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`

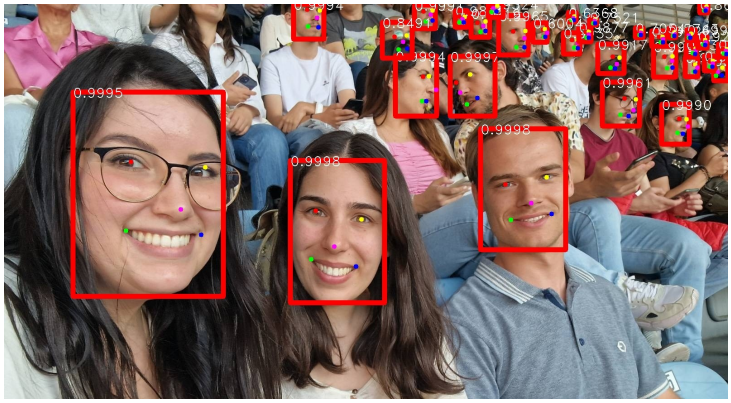

Figure 1: Results produced by the RetinaFace method over a test photo. Represented are the bounding boxes, respective confidence scores and the five facial landmarks.

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 [1] and SMOWL [2] 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 [5]. 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.* [4], 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, Resnet-34 (from the TrustID project) and iResnet-50 (with Squeeze and Excitation blocks). 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 [4] and FaceScrub [3], and outputs a 128-dimensional feature embedding..

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

0.2.3 MobileFacenet

MSM1V2

0.3 Training data

0.4 Benchmarks

0.5 Implementation tools