**IT-JIM Task 1 - DL**

Arseny Khrutsky

---

## Introduction

The goal of this project is to develop a deep learning-based binary classifier that distinguishes between images containing artefacts and those that do not. The task is a part of a technical assessment and involves training a model using a provided image dataset, evaluating its performance using a Micro F1 Score, and optionally exploring multiple modeling approaches or ensembling methods.

The dataset includes 1800 training images and 200 test images, all named in the format image_<index>_<label>.png, where <label> is either 0 (artefact) or 1 (artefactless). The classification model is expected to generalize well on unseen data, particularly focusing on optimizing the Micro F1 Score metric.

This document outlines the steps taken in the development process, from exploratory analysis and preprocessing to training, validation, testing, and packaging of the final deliverable. The project is implemented in Python using PyTorch and structured for ease of reuse and reproducibility, including modularized code, CLI commands, and a documented pipeline.

---

## Dataset Overview

The dataset consists of:

- 1800 training images
- 200 test images

### File Structure

All images are stored in two separate directories:

- data/train/
- data/test/

Each image is named using the pattern:

image_<frame_index>_<class_label>.png

- <frame_index> is a numeric identifier
- <class_label> is either 0 (artefact) or 1 (artefactless)

### Class Distribution

During initial inspection, the dataset was observed to be slightly imbalanced toward one of the classes. Stratified sampling was used during train-validation split to preserve this balance.

### Image Format

- File type: PNG
- Size: resized to a fixed dimension during preprocessing
- Channels: 3 (RGB)

---

## Development Process

The entire training and inference pipeline was implemented in Python using PyTorch. The solution is modularized into components under the src/ directory for better organization and clarity. The main phases are described below.

### Preprocessing

- All images were resized to a fixed size (224x224) using torchvision.transforms
- Normalization was applied with mean and standard deviation values of [0.5, 0.5, 0.5]
- Data was loaded via a custom ArtifactDataset class inheriting from torch.utils.data.Dataset

### Model Architecture

- The base model used was ResNet18, preloaded from torchvision.models
- The final fully-connected layer was modified to output a single logit for binary classification

### Training Setup

- Loss Function: Binary Cross-Entropy with Logits (BCEWithLogitsLoss)
- Optimizer: Adam
- Learning Rate: 1e-4
- Batch Size: 32
- Device: CUDA (if available) or CPU

**Training & Validation**

- A custom train_one_epoch function handled the training loop
- Validation was performed after each epoch using the validate function
- Micro F1 score was computed at each epoch to monitor model performance
- The model with the highest validation F1 was saved to models/best_model.pth

**Inference Pipeline**

- Loads the best model checkpoint
- Processes test images with the same transformations
- Performs forward pass and thresholding for prediction
- Saves results to a CSV and computes Micro F1 Score on labeled test set

**Metric Used**

- Micro F1 Score: Chosen as the main evaluation metric due to its balance between precision and recall, especially in slightly imbalanced datasets

---

**Experiments & Evaluation**

Throughout development, several experiments were conducted to evaluate the effect of different training settings and augmentations on model performance.

**Augmentation Trials**

- Basic training pipeline used resizing and normalization only.
- A separate run was performed using RandomHorizontalFlip, RandomRotation, and ColorJitter.
- Results showed a slight improvement in generalization when using augmentation:
    - Micro F1 score without augmentation: 0.9400
    - Micro F1 score with augmentation: 0.9600
- Therefore, augmentations were included as an optional CLI flag in the training script.

**Model Selection**

- ResNet18 chosen for its balance between simplicity and power.
- Tested other lightweight models like MobileNetV2 (results were similar).
- Ensembling approaches considered but not included due to time constraints.

**Hyperparameter Tuning**

- Best results obtained with:
    - Learning rate: 1e-4
    - Batch size: 32
    - Epochs: 15–20
- Stratified train/val split helped maintain class balance.

**Validation Performance**

- Final validation Micro F1 with augmentation: 0.9556
- Best checkpoint saved based on validation F1

**Final Test Results**

- Micro F1 on Test Set: 0.9600
- Artifact-only Accuracy: 75.00%

Note: All metrics are calculated using sklearn.metrics.f1_score with average='micro'.

---

**Conclusion and Future Work**

This project successfully achieved its goal of classifying artefact vs. artefactless images using a deep learning model with a strong performance on the test set.

**Achievements**

- Designed a fully working training/validation/test pipeline
- Reached a high Micro F1 score of 0.9600 on the test set
- Created a clean and modular codebase with CLI support
- Logged and saved best model checkpoints
- Added optional data augmentation support

**Lessons Learned**

- Basic augmentations like flipping and rotation can improve generalization slightly
- Keeping preprocessing consistent between train and test is crucial
- Over-augmentation may reduce performance if not tuned

**Future Improvements**

- Try advanced augmentation libraries (Albumentations, etc.)
- Experiment with more powerful models (e.g., EfficientNet, ViT)
- Implement ensemble voting or averaging between model variants
- Explore interpretability (e.g., Grad-CAM)
- Evaluate classical computer vision approaches for comparison

The entire codebase is organized for clarity and reusability, making it easy to extend or adapt for future use cases.