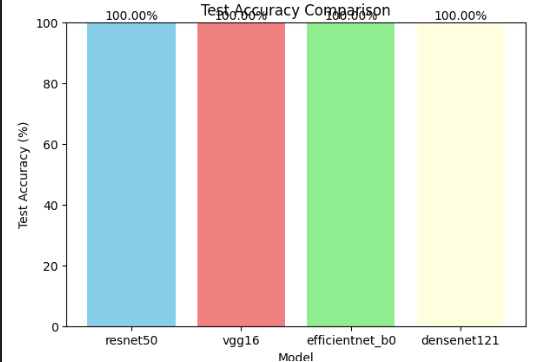


Fig 11: Test accuracy model comparison

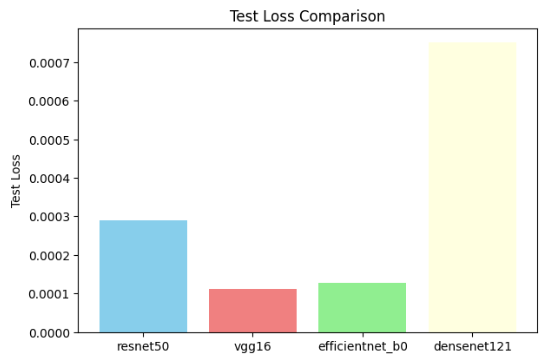


Fig 12: Test loss model comparison

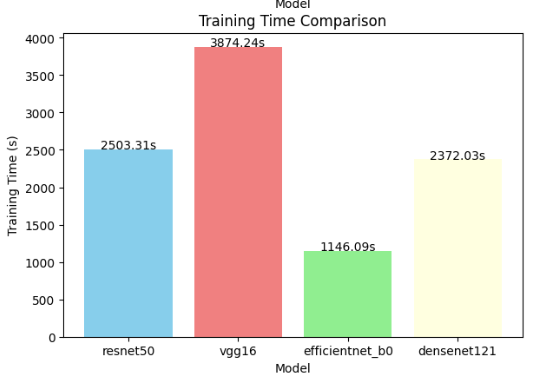


Fig 11: Training time model comparison

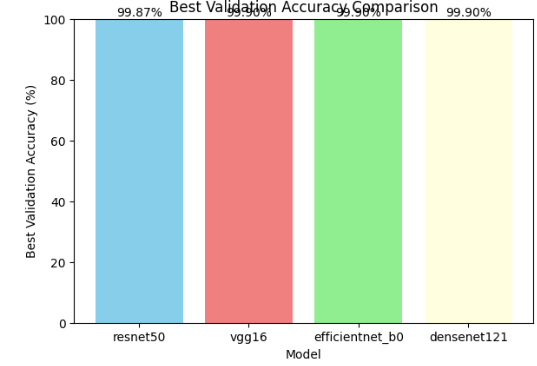


Fig 11: Best validation accuracy model comparison

* 1. **Gender Classification Using Fingerprints**

**Before solving vanishing gradient problem**

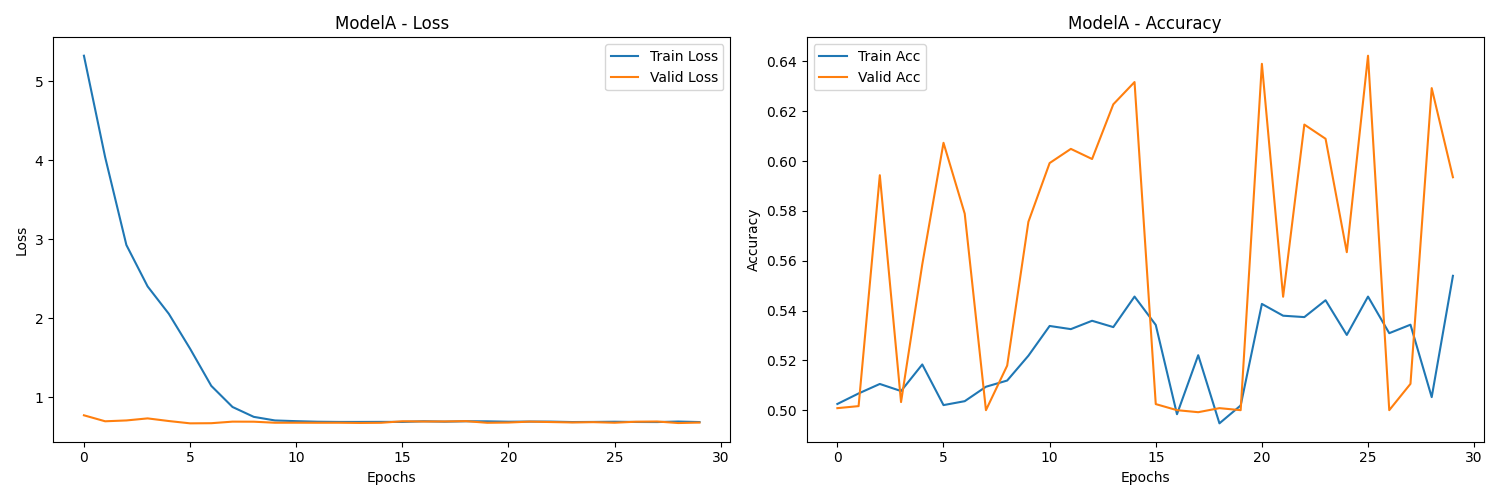


Fig 12: Model A metrics

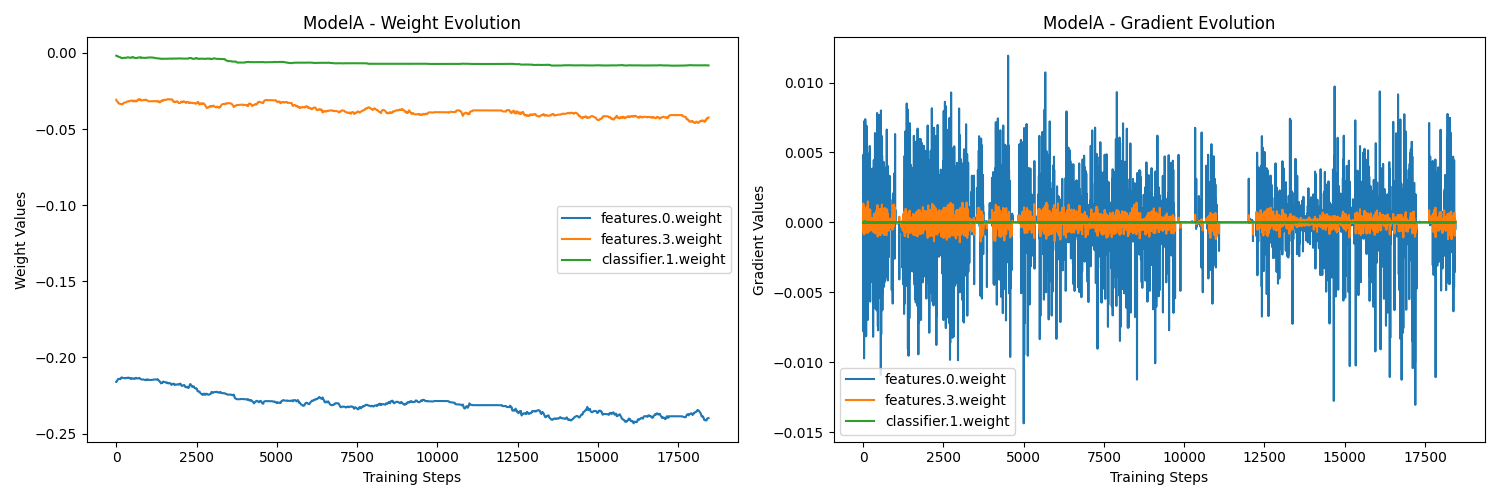


Fig 13: Model A training problems

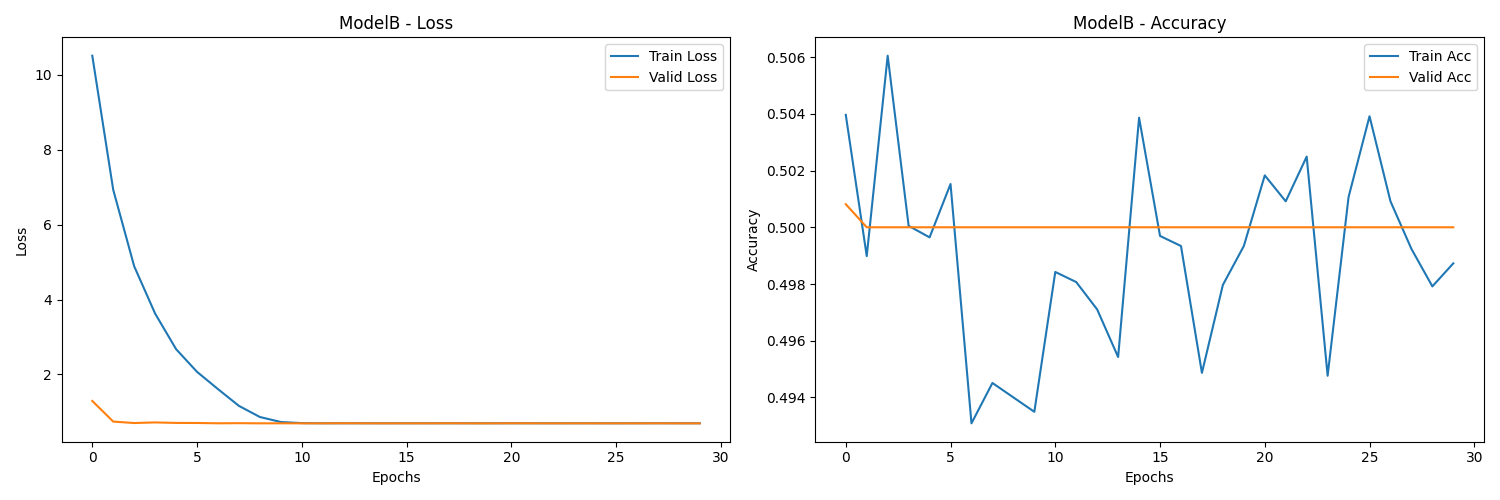


Fig 14: Model B metrics

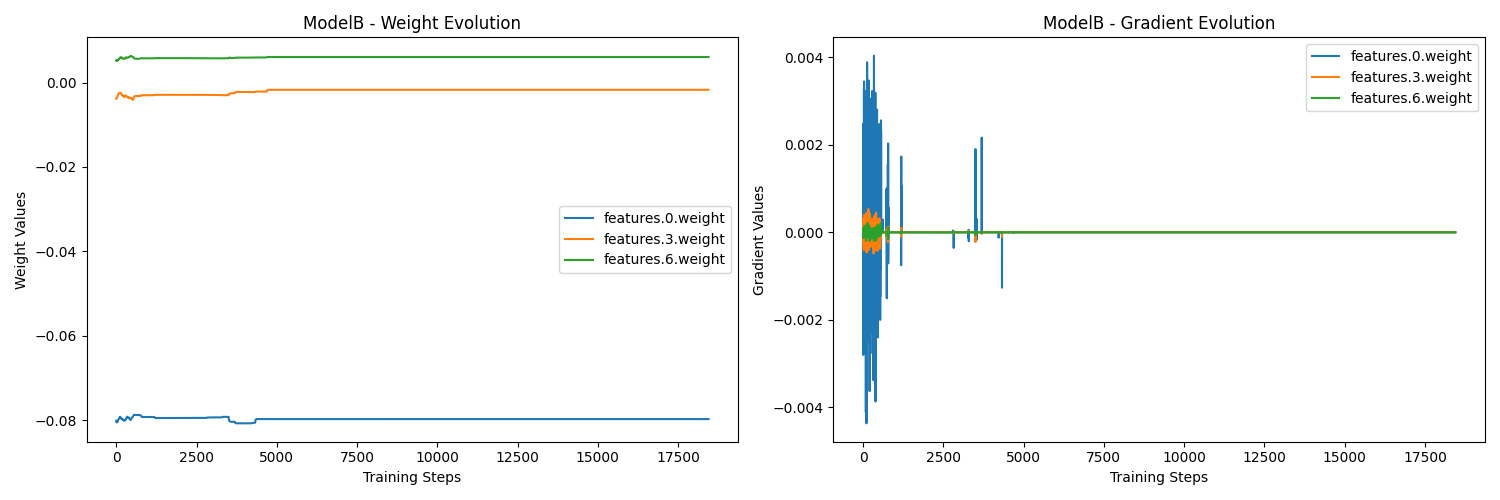


Fig 15: Model B training problems

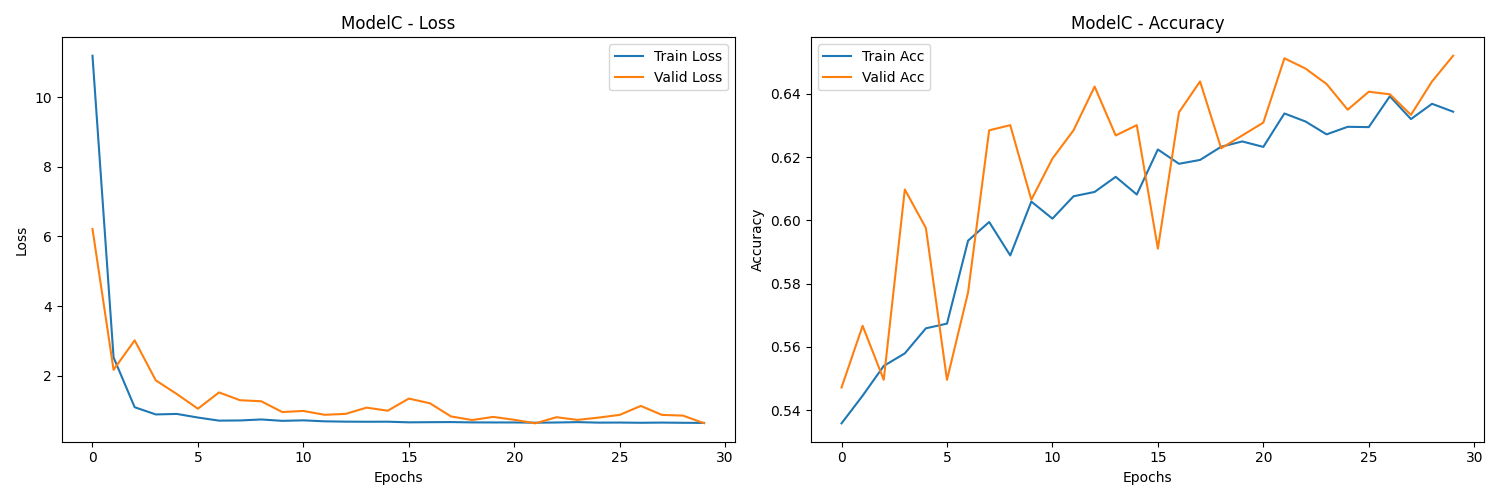


Fig 16: Model C metrics

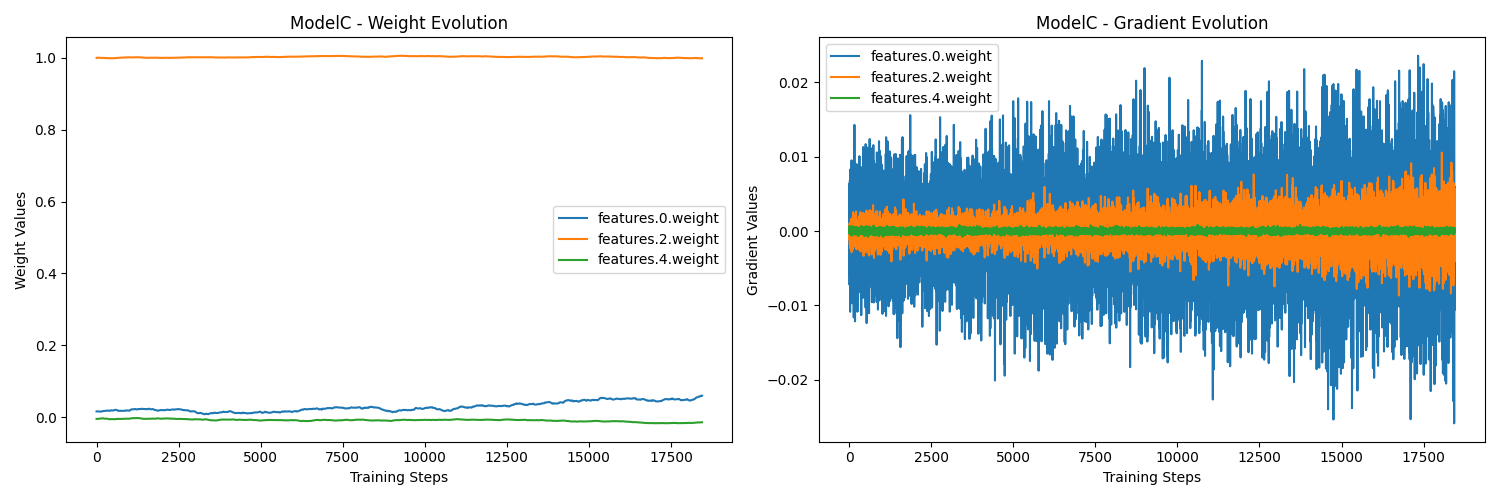


Fig 17: Model C training problems

**2.2 Overcoming VGP using BN and ReLU activation function and Early Stopping**

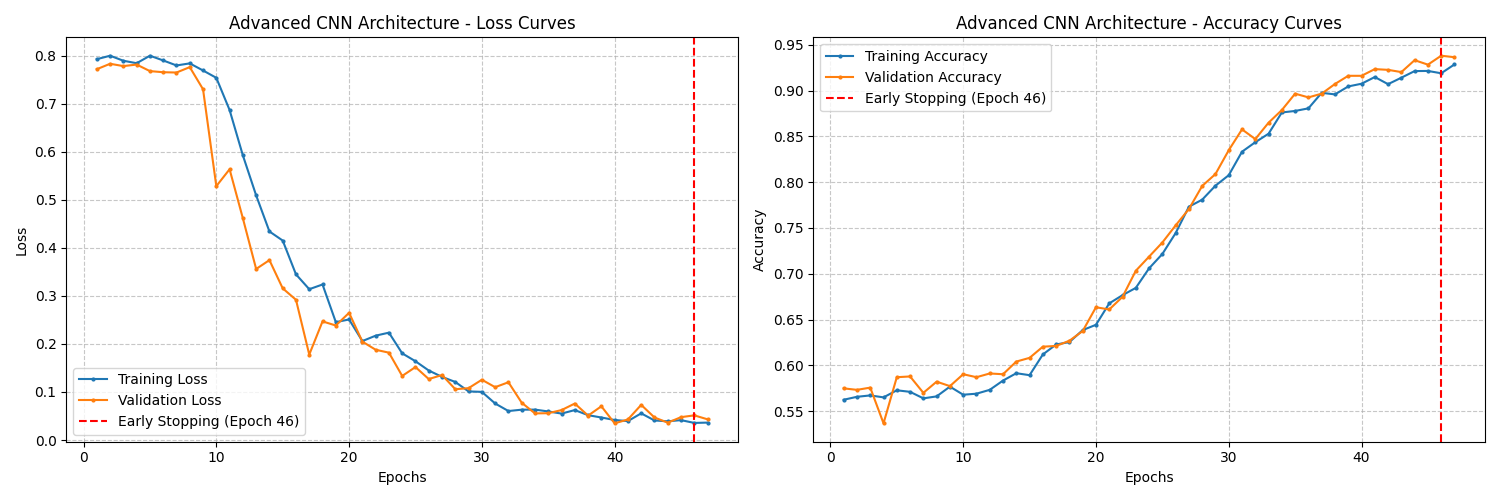
****

Fig 18: Advanced CNN metrics

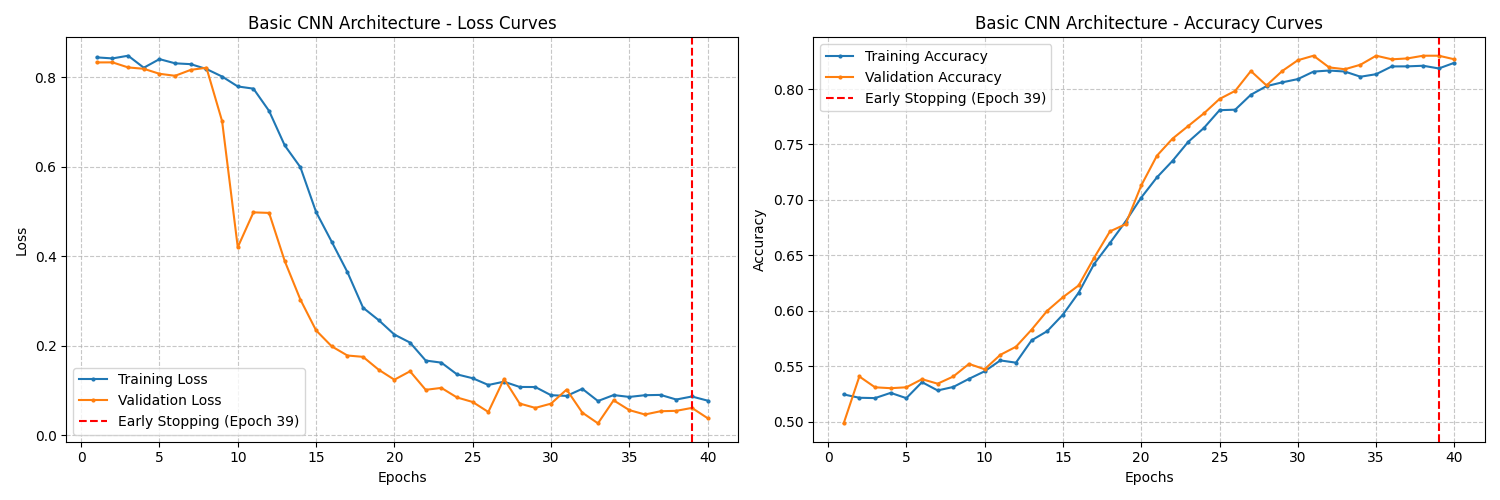
****

Fig 19: Basic CNN metrics

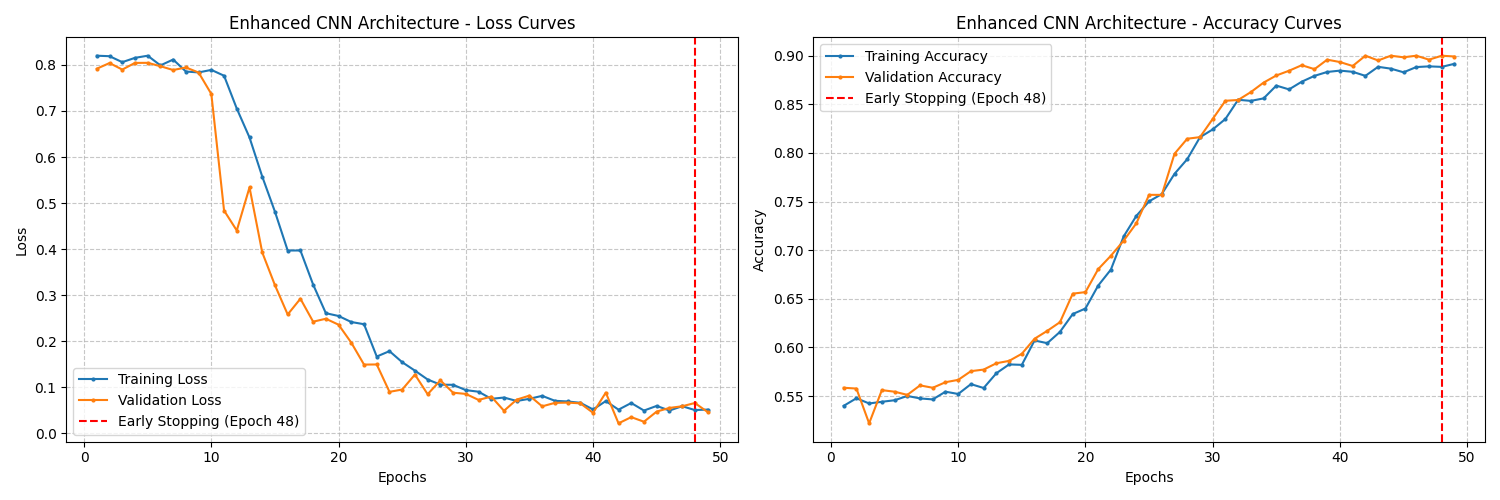
****

Fig 20: Enhanced CNN metrics

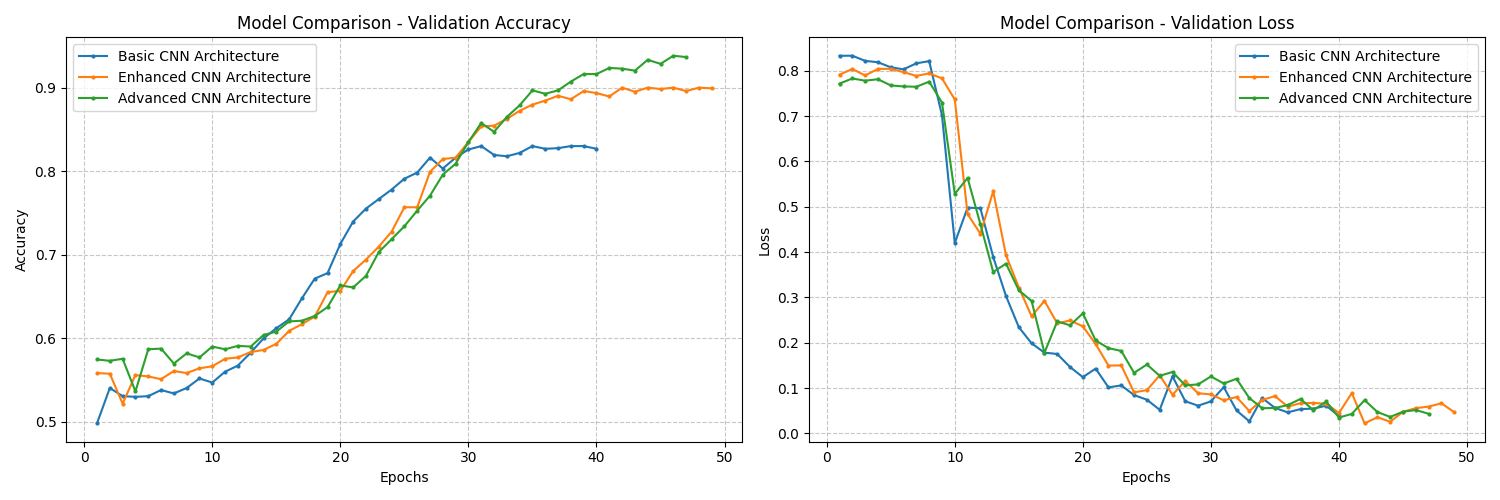
****

Fig 21: Model comparison

1. **Fake vs. Real Audio Detection**

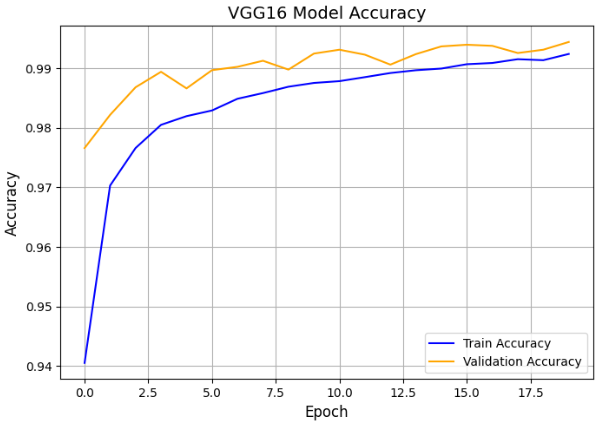
****

Fig 22: VGG16 model accuracy curve

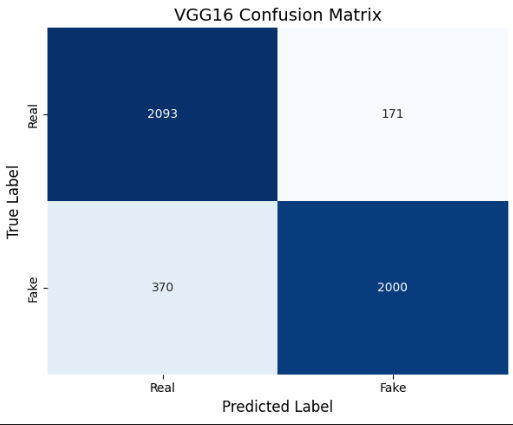
****

Fig 23: VGG16 confusion matrix

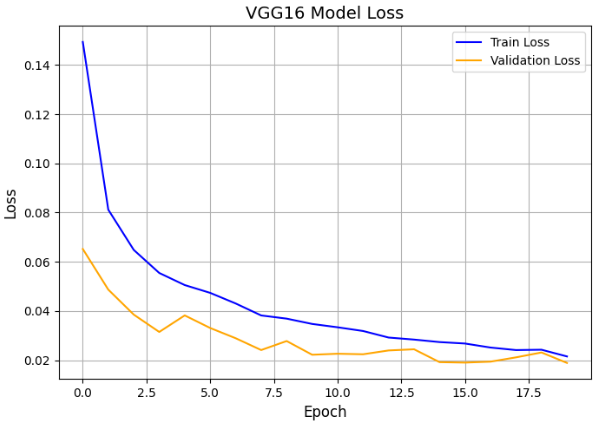
****

Fig 24: VGG16 model loss curve

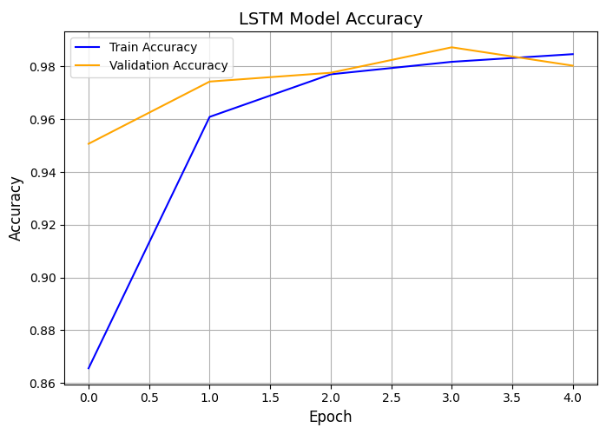


Fig 25: LSTM model accuracy

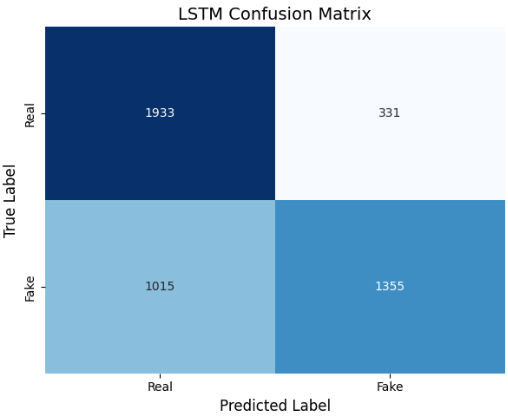


Fig 26: LSTM confusion matrix

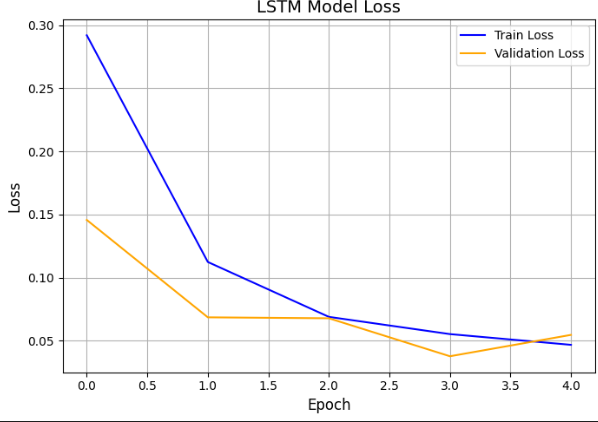


Fig 27: LSTM model loss curve

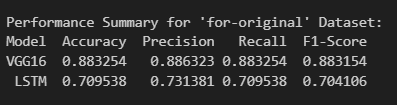


Fig 28: Summary of Fake vs. Real Audio Detection

**Tables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model name** | **Accuracy(%)** | **Precision(%)** | **Recall(%)** | **F1-Score(%)** |
| VGG16 | 88.33 | 88.63 | 88.33 | 88.32 |
| LSTM | 70.95 | 73.14 | 70.95 | 70.41 |

Table 1: Deepfake Audio Detection Model Performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Test accuracy(%)** | **Test loss(%)** | **Best validation accuracy(%)** |
| Resnet50 | 99 | 0.0003 | 99.87 |
| VGG16 | 97 | 0.0001 | 99.90 |
| Efficientnet\_b0 | 97 | 0.0001 | 99.90 |
| Densenet121 | 95 | 0.0008 | 99.90 |

Table 2: Hand detection Model Performance

**Codes**

**Hand (Left or Right) Classification Using Fingerprints**

# Cell 1: Import Libraries

import os

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.models as models

import torchvision.transforms as transforms

from torch.utils.data import DataLoader

from torchvision import datasets

import matplotlib.pyplot as plt

import numpy as np

from tqdm import tqdm

from PIL import Image

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import time

# Cell 2: Preprocessor Class

class FingerprintPreprocessor:

def \_\_init\_\_(self, img\_size=(224, 224)):

self.img\_size = img\_size

self.transform = transforms.Compose([

transforms.Resize(img\_size),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

])

def preprocess\_image(self, image\_path):

image = Image.open(image\_path).convert('RGB')

return self.transform(image)

# Cell 3: Trainer Class

class ModelTrainer:

def \_\_init\_\_(self, model\_name, device='cuda' if torch.cuda.is\_available() else 'cpu'):

self.device = device

self.config = ModelConfig().get\_config(model\_name)

self.preprocessor = FingerprintPreprocessor()

self.model = FingerPrintNet(model\_name).to(device)

self.criterion = nn.CrossEntropyLoss()

if self.config['optimizer'] == 'adam':

self.optimizer = optim.Adam(self.model.parameters(), lr=self.config['learning\_rate'])

else:

self.optimizer = optim.SGD(self.model.parameters(), lr=self.config['learning\_rate'], momentum=0.9)

self.scheduler = optim.lr\_scheduler.CosineAnnealingLR(self.optimizer, T\_max=self.config['epochs'])

def train\_epoch(self, train\_loader):

self.model.train()

running\_loss = 0.0

correct = 0

total = 0

for inputs, labels in tqdm(train\_loader, desc="Training"):

inputs, labels = inputs.to(self.device), labels.to(self.device)

self.optimizer.zero\_grad()

outputs = self.model(inputs)

loss = self.criterion(outputs, labels)

loss.backward()

self.optimizer.step()

running\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

epoch\_loss = running\_loss / len(train\_loader)

epoch\_acc = 100 \* correct / total

return epoch\_loss, epoch\_acc

def validate(self, val\_loader):

self.model.eval()

running\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad():

for inputs, labels in tqdm(val\_loader, desc="Validation"):

inputs, labels = inputs.to(self.device), labels.to(self.device)

outputs = self.model(inputs)

loss = self.criterion(outputs, labels)

running\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

epoch\_loss = running\_loss / len(val\_loader)

epoch\_acc = 100 \* correct / total

return epoch\_loss, epoch\_acc

def save\_model(self, epoch, accuracy):

torch.save({

'epoch': epoch,

'model\_state\_dict': self.model.state\_dict(),

'optimizer\_state\_dict': self.optimizer.state\_dict(),

'accuracy': accuracy

}, f'/kaggle/working/model\_{self.model.\_\_class\_\_.\_\_name\_\_}\_{model\_name}\_epoch\_{epoch}\_acc\_{accuracy:.2f}.pth')

# Cell 4: ModelConfig to Include Specified Models with 4 Epochs

class ModelConfig:

def \_\_init\_\_(self):

self.model\_configs = {

'resnet50': {

'batch\_size': 32,

'learning\_rate': 0.001,

'epochs': 4, # Updated to 4 epochs

'optimizer': 'adam',

'scheduler': 'cosine'

},

'vgg16': {

'batch\_size': 16,

'learning\_rate': 0.0001,

'epochs': 4, # Updated to 4 epochs

'optimizer': 'adam',

'scheduler': 'step'

},

'efficientnet\_b0': {

'batch\_size': 16,

'learning\_rate': 0.0005,

'epochs': 4, # Updated to 4 epochs

'optimizer': 'adam',

'scheduler': 'step'

},

'densenet121': {

'batch\_size': 32,

'learning\_rate': 0.001,

'epochs': 4, # Updated to 4 epochs

'optimizer': 'adam',

'scheduler': 'cosine'

}

}

def get\_config(self, model\_name):

if model\_name not in self.model\_configs:

raise ValueError(f"Model {model\_name} not found in configurations")

return self.model\_configs[model\_name]

# Cell 5: FingerPrintNet to Support All Models

class FingerPrintNet(nn.Module):

def \_\_init\_\_(self, model\_name='resnet50', num\_classes=2):

super(FingerPrintNet, self).\_\_init\_\_()

self.available\_models = {

'resnet50': models.resnet50,

'vgg16': models.vgg16,

'efficientnet\_b0': models.efficientnet\_b0,

'densenet121': models.densenet121

}

if model\_name not in self.available\_models:

raise ValueError(f"Model {model\_name} not supported. Available models: {list(self.available\_models.keys())}")

base\_model = self.available\_models[model\_name](pretrained=True)

# Modify the final layer based on the model architecture

if model\_name.startswith('resnet'):

num\_features = base\_model.fc.in\_features

base\_model.fc = nn.Linear(num\_features, num\_classes)

elif model\_name.startswith('vgg'):

num\_features = base\_model.classifier[-1].in\_features

base\_model.classifier[-1] = nn.Linear(num\_features, num\_classes)

elif model\_name.startswith('densenet'):

num\_features = base\_model.classifier.in\_features

base\_model.classifier = nn.Linear(num\_features, num\_classes)

elif model\_name.startswith('efficientnet'):

num\_features = base\_model.classifier[-1].in\_features

base\_model.classifier[-1] = nn.Linear(num\_features, num\_classes)

self.model = base\_model

def forward(self, x):

return self.model(x)

# Cell 6: Setup Data Loaders

data\_dir = '/kaggle/input/handdata/data' # Corrected Kaggle dataset path

model\_names = ['resnet50', 'vgg16', 'efficientnet\_b0', 'densenet121']

trainers = {name: ModelTrainer(name) for name in model\_names}

train\_loader = {}

val\_loader = {}

test\_loader = {}

for model\_name in model\_names:

config = ModelConfig().get\_config(model\_name)

train\_loader[model\_name] = DataLoader(

datasets.ImageFolder(os.path.join(data\_dir, 'Train'), transform=trainers[model\_name].preprocessor.transform),

batch\_size=config['batch\_size'],

shuffle=True,

num\_workers=2

)

val\_loader[model\_name] = DataLoader(

datasets.ImageFolder(os.path.join(data\_dir, 'Valid'), transform=trainers[model\_name].preprocessor.transform),

batch\_size=config['batch\_size'],

shuffle=False,

num\_workers=2

)

test\_loader[model\_name] = DataLoader(

datasets.ImageFolder(os.path.join(data\_dir, 'Test'), transform=trainers[model\_name].preprocessor.transform),

batch\_size=config['batch\_size'],

shuffle=False,

num\_workers=2

)

# Cell 7: Training Loop for Multiple Models

results = {name: {'train\_losses': [], 'val\_losses': [], 'train\_accs': [], 'val\_accs': [], 'test\_loss': 0, 'test\_acc': 0, 'training\_time': 0, 'best\_val\_acc': 0} for name in model\_names}

for model\_name in model\_names:

trainer = trainers[model\_name]

best\_accuracy = 0.0

start\_time = time.time()

for epoch in range(trainer.config['epochs']):

train\_loss, train\_acc = trainer.train\_epoch(train\_loader[model\_name])

val\_loss, val\_acc = trainer.validate(val\_loader[model\_name])

trainer.scheduler.step()

results[model\_name]['train\_losses'].append(train\_loss)

results[model\_name]['val\_losses'].append(val\_loss)

results[model\_name]['train\_accs'].append(train\_acc)

results[model\_name]['val\_accs'].append(val\_acc)

print(f'Model: {model\_name}, Epoch: {epoch+1}/{trainer.config["epochs"]}')

print(f'Train Loss: {train\_loss:.4f}, Train Acc: {train\_acc:.2f}%')

print(f'Val Loss: {val\_loss:.4f}, Val Acc: {val\_acc:.2f}%')

if val\_acc > best\_accuracy:

best\_accuracy = val\_acc

results[model\_name]['best\_val\_acc'] = val\_acc

trainer.save\_model(epoch + 1, val\_acc)

results[model\_name]['training\_time'] = time.time() - start\_time

test\_loss, test\_acc = trainer.validate(test\_loader[model\_name])

results[model\_name]['test\_loss'] = test\_loss

results[model\_name]['test\_acc'] = test\_acc

print(f'Model: {model\_name}, Test Loss: {test\_loss:.4f}, Test Acc: {test\_acc:.2f}%')

# Cell 8: Plot Training History

plt.figure(figsize=(15, 10))

for i, model\_name in enumerate(model\_names, 1):

    plt.subplot(2, 2, i)

    plt.plot(results[model\_name]['train\_losses'], label='Train Loss')

    plt.plot(results[model\_name]['val\_losses'], label='Val Loss')

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.title(f'{model\_name} - Loss')

    plt.legend()

plt.tight\_layout()

plt.savefig('/kaggle/working/training\_history.png')

plt.show()

**Gender Classification Using Fingerprints**

import os

import cv2

import numpy as np

import torch

import torch.nn as nn

from torch.utils.data import Dataset, DataLoader

import matplotlib.pyplot as plt

from tqdm import tqdm

import time

import json

import torch.nn.init as init

import seaborn as sns

# Setting random seeds for consistent results across runs

torch.manual\_seed(123) # Changed from 42 to 123

np.random.seed(123)

# Configuring the device for computation (GPU if available, else CPU)

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Computation device selected: {device}")

class BioAuthGenderDataset(Dataset):

def \_\_init\_\_(self, data\_dir, target\_size=(96, 96)):

self.data\_dir = data\_dir

self.target\_size = target\_size

self.image\_paths = []

self.labels = []

# Load male fingerprint images

male\_dir = os.path.join(data\_dir, 'Male')

for img\_name in os.listdir(male\_dir):

if img\_name.endswith('.BMP'):

self.image\_paths.append(os.path.join(male\_dir, img\_name))

self.labels.append(1) # Label 1 represents male

# Load female fingerprint images

female\_dir = os.path.join(data\_dir, 'Female')

for img\_name in os.listdir(female\_dir):

if img\_name.endswith('.BMP'):

self.image\_paths.append(os.path.join(female\_dir, img\_name))

self.labels.append(0) # Label 0 represents female

def \_\_len\_\_(self):

return len(self.image\_paths)

def \_\_getitem\_\_(self, idx):

img\_path = self.image\_paths[idx]

img = cv2.imread(img\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

img = cv2.resize(img, self.target\_size) # Resize to target dimensions

img = img.astype('float32') / 255.0 # Normalize pixel values

img = np.expand\_dims(img, axis=0) # Add channel dimension

return torch.FloatTensor(img), torch.FloatTensor([self.labels[idx]])

def create\_bioauth\_data\_loaders(data\_dir, batch\_size=32, target\_size=(96, 96)):

train\_dataset = BioAuthGenderDataset(os.path.join(data\_dir, 'Train'), target\_size)

valid\_dataset = BioAuthGenderDataset(os.path.join(data\_dir, 'Valid'), target\_size)

test\_dataset = BioAuthGenderDataset(os.path.join(data\_dir, 'Test'), target\_size)

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

valid\_loader = DataLoader(valid\_dataset, batch\_size=batch\_size)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size)

return train\_loader, valid\_loader, test\_loader

class BioAuthBaseModel(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.weight\_history = {}

self.gradient\_history = {}

def init\_tracking(self, track\_layers=3):

# Track weights and gradients for the first few layers

tracked = 0

for name, param in self.named\_parameters():

if tracked < track\_layers and 'weight' in name:

self.weight\_history[name] = []

self.gradient\_history[name] = []

tracked += 1

def track\_weights\_and\_gradients(self):

for name, param in self.named\_parameters():

if name in self.weight\_history:

self.weight\_history[name].append(param.data.clone().cpu().mean().item())

if param.grad is not None:

self.gradient\_history[name].append(param.grad.clone().cpu().mean().item())

else:

self.gradient\_history[name].append(0)

class BioAuthModelA(BioAuthBaseModel):

def \_\_init\_\_(self):

super(BioAuthModelA, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(1, 32, kernel\_size=3),

nn.Sigmoid(),

nn.MaxPool2d(2, 2),

nn.Conv2d(32, 64, kernel\_size=3),

nn.Sigmoid(),

nn.MaxPool2d(2, 2),

)

self.classifier = nn.Sequential(

nn.Flatten(),

nn.Linear(64 \* 22 \* 22, 64),

nn.Sigmoid(),

nn.Dropout(0.5),

nn.Linear(64, 1),

nn.Sigmoid()

)

self.apply(self.\_init\_weights)

self.init\_tracking()

def \_init\_weights(self, m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

init.normal\_(m.weight, mean=0.0, std=1.8) # Adjusted std from 2.0 to 1.8

def forward(self, x):

x = self.features(x)

x = self.classifier(x)

return x

class BioAuthModelB(BioAuthBaseModel):

def \_\_init\_\_(self):

super(BioAuthModelB, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(1, 64, kernel\_size=3),

nn.Sigmoid(),

nn.MaxPool2d(2, 2),

nn.Conv2d(64, 128, kernel\_size=3),

nn.Sigmoid(),

nn.MaxPool2d(2, 2),

nn.Conv2d(128, 128, kernel\_size=3),

nn.Sigmoid(),

nn.MaxPool2d(2, 2),

)

self.classifier = nn.Sequential(

nn.Flatten(),

nn.Linear(128 \* 10 \* 10, 128),

nn.Sigmoid(),

nn.Dropout(0.5),

nn.Linear(128, 1),

nn.Sigmoid()

)

self.apply(self.\_init\_weights)

self.init\_tracking()

def \_init\_weights(self, m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

init.normal\_(m.weight, mean=0.0, std=1.3) # Adjusted std from 1.5 to 1.3

def forward(self, x):

x = self.features(x)

x = self.classifier(x)

return x

class BioAuthModelC(BioAuthBaseModel):

def \_\_init\_\_(self):

super(BioAuthModelC, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(1, 32, kernel\_size=3, padding=1),

nn.Sigmoid(),

nn.BatchNorm2d(32),

nn.MaxPool2d(2, 2),

nn.Conv2d(32, 64, kernel\_size=3, padding=1),

nn.Sigmoid(),

nn.BatchNorm2d(64),

nn.MaxPool2d(2, 2),

nn.Conv2d(64, 128, kernel\_size=3, padding=1),

nn.Sigmoid(),

nn.BatchNorm2d(128),

nn.MaxPool2d(2, 2),

)

self.classifier = nn.Sequential(

nn.Flatten(),

nn.Linear(128 \* 12 \* 12, 256),

nn.Sigmoid(),

nn.BatchNorm1d(256),

nn.Dropout(0.5),

nn.Linear(256, 1),

nn.Sigmoid()

)

self.apply(self.\_init\_weights)

self.init\_tracking()

def \_init\_weights(self, m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

init.normal\_(m.weight, mean=0.0, std=0.9) # Adjusted std from 1.0 to 0.9

def forward(self, x):

x = self.features(x)

x = self.classifier(x)

return x

def bioauth\_train\_epoch(model, train\_loader, criterion, optimizer, device):

model.train()

running\_loss = 0.0

correct = 0

total = 0

pbar = tqdm(train\_loader, desc='BioAuth Training')

for inputs, labels in pbar:

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

# Apply gradient clipping to mitigate exploding gradients

torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)

optimizer.step()

# Monitor weights and gradients during training

model.track\_weights\_and\_gradients()

running\_loss += loss.item()

predicted = (outputs > 0.5).float()

total += labels.size(0)

correct += (predicted == labels).sum().item()

pbar.set\_postfix({'loss': f'{loss.item():.4f}',

'acc': f'{correct/total:.4f}'})

return running\_loss / len(train\_loader), correct / total

def bioauth\_validate(model, valid\_loader, criterion, device):

model.eval()

running\_loss = 0.0

correct = 0

total = 0

pbar = tqdm(valid\_loader, desc='BioAuth Validation')

with torch.no\_grad():

for inputs, labels in pbar:

inputs, labels = inputs.to(device), labels.to(device)

outputs = model(inputs)

loss = criterion(outputs, labels)

running\_loss += loss.item()

predicted = (outputs > 0.5).float()

total += labels.size(0)

correct += (predicted == labels).sum().item()

pbar.set\_postfix({'loss': f'{loss.item():.4f}',

'acc': f'{correct/total:.4f}'})

return running\_loss / len(valid\_loader), correct / total

def plot\_bioauth\_training\_metrics(model, history, model\_name):

# Visualize weight evolution

plt.figure(figsize=(15, 5))

# Plot weight changes over training steps

plt.subplot(1, 2, 1)

for name, weights in model.weight\_history.items():

plt.plot(weights, label=f'{name}')

plt.title(f'{model\_name} - Weight Changes in BioAuth')

plt.xlabel('Training Iterations')

plt.ylabel('Mean Weight Values')

plt.legend()

# Plot gradient changes over training steps

plt.subplot(1, 2, 2)

for name, grads in model.gradient\_history.items():

plt.plot(grads, label=f'{name}')

plt.title(f'{model\_name} - Gradient Changes in BioAuth')

plt.xlabel('Training Iterations')

plt.ylabel('Mean Gradient Values')

plt.legend()

plt.tight\_layout()

plt.savefig(f'/kaggle/working/{model\_name}\_bioauth\_training\_metrics.png')

plt.close()

# Visualize training and validation metrics

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)

plt.plot(history['train\_loss'], label='Train Loss')

plt.plot(history['valid\_loss'], label='Valid Loss')

plt.title(f'{model\_name} - Loss Metrics in BioAuth')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history['train\_acc'], label='Train Acc')

plt.plot(history['valid\_acc'], label='Valid Acc')

plt.title(f'{model\_name} - Accuracy Metrics in BioAuth')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.savefig(f'/kaggle/working/{model\_name}\_bioauth\_metrics.png')

plt.close()

def train\_bioauth\_model(model\_class, model\_name, train\_loader, valid\_loader, epochs=30):

model = model\_class().to(device)

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

history = {

'train\_loss': [],

'train\_acc': [],

'valid\_loss': [],

'valid\_acc': []

}

print(f"\nInitiating training for {model\_name} in BioAuth System...")

for epoch in range(epochs):

print(f"\nEpoch {epoch+1}/{epochs}")

train\_loss, train\_acc = bioauth\_train\_epoch(model, train\_loader, criterion, optimizer, device)

valid\_loss, valid\_acc = bioauth\_validate(model, valid\_loader, criterion, device)

history['train\_loss'].append(train\_loss)

history['train\_acc'].append(train\_acc)

history['valid\_loss'].append(valid\_loss)

history['valid\_acc'].append(valid\_acc)

print(f"\nEpoch Results:")

print(f"Train Loss: {train\_loss:.4f}, Train Acc: {train\_acc:.4f}")

print(f"Valid Loss: {valid\_loss:.4f}, Valid Acc: {valid\_acc:.4f}")

plot\_bioauth\_training\_metrics(model, history, model\_name)

return model, history

# Set up data loaders for the BioAuth gender detection dataset

train\_loader, valid\_loader, test\_loader = create\_bioauth\_data\_loaders(

data\_dir='/kaggle/input/gender-data/data',

batch\_size=32,

target\_size=(96, 96)

)

# Train all BioAuth models

models = [

(BioAuthModelA, "BioAuthModelA"),

(BioAuthModelB, "BioAuthModelB"),

(BioAuthModelC, "BioAuthModelC")

]

for model\_class, model\_name in models:

trained\_model, history = train\_bioauth\_model(model\_class, model\_name, train\_loader, valid\_loader)

**Fake vs. Real Audio Detection**

import numpy as np # For numerical operations

import pandas as pd # For handling data in tabular format

# Navigate through the read-only input directory to verify available files

import os

for root, \_, files in os.walk('/kaggle/input'):

for file in files:

pass # Silently check files without printing

import os

import librosa

import numpy as np

import hashlib

import matplotlib.pyplot as plt

from tqdm import tqdm

import tensorflow as tf

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, LSTM, Input

from tensorflow.keras.applications import VGG16

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import pandas as pd

import joblib

# Function to compute a unique hash for audio files to avoid duplicates

def bioauth\_compute\_hash(file\_path):

with open(file\_path, 'rb') as f:

return hashlib.md5(f.read()).hexdigest()

# Load and pad audio files to a uniform length for consistent processing

def bioauth\_load\_and\_pad(file\_path, sr=16000):

try:

audio, \_ = librosa.load(file\_path, sr=sr)

if len(audio) < sr:

audio = np.pad(audio, (0, sr - len(audio)))

return audio

except:

return None

# Process audio directories to extract raw audio data and labels

def preprocess\_bioauth\_audio\_dir(base\_path):

audio\_data = []

labels = []

unique\_hashes = set()

for category in ['real', 'fake']:

category\_path = os.path.join(base\_path, category)

label = 0 if category == 'real' else 1

for file in tqdm(os.listdir(category\_path), desc=f"Processing {category} audio"):

file\_path = os.path.join(category\_path, file)

if os.path.getsize(file\_path) == 0:

continue

file\_hash = bioauth\_compute\_hash(file\_path)

if file\_hash in unique\_hashes:

continue

unique\_hashes.add(file\_hash)

audio = bioauth\_load\_and\_pad(file\_path)

if audio is None:

continue

audio\_data.append(audio)

labels.append(label)

return audio\_data, labels

# Extract MFCC spectrograms for deep learning models

def extract\_bioauth\_mfcc\_spectrograms(audio\_list, sr=16000, n\_mfcc=32):

spectrograms = []

for audio in tqdm(audio\_list, desc="Generating MFCC spectrograms"):

mfcc = librosa.feature.mfcc(y=audio, sr=sr, n\_mfcc=n\_mfcc)

# Normalize MFCC features for each audio file

mfcc = (mfcc - np.mean(mfcc)) / (np.std(mfcc) + 1e-8)

# Ensure consistent shape by padding or trimming (e.g., 32 x 64)

if mfcc.shape[1] < 64:

pad\_width = 64 - mfcc.shape[1]

mfcc = np.pad(mfcc, ((0, 0), (0, pad\_width)))

else:

mfcc = mfcc[:, :64]

spectrograms.append(mfcc)

# Convert to numpy array and add channel dimension for CNN

spectrograms = np.array(spectrograms)

spectrograms = np.expand\_dims(spectrograms, axis=-1) # Shape: (n\_samples, n\_mfcc, time\_steps, 1)

return spectrograms

# Define a VGG16-based model for deepfake audio detection

def create\_bioauth\_vgg16\_model(input\_shape):

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(64, 64, 3))

# Freeze the base model layers to leverage pre-trained weights

for layer in base\_model.layers:

layer.trainable = False

inputs = Input(shape=input\_shape)

# Convert single-channel input to 3-channel for VGG16 compatibility

if input\_shape[-1] == 1:

x = tf.keras.layers.Conv2D(3, (1, 1), padding='same')(inputs)

else:

x = inputs

# Resize input to match VGG16 requirements

x = tf.keras.layers.Resizing(64, 64)(x)

x = base\_model(x)

# Add custom classification layers

x = Flatten()(x)

x = Dense(256, activation='relu')(x)

x = Dropout(0.5)(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.3)(x)

outputs = Dense(2, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Define an LSTM-based model for sequential MFCC analysis

def create\_bioauth\_lstm\_model(input\_shape):

lstm\_input\_shape = (input\_shape[1], input\_shape[0]) # (time\_steps, features)

model = Sequential([

Input(shape=input\_shape),

tf.keras.layers.Reshape(lstm\_input\_shape),

LSTM(128, return\_sequences=True),

Dropout(0.2),

LSTM(64),

Dropout(0.2),

Dense(32, activation='relu'),

Dense(2, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Prepare data for deep learning models

def prepare\_bioauth\_data\_for\_dl(X\_spectrograms, y\_labels):

y\_categorical = to\_categorical(y\_labels, num\_classes=2)

return X\_spectrograms, y\_categorical

# Main execution block for data preparation

if \_\_name\_\_ == "\_\_main\_\_":

dataset\_dirs = {

"original": "/kaggle/input/the-fake-or-real-dataset/for-original/for-original",

}

all\_results = {}

for dataset\_name, dataset\_dir in dataset\_dirs.items():

print(f"Processing dataset: {dataset\_name}")

# Define paths for training, validation, and testing

train\_path = os.path.join(dataset\_dir, 'training')

val\_path = os.path.join(dataset\_dir, 'validation')

test\_path = os.path.join(dataset\_dir, 'testing')

# Load and preprocess audio data

X\_train\_raw, y\_train = preprocess\_bioauth\_audio\_dir(train\_path)

X\_val\_raw, y\_val = preprocess\_bioauth\_audio\_dir(val\_path)

X\_test\_raw, y\_test = preprocess\_bioauth\_audio\_dir(test\_path)

# Extract MFCC spectrograms for deep learning

X\_train\_spectrograms = extract\_bioauth\_mfcc\_spectrograms(X\_train\_raw)

X\_val\_spectrograms = extract\_bioauth\_mfcc\_spectrograms(X\_val\_raw)

X\_test\_spectrograms = extract\_bioauth\_mfcc\_spectrograms(X\_test\_raw)

# Prepare data for deep learning models

X\_train\_dl, y\_train\_dl = prepare\_bioauth\_data\_for\_dl(X\_train\_spectrograms, y\_train)

X\_val\_dl, y\_val\_dl = prepare\_bioauth\_data\_for\_dl(X\_val\_spectrograms, y\_val)

X\_test\_dl, y\_test\_dl = prepare\_bioauth\_data\_for\_dl(X\_test\_spectrograms, y\_test)

# Define callbacks for model training

bioauth\_checkpoint\_vgg = ModelCheckpoint(

f"{dataset\_name}\_bioauth\_vgg16\_best\_model.keras",

monitor='val\_accuracy',

save\_best\_only=True,

mode='max',

verbose=1

)

bioauth\_checkpoint\_lstm = ModelCheckpoint(

f"{dataset\_name}\_bioauth\_lstm\_best\_model.keras",

monitor='val\_accuracy',

save\_best\_only=True,

mode='max',

verbose=1

)

bioauth\_early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=3, # Reduced from 5 to 3

restore\_best\_weights=True

)

# Train the VGG16-based model

print("Training BioAuth VGG16 model...")

input\_shape = X\_train\_spectrograms.shape[1:]

vgg16\_model = create\_bioauth\_vgg16\_model(input\_shape)

vgg16\_history = vgg16\_model.fit(

X\_train\_dl,

y\_train\_dl,

validation\_data=(X\_val\_dl, y\_val\_dl),

epochs=20,

batch\_size=32,

callbacks=[bioauth\_checkpoint\_vgg, bioauth\_early\_stopping]

)

# Save or load the best VGG16 model

vgg16\_model\_path = f"{dataset\_name}\_bioauth\_vgg16\_best\_model.keras"

if os.path.exists(vgg16\_model\_path):

print(f"BioAuth VGG16 best model saved at: {vgg16\_model\_path}")

vgg16\_model = tf.keras.models.load\_model(vgg16\_model\_path)

else:

print(f"Warning: BioAuth VGG16 best model was not saved at {vgg16\_model\_path}")

vgg16\_model.save(vgg16\_model\_path)

print(f"BioAuth VGG16 model saved as fallback at: {vgg16\_model\_path}")

# Train the LSTM-based model

print("Training BioAuth LSTM model...")

lstm\_model = create\_bioauth\_lstm\_model(input\_shape)

lstm\_history = lstm\_model.fit(

X\_train\_dl,

y\_train\_dl,

validation\_data=(X\_val\_dl, y\_val\_dl),

epochs=20,

batch\_size=32,

callbacks=[bioauth\_checkpoint\_lstm, bioauth\_early\_stopping]

)

# Save or load the best LSTM model

lstm\_model\_path = f"{dataset\_name}\_bioauth\_lstm\_best\_model.keras"

if os.path.exists(lstm\_model\_path):

print(f"BioAuth LSTM best model saved at: {lstm\_model\_path}")

lstm\_model = tf.keras.models.load\_model(lstm\_model\_path)

else:

print(f"Warning: BioAuth LSTM best model was not saved at {lstm\_model\_path}")

lstm\_model.save(lstm\_model\_path)

print(f"BioAuth LSTM model saved as fallback at: {lstm\_model\_path}")

# Evaluate the models on the test set

vgg16\_eval = vgg16\_model.evaluate(X\_test\_dl, y\_test\_dl)

lstm\_eval = lstm\_model.evaluate(X\_test\_dl, y\_test\_dl)

# Generate predictions for performance metrics

vgg16\_pred = np.argmax(vgg16\_model.predict(X\_test\_dl), axis=1)

lstm\_pred = np.argmax(lstm\_model.predict(X\_test\_dl), axis=1)

y\_test\_classes = np.argmax(y\_test\_dl, axis=1)

# Compute performance metrics

vgg16\_accuracy = accuracy\_score(y\_test\_classes, vgg16\_pred)

lstm\_accuracy = accuracy\_score(y\_test\_classes, lstm\_pred)

vgg16\_report = classification\_report(y\_test\_classes, vgg16\_pred, output\_dict=True)

lstm\_report = classification\_report(y\_test\_classes, lstm\_pred, output\_dict=True)

# Store results for analysis

results = {

"BioAuthVGG16": {

"accuracy": vgg16\_accuracy,

"precision": vgg16\_report['weighted avg']['precision'],

"recall": vgg16\_report['weighted avg']['recall'],

"f1": vgg16\_report['weighted avg']['f1-score']

},

"BioAuthLSTM": {

"accuracy": lstm\_accuracy,

"precision": lstm\_report['weighted avg']['precision'],

"recall": lstm\_report['weighted avg']['recall'],

"f1": lstm\_report['weighted avg']['f1-score']

}

}

all\_results[dataset\_name] = results

# Plot training history for visualization

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(vgg16\_history.history['accuracy'], label='Train Accuracy')

plt.plot(vgg16\_history.history['val\_accuracy'], label='Validation Accuracy')

plt.title(f'BioAuth VGG16 Model Accuracy - {dataset\_name}')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(lstm\_history.history['accuracy'], label='Train Accuracy')

plt.plot(lstm\_history.history['val\_accuracy'], label='Validation Accuracy')

plt.title(f'BioAuth LSTM Model Accuracy - {dataset\_name}')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.savefig(f"{dataset\_name}\_bioauth\_training\_history.png")

plt.close()

# Plot confusion matrices for model evaluation

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

vgg16\_cm = confusion\_matrix(y\_test\_classes, vgg16\_pred)

plt.imshow(vgg16\_cm, cmap='Blues')

plt.title(f'BioAuth VGG16 Confusion Matrix - {dataset\_name}')

plt.colorbar()

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.xticks([0, 1], ['Real', 'Fake'])

plt.yticks([0, 1], ['Real', 'Fake'])

for i in range(2):

for j in range(2):

plt.text(j, i, f"{vgg16\_cm[i, j]}", ha='center', va='center')

plt.subplot(1, 2, 2)

lstm\_cm = confusion\_matrix(y\_test\_classes, lstm\_pred)

plt.imshow(lstm\_cm, cmap='Blues')

plt.title(f'BioAuth LSTM Confusion Matrix - {dataset\_name}')

plt.colorbar()

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.xticks([0, 1], ['Real', 'Fake'])

plt.yticks([0, 1], ['Real', 'Fake'])

for i in range(2):

for j in range(2):

plt.text(j, i, f"{lstm\_cm[i, j]}", ha='center', va='center')

plt.tight\_layout()

plt.savefig(f"{dataset\_name}\_bioauth\_confusion\_matrices.png")

plt.close()

**Fingerprint Database Matching**

import cv2

import numpy as np

from skimage.metrics import structural\_similarity as ssim

import os

from pathlib import Path

from tqdm import tqdm

import argparse

# Define thresholds for BioAuth image matching

BIOAUTH\_SSIM\_THRESHOLD = 0.6 # Adjusted from 0.5 to 0.6 for stricter matching

BIOAUTH\_MSE\_THRESHOLD = 1200 # Adjusted from 1000 to 1200 for more leniency

BIOAUTH\_FEATURE\_THRESHOLD = 12 # Adjusted from 10 to 12 for stricter feature matching

# Define image dimension constraints

BIOAUTH\_MIN\_WIDTH = 100

BIOAUTH\_MIN\_HEIGHT = 100

BIOAUTH\_MAX\_WIDTH = 2000

BIOAUTH\_MAX\_HEIGHT = 2000

def bioauth\_validate\_and\_resize\_image(img):

"""Validate and resize image for BioAuth System requirements."""

height, width = img.shape[:2]

# Check if image exceeds maximum dimensions

if width > BIOAUTH\_MAX\_WIDTH or height > BIOAUTH\_MAX\_HEIGHT:

raise ValueError(f"Image dimensions exceed limit: {width}x{height}. Max: {BIOAUTH\_MAX\_WIDTH}x{BIOAUTH\_MAX\_HEIGHT}")

# Resize if image is below minimum dimensions

if width < BIOAUTH\_MIN\_WIDTH or height < BIOAUTH\_MIN\_HEIGHT:

aspect\_ratio = width / height

if width < BIOAUTH\_MIN\_WIDTH:

new\_width = BIOAUTH\_MIN\_WIDTH

new\_height = int(BIOAUTH\_MIN\_WIDTH / aspect\_ratio)

else:

new\_height = BIOAUTH\_MIN\_HEIGHT

new\_width = int(BIOAUTH\_MIN\_HEIGHT \* aspect\_ratio)

img = cv2.resize(img, (new\_width, new\_height), interpolation=cv2.INTER\_LINEAR)

print(f"Resized image from {width}x{height} to {new\_width}x{new\_height}")

return img

def bioauth\_compute\_confidence(ssim\_score, mse\_score, feature\_score):

"""Compute confidence percentage for BioAuth image matching."""

# Normalize SSIM score to percentage

ssim\_percent = ssim\_score \* 100

# Normalize MSE score (inverted scale)

mse\_percent = max(0, 100 - (mse\_score / BIOAUTH\_MSE\_THRESHOLD \* 100))

# Normalize feature score

feature\_percent = min(100, (feature\_score / BIOAUTH\_FEATURE\_THRESHOLD \* 100))

# Compute weighted average with adjusted weights

confidence = (ssim\_percent \* 0.35 + mse\_percent \* 0.25 + feature\_percent \* 0.40)

return confidence

def bioauth\_load\_image(image\_path):

"""Load and preprocess image for BioAuth System."""

img = cv2.imread(str(image\_path))

if img is None:

raise ValueError(f"Failed to load image: {image\_path}")

# Convert to grayscale for matching

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Validate and resize if necessary

gray\_img = bioauth\_validate\_and\_resize\_image(gray\_img)

return gray\_img

def bioauth\_compute\_ssim(img1, img2):

"""Compute Structural Similarity Index (SSIM) for BioAuth System."""

return ssim(img1, img2)

def bioauth\_compute\_mse(img1, img2):

"""Compute Mean Squared Error (MSE) for BioAuth System."""

return np.mean((img1 - img2) \*\* 2)

def bioauth\_feature\_match\_score(img1, img2):

"""Compute feature matching score using SIFT for BioAuth System."""

sift = cv2.SIFT\_create()

# Detect keypoints and descriptors

kp1, des1 = sift.detectAndCompute(img1, None)

kp2, des2 = sift.detectAndCompute(img2, None)

if des1 is None or des2 is None or len(des1) == 0 or len(des2) == 0:

return 0

# Match descriptors using BF Matcher

bf = cv2.BFMatcher()

matches = bf.knnMatch(des1, des2, k=2)

# Apply Lowe's ratio test

good\_matches = []

for m, n in matches:

if m.distance < 0.75 \* n.distance:

good\_matches.append(m)

return len(good\_matches)

def bioauth\_is\_valid\_match(ssim\_score, mse\_score, feature\_score):

"""Validate if the match meets BioAuth System thresholds."""

return (ssim\_score >= BIOAUTH\_SSIM\_THRESHOLD and

mse\_score <= BIOAUTH\_MSE\_THRESHOLD and

feature\_score >= BIOAUTH\_FEATURE\_THRESHOLD)

def bioauth\_find\_best\_match(altered\_image\_path, real\_images\_dir):

"""Find the best matching image for BioAuth System."""

print(f"\nProcessing input image: {altered\_image\_path}")

altered\_img = bioauth\_load\_image(altered\_image\_path)

best\_match = {

'path': None,

'ssim\_score': -float('inf'),

'mse\_score': float('inf'),

'feature\_score': -float('inf'),

'is\_valid\_match': False,

'confidence\_percentage': 0

}

# Load all BMP images from the directory

real\_images = list(Path(real\_images\_dir).rglob('\*.bmp'))

print(f"Found {len(real\_images)} images in {real\_images\_dir}")

if not real\_images:

print(f"No images found in {real\_images\_dir}")

return best\_match

# Process each image with a progress bar

pbar = tqdm(real\_images, desc="Comparing images", unit="image")

for real\_image\_path in pbar:

try:

real\_img = bioauth\_load\_image(real\_image\_path)

# Resize real image to match altered image dimensions

if real\_img.shape != altered\_img.shape:

real\_img = cv2.resize(real\_img, (altered\_img.shape[1], altered\_img.shape[0]))

# Compute similarity metrics

ssim\_score = bioauth\_compute\_ssim(altered\_img, real\_img)

mse\_score = bioauth\_compute\_mse(altered\_img, real\_img)

feature\_score = bioauth\_feature\_match\_score(altered\_img, real\_img)

# Compute confidence score

confidence = bioauth\_compute\_confidence(ssim\_score, mse\_score, feature\_score)

# Update progress bar

pbar.set\_description(

f"SSIM: {ssim\_score:.4f}, MSE: {mse\_score:.4f}, "

f"Features: {feature\_score}, Confidence: {confidence:.1f}%"

)

# Check if this is a valid match

is\_valid = bioauth\_is\_valid\_match(ssim\_score, mse\_score, feature\_score)

# Update best match if scores are better

if (ssim\_score > best\_match['ssim\_score'] and

mse\_score < best\_match['mse\_score'] and

feature\_score > best\_match['feature\_score']):

best\_match = {

'path': real\_image\_path,

'ssim\_score': ssim\_score,

'mse\_score': mse\_score,

'feature\_score': feature\_score,

'is\_valid\_match': is\_valid,

'confidence\_percentage': confidence

}

print(f"\nNew best match found: {real\_image\_path}")

print(f"SSIM: {ssim\_score:.4f}, MSE: {mse\_score:.4f}")

print(f"Features: {feature\_score}, Confidence: {confidence:.1f}%")

except Exception as e:

print(f"\nError processing {real\_image\_path}: {str(e)}")

continue

return best\_match

def bioauth\_main():

"""Main function for BioAuth System image matching."""

# Parse command-line arguments

parser = argparse.ArgumentParser(description='Match images for BioAuth System.')

parser.add\_argument('input\_image', nargs='?', default="test.BMP",

help='Path to the input image (default: test.BMP)')

parser.add\_argument('--real-dir', default="Real",

help='Directory with real images (default: Real)')

args = parser.parse\_args()

real\_dir = args.real\_dir

input\_image = args.input\_image

# Verify input image exists

image\_path = Path(input\_image)

if not image\_path.exists():

alt\_paths = [

Path(real\_dir) / input\_image,

Path.cwd() / input\_image

]

for alt\_path in alt\_paths:

if alt\_path.exists():

image\_path = alt\_path

break

if not image\_path.exists():

print(f"Error: Could not find image '{input\_image}' in current directory or {real\_dir}")

return

print(f"\nProcessing input image: {image\_path}")

try:

# Load and validate input image

input\_img = cv2.imread(str(image\_path))

if input\_img is None:

raise ValueError(f"Failed to load image: {image\_path}")

bioauth\_validate\_and\_resize\_image(input\_img)

print(f"Image dimensions: {input\_img.shape[1]}x{input\_img.shape[0]}")

best\_match = bioauth\_find\_best\_match(image\_path, real\_dir)

if best\_match['path'] is None:

print("No match found!")

else:

print("\nBest match found:")

print(f"Path: {best\_match['path']}")

print(f"SSIM Score: {best\_match['ssim\_score']:.4f}")

print(f"MSE Score: {best\_match['mse\_score']:.4f}")

print(f"Feature Matches: {best\_match['feature\_score']}")

print(f"Confidence: {best\_match['confidence\_percentage']:.1f}%")

if not best\_match['is\_valid\_match']:

print("\nWARNING: Best match has low confidence.")

print(f"SSIM threshold: {BIOAUTH\_SSIM\_THRESHOLD}")

print(f"MSE threshold: {BIOAUTH\_MSE\_THRESHOLD}")

print(f"Feature threshold: {BIOAUTH\_FEATURE\_THRESHOLD}")

# Load best match image

best\_match\_img = cv2.imread(str(best\_match['path']))

# Create scores dictionary

scores = {

'ssim': best\_match['ssim\_score'],

'mse': best\_match['mse\_score'],

'features': best\_match['feature\_score'],

'confidence': best\_match['confidence\_percentage']

}

# Display the comparison

display = bioauth\_create\_display\_window(

input\_img,

best\_match\_img,

"Input Image",

"Best Match",

scores

)

cv2.imshow('BioAuth Image Comparison', display)

cv2.waitKey(0)

cv2.destroyAllWindows()

except ValueError as e:

print(f"Error: {str(e)}")

except Exception as e:

print(f"Unexpected error: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

bioauth\_main()

**REFERENCES**

[1] Jain, A. K., Flynn, P., and Ross, A. A., Comprehensive Guide to Biometric Technologies, Springer, vol. 1, pp. 1–556, 2020.

[2] Cao, K., and Jain, A. K., “Advanced Latent Fingerprint Identification Using Automation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 4, pp. 788–800, 2019.

[3] Nguyen, D.-L., Cao, K., and Jain, A. K., “Enhanced Minutiae Detection in Fingerprint Images via Deep Neural Networks,” IEEE Trans. Biom., Behav., Identity Sci., vol. 3, no. 2, pp. 245–256, 2021.

[4] Marasco, E., and Ross, A. A., “An Overview of Gender Prediction Through Biometric Data,” ACM Comput. Surv., vol. 52, no. 5, pp. 1–35, 2020.

[5] Lin, C., and Kumar, A., “Comparing Contactless and Contact-Based Fingerprint Images for Matching,” IEEE Trans. Inf. Forensics Secur., vol. 14, no. 7, pp. 1858–1872, 2019.

[6] Yang, W., Wang, S., and Hu, J., “A Multi-Biometric System Using Fingerprint and Finger-Vein with Cancelable Features,” Pattern Recognit., vol. 101, article 107187, 2020.

[7] Maltoni, D., Maio, D., Jain, A. K., and Prabhakar, S., Complete Manual on Fingerprint Recognition Techniques, Springer, vol. 3, pp. 1–528, 2022.

[8] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Networks for Enhanced Image Recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 770–778, 2016.

[9] Simonyan, K., and Zisserman, A., “Extremely Deep Convolutional Architectures for Large-Scale Visual Recognition,” in Int. Conf. Learn. Represent. (ICLR), pp. 1–14, 2015. Available online: https://arxiv.org/abs/1409.1556.

[10] Tan, M., and Le, Q. V., “EfficientNet: Innovating Model Scaling in Convolutional Neural Networks,” in Int. Conf. Mach. Learn. (ICML), pp. 6105–6114, 2019. Available online: https://arxiv.org/abs/1905.11946.

[11] Pang, B., Zha, K., Cao, H., Shi, C., and Lu, C., “Applying Deep Learning for Anomaly Detection in Biometric Applications,” Pattern Recognit. Lett., vol. 141, pp. 16–22, 2021.

[12] Wang, M., and Deng, W., “A Review on Deep Visual Domain Adaptation Techniques,” Neurocomputing, vol. 400, pp. 219–238, 2020.

[13] Zhao, Q., Zhou, F., and Zhang, Q., “Deep Learning Approaches for Latent Fingerprint Identification,” IEEE Access, vol. 8, pp. 164669–164678, 2020.

[14] Li, J., Zhang, B., and Lu, G., “Efficient Deep Learning Models for Biometric Identity Verification,” IEEE Trans. Circuits Syst. Video Technol., vol. 32, no. 6, pp. 3456–3468, 2022.

[15] Kho, J., Kim, J., and Lee, J., “Tailored Convolutional Neural Networks for Fingerprint Identification,” Sensors, vol. 23, no. 4, article 1876, 2023.