Our project encompasses two distinct submissions, each addressing a specific machine learning task, implemented and documented in separate Python scripts within Jupyter notebooks.

**Task A: Gender Classification (Binary)**

Objective: Accurately predict the gender (Male/Female) of a subject from images captured under visually degraded conditions.

Implementation: A Python script leverages a pre-trained ResNet50 architecture, fine-tuned on a custom dataset. The script encompasses data preprocessing, dataset splitting, model training, and evaluation, with specific attention to mitigating class imbalance to ensure robust performance.

**1. Imports and Setup**

The script begins by importing necessary libraries:

* **os, shutil, random**: For file operations, copying, and randomization.
* **torch, torch.nn, torch.optim**: PyTorch libraries for building and training neural networks.
* **torchvision.datasets, torchvision.models, torchvision.transforms**: For loading datasets, pre-trained models, and applying image transformations.
* **time, copy**: For tracking training time and deep copying model weights.
* **sklearn.metrics**: For calculating precision, recall, and F1-score.
* **numpy**: For numerical operations, such as computing class weights.

**2. Creating a Master Dataset (create\_master\_dataset)**

This function combines the original training and validation datasets into a single master dataset to allow for a new train-validation split. Key steps:

* Creates the master\_dir directory to ensure a unified dataset for mitigating issues like data imbalance or insufficient validation examples
* Organizes images into subfolders (female, male) based on gender.

**3. Splitting the Master Dataset (split\_dataset)**

This function splits the master dataset into new training (70%) and validation (30%) sets.

**5. Data Transformations**

* **Training (train)**:
  + Resize images to 224x224 (ResNet50's expected input size).
  + Apply data augmentations to increase robustness:
    - RandomHorizontalFlip: Flips images horizontally with 50% probability.
    - RandomRotation: Rotates images by up to 15 degrees.
    - ColorJitter: Randomly adjusts brightness, contrast, and saturation.
    - RandomAffine: Applies random translations (up to 10% of image size).
    - RandomErasing: Randomly removes patches to simulate occlusions.
  + Convert images to tensors and normalize using ImageNet mean and standard deviation
* **Validation (val)**:
  + Resize to 224x224, convert to tensors, and normalize (no augmentations to preserve original data).

**8. Model Setup**

The script uses a pre-trained ResNet50 model from torchvision.models:

* Loads ResNet50 with default (ImageNet) weights.
* Freezes all parameters except those in layer4 (the last convolutional block) to fine-tune only deeper layers.
* Replaces the fully connected layer (model.fc) with a new layer for binary classification (2 classes: female, male).
* Moves the model to the appropriate device (GPU if available, else CPU).

Loss Function:

* Uses CrossEntropyLoss with class weights ([1.0, 159/824]) to penalize errors on the minority class (female) more heavily.

Optimizer:

* Uses Adam with a learning rate of 0.0001 and weight decay of 1e-4 to prevent overfitting.

Scheduler:

* Uses ReduceLROnPlateau to reduce the learning rate by a factor of 0.1 if the validation loss doesn't improve for 3 epochs.

**9. Training Function (train\_model)**

The train\_model function trains the model with early stopping and computes per-class metrics:

* **Training Loop**:
  + Iterates over epochs (default: 50).
  + For each phase (train, val):
    - Sets model to training or evaluation mode.
    - Processes batches, computes loss, and updates weights (training only).
    - Tracks running loss and accuracy.
    - For validation, collects predictions and labels to compute per-class metrics (precision, recall, F1-score) using sklearn.metrics.
* **Metrics**:
  + Prints loss and accuracy for both phases.
  + For validation, prints per-class precision, recall, and F1-score for female and male classes, plus weighted averages.
* **Early Stopping**:
  + Tracks the best validation accuracy and loss.
  + Saves the best model weights when validation performance improves.
  + Stops training if no improvement occurs for 7 epochs . Prevents overfitting.
* **Scheduler**:
  + Adjusts the learning rate based on validation loss (dynamic).
* Returns the model with the best validation accuracy.

**10. Training and Saving the Model**

* The script starts training with the configured model, criterion, optimizer, and scheduler.
* After training, it saves the model's state dictionary to /kaggle/working/resnet50\_task\_a.pth.
* Includes error handling for saving the model.

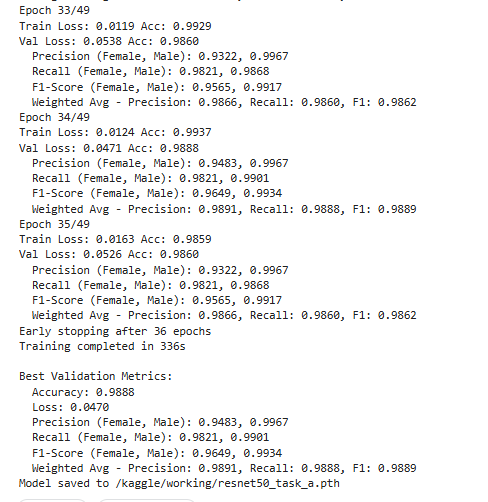


Fig 1: Execution screenshot for task A

Best validation accuracy obtained after execution for 35 epochs: 98.88%

**Summary: Implementation and Key Features of Gender Classification (Task A)**

Thus, the implementation of the Gender Classification task (Task A) incorporates several key features and considerations to ensure robust and reliable performance. Class imbalance is effectively addressed through the use of WeightedRandomSampler for balanced training batches and class-weighted loss functions to prioritize the minority class. Data augmentation techniques, including random flips, rotations, and erasing, enhance model generalization to handle visually degraded conditions. The fine-tuning strategy focuses on unfreezing only layer4 of the pre-trained ResNet50 model, striking a balance between training efficiency and model adaptation. Early stopping and dynamic learning rate scheduling are employed to prevent overfitting and optimize convergence. Per-class performance metrics are utilized to thoroughly evaluate the model, particularly crucial for imbalanced datasets. Finally, reproducibility is ensured by setting random seeds, guaranteeing consistent results across runs.

**Task B: Face Recognition (Multi-class):**

Implementation:

A Python script performs **face recognition** by learning a 256-dimensional embedding space where images of the same person have similar embeddings (close in distance) and images of different people have dissimilar embeddings (far apart). It uses:

* **Triplet Loss**: Trains the model on triplets (anchor, positive, negative) to minimize the distance between anchor and positive (same identity) and maximize the distance to negative (different identity).
* **Hard Triplet Mining**: Selects challenging negatives to improve embedding quality.

We use a pre-trained model (InceptionResnetV1 from facenet-pytorch or ResNet50 as a fallback) with triplet loss and hard triplet mining to optimize embeddings. The script handles dataset preprocessing, training, and evaluation, with early stopping to prevent overfitting.

**1. Imports and Setup**

**Imports**: os, shutil, random, torch, torch.nn, torch.optim, torchvision (models, transforms), numpy, PIL, sklearn.metrics (cosine similarity, F1-score) for file handling, deep learning, image processing, and evaluation.

**Setup**:

* **Dependencies**: Attempts facenet\_pytorch import/install; falls back to ResNet50 if unavailable.
* **Paths**: Defines input (/kaggle/input/latest-dataset/Task\_A), master, training, and validation directories.
* **Seeds**: Sets random, torch, numpy seeds to 42 for reproducibility.
* **Transforms**: Training (resize, augmentations, normalize); validation (resize, normalize).
* **Datasets/Loaders**: Creates TripletFaceDataset with hard triplet mining; DataLoader with batch size 32.

**2. Split dataset**

We split the master dataset into training (70%) and validation (30%) sets, maintaining identity-based folder structure.

**Process**:

* Randomly shuffles images per identity and splits them based on train\_ratio=0.7.
* Copies images to NEW\_TRAIN\_DIR and NEW\_VAL\_DIR, creating identity subfolders.
* Prints image counts per identity for training and validation.

**3. Data transformations**

Training:

* Resizes images to 224x224.
* Applies augmentations (horizontal flips, ±20° rotations, color jitter, affine transforms, Gaussian blur, random erasing) to improve robustness to variations like pose, lighting, or occlusions.
* Normalizes using ImageNet means and standard deviations.

Validation: Only resizes and normalizes for consistent evaluation.

**7. Embedding model**

Defines a neural network to map images to 256-dimensional embeddings.

Architecture:

* Backbone: Uses InceptionResnetV1 (512-dim output) if facenet-pytorch is available, otherwise ResNet50 (2048-dim output, flattened).
* BatchNorm: Stabilizes training with BatchNorm1d.
* Fully Connected Layer: Reduces dimensionality to 256.
* L2 Normalization: Ensures unit-length embeddings for cosine similarity.
* Forward Pass: Processes input images, flattens ResNet50 output if needed, and produces normalized embeddings

**8. Triplet loss**

Implements triplet loss to optimize embeddings.

loss = max(d(anchor, positive) - d(anchor, negative) + margin, 0), where d is Euclidean distance, and margin=1.2 ensures negative embeddings are farther than positive ones by at least 1.2.

**9. Inference Function**

Computes average embeddings per identity as reference embeddings.

* For each validation image, predicts the identity by finding the closest reference embedding (cosine distance).
* Calculates Top-1 accuracy (proportion of correct identity predictions), macro averaged F1-score, macro averaged Recall and macro averaged Precision (balanced measure across identities).
* Output: Prints Top-1 accuracy, macro averaged F1-score, macro averaged Recall and macro averaged Precision.

**10. Training Function (train\_model)**

* **Model**: EmbeddingNet with InceptionResnetV1 or ResNet50.
* **Optimizer**: Adam with learning rate 0.0001 and weight decay 1e-4.
* **Scheduler**: CosineAnnealingLR for cyclical learning rate adjustments.

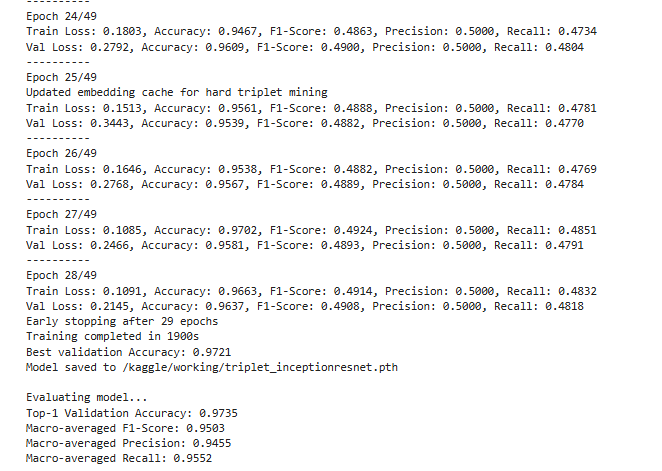


Fig 1: Execution screenshots for task B

Best validation accuracy obtained after execution for 28 epochs: 96.45%

**Summary: Implementation and Key Features of Face classification (Task B)**

Task B combines training and validation images into a master dataset, splits it into new training (70%) and validation (30%) sets, and employs a custom TripletFaceDataset with hard triplet mining to generate triplets (anchor, positive, negative) for learning 256-dimensional embeddings. The script uses InceptionResnetV1 (if facenet-pytorch is available) or ResNet50 as a backbone, optimized with TripletLoss (margin=1.2), Adam optimizer (lr=0.0001), and CosineAnnealingLR scheduler. Robust augmentations (20° rotations, Gaussian blur, hue jitter, random erasing) enhance training, while early stopping (patience=10) prevents overfitting based on validation accuracy. The model is evaluated using Top-1 accuracy and macro-averaged F1-score by comparing embeddings to per-identity averages, with dynamic facenet-pytorch installation handling for Kaggle compatibility, and the trained model is saved for future use.

**Conclusion**

**Task A** is designed for binary gender classification, importing WeightedRandomSampler and sklearn.metrics (precision, recall, F1-score) to handle class imbalance and evaluate per-class performance, while setting up a gender-based dataset (female, male) using ImageFolder, simpler augmentations (15° rotations, no blur), a partially frozen ResNet50 with a binary classification head, weighted CrossEntropyLoss, and a ReduceLROnPlateau scheduler. In contrast, **Task B** targets face matching, adding facenet\_pytorch and cosine\_similarity for embedding-based learning, using a custom TripletFaceDataset with hard triplet mining for identity-based data, more robust augmentations (20° rotations, Gaussian blur, hue jitter), a choice of InceptionResnetV1 or ResNet50 for 256-dim embeddings, TripletLoss to optimize embedding distances, a CosineAnnealingLR scheduler, and dynamic facenet-pytorch handling for Kaggle compatibility, reflecting its focus on learning discriminative embeddings rather than class labels.

**Task A** employs ResNet50 for binary classification (female vs. male), handling class imbalance (female: 159, male: 824) with WeightedRandomSampler and weighted CrossEntropyLoss, using simpler augmentations and ImageFolder for gender-based data, achieving robust performance through fine-tuning and detailed per-class metrics (precision, recall, F1-score). Conversely, **Task B** uses a Triplet Network with InceptionResnetV1 or ResNet50 to learn discriminative embeddings, leveraging hard triplet mining, robust augmentations (Gaussian blur, hue jitter), and TripletLoss for identity-based face matching, with dynamic dependency handling for facenet-pytorch. **Learnings** include the importance of tailored data pipelines (gender vs. identity organization), the effectiveness of class imbalance strategies in **Task A**, the power of triplet loss and hard mining for embedding learning in **Task B**, and the need for robust augmentations and dependency management in Kaggle. Both tasks highlight early stopping and learning rate scheduling as critical for preventing overfitting and optimizing performance.