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: 25).
  + 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.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 1: Execution screenshot for task A

Best validation accuracy obtained after execution for 20 epochs: 99.16%

**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):**

Objective: Correctly assign a face image to the corresponding person’s identity from a predefined set of individuals.  
Implementation: A separate Python script implements a face recognition model, designed to handle multi-class classification, with details on preprocessing, model architecture, training, and evaluation to be provided in the respective notebook.

The script implements a face recognition pipeline in a Kaggle environment:

* Combines training and validation datasets into a master dataset, handling normal and distorted images.
* Splits the dataset into new training (70%) and validation (30%) sets per person.
* Applies standard image preprocessing (resize, normalize) without augmentation.
* Trains a ResNet50 model with mixed precision and early stopping.
* Evaluates the model using Top-1 accuracy and macro-averaged F1-score.
* Saves the best and final model weights.

This is a robust setup for multi-class face recognition, optimized for performance and handling complex dataset structures with distorted images. The Python script implements a face recognition task using a ResNet50 model on a dataset of images organized by person, including both normal and distorted images. The script combines training and validation datasets, splits them into new training and validation sets, preprocesses the data, trains a deep learning model with mixed precision training, and evaluates its performance.

**1. Mixed Precision Training**:

* Uses GradScaler and autocast to perform computations in lower precision (e.g., FP16) on GPU, reducing memory usage and speeding up training.
* Scales gradients to prevent underflow in mixed precision.
* Iterates over epochs (default 30).
* Training phase: Computes loss, updates weights using Adam, and tracks loss and accuracy.
* Validation phase: Evaluates loss and accuracy without gradient computation.
* Saves the model with the best validation loss.
* Early stopping: Stops training if validation loss doesn’t improve for 5 epochs (patience).

**Output**: Prints training and validation loss/accuracy per epoch and saves the best model.

**2. Model Evaluation**

Metrics:

* **Top-1 Accuracy**: Percentage of correctly predicted classes.
* **Macro-averaged F1-Score**: Harmonic mean of precision and recall, averaged across all classes (useful for imbalanced datasets).

A screenshot of a computer

AI-generated content may be incorrect.

Fig 2: Execution screenshot of Task B

Best accuracy obtained after execution for 19 epochs (early stopping): 96.86%

**Key features of Facial image classification (Task B)**

The face recognition script for Task B performs multi-class classification to identify individuals from face images, contrasting with the previous script’s binary gender classification. Unlike the gender classification dataset, which organizes images into female and male folders, this script structures data by person-specific folders, with optional subfolders for distortions. Data transformations exclude augmentations like random flips or rotations, differing from the gender script’s robust augmentation strategy, to preserve identity-specific features. The script employs the Adam optimizer for faster convergence, replacing the SGD used in some alternative approaches, and incorporates mixed precision training with torch.cuda.amp for GPU optimization, a feature absent in the gender classification script. Evaluation metrics include macro-averaged F1-score alongside accuracy, suitable for the multi-class nature of face recognition, compared to the per-class metrics of the previous task. Additionally, the script handles edge cases by duplicating single images for both training and validation, ensuring robust dataset handling distinct from the gender classification approach.

**Conclusion**

In conclusion, the successful implementation of Task A (Gender Classification) and Task B (Face Recognition) demonstrates robust approaches to handling distinct machine learning challenges in facial image analysis. Task A effectively addressed binary classification under visually degraded conditions using a fine-tuned ResNet50 model, with strategies like WeightedRandomSampler, class-weighted loss, and extensive data augmentation to mitigate class imbalance and enhance generalization. Task B tackled multi-class face recognition with a person-specific dataset structure, employing Adam optimizer, mixed precision training for GPU efficiency, and macro-averaged F1-score to ensure balanced performance across multiple identities, while omitting augmentations to preserve identity-specific features. Key learnings include the importance of tailored preprocessing and optimization strategies to address task-specific challenges, such as class imbalance in gender classification and the need for precise feature preservation in face recognition. The use of advanced techniques like mixed precision training and per-class metrics highlights the adaptability of deep learning pipelines to diverse classification tasks.

For future scope, both tasks could benefit from exploring advanced architectures like Vision Transformers to potentially improve performance on complex datasets. Incorporating adversarial training or domain adaptation could enhance robustness to visual degradations, particularly for Task A. For Task B, integrating few-shot learning or continual learning could improve scalability to larger or dynamic sets of identities. Additionally, combining both tasks into a unified pipeline, leveraging shared feature extractors, could optimize computational efficiency and enable joint gender and identity prediction. Exploring real-time deployment and privacy-preserving techniques, such as federated learning, could further extend the practical applicability of these models.