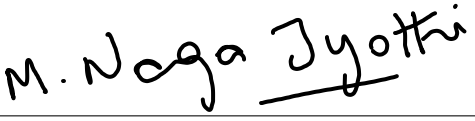
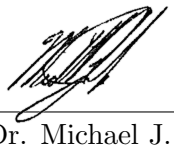
Face Recognition on Edge Devices

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. Further, the content of this thesis is truthful in regards to my own work and the portrayal of others' work. This thesis does not include proprietary or classified information.



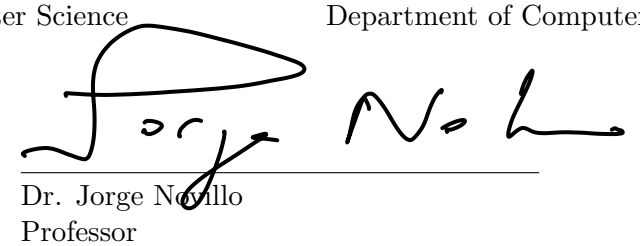
Naga Jyothi Madamanchi

Certificate of Approval:



Dr. Michael J. Reale
Associate Professor
Department of Computer Science

Dr. Chen-fu Chiang
Associate Professor
Department of Computer Science



Dr. Jorge Novillo
Professor
Department of Computer Science

# Face Recognition on Edge Devices

Naga Jyothi Madamanchi

A Project

Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
January 15, 2024

Face Recognition on Edge Devices

Naga Jyothi Madamanchi



M. Naga Jyothi

Signature of Author

15/01/2024

Date of Graduation

Project Abstract

Face Recognition on Edge Devices

Naga Jyothi Madamanchi

Master of Science, January 15, 2024
(B.S., SUNY Polytechnic Institute, 2019)

43 Typed Pages

Directed by Michael J. Reale

In the rapidly evolving landscape of face recognition technologies, this project addresses the pressing need to evaluate and optimize diverse face recognition models for deployment on edge devices. Spanning traditional methods such as Eigenfaces to contemporary deep learning approaches like VGG Face, DeepFace, and ArcFace, the study conducts a rigorous comparative analysis across various operational settings. The central focus is on elucidating the impact of Graphics Processing Units (GPUs) in enhancing model performance, particularly within the resource constraints inherent to edge devices.

Empirical testing unfolds on a dataset that's widely recognized as the Labeled Faces in the Wild (LFW) dataset. Employing Python as the primary programming language, TensorFlow as the machine learning backbone, and leveraging the capabilities of Keras, OpenCV, and Adam Geitgey's Face Recognition library, the project assembles a versatile toolkit for model evaluation.

Upon completion, this project is poised to contribute substantial insights into the deployability of face recognition models on edge devices, providing practical guidance for developers, engineers, and decision-makers. The anticipated outcomes encompass a nuanced understanding of the models' adaptability, limitations, and the role of GPUs in enhancing performance. As technology converges towards decentralized computing paradigms,

this study is positioned to play a pivotal role in shaping the landscape of face recognition technologies on the edge.

## Acknowledgments

I would like to express my heartfelt gratitude to my professors for their invaluable guidance, unwavering support, and insightful feedback throughout this project. Your expertise and encouragement have been pivotal in my journey of learning and discovery. This endeavour, undertaken independently, was significantly enriched by your collective wisdom and dedication. Thank you for inspiring me and shaping my academic journey.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

Table of Contents

# List of Figures

## List of Tables

# Chapter 1

## Introduction

The rapid evolution of face recognition technologies has ushered in transformative possibilities across diverse sectors, from security and surveillance to user authentication and human-computer interaction. In this dynamic landscape, the deployment of face recognition on edge devices has emerged as a pivotal frontier, presenting unique challenges and opportunities. Edge devices, characterized by their resource constraints and decentralized processing capabilities, offer a promising avenue for real-time and context-aware face recognition applications.

This project seeks to delve into the intricacies of face recognition on edge devices, conducting a meticulous examination of various models to discern their efficacy in both traditional computing environments and resource-constrained edge devices. The selection of models spans a spectrum of techniques, encompassing traditional methods like Eigenfaces [10] and contemporary deep learning approaches such as VGG Face [3], DeepFace[8], Face recognition (Dlib) [4]. This diverse set of models aims to provide a comprehensive overview, elucidating the strengths and weaknesses inherent in different face recognition paradigms.

The primary focus will be on unraveling the role of Graphics Processing Units (GPUs) in augmenting the performance of these models. GPUs, renowned for their parallel processing capabilities, are pivotal in enhancing the computational efficiency of face recognition tasks. As such, this study endeavors to decipher the impact of GPU utilization on the speed and accuracy of face recognition models, particularly in the context of edge computing.

The publicly available Labeled Faces in the Wild (LFW) dataset [1]. This dataset approach aims to encapsulate a broad spectrum of conditions, ensuring the findings are robust, applicable, and reflective of real-world scenarios.

Python, TensorFlow, Keras, OpenCV, and Adam Geitgey's Face Recognition library constitute the toolkit for this investigation. Leveraging these tools, the project aspires to offer actionable insights for developers, engineers, and decision-makers contemplating the integration of face recognition technologies in the burgeoning landscape of edge computing. As technology converges toward decentralized and edge-centric paradigms, the outcomes of this study are poised to enrich the understanding of deploying face recognition models in resource-constrained environments, paving the way for informed decision-making and innovation.

### 1.1 Motivation

In an era of rapid technological transformation, integrating face recognition on edge devices holds immense potential for real-time applications. This project is motivated by the need to comprehensively assess and optimize face recognition models, from traditional to deep learning approaches, for efficient deployment on resource-constrained edge devices. The outcomes promise to unlock new possibilities in security, authentication, and human-computer interaction, catering to the evolving demands of decentralized computing environments.

### 1.2 Problem Statement

The challenge lies in adapting face recognition models to perform optimally on edge devices with limited resources. Existing models may struggle with speed and accuracy in these decentralized environments, hindering their practical application. This project aims to identify and address these challenges, exploring the impact of Graphics Processing Units (GPUs) on model performance. The ultimate goal is to optimize face recognition for seamless integration into edge computing, overcoming current limitations and enhancing practical usability.

## 1.3 Objective of the project

The project aims to achieve a multifaceted set of objectives: evaluate the performance of diverse face recognition models, ranging from traditional to deep learning approaches, in both standard computing environments and resource-constrained edge devices. It seeks to ascertain the influence of Graphics Processing Units (GPUs) on model efficiency. Through empirical testing on publicly available datasets, the project aims to provide valuable insights into the adaptability and deployability of these models. The overarching goal is to contribute to the advancement of face recognition technologies tailored for edge computing contexts, fostering innovation and informed decision-making for developers and decision-makers.

## 1.4 Approach to the Problem

The approach involves a meticulous comparative analysis of diverse face recognition models, spanning both traditional and deep learning paradigms. Rigorous testing is conducted across various operational settings, including standard computing environments and resource-constrained edge devices. The impact of Graphics Processing Units (GPUs) on model performance is a focal point, evaluated through empirical testing on publicly available datasets.

## 1.5 Novelty of Approach

What sets this approach apart is its holistic evaluation spanning traditional and deep learning models, combined with a specific focus on edge device deployment. The inclusion of GPU analysis enhances the depth of understanding, addressing a crucial aspect often overlooked in existing studies. The use of a public dataset ensures a diverse range of testing conditions, contributing to the robustness and generalized of the findings. This comprehensive approach, combining traditional and cutting-edge techniques, positions the study at the forefront of understanding face recognition in the evolving landscape of edge computing.

## 1.6 Scope and Limitations

The scope encompasses a comprehensive evaluation of face recognition models on edge devices, exploring their adaptability and efficiency. However, the study is delimited by the constraints of specific models, datasets, and potential variations in edge device specifications. Limitations include the potential influence of external factors on model performance, and the study's focus solely on technical aspects, excluding broader ethical considerations associated with face recognition technology deployment on edge devices.

# Chapter 2

## Related Work

**An Optimized Face Recognition for Edge Computing [1]:** Xie et al. introduce a streamlined approach to face recognition tailored for edge devices. The authors replace VGG16 with the more compact MobileNet for face detection, enhancing efficiency. They employ 2-D face key point detection and retrain SphereFace using FP16 for resource optimization. The implementation on a Movidius NCS2 device for acceleration and a Raspberry Pi 3B+ for face alignment achieves a remarkable over 60 times speedup, enabling real-time face recognition at 7.031 FPS with 93 percent accuracy. The system's low power consumption, at 6.7W, is 17 times lower than CPU and GPU solutions. This research aligns with our project by providing insights into optimizing face recognition models for edge devices, potentially enhancing processing speed and energy efficiency in our comparative analysis.

**Characterizing Face Recognition for Resource Efficient Deployment on Edge [2]:** Biswas et al. address the challenge of tailoring state-of-the-art (SOTA) face recognition solutions for specific edge devices, an area that remains largely unexplored. The authors propose an approximation method to establish the relationship between model architecture and inference time in an edge deployment scenario, enabling predictions of custom model throughput with a small number of data points. They conduct an efficient analysis of approaches, including Face Anti-Spoofing modules, to discern the most hardware-efficient methods in terms of accuracy and error rates. This novel research provides valuable insights into optimizing face recognition for edge deployment, offering a potential benchmark for our project by guiding the selection and adaptation of models based on resource-efficient deployment considerations. The availability of data and code on GitHub enhances its reproducibility and future applicability.

**Real-Time Face Recognition on an Edge Computing Device [3]:** Samarth Gupta et al. presented at ICSCA '20, addresses the challenges associated with deploying deep learning-based face recognition systems on resource-constrained edge devices. The author reviews different face recognition applications and algorithms, emphasizing the limitations of deep learning models such as extensive training data requirements and high computational demands. The paper explores how edge computing devices, like the Intel Neural Stick, bridge these gaps due to their advantages in power efficiency and flexibility. Gupta provides a valuable flowchart detailing the deployment of a Convolutional Neural Network (CNN)-based face recognition model on an edge computing device, offering insights that can guide our project towards efficient and real-time face recognition implementations on edge devices, aligning with the resource constraints typical in such environments.

**Edge-AI based Face Recognition System: Benchmarks and Analysis [4]:** Anwar et al. discusse the integration of a multi-face recognition system into compact and low-power edge devices, particularly NVIDIA Jetson (Nano, TX2). Utilizing the 'FaceBoxes' face detector for multi-face detection and extraction and 'FaceNet' for face recognition, the proposed system employs a 'Multilayer Perceptron (MLP)' for face classification. The performance is benchmarked on NVIDIA Jetson Edge computing boards, comparing TensorRT-based optimization with a typical Tensorflow model. The analysis covers accuracy, frames per second (FPS), execution time, memory usage, and energy consumption. Furthermore, a comparison with the state-of-the-art MTCNN-based face recognition system is presented. This paper's insights into the bench-marking and deployment of a face recognition framework on specific Edge-GPU platforms can inform our project's decision-making, guiding us in optimizing for performance metrics relevant to our application.

# Chapter 3

# Method

In this chapter, I will outline the approach I took in completing this work.

## 3.1 Datasets

The Labeled Faces in the Wild (LFW) [1] dataset stands as a seminal resource in the domain of face recognition, offering a diverse and challenging set of images for model evaluation. Compiled by the University of Massachusetts, LFW contains over 13,000 labeled images featuring faces collected from the wild, encompassing a broad spectrum of real-world conditions. Originally created to foster advancements in face recognition algorithms, the dataset includes variations in illumination, pose, and facial expressions, mirroring the complexities encountered in practical applications.

Figure 3.1: LFW_Dataset [1]

LFW gained prominence due to its role in benchmarking face recognition models, providing a standardized evaluation platform for researchers and practitioners. The dataset features images of more than 5,000 individuals, with a notable emphasis on unconstrained, uncontrolled settings. This characteristic challenges models to exhibit robustness and generalization across diverse scenarios, making it particularly valuable for assessing real-world applicability.

Researchers leverage LFW to gauge the effectiveness of their models in unconstrained face recognition tasks, often employing it as a benchmark to compare algorithmic performance. Its widespread adoption has contributed to the advancement of face recognition technology by fostering healthy competition and collaboration within the research community. The Labeled Faces in the Wild dataset remains instrumental in pushing the boundaries of face recognition research and remains an essential resource for evaluating the capabilities of face recognition models in the wild.

When working with the LFW dataset, I encountered several notable challenges:

- **Limited Pose Variety and Sample Size:** The dataset presented constraints with a restricted range of image poses and a relatively small sample size. This limitation hindered the desired variability and robustness in the model training process.

- **Dataset Augmentation and Training Duration:** To overcome the limited dataset, I applied augmentation techniques, expanding it to over 19,000 images. However, this significant increase resulted in a considerable extension of training time. Even for a relatively straightforward encoding-based model, the training duration exceeded 3 hours, proving impractical for efficient development and iteration.

### 3.2 Project Flow

The project flow follows the steps below:

- **Step-1:**Data Collection and Pre-processing

- **Step-2:**Feature Extraction
- **Step-3:**Model Selection and Training
- **Step-4:**Model Evaluation and Comparison
- **Step-5:**Result Analysis

### 3.2.1 Data Collection and Pre-processing

- **Data Collection**: The data collection process for this project involves using the LFW dataset. The Labeled Faces in the Wild (LFW) dataset, is a publicly available resource renowned for its diversity and real-world applicability. LFW incorporates over 19,000+ labelled images with more than 4000+ unique labels, and variations in lighting, pose, and expressions, offering a challenging set for model testing.
- **Pre-processing Techniques**: Data pre-processing is a critical phase to enhance the reliability and effectiveness of the evaluation. Standardization of image sizes, normalization of pixel values, and handling variations in lighting and pose are crucial pre-processing steps. For LFW, special attention is given to ensuring consistency in labelling and addressing potential biases within the dataset. Additionally, facial landmarks may be detected and aligned to mitigate variations in facial poses. These pre-processing techniques collectively aim to create a standardized and representative dataset, fostering a realistic assessment of the selected face recognition models under diverse conditions. The pre-processed data sets will serve as the foundation for empirical testing, ensuring that the models are evaluated under conditions that mirror real-world scenarios.

  **Challenges in LFW dataset**

  - Folders with a single image provide inadequate test samples for effective evaluation.

- Images containing multiple faces pose challenges in accurately identifying and labeling each face.
- Folders with fewer than 5 samples are insufficient for training, given the dataset contains over 4000+ labeled faces in total.

### 3.2.2 Model Selection

The project adopts a comprehensive approach to model selection, encompassing both traditional and contemporary face recognition techniques to ensure a well-rounded evaluation. Models chosen include Eigenfaces [10] and contemporary deep learning approaches such as VGG Face[3], DeepFace[8], Face recognition (Dlib)[4]. Each model represents a distinct paradigm, ranging from classic dimensionality reduction methods to advanced deep learning architectures. This diverse selection aims to provide a holistic understanding of face recognition efficacy across various operational contexts, with a particular emphasis on the suitability for edge devices.

- **Training Strategies**: For traditional models like Eigenfaces and Adam Geitgey's library, training involves capturing the key facial features or embeddings. In contrast, deep learning models like VGG Face, DeepFace undergo extensive training on large-scale datasets to learn intricate hierarchical representations. The training process entails adjusting model parameters to minimize prediction errors and optimize feature extraction.
- **Transfer Learning and Fine-tuning**: Given the resource constraints of edge devices, transfer learning and fine-tuning strategies are explored. Pre-trained models, especially those on large datasets like ImageNet, serve as a foundation. The models are then fine-tuned on the specific face recognition datasets to adapt to the nuances of facial features and variations encountered in the project's LFW dataset.

The model selection and training phase aim to strike a balance between model complexity, accuracy, and deployability on edge devices, laying the groundwork for the subsequent empirical evaluation.

## 3.3 Model Explanation

This chapter addresses different models used in this project

### 3.3.1 Face Recognition Library

A generic term, "Face Recognition Library" refers to a software toolkit or framework designed for developing facial recognition applications. Such libraries often provide pre-trained models, APIs, and utilities to detect and recognize faces in images or videos. They encapsulate complex algorithms and models, making it easier for developers to integrate face recognition capabilities into their applications. Popular face recognition libraries include OpenCV, Dlib, and FaceNet, each offering unique features and compatibility with various programming languages.

### 3.3.2 Eigenfaces

Eigenfaces is a fundamental approach to facial recognition based on the Principal Component Analysis (PCA) [9] technique. It involves transforming facial images into a set of eigenfaces, which are the principal components capturing the most significant variations in the dataset. By representing faces as linear combinations of these eigenfaces, the method reduces dimensionality and aids in efficient recognition. Despite its simplicity, Eigenfaces has been influential in the development of subsequent face recognition techniques and serves as a foundation for understanding facial feature extraction.

Face Recognition SVM: Face Recognition with Support Vector Machines (SVM) [5] involves using SVM algorithms for classifying faces. SVMs are supervised machine learning models capable of separating classes in a high-dimensional feature space. In face recognition, SVMs learn to discriminate between different individuals based on extracted facial features.

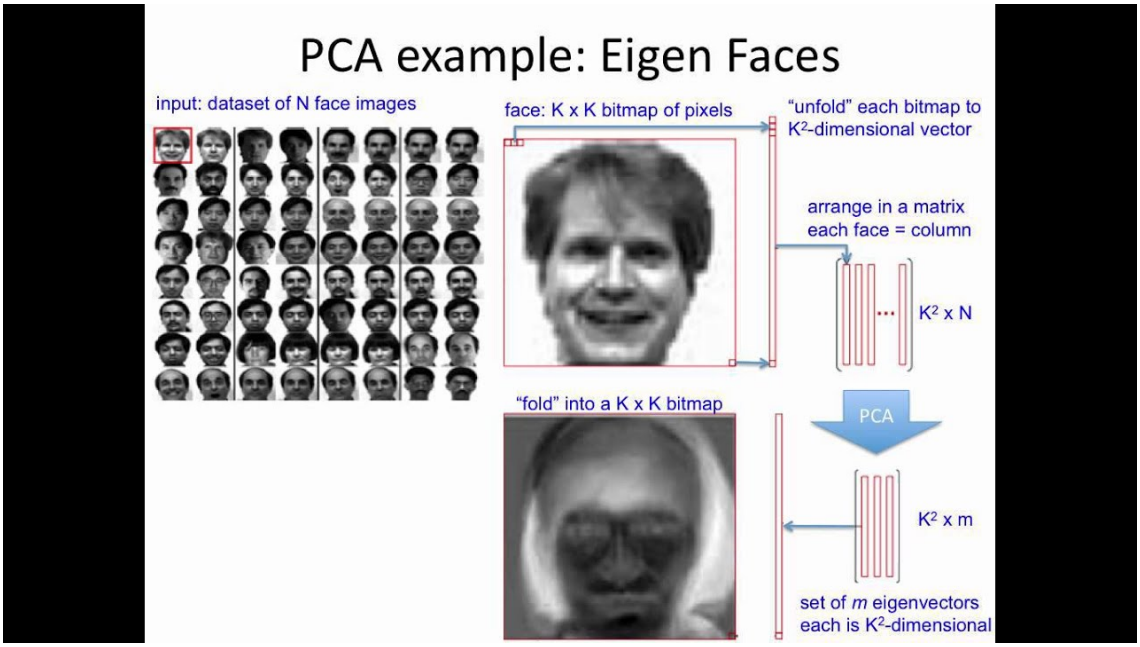



Figure 3.2: EigenFaces [7]

These features can be eigenfaces, deep embeddings, or other representations. SVMs offer robustness to variations in facial expressions and lighting conditions, making them suitable for face recognition tasks where a clear margin between classes is crucial.

### 3.3.3 VGG Face Recognition (VGG 16)

VGG Face is a deep learning model specifically designed for face recognition. It is built on the VGG 16 architecture, known for its simplicity and effectiveness. VGG Face is trained on a large dataset of faces, learning hierarchical features that are instrumental in distinguishing facial identities. The model's architecture consists of multiple convolutional layers, enabling it to capture complex patterns and representations in facial images. VGG Face is widely used in face recognition applications for its impressive performance and ability to generalize well across diverse datasets.

**VGG16 Architecture**

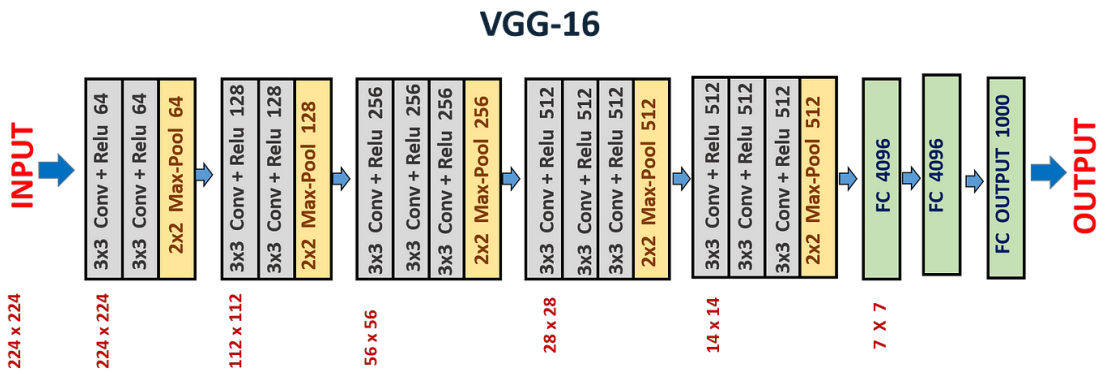



Figure 3.3: VGG16 Architecture [6]

VGG16, short for Visual Geometry Group 16, is a convolutional neural network (CNN) architecture renowned for its simplicity and effectiveness in image classification. Comprising 16 layers, it features convolutional and max-pooling layers, culminating in three fully connected layers. The convolutional layers employ small 3x3 filters, maintaining a consistent receptive field. The architecture's deep stack of layers enables it to learn intricate hierarchical features in images. VGG16 gained prominence for its compelling performance in the ImageNet Large Scale Visual Recognition Challenge, showcasing its ability to capture complex patterns and representations, setting a benchmark for subsequent deep learning models.

### 3.3.4 Deep Face Recognition (FaceNet)

FaceNet represents a breakthrough in face recognition by employing deep neural networks. Developed by Google, FaceNet generates high-dimensional embeddings, or face vectors, that encode facial features. The model is trained to ensure that similar faces are closer together in the embedding space, facilitating accurate face matching. One of its key contributions is the Triplet Loss function, which enforces a margin between the similarity scores of positive and negative face pairs during training. FaceNet excels in face verification

and identification tasks and has influenced the development of other deep face recognition systems. Its architecture emphasizes learning discriminative features, making it highly effective even in challenging scenarios.

**FaceNet Model Architecture**

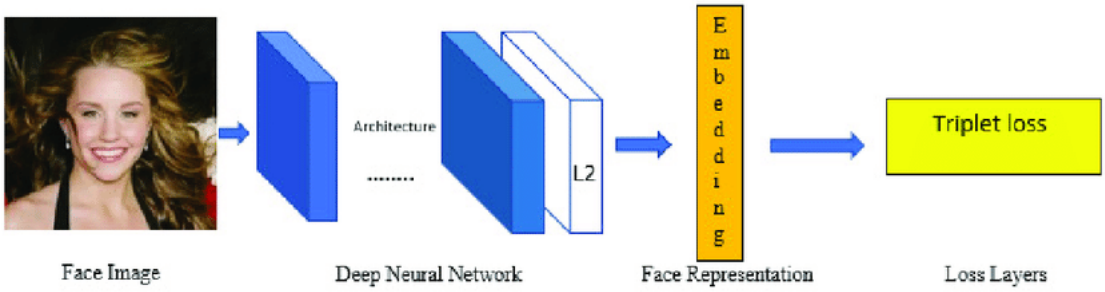



Figure 3.4: FaceNet Architecture [2]

FaceNet revolutionized face recognition with its innovative architecture. It employs a deep convolutional neural network (DCNN) to generate discriminative embeddings for facial images. The model begins with a batch input layer, followed by a series of convolutional and pooling layers that capture intricate facial features. Crucially, FaceNet incorporates L2 normalization, enhancing the model's ability to create compact and invariant face representations. The novel Triplet Loss function plays a pivotal role during training, enforcing that the distance between anchor-positive pairs is minimized, while anchor-negative pairs maintain a margin. This methodology facilitates the learning of a face embedding space where similar faces are clustered together. FaceNet's architecture, along with its emphasis on learning highly discriminative features, has set new standards in face recognition, contributing significantly to the evolution of deep learning in this domain.

## 3.4 Key challenges faced during the project

- **Resource Constraints on Edge Devices** - The primary challenge was adapting face recognition models to function optimally within the limitations of edge devices.

These devices often have constrained computational resources, including limited memory and processing power. Overcoming these limitations while maintaining acceptable performance posed a significant hurdle.

- **Model Compatibility and Optimization** - Integrating diverse face recognition models onto edge devices required careful consideration of compatibility issues. Optimizing models for efficient execution on varying hardware configurations demanded a nuanced approach. Balancing accuracy and inference speed while ensuring seamless integration added complexity to the optimization process.

- **Real-world Variability in Testing Conditions** - Testing with publicly available LFW dataset introduced challenges related to the diverse and unpredictable conditions of real-world scenarios. Addressing variations in lighting, pose, and facial expressions became crucial to providing a realistic assessment of model performance. This required meticulous dataset curation and model tuning to enhance robustness across different operational settings.

### 3.5 Potential challenges and risks in doing the project

- Training the model proved to be a complex task due to the extensive time required for training, coupled with the additional challenge of the system going into sleep mode during long training sessions.

- Technical issues such as hardware or software failures, network connectivity, and memory limitations.

### 3.6 Python Packages and Libraries

Python has a variety of package managers like pip, conda. For this project, libraries are installed. Using pip package manager.

### 3.7 Code Explanation

The code implementation is taken forward in different .py file which are listed below

1. *process_images.py*
2. $face\_model1.py$
3. $face\_model2.py$
4. $face\_model3.py$
5. $face\_model4.py$
6. $face\_model5.py$

#### 3.7.1 $process_images.py$

The "process_images.py" script is crucial to the project as it focuses on the preprocessing and augmentation of face images for building a robust and diverse dataset. This script enhances the quality and variety of the dataset by incorporating data augmentation techniques and managing the distribution of images between training and testing sets. The primary functionalities of this script are as follows:

1. Data Augmentation: The script employs various augmentation techniques, including horizontal flipping, slight rotation, and brightness variation, to generate augmented versions of the original images. This diversifies the dataset, making the model more resilient to variations in facial appearances.
2. Face Detection and Validation: The MTCNN (Multi-task Cascaded Convolutional Networks) detector is used to identify and validate faces in images. Only images with a single detected face are considered valid, ensuring the quality of the dataset.
3. Train-Test Split: The script divides the dataset into training and testing sets based on a specified ratio. This split is crucial for evaluating the model's performance on unseen data during testing.

4. Logging Skipped Images and Folders: The script maintains logs of skipped images and folders, providing transparency on excluded data. This information is valuable for understanding potential limitations or issues in the dataset.

5. Data Persistence: The processed and augmented data, along with the train-test split, are serialized and saved in a pickle file ('data.pkl'). This file becomes the input for subsequent phases of the project, ensuring reproducibility and consistent experimentation.

In short, "process_images.py" plays a pivotal role in curating a high-quality, diverse dataset, contributing to the overall effectiveness of the face recognition models developed in the project.

### 3.7.2 *face_model1.py*

The "face_model1.py" script is a crucial component of the project, serving the purpose of training and evaluating a face recognition model using the face_recognition library. This script leverages the powerful face recognition capabilities provided by the library to encode facial features, recognize faces, and assess the accuracy of the recognition process.

The script begins by loading pre-processed face data from a pickle file, separating it into training and testing sets. It then utilizes the face_recognition library to train the face recognition model by encoding the facial features of known individuals. The encoding process involves extracting facial landmarks and creating numerical representations of the faces.

Following the training phase, the script proceeds to recognize faces in the test set using the trained model. For each test image, the script compares the computed face encodings with the known encodings, determining the recognized names of individuals. The accuracy of the recognition process is calculated by comparing the recognized names with the ground truth labels.

The script's significance lies in its ability to provide a practical implementation of face recognition using a readily available library. It showcases the integration of face recognition into real-world scenarios, demonstrating its potential for applications such as security, access control, and personal identification. Additionally, the script offers a performance evaluation metric, accuracy, to quantify the effectiveness of the trained face recognition model on the provided test dataset.

### 3.7.3 *face_model2.py*

The "face_model2.py" script is of significant importance in the project, as it introduces a different approach to face recognition by utilizing a Support Vector Machine (SVM) classifier. This script leverages the face_recognition library for encoding facial features and the svm.SVC implementation from scikit-learn for training and evaluating the face recognition model.

The script starts by loading pre-processed face data from a pickle file, containing both training and testing sets. It employs the face_recognition library to encode the facial features of known individuals. The encoded face representations, along with the corresponding labels, are then used to train an SVM classifier using scikit-learn's SVM implementation.

During the testing phase, the script applies the trained SVM classifier to recognize faces in the test set. For each test image, the face encoding is computed, and the SVM classifier predicts the individual's identity. The accuracy of the SVM classifier is calculated by comparing the predicted labels with the ground truth labels.

This script's significance lies in showcasing an alternative methodology for face recognition, employing machine learning techniques like SVMs. It provides a tangible example of integrating SVMs with facial feature encodings for recognition purposes, offering a different perspective on face recognition model development within the project's broader scope. The script contributes to the versatility of the project by exploring different algorithms for face recognition beyond the conventional approaches.

### 3.7.4 *face_model3.py*

The "face_model3.py" script is of significant importance in the project as it introduces a novel approach to face recognition utilizing Eigenfaces and a Support Vector Machine (SVM) classifier. This script incorporates dimensionality reduction through Principal Component Analysis (PCA) on preprocessed facial images, transforming them into Eigenfaces. The Eigenfaces, along with corresponding labels, are then used to train an SVM classifier for facial recognition.

The script begins by loading preprocessed face data from a pickle file, separating it into training and testing sets. It preprocesses and resizes the images, converting them into flattened arrays for efficient PCA. The PCA is employed to extract principal components (Eigenfaces) that capture essential facial features while reducing dimensionality.

The SVM classifier is trained on the transformed Eigenfaces, enabling efficient classification during the testing phase. The accuracy of the model is then evaluated on the testing set, providing insights into its recognition capabilities.

The script's importance lies in its innovative combination of Eigenfaces and SVM, showcasing an alternative technique for face recognition that leverages dimensionality reduction. This approach not only demonstrates the versatility of face recognition methodologies within the project but also offers a unique perspective on the integration of traditional computer vision techniques with machine learning algorithms for effective facial recognition.

The experiments are carried out by using all components and also by using only 100 components. The results produced were the same and there was no performance improvement observed.

### 3.7.5 *face_model4.py*

The "face_model4.py" script is a crucial component of the project, designed for training and evaluating a face recognition model. This Python script employs the VGG16 architecture, leveraging ImageNet weights as initial weights for the model. The script combines

aspects of computer vision, deep learning, and model evaluation to achieve accurate face recognition.Here VGG16 architecture is trained to classify the faces.

The process begins with loading pre-processed face data from a pickle file, which includes training and testing images along with their corresponding labels. It utilizes a custom data generator, 'FaceDataGenerator,' for dynamically loading and preprocessing batches of images during training. The script offers a choice of face detection method, either Haar Cascade or MTCNN, allowing adaptability in various environments.

The core of the script involves utilizing the VGG16 model, fine-tuning it with custom layers specifically for face recognition. After compiling and training on the provided datasets, the model is saved for future use.

For evaluation, the script generates a confusion matrix, a classification report with F1 scores, and precision-recall curves, offering insights into the model's performance and highlighting areas for improvement. Additionally, it plots accuracy and loss curves during training, providing a visual representation of the model's learning progress.

### 3.7.6 *face_model5.py*

The "face_model5.py" script plays a crucial role in the project by implementing the training and evaluation pipeline for a face recognition model using the FaceNet architecture. This script is instrumental in transforming raw facial images into embeddings, leveraging a pre-trained FaceNet model. It further trains a Support Vector Machine (SVM) classifier on these embeddings, creating a robust face recognition system.

The script begins by loading pre-processed face data from a pickle file, including training and testing images along with corresponding labels. It utilizes the FaceNet embedder to generate embeddings for each facial image, capturing essential facial features in a numerical representation. The embeddings are then used to train an SVM classifier, employing a linear kernel for efficient classification.

The accuracy of the trained model is evaluated on both the training and testing datasets, providing insights into its generalization capabilities. Confusion matrices are

generated to visualize the model's performance, highlighting areas of accurate predictions and potential misclassifications. The script concludes by saving the trained SVM model and label encoder for future deployment.

# Chapter 4

# Experiments and Results

## 4.1 Model1

In the model1, the 'face recognition' library by Adam Geitgey was employed for model training and facial recognition. A dataset comprising 2317 images for training and 268 for recognition was processed, resulting in a commendable face recognition accuracy of 86.19%. The entire training and recognition cycle was completed, with performance metrics meticulously logged. Notably, the model exhibited a training time of 7.57 minutes and a recognition time of 54.67 seconds, achieving processing speeds of 5.10 images/second during training and 4.90 images/second during recognition. Both training and recognition phases were successfully executed with the provided dataset. However, despite the model's good accuracy, there is a suggestion for potential optimization to enhance either speed or accuracy further in future iterations.

| Model1 | Training Time (in sec) | Recognition Time (in sec) | Accuracy(%) |
|---|---|---|---|
| Windows | 130.44 | 86.19 | 86.19 |
| Linux (Nvidia) | 454.28 | 54.67 | 86.19 |
| Raspberrypi | 166.19 | 21.01 | 86.19 |

Table 4.1: A consolidated model1 report

## 4.2 Model2

In this set of experiments, the combination of the 'face recognition' library and an SVM classifier was applied to enhance facial recognition capabilities. The encoding and testing processes were completed, and the output confirmed successful execution. Processing the same number of images as in Model 1 (2317 for training and 268 for testing), this model

achieved a notable improvement with a higher accuracy of 96.27% in face recognition. Despite a slight increase in encoding time to approximately 7.65 minutes, testing time remained consistent at around 54 seconds. The processing speed maintained its efficiency, with 5.05 images/second for encoding and 4.96 images/second for testing. The integration of the SVM classifier proved to be advantageous, significantly boosting accuracy compared to Model 1. Overall, the model demonstrated high efficiency and accuracy, signaling robust performance in facial recognition tasks.

| Model2 | Training Time (in sec) | Recognition Time (in sec) | Accuracy(%) |
|---|---|---|---|
| Windows | 459.13 | 54.01 | 96.26 |
| Linux (Nvidia) | 134.54 | 15.63 | 96.26 |
| Raspberrypi | 166.58 | 19.55 | 96.26 |

Table 4.2: A consolidated model2 report

### 4.3 Model3

On conducting model3 experiments with the Eigenfaces method, incorporating Principal Component Analysis (PCA) for dimensionality reduction and Support Vector Machine (SVM) for classification, Model 3 aimed to optimize speed. The process was completed, yielding metrics for the PCA + SVM model's performance. Despite maintaining a dataset size consistent with prior models (2317 images for training, 268 for testing), this model exhibited a significantly lower accuracy of 15.29% in face recognition. Remarkably, training time was drastically reduced to approximately 8.02 seconds, and testing time to about 0.86 seconds, while processing speed remained steady at 5.05 images/second for encoding and 4.96 images/second for testing. The substantial decrease in training and testing time implies a notably faster model attributed to PCA's dimensionality reduction. However, the low accuracy suggests ineffectiveness for the dataset or potential misconfiguration, necessitating further investigation and refinement.

| Model2 | Training Time (in sec) | Recognition Time (in sec) | Accuracy(%) |
|---|---|---|---|
| Windows | 10.84 | 0.86 | 15.29 |
| Linux (Nvidia) | 8.02 | 0.75 | 15.29 |
| Raspberrypi | 262.6 | 20.9 | 15.29 |

Table 4.3: A consolidated model3 report

### 4.4 Model4

In Model 4, the facial recognition task was tackled using the VGG16 convolutional neural network architecture, implemented with a TensorFlow backend and GPU acceleration. The training and validation processes were successfully completed, showcasing remarkable results. The model achieved a peak training accuracy of almost 100%, with validation accuracy consistently surpassing 99%. The confusion matrix revealed high true positives and minimal misclassifications, indicating robust model performance. Training, executed over 10 epochs, exhibited a significant decrease in loss and continual improvement in accuracy. The use of an NVIDIA GeForce RTX 4060 underscored the substantial GPU computational power employed. Model 4 demonstrated exceptional accuracy and validation performance, outperforming previous models. The speed of training and validation implied efficient optimization, highlighting the potent processing capabilities inherent in the VGG16 architecture.

| Model2 | Training Time (in sec) | Recognition Time (in sec) | Accuracy(%) |
|---|---|---|---|
| Windows | 212.63 | 7.99 | 99.32 |
| Linux (Nvidia) | 1615.38 | 60.81 | 99.32 |
| Raspberrypi | NA | NA | NA |

Table 4.4: A consolidated model4 report

### 4.5 Model5

Model 5 employed DeepFace with the FaceNet architecture for facial recognition tasks, conducting successful training and testing with results captured through a confusion matrix and performance metrics. The confusion matrix depicted a well-distributed set of true positives across various classes, with some minor misclassifications. Performance metrics indicated a nearly flawless recognition rate during testing, highlighting the model's high effectiveness. Notably, the testing phase demonstrated an exceptionally fast execution time, with steps completed in milliseconds, showcasing the model's efficiency. Face recognition was virtually instantaneous, reinforcing the model's rapid processing capabilities. The results, characterized by high true positive rates and minimal misclassifications, underscored the model's accuracy and reliability. Given the swift testing time and outstanding performance, Model 5 proves to be well-suited for real-time face recognition applications, showcasing its practicality in dynamic environments.

| Model2 | Training Time (in sec) | Recognition Time (in sec) | Accuracy(%) |
|---|---|---|---|
| Windows | 10.84 | 0.005 | 97.38 |
| Linux (Nvidia) | 8.02 | 0.005 | 97.38 |
| Raspberrypi | NA | NA | NA |

Table 4.5: A consolidated model5 report

### 4.6 Consolidated Results

The following sections provide a comprehensive overview of diverse models, encapsulating key metrics such as training and testing accuracy, execution times, and suitability for real-time applications. The models are ranging from traditional methods to deep learning architectures.

#### 4.6.1 Timing

The below table gives a consolidated performance metrics of different models.

| Models | Python files | Windows | Ubuntu | Raspberry |
|---|---|---|---|---|
| face recognition | face_mode1.py | 54.67 | 15.8 | 21.01 |
| face recognition svm | face_mode2.py | 54.03 | 15.63 | 19.55 |
| Eigen faces | face_mode3.py | 0.8614 | 0.75 | 20.9 |
| VGG16 | face_mode4.py | 7.99 | 60.81 | |
| Facenet model | face_mode5.py | 0.005 | 0.01 | |

Table 4.6: Performance metrics by different models (in seconds)

### 4.6.2 Accuracy

The below table gives a consolidated accuracies report for different models.

| Models | Python files | Accuracy |
|---|---|---|
| face recognition | face_mode1.py | 86.19 |
| face recognition svm | face_mode2.py | 96.26 |
| Eigen faces(PCA and SVM) | face_mode3.py | 15.29 |
| VGG16 | face_mode4.py | 99.32 |
| Facenet model | face_mode5.py | 97.38 |

Table 4.7: Accuracy metrics by different models(in %)

# Chapter 5

# Conclusion

The comparative analysis of face recognition models across different platforms highlights distinct performance characteristics. On the (Nvidia) Ubuntu platform, the VGG16 model, demonstrates superior efficiency, processing over 250+ images in just 7.99 seconds. This efficiency suggests that the VGG16 architecture, especially when combined with GPU computational strengths, is highly effective for face recognition tasks. However, the VGG16 model was not running on the Raspberry Pi, because of resource constraints.

The other face recognition models, particularly those in the face_model1.py and face_model2.py scripts, show consistency across Windows and (Nvidia)Ubuntu. The performance on the Ubuntu system is notably faster, highlighting the operating system's suitability for image processing tasks. In contrast, the Eigenfaces model, exhibits the fastest recognition times but very less accuracy on all platforms. This suggests that Eigenfaces may not be as efficient as other options like VGG, Facenet exlored models for processing large batches of images.

Lastly, the Facenet model is exceptional on the (Nvidia)Ubuntu platform, with a recognition time of merely 0.1 seconds, indicating extreme optimization for that operating system. However, the VGG16 model, this model was not operational on the Raspberry Pi. Because of the hardwre contraints. Overall, Ubuntu emerges as the most capable platform for running these face recognition tasks with impressive speed, while Windows presents as a viable alternative. The Raspberry Pi's performance indicates its limitations for running complex models, necessitating the consideration of less resource-intensive solutions for face recognition on such devices.

In conclusion, the choice of face recognition model and the platform must be carefully matched to the application's requirements, considering both the computational resources available and the desired speed of image processing. For high-speed and high-volume image

recognition tasks, the VGG16 and Facenet models are preferable on robust systems like (Nvidia) Ubuntu, while alternative, less demanding models may be required for platforms with limited processing power such as the Raspberry Pi.

# Chapter 6

## Future Work

Future development in face recognition technology should concentrate on fortifying model robustness and optimizing performance for edge computing, where resources are limited. This includes refining algorithms for devices such as the forthcoming Raspberry Pi 5, which promises enhancements in processing capabilities. Adapting these models to accurately interpret dynamic, real-world conditions will also be crucial for their practical application in fields ranging from security to consumer electronics.

Further explorations might leverage 'tiny' models that optimize resource usage without compromising on performance, ideal for edge devices. Comprehensive testing on platforms like the Raspberry Pi 5 will be indispensable to understand the scalability of face recognition technologies. Such focused and forward-thinking research efforts will ensure that the benefits of advancements are widely accessible, establishing a versatile and secure framework for the burgeoning landscape of face recognition applications.

# Bibliography


[1] Labeled faces in the wild home. https://vis-www.cs.umass.edu/lfw/.

[2] Sirine Ammar, Thierry Bouwmans, and Mahmoud Neji. Face identification using data augmentation based on the combination of dcgans and basic manipulations. https://www.mdpi.com/2078-2489/13/8/370.

[3] Arun Kumar Dubey and Vanita Jain. Automatic facial recognition using vgg16 based transfer learning model. *Journal of Information and Optimization Sciences*, 41(7):1589–1596, 2020.

[4] Adam Geitgey. Face recognition documentation. *Release*, 1(3):3–37, 2019.

[5] Guodong Guo, Stan Z Li, and Kapluk Chan. Face recognition by support vector machines. https://www.researchgate.net/publication/2427763_Face_Recognition_by_Support_Vector_Machines.

[6] Vaibhav Khandelwal. The architecture and implementation of vgg-16. https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b.

[7] Victor Lavrenko. Pca 10: eigen-faces. https://www.youtube.com/watch?app=desktop&v=_lY74pXWlS8. YouTube video.

[8] Lam Duc Vu Nguyen, Van Van Chau, and Sinh Van Nguyen. Face recognition based on deep learning and data augmentation. In *International Conference on Future Data and Security Engineering*, pages 560–573. Springer, 2022.

[9] Jonathon Shlens. A tutorial on principal component analysis. https://www.cs.cmu.edu/~elaw/papers/pca.pdf.

[10] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, pages 586–587. IEEE Computer Society, 1991.

# Appendix A
## Notes from the Original Style Files

These style-files for use with LaTeX are maintained by Darrel Hankerson[1] and Ed Slaminka[2].

In 1990, department heads and other representatives met with Dean Doorenbos and Judy Bush-Crofton (then responsible for manuscript approval). This meeting was prompted by a memorandum[3] from members of the mathematics departments concerning the *Thesis and Dissertation Guide* and the approval process. There was wide agreement among the participants (including Dean Doorenbos) to support the basic recommendations outlined in the memorandum. The revised *Guide* reflected some (but not all) of the agreements of the meeting.

Ms Bush-Crofton was supportive of the plan to obtain "official approval" of these style files.[4] Unfortunately, Ms Bush-Crofton left the Graduate School before the process was completed. In 1994, we were revisiting some of the same problems which were resolved at the 1990 meeting.

In Summer 1994, I sent several memoranda to Ms Ilga Trend of the Graduate School, reminding her of the agreements made at the 1990 meeting. Professors A. Scottedward Hodel and Stan Reeves provided additional support. In short, it is essential that the Graduate School honor its commitments of the 1990 meeting. It should be emphasized that Dean Doorenbos is to thank for the success of that meeting.

Maintaining these LaTeX files has been more work than expected, in part due to continuing changes in requirement by the graduate school. The Graduate School occasionally has complete memory loss about the agreements of the 1990 meeting. If the Graduate School rejects your manuscript based on items controlled by the style-files, ask your advisor to contact the Graduate school (and copy to the chair) to urge cooperation.

Finally, there have been several requests for additions to the package (mostly formatting changes for figures, etc.). While such changes are not really part of the thesis-style package, it could be beneficial to collect these options and distribute with the package (making it easier on the next student). I'm especially interested in changes needed by various departments.

---

[1] Mathematics and Statistics, 221 Parker Hall, 844-3641, `hankedr@auburn.edu`

[2] Mathematics and Statistics, 218 Parker, `slamiee@auburn.edu`

[3] Originally, the memorandum was presented to Professor Larry Wit. A copy is available on request.

[4] Followup memoranda gave a definition of "official approval." Copies will be sent on request.

# Appendix B

## B.1 Software Requirements

- **Python IDLE:** Python version 3.8.2 on windows and 3.10 on Ubuntu and Raspberrypi
- Python Anaconda (conda)
- **Virtual Machine with LINUX Operating System:** Ubuntu (Nvidia) Version : Ubuntu 22.04 RAM : 4GB GPU core : single core CPU cores : 4 cores Dist space: 64GB
- **Virtual Machine with Raspbian Operating System:** Rapbian (Raspberry Pi) Version : Raspbian buster RAM : 4GB CPU cores : 2 cores Dist space: 64GB