SUR Project

Person classification from images and voice 2024/2025

Face classification

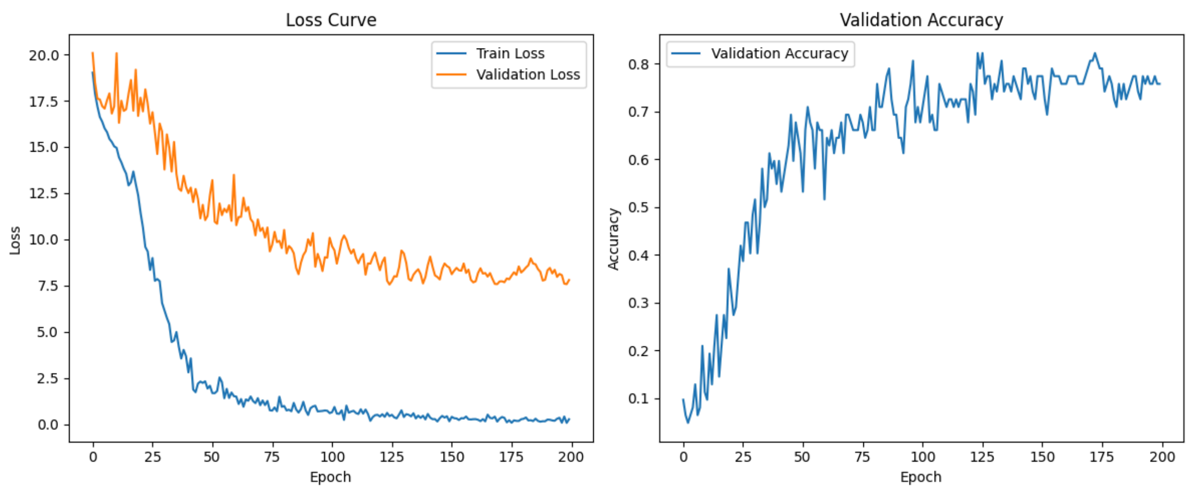
For facial image inputs, I designed a convolutional neural network inspired by ResNet18. The architecture uses a sequence of residual blocks, each with stride=2, to progressively reduce spatial resolution in place of traditional pooling layers. Dropout (p=0.2 during final training) was applied within the residual blocks and proved effective for improving generalization on moderately sized datasets. The network ends with an adaptive average pooling layer, followed by a fully connected layer that projects the features into a 256-dimensional embedding space (128-dimensions was performing worse and 512 did not show any improvements over 256).

Deeper CNN variants consistently outperformed shallower ones, which was somewhat surprising given the limited size of the training set. To avoid immediate overfitting, data augmentation was essential—but only when carefully balanced. Excessive augmentation degraded performance by distorting the semantic content of the images, while insufficient augmentation led to overfitting.

In the final setup, augmentation was applied with a probability of 0.5 per image. The transformation pipeline included controlled operations such as color jittering, random grayscale conversion, Gaussian blur, perspective distortion, horizontal flipping, small rotations, and random resized cropping. After augmentation, all images were normalized to zero mean and unit variance per channel.

For classification, I replaced the standard softmax cross-entropy loss with the ArcFace loss function, which encourages higher inter-class separability and intra-class compactness in the embedding space. Configuring the ArcFace with scale=30 and margin=0.5 for a 256-dimensional embedding and 31 output. This setup consistently outperformed traditional softmax loss in my experiments.

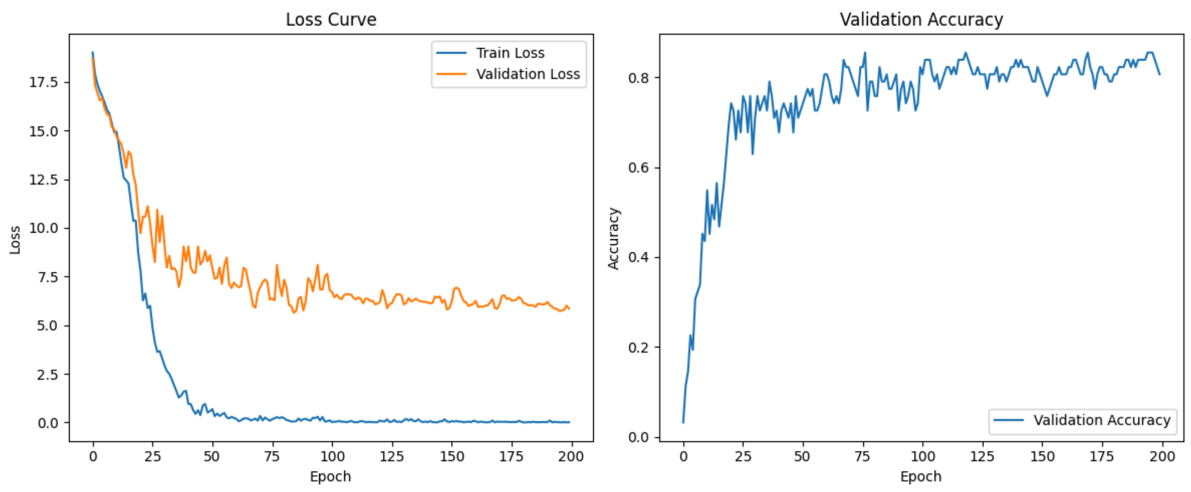
I trained the model for 200 epochs and saw stable convergence without overfitting. For the final model I combined the training and dev sets and ran a blind training for 200 epochs to make full use of the data.



Training on train and validation on dev dataset

Speaker classification

I used an x-vector architecture with MFCCs (13 coefficients; using more harmed training) as input. Adding deltas degraded performance, so I stuck to static MFCCs. To handle variable-length inputs, I padded MFCCs with zeros but only pooled over valid segments during statistical pooling. Segment dimensions were set to 128; 256 yielded similar results, while 512 proved too difficult to train. Also I used one less segment layer than the original paper, since it did not yield better results. The final speaker embedding had 256 dimensions. Although I experimented with data augmentation, even slight augmentation made training have bad accuracy on validation, so I trained without it. The model includes dropout (0.1 worked well—0.2 significantly hurt results). I also batch normalization. The model was trained for 200 epochs, just like my image setup, using the same classification head.



Combined model

I created a simple combined model that sums the class probabilities from each individual model and selects the class with the highest total score as the final prediction.

Fusion model

I also created a fusion model that trained the image and audio branches simultaneously using the same architectures, with batch normalization and dropout set to 0.2. Image augmentation was applied with a 60% probability, and I added MFCC-based audio augmentation using time masking, frequency masking, and noise. To combine the 256-dimensional embeddings from both modalities, I used a soft attention mechanism that learns to weight each input based on its relevance—giving more emphasis to the cleaner or more informative features. The resulting fused embedding, also 256-dimensional, was passed through the same classification head and trained for 100 epochs. There was no improvement beyond 100 epochs so I stopped the training earlier.

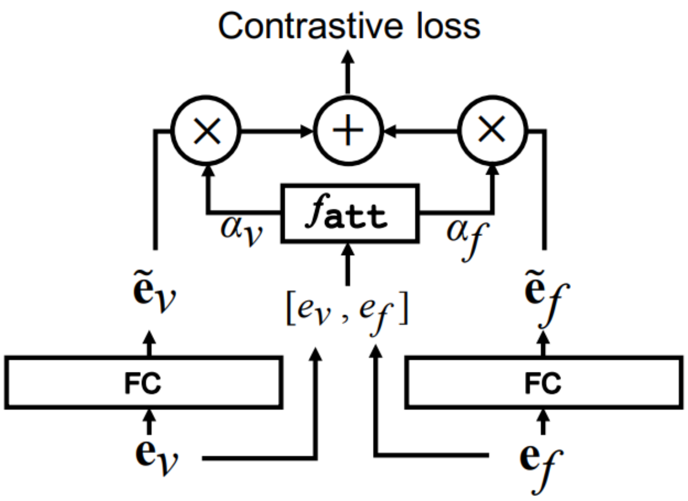


Diagram of soft attention fusion, e_v is voice embedding, e_f is face embedding
(don't mind the contrastive loss)

Environment Setup

To run this code, you need a Python environment with the following libraries installed: torch, torchvision, and torchaudio. The recommended way to set up the environment is by using Conda. You can easily install all required dependencies, including the correct versions and GPU support, by running the following command in your terminal:
conda env create -f environment.yml
Then you can activate it with:
conda activate sur_env
After that you can run my code with python commands.

Replicating results

To replicate the results, place the code in the same directory as the data folders. In *main.py*, you can configure parameters such as whether to use augmentation, batch normalization, dropout probability, etc. Set the mode parameter to either "audio", "images", or "both" — this determines whether you train the audio (x-vector), image (ResNet), or fusion model. Also, specify where to save the trained weights.

To evaluate, use *eval.py*. Again, set the mode. If you want to combine model outputs by adding class probabilities, set mode="both" and fusion=False. To evaluate the soft attention fusion model, use mode="both" and fusion=True. Then set the paths to the trained weights and the output file, and run the script — it will generate the output in the required assignment format.