Face Recognition Pipeline

Jastrzębski, Mikołaj
mikjas02@gmail.com

Ner, Jakub
kubaner1@gmail.com

November 29, 2024

Contents

1 Introduction

The triplet loss function facilitates the calculation of similarity between facial embeddings [1], making it a core component in face recognition systems. However, relying solely on this method in access control environments poses significant computational demands on servers. To alleviate this burden and enable efficient on-the-edge computations, we propose incorporating two preliminary stages that reduce the volume of requests sent to the face recognition server. These stages are as follows:

1. **Face Detection and Preprocessing:** This stage involves detecting faces, correcting any rotation, and cropping the face region to prepare the image for further analysis.
2. **Image Quality Assessment:** Low-quality images are filtered out at this stage to ensure that only high-quality inputs are processed further.

1.1 Objectives

The primary objective of this work is to develop an efficient scalable face recognition pipeline suitable for collecting face images and further identification. Dividing the problem into separate stages gives more control over the flow and enables adding new layers or replacing existing ones.

2 Problem Analysis and Research

2.1 Selecting Face Detection Architecture

Running face detection on edge devices necessitates a robust yet compact model. While accuracy remains an important consideration, it is secondary to the need for efficiency and scalability. Among the most suitable architectures for this task are Single Shot Detector (SSD) [2] and YOLO [3]. Both utilize anchor boxes and require non-maximum suppression (NMS) for post-processing. For this work, we adopted the SSD-based BlazeFace model [4], which is specifically optimized for face detection. The choice was motivated by the following advantages:

1. **Speed:** The BlazeFace model employs only 896 anchor boxes, compared to YOLOv8's 8,400 anchor boxes, resulting in significantly faster post-processing due to reduced NMS overhead. Additionally, BlazeFace uses an input resolution of 128x128, whereas YOLO models typically require much larger inputs, ranging from 448x448 to 1280x1280, depending on the version.
2. **Compactness:** BlazeFace has a model size of just 224 KB, making it far smaller than YOLOv8, which requires 11 MB of storage.
3. **Convenience:** Along with bounding boxes, BlazeFace provides facial landmarks, such as coordinates for the eyes, nose, mouth, and ears, which are beneficial for subsequent processing tasks.

2.2 Review of Face Recognition Problem

2.2.1 One-Shot Learning - The Basis of Face Verification

One-shot learning focuses on learning from a single example [5]. For instance, in a face verification system, there may be only one photo of each individual in the database, such as the photo used for employee identification. The goal is to verify an individual's identity by comparing their photo with the stored reference image.

One naive approach is to design a Convolutional Neural Network with multiple outputs - one for each employee and an additional output for an "unknown" class. However, this method has two significant drawbacks:

1. **Insufficient training data:** The limited examples per person make it challenging to train the CNN effectively.

2. **Scalability issues:** Adding a new individual to the system would require retraining the model and modifying its architecture to include an additional output.

Instead, a more effective solution involves learning a similarity function, $d(\text{img}_1, \text{img}_2)$, which measures the degree of difference between two images. If the similarity score is below a certain threshold τ, the two images are considered to belong to the same person:

$$\text{If } d(\text{img}_1, \text{img}_2) \leq \tau, \text{ then the images belong to the same person.}$$

$$\text{Otherwise, } d(\text{img}_1, \text{img}_2) > \tau, \text{ they belong to different individuals.}$$

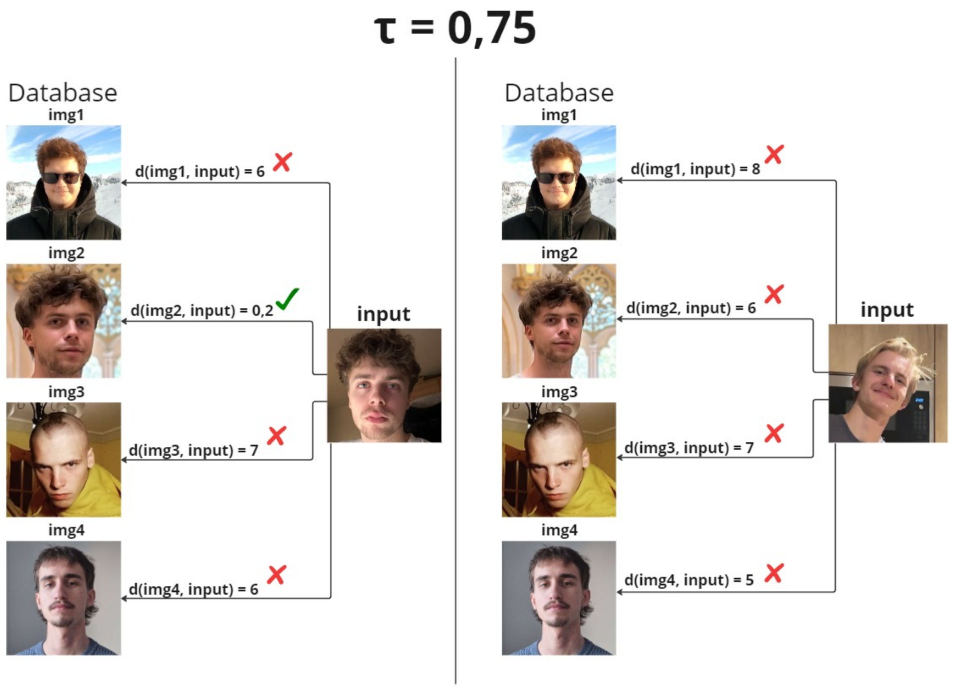


Figure 1: Illustration of the One-Shot Learning Comparison.

2.2.2 Siamese Neural Network - Encoding Image Similarity

The Siamese neural network [5] is designed to compute encoding vectors for two input images, $f(x_1)$ and $f(x_2)$. These encoding vectors represent high-dimensional feature embeddings that capture the unique characteristics of the input images. The similarity function $d(\text{img}_1, \text{img}_2)$ then computes the squared Euclidean distance between the encodings.

$$d(\text{img}_1, \text{img}_2) = \left\| f\left(x^{(1)}\right) - f\left(x^{(2)}\right) \right\|_2^2 \tag{1}$$

The architecture consists of two sister networks that share the same weights and parameters. During training, the parameters of the CNN are optimized using backpropagation to minimize the following objectives:

- If x_1 and x_2 represent the same person, the Euclidean distance $d(\text{img}_1, \text{img}_2)$ should be small.
- If x_1 and x_2 represent different individuals, $d(\text{img}_1, \text{img}_2)$ should be large.

The ultimate goal is to learn a compact 128-dimensional encoding vector for each image such that:

$$d(f(x_1), f(x_2)) \text{ is minimized for same individuals,}$$
$$d(f(x_1), f(x_2)) \text{ is maximized for different individuals.}$$

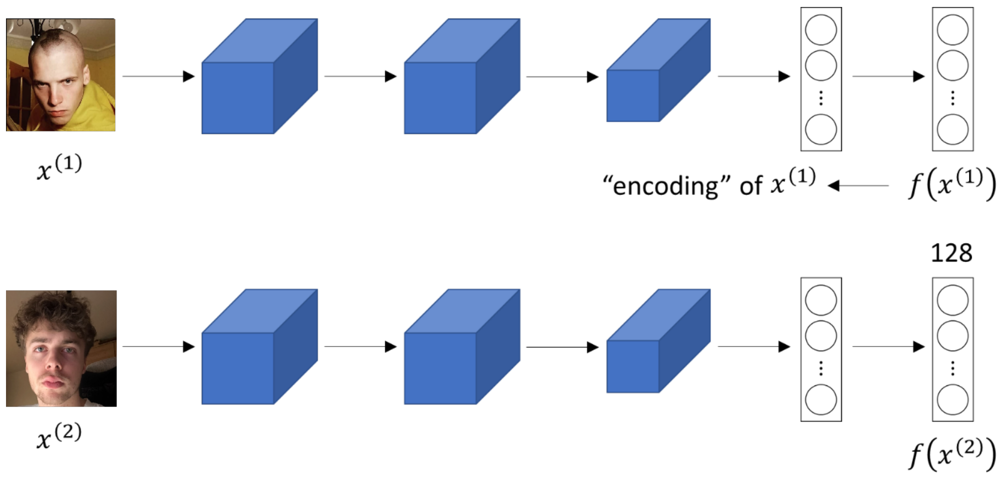


Figure 2: Schematic of a Siamese Neural Network.

2.2.3 Triplet Loss - A More Robust Learning Approach

An advanced strategy to train Siamese networks is using the *Triplet Loss*. During each iteration, the network processes three images:

- **Anchor (A):** A reference image.
- **Positive (P):** An image of the same person as the anchor.
- **Negative (N):** An image of a different person.

The images are then paired, as (**Anchor**, **Positive**) and (**Anchor**, **Negative**).

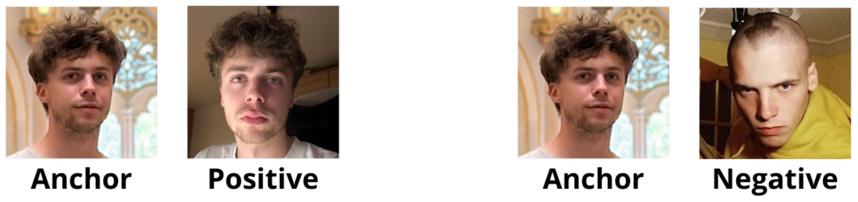


Figure 3: Illustration of Anchor, Positive, and Negative Images in Triplet Loss.

The objective is to ensure that the distance between the anchor and positive images is smaller than the distance between the anchor and negative images:

$$d(A, P) + \alpha < d(A, N)$$

Here, α is a margin hyperparameter that ensures a sufficient separation between positive and negative pairs.

Without proper constraints, the network could trivially set all weights to zero, satisfying the equation above without meaningful learning. To prevent this, the loss function is defined as:

$$\mathcal{L} = \max(0, d(A, P) - d(A, N) + \alpha)$$

The formal definition:

$$\mathcal{L}(A, P, N) = \max\left(\|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha, 0\right)$$

To ensure effective training, the selection of triplets is critical. The network performs best when trained on *hard triplets*, where:

$$d(A, P) \approx d(A, N)$$

These challenging examples force the network to improve its discrimination capability by pushing the distances apart.

3 Pipeline Development

3.1 Stage 1: Face Detection

The first stage of the pipeline focuses on detecting and preprocessing facial images to ensure they are ready for further processing. The steps involved are as follows:

1. **Preprocessing:** To an input image is added a padding either horizontal or vertical to maintain square ratio and prevent the images from being stretched. Then the image is resized to match BlazeFace's required resolution of 128x128. Additionally, the image channels are checked and adjusted to ensure compatibility, as OpenCV utilizes the BGR image format.

2. **Face Detection:** We utilize Hollance's [6] implementation of the BlazeFace model to perform inference on the resized image and extract face detections from the image.

3. **Postprocessing:**

   (a) *Selection:* The largest detected face, assumed to be the one closest to the camera, is selected for further processing.

   (b) *Rotation Correction:* By calculating the arc-tangent of the angle between the subject's eyes, any rotational misalignment in the face is corrected. It is applied to the input image with padding.

   (c) *Cropping:* New detection coordinates are computed, and the face is cropped from the image. A margin is added around the face to ensure the image maintains context. Finally the horizontal or vertical padding is added to maintain square ratio.

3.2 Stage 2: Image Quality Assessment

3.2.1 Dataset Preparation and Augmentation

The preparation of the dataset was a crucial stage in ensuring the effectiveness of the image quality assessment module. The dataset was divided into three distinct classes:

1. **Standard Photos** – High-quality images of faces, primarily captured from a frontal angle, without occlusions or distortions. The only permissible obstructions are natural elements, such as hair partially covering the face.

2. **Obstructed Face Photos** – Images of faces captured from various angles where the face is partially or fully covered by one or both hands, or where the hands are in significant contact with the face.

3. **Degraded Quality Photos** – Originally high-quality facial images that have been deliberately degraded using a custom augmentation pipeline to simulate low-quality conditions.

To build a comprehensive dataset, we evaluated multiple existing datasets and adopted various augmentation techniques tailored to our requirements.

Standard Photos.
For the class of normal photos, we considered two prominent datasets: CelebA-HQ [7] and Labeled Faces in the Wild (LFW) [8]. After thorough analysis, CelebA-HQ was selected as the primary source for the following reasons:

- It is a newer and more robust dataset, providing high-resolution images with better consistency in image quality.
- The dataset includes a wide range of face variations, making it suitable for our pipeline's generalization.

Low-Quality Photos.
The low-quality photo category required simulating natural occlusions and distortions commonly observed in real-world scenarios. These occlusions were crafted using various transformations, such as extreme brightness variations, blur, and added noise. Images from CelebA-HQ, not already utilized for normal photos, were augmented using the `albumentations` library. The transformations applied included:

- **RandomBrightnessContrast**: Adjusts brightness, either increase or decrease it severely.
- **RandomSunFlare**: Adds sun flare effects that simulates the presence of bright source of light.
- **RandomRain**: Simulates rain with adjustable slant and drop width, whilist bluring the image.
- **GaussianBlur**: Applies basic blur.
- **Downscale**: Reduces image resolution.
- **GaussNoise**: Adds noise that is very common in old selfie cameras.

Each augmentation's parameters were fine-tuned through extensive research and experimentation, leveraging the documentation and exploratory tools available on the `albumentations` website.

Face-Covered Photos.
Creating a dataset of faces covered by hands presented the most significant challenge due to the lack of suitable pre-existing datasets. Our approach combined manual efforts, crowd-sourcing, and augmentation:

1. **Manual extraction:** We manually scanned the CelebA-HQ dataset to identify images containing hands covering faces. Approximately 300 images were extracted through this process.
2. **Crowd-sourcing:** We requested friends to contribute 3–5 images each of themselves covering their faces with their hands, further diversifying the dataset.
3. **External datasets:** We explored publicly available datasets, such as VLM-Hand Over Face Dataset [9], Real-World Occluded Faces (ROF) [10], and Caltech Occluded Faces in the Wild (COFW) [11]. These datasets contained some usable images, but they required manual filtering to remove irrelevant photos, such as those with non-hand occlusions or faces too far from the camera.

After compiling these sources, all images were processed through our Phase 1 pipeline, which cropped the photos to focus on faces and hands, ensuring consistency in the dataset.

Final Dataset and Augmentation.
The final dataset comprised the following:

- 4000 normal photos
- 3500 low-quality photos
- 1750 hand-covered photos

Despite the high quality of the hand-covered photos, the initial quantity was insufficient. To address this imbalance, we doubled the number of hand-covered images through augmentation, creating 3500 total images. This resulted in a more balanced data split.

The dataset was then divided into training and testing sets using the `train_test_split` function from `sklearn.model_selection`.

During this process, we observed a critical issue: augmented pairs (original and augmented versions of the same image) occasionally split across training and testing sets. This risked overfitting by allowing the model to train on the original and validate on its augmented counterpart. To prevent this, the dataset was manually split to ensure true independence between the training and testing sets.

3.2.2 Model Training for Quality Assessment

The training process for quality assessment was a structured and iterative approach, aimed at developing robust models capable of accurately classifying data across multiple classes. This process involved loading the dataset, normalizing the data, creating multiple custom neural networks, optimizing hyperparameters, experimenting with various architectures, and training models with different strategies to achieve optimal performance.

Data Loading and Preparation.
We utilized the TensorFlow `image_dataset_from_directory` function to load and preprocess the dataset, ensuring a standardized input size of 128×128 pixels and a batch size of 16. The dataset was split into training and testing subsets, with the classes inferred automatically. Data normalization was performed by applying a `Rescaling` layer, which scaled pixel values to the range [0, 1]. To improve training efficiency, we implemented caching and prefetching using TensorFlow's `AUTOTUNE` to minimize bottlenecks caused by disk I/O and GPU communication.

Hyperparameter Tuning and Architecture Exploration.
Throughout the training process, significant effort was invested in experimenting with hyperparameters such as learning rates, optimizers, batch sizes, and regularization techniques. We developed approximately 15 custom convolutional neural network architectures with varying depths and layer configurations. Key design choices included the addition of dropout layers to mitigate overfitting, the use of L2 regularization in convolutional layers, and tuning of the kernel sizes and activation functions.

Experimenting with MobileNetV2.
In addition to custom architectures, we trained and fine-tuned MobileNetV2, a pre-trained lightweight CNN architecture, using multiple strategies. These strategies included freezing the base layers to leverage transfer learning, fine-tuning the entire network to adapt to the specific dataset, and experimenting with different pooling and classification head configurations. Each approach was trained over multiple epochs, with training data augmented dynamically to enhance generalization.

Training and Model Saving.
Each model was trained for up to 40 epochs, using the Adam optimizer and sparse categorical cross-entropy loss. Model performance was monitored using validation accuracy and loss. Two key callbacks were implemented to optimize training:

- `ModelCheckpoint`: To save the model with the lowest validation loss during training, ensuring only the best model was retained.
- `ReduceLROnPlateau`: To dynamically reduce the learning rate when validation loss plateaued, thereby stabilizing training.

The models were saved in a structured directory for future use, and each model's weights and metadata were documented systematically.

Result Visualization and Analysis.
Throughout the training process, results were visualized to evaluate progress and compare performance across experiments. Training and validation accuracy/loss curves were plotted to assess convergence, while confusion matrices were generated to analyze classification performance in detail. Custom utilities, such as plotting normalized confusion matrices, were developed to provide additional insights into model predictions.

3.2.3 Evaluation of Image Quality Assessment Model

The evaluation of our image quality assessment model involved a comprehensive analysis of multiple architectures, including both custom-designed convolutional neural networks and variations of the pre-trained MobileNetV2. The primary metrics for comparison included test accuracy, test loss, confusion matrix scores, and manual evaluation on a set of critical test images.

Implementation Overview.
The models were developed and trained using the following key libraries and frameworks:

- `numpy` for numerical operations,
- `matplotlib.pyplot` for result visualization,
- `tensorflow` for deep learning model design and training,
- `sklearn.metrics` for generating and analyzing confusion matrices,
- `tensorflow.keras.applications.MobileNetV2` for leveraging pre-trained MobileNetV2,
- `tensorflow.keras.preprocessing` for dataset loading and preprocessing,
- `tensorflow.keras.layers, models, optimizers, callbacks` for constructing, optimizing, and managing the training pipeline.

We designed several models to explore a wide range of complexities and capacities, including architectures classified as small, small-medium, medium, medium-big, and big. Additionally, different variations of MobileNetV2 were fine-tuned and evaluated for comparison. The analysis of each model's performance was based on detailed metrics and qualitative insights.

Custom Architectures.
Among the custom architectures, three designs emerged as the most interesting:

- **Medium Model:** This model demonstrated the best balance of simplicity, efficiency, and accuracy. Its architecture comprised:
  - Three convolutional layers with 3x3 kernels (`Conv2D`), each followed by batch normalization and pooling layers.
  - Regularization applied to the second and third convolutional layers using an L2 kernel regularizer ($\lambda = 0.01$).
  - Global average pooling to reduce dimensionality and prevent overfitting.
  - A dense layer with 128 neurons, ReLU activation, and a dropout of 50% for regularization.
  - An output layer with three neurons (one per class), using softmax activation.
- **Alternative Medium Model:** A variation of the medium model, this architecture featured:
  - An increased number of neurons (256) in the dense layer.
  - Additional L2 regularization applied to the first convolutional layer.

  Despite its slightly larger size, the model's performance was comparable to the MobileNetV2 baseline.
- **Smallest Model:** This compact architecture struck a balance between simplicity and performance:

- Two lightweight convolutional layers with 32 filters and 3x3 kernels, followed by pooling and batch normalization.
- A single convolutional layer with 64 filters and L2 regularization.
- Global average pooling, a dense layer with 128 neurons and dropout, and a softmax output layer.

The small size of this model made it highly efficient, with a performance that surpassed expectations for its size.

MobileNetV2 Architectures.
MobileNetV2 served as the baseline for our evaluation. The pre-trained model, initialized with `imagenet` weights, was modified as follows:

- The base layers were frozen to retain the learned feature representations.
- A global average pooling layer was added to flatten the features.
- A dropout layer with a rate of 50% was introduced to improve generalization.
- The final classification head consisted of a dense layer with softmax activation for multi-class classification.

The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss. Training was conducted for 30 epochs, with the best model saved using a `ModelCheckpoint` callback based on validation loss.

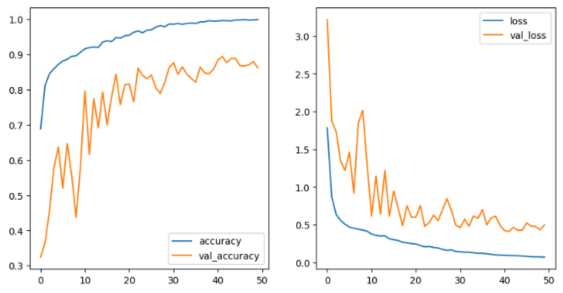
The performance of MobileNetV2 was comparable to that of the second-best custom architecture, reinforcing the effectiveness of our design choices.

Models Accuracy and Loss

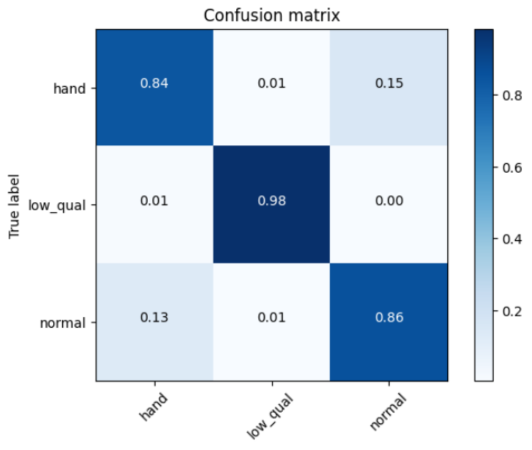
Table 1: Accuracy and Loss Results for Different Models

Model	**Training Accuracy**	**Validation Accuracy**	**Training Loss**	**Validation Loss**
Medium Model	0.9989	0.8823	0.0705	0.4746
Alternative Medium Model	0.9984	0.8677	0.0927	0.5091
Small Model	0.9938	0.8651	0.0919	0.4402
MobileNetV2	0.8924	0.8326	0.2928	0.4423

Training and validation graphs with Confusion Matrices of Models

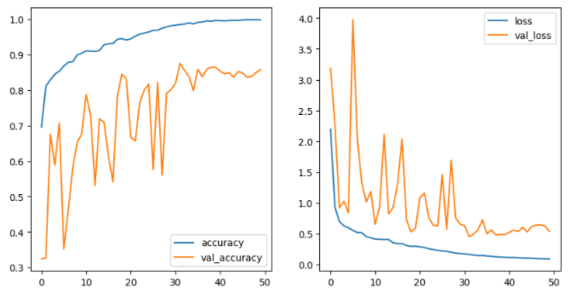


(a) Training and validation metrics for the Medium Model.

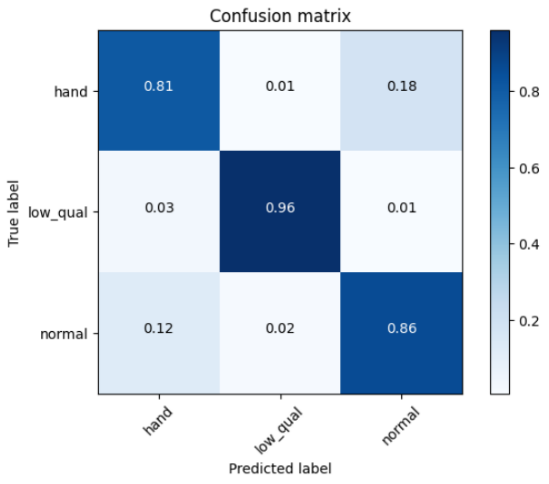


(b) Confusion Matrix for the Medium Model.

Figure 4: Results for the Medium Model.

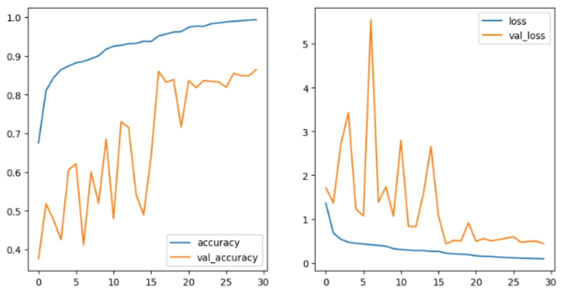


(a) Training and validation metrics for the Alternative Medium Model.

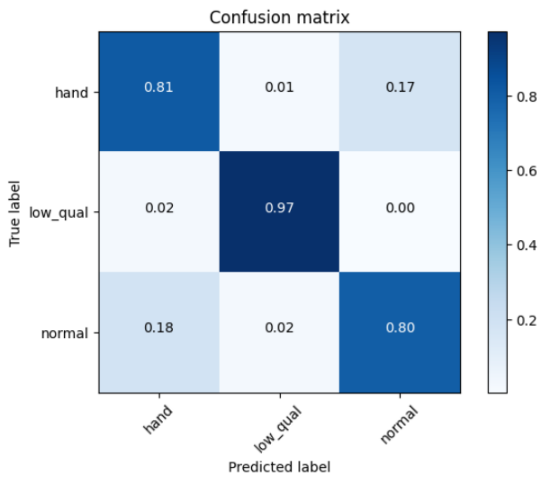


(b) Confusion Matrix for the Alternative Medium Model.

Figure 5: Results for the Alternative Medium Model.

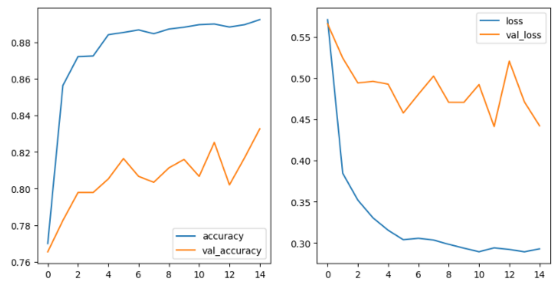


(a) Training and validation metrics for the Small Model.

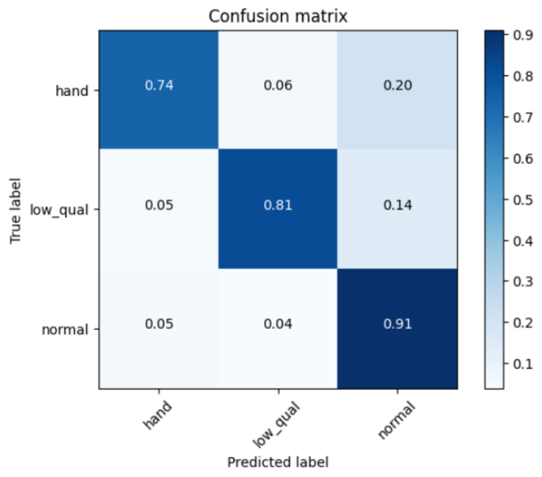


(b) Confusion Matrix for the Small Model.

Figure 6: Results for the Small Model.



(a) Training and validation metrics for the MobileNetV2 Model.



(b) Confusion Matrix for the MobileNetV2 Model.

Figure 7: Results for the MobileNetV2 Model.

Summary

1. **Medium Model:**
   - The medium model achieves the highest validation accuracy of 88.23% while maintaining an excellent training accuracy of 99.89%. This indicates its effectiveness in learning dataset features and generalizing well to unseen data.
   - Despite its high accuracy, the validation loss (0.4746) is slightly higher than that of the small model. This suggests the potential need for additional regularization or fine-tuning to further reduce overfitting.

- The confusion matrix highlights strong classification performance across all classes, with particularly high precision for the 'low_qual' class (0.98). This model demonstrates excellent performance but shows some signs of overfitting that may warrant additional regularization techniques.
- Despite some overfitting, the model performed the best out of all previously trained ones and was used as the final model for our project.

2. **Alternative Medium Model:**
   - The alternative medium model performs slightly worse than the medium model, with a validation accuracy of 86.77% and a higher validation loss of 0.5091.
   - The increased training loss (0.0927 compared to 0.0705 for the medium model) suggests that the architectural modifications, such as increasing the number of dense layer neurons and adding kernel regularization to the first convolutional layer, led to increased complexity without substantial improvement in performance.
   - Despite this, the confusion matrix reflects consistent performance across most classes, with particularly robust differentiation of the 'low_qual' class.
   - The Alternative Medium Model performs well overall but is marginally less effective than the Medium Model.

3. **Small Model:**
   - The smallest model demonstrates a strong validation performance with an accuracy of 86.51% and the lowest validation loss of 0.4402. This highlights its efficiency despite its compact size.
   - Its slightly lower training accuracy (99.38%) compared to the medium model indicates that the reduced complexity may have limited its ability to capture intricate features, but its generalization capability is notable.
   - The confusion matrix shows a slight reduction in classification accuracy for the 'normal' class, which could indicate challenges in learning the distinguishing features for this class.
   - The Small Model exhibits balanced performance, making it a viable option for applications where simplicity and lower computational cost are prioritized.

4. **MobileNetV2:**
   - MobileNetV2, although a pre-trained model, achieves a validation accuracy of 83.26%, which is lower than all custom architectures. This result underscores the potential of custom architectures tailored specifically for the dataset to outperform generic pre-trained models.
   - The training accuracy (89.24%) and relatively high training loss (0.2928) reflect MobileNetV2's difficulty in fully adapting to the dataset. This may be due to the freezing of its base layers during training, which limits its learning capacity for the new task.
   - The MobileNetV2 did not beat your best performing model at the task of classifying low quality images and images containing occlusions such as hand.

Conclusion
Among the four models, the Medium Model emerged as the most effective, achieving the highest validation accuracy and robust classification performance. However, its relatively higher validation loss indicates some potential overfitting. The Alternative Medium Model performed slightly less effectively, with higher validation loss but consistent class differentiation. The Small Model, with its lower validation loss, balanced performance, and compact architecture, offers a strong trade-off between accuracy and efficiency. Finally, the MobileNetV2 model, while computationally lightweight, underperformed relative to the other models, suggesting its architecture may not be well-suited for the task.

3.3 Stage 3: Face Recognition

The face recognition model was designed to prioritize high accuracy while maintaining a compact and efficient architecture. Its design draws significant inspiration from the Inception network family, particularly the Inception-v1 structure [12], and incorporates a customized implementation of the FaceNet model. The input to the model is a (96, 96) image, and the model itself has a medium-weight size of approximately 14 MB. This design ensures computational efficiency and accuracy, making it an optimal choice for the intended use case. The model's design philosophy and components are outlined below:

- **Inception Blocks**: The model incorporates modular inception blocks (e.g., `inception_3a`, `inception_3b`, etc.), where multiple parallel convolutional paths (`1x1`, `3x3`, `5x5`) and pooling operations are concatenated to capture multi-scale feature representations.
- **Branching and Concatenation**: Each block is composed of branches for different operations (e.g., convolutions and pooling), which are concatenated at the output, facilitating efficient multi-path feature extraction.
- **Intermediate Layers**: Every convolutional layer is followed by Batch Normalization to improve convergence and a ReLU activation function for non-linearity.
- **Downsampling**: Certain blocks (e.g., `inception_3c`, `inception_4e`) use stride-based convolution and pooling for spatial dimensionality reduction while maintaining critical feature information.
- **Hierarchical Structure**: The architecture begins with simpler convolutional layers (`conv1`, `conv2`, `conv3`) before transitioning into more complex inception blocks, progressively learning high-level features.
- **Output Layer**: The model is adaptable for task of embedding generation for face recognition, therefor it ends with a Dense Layer of 128 number of outputs that symbolise the embeding.

Once the model architecture is finalized, it can be applied to the face recognition task. This process requires a database containing a single image of each individual to be recognized. To enhance privacy and computational efficiency, facial embeddings of the images are precomputed and stored instead of the raw images themselves.

During the recognition process, the model takes a new image captured by a camera as input and generates its facial embedding. The system then iterates through the database, comparing the embedding of the input image to those stored in the database. If the embedding matches that of an individual in the database within a defined threshold, the system outputs the identity of the person in the image. Conversely, if no match is found, the system classifies the person as a stranger. This embedding-based approach ensures a robust, efficient, and privacy-conscious method for face recognition.

3.4 Face Recognition Pipeline: Integration of All Stages

The face recognition pipeline integrates three stages to ensure efficient and accurate and flexible processing of input images.

The first stage is dedicated to detecting faces within the input image. Once a face is detected, the resulting data is passed to the second and third stages. However, the execution of the third stage is conditional on the outcome of the Image Quality Assessment (Stage 2). This dependency ensures that only high-quality images proceed further in the pipeline, reducing unnecessary computational load and improving recognition accuracy.

The predictions from Stage 2 are also utilized in a thresholding formula to evaluate the suitability of the image for further processing. The formula is defined as follows:

$$f(\text{hand}, \text{low\_qual}, \text{normal}) = \text{normal} \cdot A - (\text{hand} \cdot B + \text{low\_qual} \cdot C)$$

Here:

- **hand**: Represents images occluded by a hand, mask, or similar obstructions.

- **low_qual**: Denotes images with quality issues such as poor lighting conditions, low resolution, or blurriness.
- **normal**: Indicates images of sufficient quality that are deemed acceptable for recognition.
- **A, B, C**: Constants used to weight the impact of each category in the formula.

This formula balances the system's performance by penalizing low-quality and manually flagged images while prioritizing normal-quality images for recognition. The integration of all stages ensures that the pipeline remains efficient and robust. Additionally, Parametrized formula ensures flexibility and control over the flow.

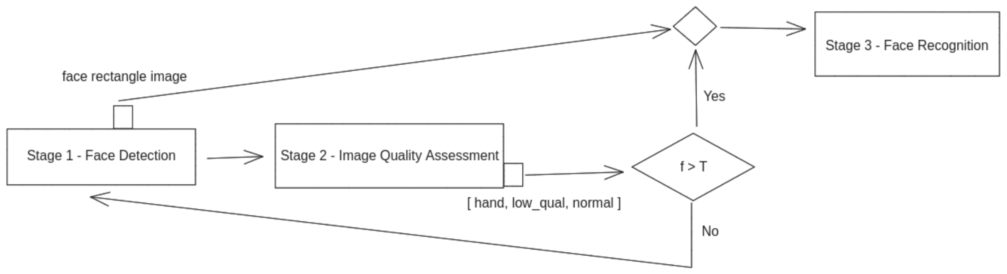

face rectangle image
Stage 1 - Face Detection
Stage 2 - Image Quality Assessment
[hand, low_qual, normal]
f > T
Yes
No
Stage 3 - Face Recognition

Figure 8: Activity diagram for the pipeline

4 Conclusion

4.1 Summary of Achievements

This project successfully developed a scalable and efficient face recognition pipeline tailored for real-world applications, emphasizing modularity, computational efficiency, and robustness. Key achievements include:

- **Efficient Face Detection:** The implementation of the BlazeFace model, optimized for edge devices, ensured fast and accurate face detection with minimal computational overhead. The model effectively handled preprocessing tasks, including rotation correction and cropping, to prepare images for downstream processing.
- **Robust Image Quality Assessment:** A comprehensive dataset was curated and augmented to train custom convolutional neural networks for image quality classification. The final medium-sized model achieved the highest validation accuracy of 88.23% thanks to the custom dataset developed for the project, demonstrating its capability to filter out low-quality and occluded images effectively, ensuring only high-quality inputs proceed to the recognition stage.
- **Compact and Accurate Face Recognition:** By incorporating a lightweight model inspired by the Inception network and FaceNet, the system achieved a balance between accuracy and efficiency. The embedding-based recognition approach facilitated scalable and privacy-conscious identity verification.
- **Pipeline Integration:** All stages—face detection, image quality assessment, and face recognition—were seamlessly integrated into a unified pipeline, achieving modularity and ease of scalability. This integration optimized resource utilization and reduced the computational load on the server by filtering non-essential inputs early in the pipeline.

4.2 Future Work and Recommendations

While the proposed face recognition pipeline demonstrates significant advancements in efficiency, scalability, and accuracy, there are several areas for potential improvement and expansion:

- **Enhanced Dataset Quality and Diversity:** The effectiveness of the pipeline heavily relies on the dataset used for training and evaluation. Incorporating larger, more diverse, and higher-quality datasets could improve model robustness, particularly in handling edge cases such as extreme lighting conditions, unusual occlusions, and diverse ethnicities.
- **Detection of Spoofing Attacks:** An additional layer of verification could be introduced to detect attempts at spoofing the system using photos, mannequins, or holograms. Techniques such as liveness detection, motion analysis, or infrared-based depth detection could enhance the system's security.
- **Improved Face Recognition Model:** Although the current model is efficient and lightweight, further exploration of advanced architectures and training techniques, such as knowledge distillation or attention mechanisms, could yield even better performance in terms of accuracy and generalization.
- **Adopting RT-DETR for Object Detection:** While BlazeFace was effective for face detection, the recent RT-DETR (Real-Time DEtection TRansformer) model offers a promising alternative to YOLO for real-time object detection tasks. Its advanced design may provide superior accuracy and speed, particularly in dynamic and complex environments.
- **Improved Augmentation and Preprocessing Techniques:** Additional preprocessing strategies, such as adaptive augmentation pipelines that simulate more real-world scenarios, could further enhance the quality and variability of training data, making the pipeline more robust against unseen conditions.
- **On-the-Edge Optimization:** The integration of more advanced model compression techniques, such as quantization or pruning, could allow the pipeline to function with even greater efficiency on edge devices without sacrificing accuracy.
- **Real-Time Monitoring and Feedback:** Introducing a real-time monitoring system to evaluate pipeline performance in deployed environments could identify bottlenecks or edge cases, providing valuable feedback for continuous improvement.

References

[1] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).* 2015, pp. 815–823.

[2] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Proceedings of the European conference on computer vision (ECCV).* 2016, pp. 21–37.

[3] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).* 2016, pp. 779–788.

[4] Valentin Bazarevsky et al. "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs". In: *arXiv preprint arXiv:1907.05047* (2019).

[5] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML) Deep Learning Workshop.* 2015.

[6] Matthijs Hollemans. "BlazeFace-PyTorch". In: (2021). Available online: `https://github.com/hollance/BlazeFace-PyTorch` (accessed on [10/10/2024]).

[7] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *arXiv preprint arXiv:1710.10196* (2018).

[8] Gary B Huang et al. "Labeled Faces in the Wild: A database for studying face recognition in unconstrained environments". In: *University of Massachusetts, Amherst, Technical Report* 1.2 (2008).

[9] Sakher Ghanem. *VLM-Hand Over Face Dataset.* Available online: `https://sites.google.com/view/sghanem/vlm-handoverface` (accessed on [10/11/2024]).

[10] Hazim Kemal Ekenel Mustafa Ekrem Erakin Uğur Demir. “On Recognizing Occluded Faces in the Wild”. In: (2021). Available online: `https://paperswithcode.com/paper/on-recognizing-occluded-faces-in-the-wild` (accessed on [01/11/2024]).

[11] Xavier P Burgos-Artizzu, Pietro Perona, and Piotr Dollár. “Robust Face Landmark Estimation under Occlusion”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013, pp. 1513–1520.

[12] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2015, pp. 1–9.