From the previously described systems in Section **??**, it is possible to infer that a DL approach can be grouped in two implementation choices: the methods of 1) face detection and 2) face representation (that include the feature extracting backbone neural network and the loss function necessary for training[1]). The next sections describe the choices of those and reasons behind them, as well as the implementation and steps necessary for that effect.

## 0.1 Face detection

Following the proposed pipeline for a Face Recognition system, the initial design choice pertains to the Face Detection module, responsible for detecting, selecting and standardizing the faces. A comprehensive study of solutions was presented in Section 2.3.1, wherein RetinaFace emerged as the best overall solution for a student monitoring context. This approach accepts an image as input and produces multiple outputs, including a bounding box, five key facial landmarks (representing the center of the eyes, nose, and corners of the mouth), and a confidence score that reflects the likelihood of the detection accurately identifying a face. The selection of this option was predominantly influenced by three critical factors: 1) Adaptability to changes in light, pose, facial expressions, etc., facilitated by the DCNN backbone, 2) incorporation of a single-stage approach and leveraging multi-task learning, enabling efficient real-time detection of facial landmarks using just a single CPU core (for VGA resolution), and 3) availability of readily implemented solutions with ample support.

The implementation of choice is a Pytorch implementation[2] of the original code that offers both a MobileNet-0.25 or ResNet-50 as backbones pre-trained on ImageNet. To better suit our application, further development over the original code was needed. RetinaFace is distributed as a general face detection algorithm that's deployed on data with multiple faces to be handled, therefore, there's no built-in methods for face selection or transformations, like alignment and resizing. However, on a student's monitoring scenario, only one face is relevant, and as consequence of that, the face recognition systems are to be trained and validated on single face pictures normalized to a canonical view.

[1] As this dissertation specifically concentrates on the face recognition domain, the loss function pertains exclusively to the training of that stage.

[2] `https://github.com/biubug6/Pytorch_Retinaface`

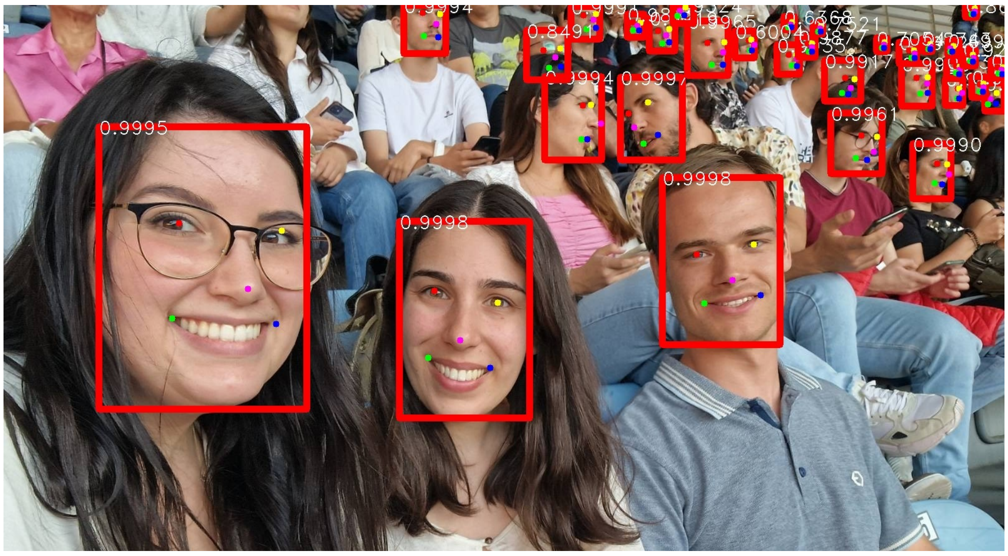

Figure 1: Results produced by the RetinaFace method over a test photo. Represented are the bounding boxes, respective confidence scores and the five facial landmarks.

Instantiating the model with default threshold parameters and Resnet-50 as the backbone, it produces the results seen in (Figure 1). From all the bounding boxes available, one must be picked, and one way of doing so is by assuming that the more relevant face is closer to the camera, hence the area of its bounding box will be greater.

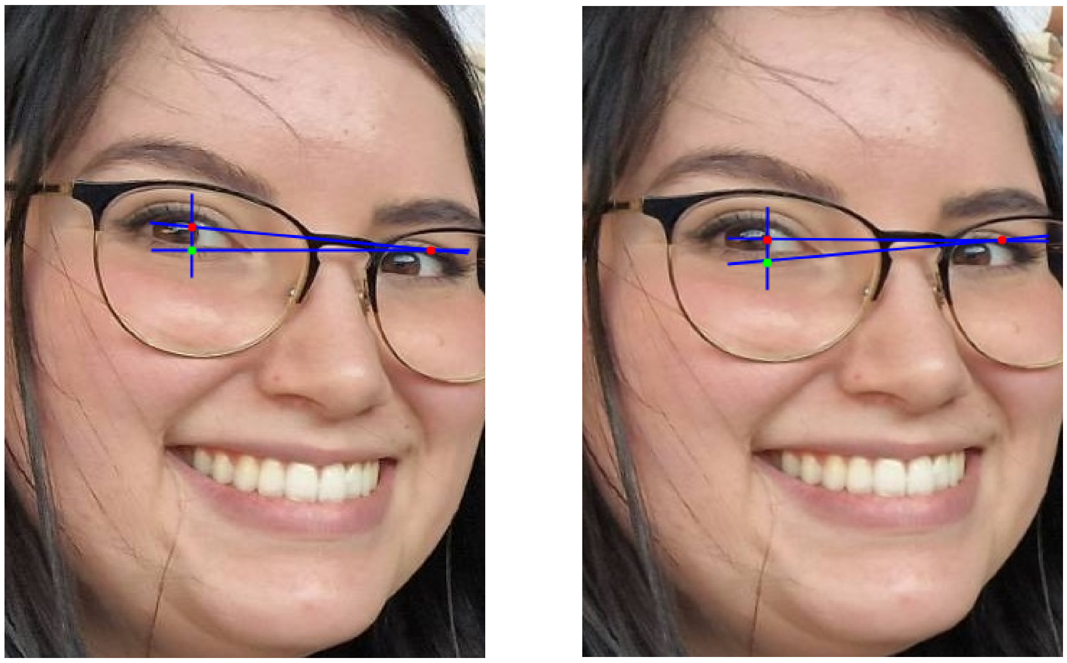

Figure 2: Landmark based alignment. The green dot serves as an auxiliary point, resulting from the intersection of a horizontal line originating at the pivot eye and a vertical line projected from the other eye. Subsequently, the rotation angle is determined by computing the arctangent of two distances: the distance between the higher eye and the auxiliary point, and the distance from the auxiliary point to the pivot eye.

The alignment is done with the help of the eyes landmarks as seen in (Figure 2). The eye lower relative to the other is declared as the pivot and starting point of a horizontal line that acts as the reference to calculate the angle of rotation. Following the selection, cropping, and alignment of the face, it is resized to fit the required dimensions for the subsequent phase.

## 0.2 Face Representation

The Face Representation stage has the task of handling the features of each face and includes a feature extractor and a loss function. Following the review on related works in Section **??** Ganidisastra and Bandung [3] and SMOWL [4] both utilize Facenet trained with triplet loss for face recognition. This naturally prompts an investigation into the potential effects on performance when utilizing a more modern network (ResNet variations), as well as a lightweight network with fewer parameters (MobileFaceNet). Furthermore, considering the drawbacks associated with triplet loss training, examining the impact of a well-established loss function, like ArcFace, which encourages intra-class compactness and inter-class distance, would also be noteworthy. Due to time constraints, the methods of choice are all pretrained and subsequently finetuned on datasets appropriate to the model's scenario deployment.

### 0.2.1 FaceNet

For this method, we utilize a highly regarded PyTorch implementation[3] that offers pretrained models in either the CASIA-WebFace or VGGFace2 dataset. The images' faces are cropped and aligned (but not rotated) using MTCNN, and resized to $160 \times 160 \times 3$. However, this implementation has differences compared to the Facenet described in the original paper [7]. Firstly, differing from the original adopted GoogLeNet-style Inception model, the backbone network employed here is the Inception Resnet V1, which integrates residual blocks into the Inception architecture. As highlighted in Section **??**, this modification streamlines the training process by addressing issues such as accuracy degradation and gradient vanishing. Secondly, adhering to the recommendations by Parkhi *et al.* [6], the network was trained as a classifier using softmax loss, departing from the original's triplet loss metric learning approach. Finally, the method's output is a 512-dimensional embedding, differing from the 128-dimensional output reported in the original paper.

### 0.2.2 ResNet

The choices for this architecture we're designed to cover a range of neural network depth, complexity and trainable parameters. Henceforth, there are 3 objects of study, ranging from less to deeper ones: iResnet-18 [4], a 29 layer version of Resnet-34 [5] (used in the TrustID project) and iResnet-50 (with Squeeze and Excitation blocks)[6]. Both the iResnet-18 and iResnet-50 were trained on $112 \times 112 \times 3$ faces from the MS1MV2 dataset, using ArcFace as the loss function, and output 512-dimensional feature embeddings. Regarding the Resnet-34, it was trained with triplet loss on a custom dataset comprised of 3 million $150 \times 150 \times 3$ images from the Visual Geometry Group Face [6] and FaceScrub [5], and outputs a 128-dimensional feature embedding.

[3] https://github.com/timesler/facenet-pytorch

[4] `https://github.com/deepinsight/insightface`

[5] `http://dlib.net/face_recognition.py.html`

[6] `https://github.com/TreB1eN/InsightFace_Pytorch`

### 0.2.3 MobileFaceNet

MobileFaceNet [6] is the lightscale approach to face recognition that will further aid the study of the computational cost and model's performance trade-off. Being that is made available by the same author as the iResnet-50, it has technical similarities. Once again, it is trained with ArcFace loss on $112 \times 112 \times 3$ images from the MS1MV2 dataset and outputs the usual 512-dimensional embedding.

## 0.3 Finetuning data

Considering that the intended application of the face recognition systems is to monitor students, it is not unreasonable to assume that, due to the nature of the capture device (webcam or smartphone) or its positioning, that will be reflect on the quality of the data. For instance, one potential scenario involves a student with one monitor and a laptop placed to the side of it, accompanied by a lamp on the opposite side. This will configuration will result in face captures with very different resolutions, poses and lightning compared to another student using a laptop, against a well lit window, facing the subject. Therefore, it is crucial to have a robust system that is capable of being insensitive to these variations, hence the possible need to finetune pretrained models for that. In this regard, we will examine two distinct datasets: DigiFace-1M and QMUL-SurvFace.

Amidst the controversies surrounding the methods of data acquisition for some publicly distributed datasets, including MS-Celeb-1M, MegaFace, FaceScrub, IJB-C or VGGFace, Bae *et al.* developed DigiFace-1M [1] with those concerns in mind. This fully synthetic dataset emulates different scenarios with variant poses, light, expression and accessories (hats, masks, makeup, etc.), and it serves as an interesting object of study of the potential of this type of datasets to diminish the reliance over real face data to finetune a model for more adverse scenarios.

The second dataset used to finetune our models of choice is QMUL-SurvFace [2] by Cheng *et al.*. Initially described as a benchmarking dataset, its unconstrained way of capturing resulted on faces with very high variance in resolution, capturing angles, poses, light or accessories, poses as a perfect source to further adapt the models to said scenarios. Another noteworthy benefit of this dataset is that the images we're collected with the consent of the individuals, meaning that ethical and privacy dilemmas are not at play.

## 0.4 Benchmarks

To comprehensively assess the performance of the chosen models, we will subject them to testing across eight diverse datasets: VGGFace2, AgeDB30, two CFP variations (CFP-FF for frontal-frontal pairs and CFP-FP for frontal-profile pairs), CALFW, CPLFW, XQLFW and LFW. This approach is designed to capture a wide array of characteristics, closely resembling real-world scenarios for a thorough evaluation, therefore, four dataset groups based on their characteristics are defined: frontal, age, pose and hard. The frontal group is composed of the CFP-FF and LFW, easier datasets with only frontal views of the face. The age group is designed to evaluate the model's sensitivity to age variations, and gathers the AgeDB30 and CALFW datasets. The pose one measures the system's performance when tested against faces with a wide range of poses by evaluating in the CFP-FP and CPLFW datasets. Finally, the hard group includes VGGFace2, a large scale benchmark with great variation in pose, age, light, etc., and XQLFW, a very difficult dataset with degraded image quality.

The evaluation will be conducted on the processed dataset to match the size of the training data for each model. First with the original pretrained model and then with the finetuned version of the selected one. The chosen evaluation metrics, performed with 10-fold cross validation, are based on biometric systems of verification and encompass Accuracy, the Receiver Operating Characteristic (ROC) and Detection Error Trade-off (DET) plots obtained by sweeping an interval of thresholds and calculating the True Acceptance Rate (TAR), False Acceptance Rate (FAR) and the False Reject Rate (FRR), and the Equal Error Rate (EER).

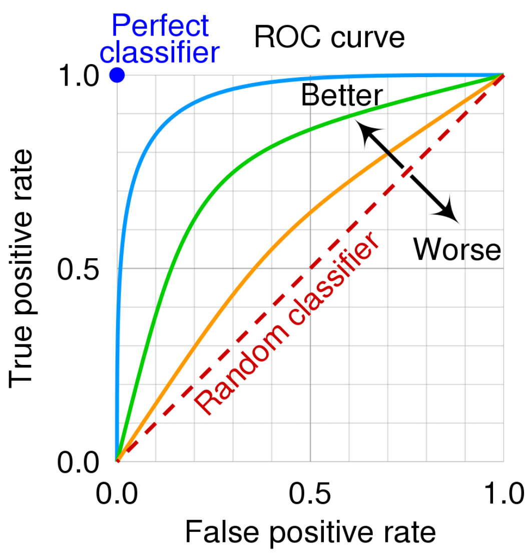

Figure 3: The y-axis represents the True Positive Rate (or TAR) and the x-axis represents the False Positive Rate (or FAR). The closer the ROC curve is to the top left corner, the better performance the model has, since that means that it is able of correctly identify more genuine face matches (TAR) while minimizing the incorrectly classified matches (FAR).

The $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ is a measure for the number of correct predictions, $TAR = \frac{TP}{TP+FN}$ indicates the proportion of genuine face pairs correctly classified and $FAR = \frac{FP}{FP+TN}$ indicates the quantity of imposter face pairs incorrectly classified as matches, where TP is True Positive, TN is True Negative, FP is False Positive and FN is False Negative. With the previous metrics is possible to obtain the ROC plot and study the trade-off between the TAR and FAR at different thresholds and evaluate the performance of different models (Figure 3). $FRR = 1 - TAR$ calculates the ratio of genuine matches that are classified as a non-match and allows to generate the DET curve which serves a similar purpose to the ROC curves, but with a better performance visualization due to the axis' logarithmic scale. Finally, the EER is the point where FRR = FAR and the lower it is, the better performing the system is.

## 0.5 Implementation details

All the training, processing, and benchmarking code was developed in a Docker environment to ensure reproducibility. The code was written using Python 3.8.10 and relies on PyTorch 1.14.0 and Torchvision 0.15.0, along with their corresponding required libraries. The training process is conducted exclusively on a single GPU, specifically NVIDIA's GeForce RTX 3080 Ti 12Gb. Regarding the codes, they were developed with modularity in mind. First, the data is processed with the custom RetinaFace algorithm and saved. Then, it is proceeded forward to the neural network that produces a feature embedding for each image. If the model is being trained, the embedding will be passed on to the ArcFace module, which will output the feature's logits. Following the original paper, the logits are turned to probabilities by applying the softmax activation function and then contributing to the final step, i.e, the cross entropy loss. This is handled by Pytorch's function CrossEntropyLoss, since it performs both steps simultaneously. Because ArcFace is a separate structure that does not integrate the backbone networks as a custom layer, the neural networks can be quickly and easily changed.