# ECE381V Mini Project 1

Rasmus Laansalu
The University of Texas at Austin
rl37272@my.utexas.edu

Kutay Tire
The University of Texas at Austin
kutaytire@utexas.edu

Vignesh Nandakumar
The University of Texas at Austin
vnandakumar@utexas.edu

Fatima Al-Janahi
The University of Texas at Austin
fatimaaljanahi@my.utexas.edu

## Abstract

*We present two modular mini-projects under a unified evaluation harness. Project 1 builds and evaluates image classifiers on a 30-class subset of Food-101 using CNN and ResNet backbones, trained from scratch, via transfer learning, and with fine-tuning. Project 2 studies tabular learning on three datasets, comparing strong tree baselines (XGBoost, LightGBM, CatBoost) with TabPFN/TabNet, plus ensembling and a small transfer/fine-tuning experiment.*

## 1 Project 1 — Image Classification with Deep Learning

### 1.1 Introduction

Image classification aims to automatically assign semantic labels to images, a fundamental task in computer vision. This project explores deep learning approaches for food image classification using the Food-101 dataset, focusing on a subset of 30 classes to balance accuracy and computational efficiency. We compare three training regimes: training from scratch, transfer learning with a frozen pretrained backbone, and fine-tuning of the upper layers. To improve generalization and handle class imbalance, we apply data augmentation and class-aware sampling strategies. The goal is to evaluate how model initialization, augmentation, and balancing techniques influence performance and to identify the most effective training strategy for achieving robust classification results.

### 1.2 Datasets

The experiments use the Food-101 dataset, which contains 101 food categories and approximately 1,000 images per class collected from food photography websites. Each image is labeled and organized by class name, making it suitable for supervised classification. To reduce computational cost, we selected a subset of 30 visually diverse classes from the official list provided in `config.json`. Images were resized to $128 \times 128$ pixels to standardize input dimensions and enable faster training.

For evaluation, we adopted the standard Food-101 train–test split, which assigns 800 images per class for training and 200 for testing, filtered to include only the chosen 30 classes. This setup provides a balanced yet challenging benchmark for analyzing different training regimes and augmentation strategies.

### 1.3 Setup & Reproducibility

This project has certain python package requirements, which can be found in table 1. These are required to run our jupyter notebook and can be installed via your preferred package manager, e.g. pip, conda, etc.

Table 1. Python packages required for using the Food-101 dataset.

| Package | Version (Recommended) |
|---|---|
| `torch` | $\geq 2.0.0$ |
| `torchvision` | $\geq 0.15.0$ |
| `numpy` | $\geq 1.23.0$ |
| `pillow` | $\geq 9.0.0$ |
| `scikit-learn` | $\geq 1.0.0$ |
| `matplotlib` | $\geq 3.5.0$ |
| `seaborn` | $\geq 0.12.0$ |

After setting up the python environment, the project can be downloaded by cloning the GitHub repository `https://github.com/rasmus11423/Image_Classification_DL/tree/main`. In

the main folder of the project, open the notebook **01_download_food101** since this will prompt the download of the Food101 dataset. In this notebook, change the **PROJECT_ROOT** variable to reflect the path where the repository has been downloaded. Then run through the cells until the dataset has been successfully downloaded.

To run the classification code, open the **02_train_food** notebook in the project directory and change **PROJECT_ROOT** and **CONFIG_PATH** to reflect the locations of the project directory and the config file in your system. The rest of the code can be run continuously to generate classification results.

## 1.4 Hyperparameters

Table 2 lists the main hyperparameter configurations used for the three training regimes: training from scratch, transfer learning, and fine-tuning.

Table 2. Summary of hyperparameter configurations across training regimes.

| Hyperparameter | Scratch (CNN) | Transfer (ResNet-50) | Fine-Tune (ResNet-50) |
|---|---|---|---|
| Optimizer | AdamW | AdamW | AdamW |
| Learning Rate | $1 \times 10^{-3}$ | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| Weight Decay | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| Batch Size | 32 | 32 | 32 |
| Image Size | $128 \times 128$ | $128 \times 128$ | $128 \times 128$ |
| Epochs | 30 | 10 | 10 |
| Backbone | Simple CNN | ResNet-50 (pretrained) | ResNet-50 (pretrained) |
| Trainable Layers | All | Head only | Last block + head |
| Augmentation | None / Basic aug | Same | Same |
| Class Balancing | None / Weighted | Weighted | Weighted |
| Loss Function | Cross-Entropy | Cross-Entropy | Cross-Entropy |
| Scheduler | None | None | None |

The selected hyperparameters were chosen for stable convergence and fair comparison. AdamW provided consistent optimization across regimes, with lower learning rates for pretrained models to preserve learned features. A batch size of 32 and $128 \times 128$ resolution balanced speed and detail, while basic augmentation and weighted sampling improved robustness under class imbalance. Ten epochs were sufficient for convergence in all settings.

## 1.5 Results

Table 3. Performance comparison across training regimes.

| Training Regime | Accuracy (Val) | Macro-F1 (Val) | Train Acc. |
|---|---|---|---|
| From Scratch (CNN) | 0.167 | 0.150 | 0.162 |
| Transfer Learning (ResNet-50) | 0.688 | 0.648 | 0.693 |
| Fine-Tuning (ResNet-50) | 0.863 | 0.862 | 0.989 |

Table 3 summarizes the performance metrics obtained from the three training regimes in the 30-class Food-101 subset. Each model was evaluated using validation accuracy and macro-F1 to measure overall classification quality and class balance. The fine-tuned ResNet-50 achieved the highest performance across all metrics, confirming the effectiveness of transfer learning and selective layer unfreezing.

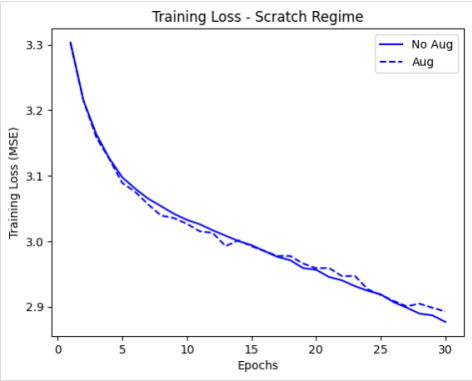


Figure 1. Training loss for scratch regime.

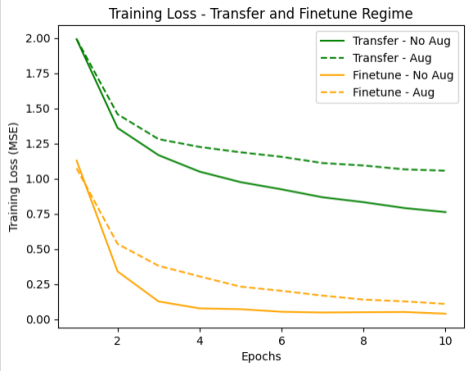


Figure 2. Training loss for transfer and fine-tune regime.

**Evaluating Training Loss** Figures 1 and 2 highlight the training loss in all three regimes with and without data augmentation. It can be seen that the loss for the scratch regime is the highest, followed by the transfer and fine-tune models. This is expected since the scratch model is randomly initializing its weights, whereas the other two regimes have a given weight distribution. Furthermore, the fine-tuning model has the most optimal weight initialization out of the tree regimes, resulting in a lower training loss starting point.

All three regimes have a similar descent, yet the transfer and fine-tune models without data augmentation have a lower loss than their data-augmented counterparts. This could be because adding augmented samples makes it harder for the model to learn/generalize across varying samples, leading to a higher loss. However, testing against the validation set that is not augmented yields better performance, highlighting the efficacy of augmentation. The scratch regime shows similar levels of loss because the model is still in the early stages of training. If given more epochs, then a similar divergence between the non-augmented and augmented curves would be seen.

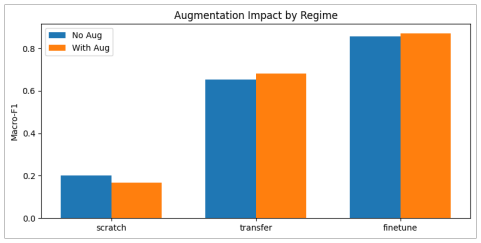


Figure 3. Macro-F1 impact by training regime.

**Augmentation and Imbalance Study.** Figure 3 shows the macro-F1 scores with and without augmentation for each regime, highlighting the clear improvement with pre-trained models. An interesting observation is that including augmented samples when training from scratch decreased the macro-F1 score. However, this could be attributed to only training the model over 30 epochs since having a larger variance across samples may take longer to learn. In both the transfer and fine-tune regime, adding augmented samples slightly improved the overall macro-F1 score. Since both of these models were only trained over 10 epochs, increasing the training time may lead to a larger performance gap with data augmentation.

The confusion matrix in Figure 4 demonstrates strong diagonal dominance, indicating effective per-class recognition after fine-tuning.

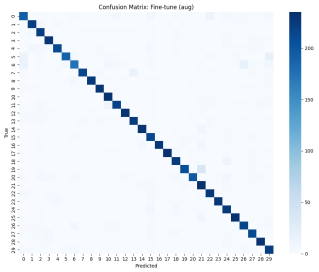


Figure 4. Confusion matrix for the fine-tuned ResNet-50 model with augmentation.

### 1.6 Discussion

This project demonstrated the effectiveness of deep learning approaches for food image classification on a 30-class subset of Food-101. Transfer learning and fine-tuning significantly outperformed training from scratch, confirming the benefit of leveraging pretrained features. Augmentation and class balancing improved robustness and macro-F1 performance across all regimes. Future work could explore larger subsets, caption-based multimodal training, and more advanced augmentation strategies to further enhance model generalization.

## 2 Project 2 — Towards Foundation Models for Tabular Data

### 2.1 Introduction

The goal of this project is to examine when modern tabular deep learning models can match or surpass strong classical tree ensembles under a unified, reproducible evaluation. While XGBoost, LightGBM, and CatBoost remain the de facto choices for many tabular problems, recent "foundation" approaches such as TabPFN and representation-learning models like TabNet promise competitive performance with less tuning and potential for transfer across datasets. We evaluate both families side-by-side on three heterogeneous datasets that jointly cover multi-class numeric classification, binary categorical-heavy classification, and real-valued regression. Beyond head-to-head accuracy, we also study a simple ensemble that combines a deep model with a tree model, and we test whether a model pre-trained on one dataset can be fine-tuned to another with a measurable benefit.

### 2.2 Datasets

We select three datasets to cover complementary regimes and to align with prior evaluations in the TabPFN/TabNet literature. *mfeat-fourier* (OpenML ID 14; `https://www.openml.org/d/14`) provides a clean, purely numeric, 10-class classification task where models must capture smooth class boundaries from Fourier descriptors. *German Credit* (credit-g, OpenML ID 31; `https://www.openml.org/d/31`) is a binary, categorical-heavy dataset that stresses encoders and native categorical handling; tree ensembles are known to be competitive here. *SARCOS inverse dynamics* (`https://www.openml.org/d/43873`) is a realistic robotics regression benchmark with 21 inputs and seven torque targets; we follow common practice and model a single target (y1). Together these datasets allow us to test multi-class vs. binary classification, categorical vs. numeric features, and small/medium classification vs. larger regression.

The following three datasets are used for *all experiments except transfer learning* (i.e., for classical baselines, DL models, ensembling, and fine-tuning). For transfer learning specifically, we adopt different pairs that have compatible feature schemas.

Table 4. Datasets and schema used for *non-transfer* experiments in Project 2. Exact counts are printed by our loaders in the notebook; the values above match commonly reported versions.

| Dataset | Task | #Samples | #Feat. | Cat/Num |
|---|---|---|---|---|
| mfeat-fourier | 10-class cls | ∼2,000 | 76 | 0 / 76 |
| credit-g | binary cls | 1,000 | 20 | 13 / 7 |
| SARCOS (y1) | regression | ∼53,000 | 21 | 0 / 21 |

We chose *mfeat-fourier*, *credit-g*, and *SARCOS* to represent the most significant regimes where table-based approaches differ: (i) a straightforward, purely numeric multiclass problem (mfeat-fourier) that tests representation learning in the absence of categorical confounding; (ii) a small, categorical-heavy binary problem (credit-g) where tree-based ensembles are conventionally dominant and deep networks must handle discrete features consistently; and (iii) a large, real-valued regression task with non-trivial dynamics (SARCOS) that tests stability and calibration with continuous prediction. This mix enables us to inquire when classical trees dominate, when foundation-model-type models can match or surpass them, and whether a model trained on one domain can generalize to another with minimal adjustment.

**Transfer learning datasets.** For transfer learning we require datasets with fully or partially aligned feature schemas. We therefore use two pairs from the TabPFN evaluation suite on OpenML: *heart-statlog* (`https://www.openml.org/d/53`) → *heart-c* (`https://www.openml.org/d/49`) for *partially* aligned mixed-type features, and *monks-problems-1* (`https://www.openml.org/d/333`) → *monks-problems-2* (`https://www.openml.org/d/334`) for *fully* aligned categorical features.

Table 5. Datasets used *only* for transfer-learning experiments. We select pairs with compatible schemas. All are from the TabPFN evaluation suite on OpenML. Exact counts are printed by our loaders in the notebook.

| Dataset | Task | #Samples | #Feat. | Cat/Num |
|---|---|---|---|---|
| heart-statlog | binary cls | ∼270 | 13 | 6 / 7 |
| heart-c | binary cls | ∼303 | 13 | 6 / 7 |
| monks-problems-1 | binary cls | ∼432 | 6 | 6 / 0 |
| monks-problems-2 | binary cls | ∼432 | 6 | 6 / 0 |

## 2.3 Setup & Reproducibility

All models are trained with a global seed of 42 and evaluated under 5-fold cross-validation (stratified for classification). For tree baselines we follow widely used strong defaults to reduce overfitting and keep comparisons fair. For deep models, we use AdamW with ReduceLROnPlateau and early stopping; batch sizes are set large enough to exploit GPU throughput without exhausting memory. TabPFN uses a small ensemble count for speed and, where needed, the library's pretraining-limit override to handle larger splits. We report mean ± std across folds and save the raw CSV outputs used in the figures/tables.

Table 6. Classical baselines: fixed hyperparameters used across datasets.

| Hyperparameter | XGBoost | LightGBM | CatBoost |
|---|---|---|---|
| Task (Cls / Reg) | Both | Both | Both |
| Learning Rate | 0.05 | 0.03 | 0.05 (Cls) / 0.03 (Reg) |
| # Trees (Estimators) | 600 (Cls) / 800 (Reg) | 1200 | 1200 (Cls) / 1500 (Reg) |
| Depth / Leaves | max_depth=6 / 8 | num_leaves=63 / 127 | depth=6 / 8 |
| Subsample | 0.8 | 0.8 | – |
| Colsample_bytree | 0.8 | 0.8 | – |
| Tree Method | `hist` | default | Oblivious trees |
| Categorical Handling | Ordinal encoder | Built-in + encoder | Native categorical |
| Eval Metric | logloss (Cls) | default | default |
| Random Seed | 42 | 42 | 42 |

Table 7. Deep tabular models: unified hyperparameters. TabPFN uses amortized inference (no SGD); TabNet is trained per dataset.

| Hyperparameter | TabPFN (Cls/Reg) | TabNet (Cls) | TabNet (Reg) |
|---|---|---|---|
| Optimizer | n/a (posterior) | AdamW | AdamW |
| Learning rate | n/a | $1 \times 10^{-3}$ | $1 \times 10^{-3}$ |
| Weight decay | n/a | $1 \times 10^{-5}$ | $1 \times 10^{-5}$ |
| # epochs | n/a | 250–300 (ES) | 300–400 (ES) |
| Patience (early stop) | n/a | 30–40 | 45–60 |
| Batch size | n/a | 2048 | 512 |
| Virtual batch size | n/a | 256 | 128 |
| $n_d/n_a$ | n/a | 32 / 32 | 32 / 32 |
| $n_{\text{steps}}$ | n/a | 5 | 5 |
| $\gamma$ | n/a | 1.5 | 1.5 |
| $\lambda_{\text{sparse}}$ | n/a | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| Scheduler | n/a | ReduceLROnPlateau | ReduceLROnPlateau |
| Ensemble size ($N$) | 32 | n/a | n/a |
| Device | auto (CUDA/CPU) | auto (CUDA/CPU) | auto (CUDA/CPU) |
| Target standardization | n/a | no | per-fold (fit/invert) |

**Fine-tuning.** Following the baseline setup, we additionally evaluate cross-dataset transfer by fine-tuning a TabNet model pretrained on a source dataset and adapted to a target dataset. Because TabNet requires identical feature dimensions for both pretraining and fine-tuning, we apply Principal Component Analysis (PCA) to reduce the higher-dimensional input to match the lower-dimensional one, ensuring a consistent feature interface for transfer learning. For both stages, we employ the same TabNet configuration shown in Table 7, using a smaller learning rate during fine-tuning ($2 \times 10^{-3}$). After pretraining, the resulting model checkpoint is saved and reloaded for fine-tuning to transfer the learned parameters onto the target dataset.

**Transfer learning.** We further evaluate TransTab, a transformer-based foundation model for tabular data that

Table 8. Fine-tuning hyperparameters (TabNet; classification). Values reflect the `_tabnet_finetune_cls` function.

| Hyperparameter | Setting |
|---|---|
| Backbone (TabNet) | $n_d$=32, $n_a$=32, $n_{\text{steps}}$=5, $\gamma$=1.3, $\lambda_{\text{sparse}}=10^{-4}$ |
| Optimizer | Adam (`optimizer_params`: lr $= 2 \times 10^{-3}$) |
| Max epochs / Patience | 250 / 20 (early stopping on val metric) |
| Batch size / Virtual | 1024 / 128 |
| Eval metric | AUC (binary) or Accuracy (multiclass) |
| Class weights | `balanced` (sample weights on train) |
| Seed / Verbose | 42 / 0 |
| Device | auto (CUDA/CPU) |

Table 9. Transfer-learning hyperparameters (TransTab). Values reflect `cv_transtab_5fold` with transfer enabled.

| Hyperparameter | Setting |
|---|---|
| Checkpoint init | Load from `file_name` (pretrained A) |
| Epochs / Patience | 50 / 15 (early stopping) |
| Batch size | 64 |
| Learning rate | $5 \times 10^{-5}$ |
| Optimizer | library default (unchanged) |
| Pin memory | `True` |
| Schema update | {`cat`, `num`, `bin`} set to target B |
| Device | auto (CUDA/CPU) |
| Prediction threshold | fixed at 0.5 (no threshold tuning) |

enables transfer learning across datasets with partially or fully aligned feature spaces. In contrast to TabNet, TransTab can accommodate differing column sets by learning column-type embeddings and a cross-feature attention mechanism. To prepare the datasets, we categorize each feature into categorical, numerical, and binary groups using an automated column parser. This information is passed to TransTab's encoder so that categorical embeddings and numerical projections are processed jointly within the model. For each dataset pair, we perform 5-fold cross-validation using the same hyperparameters. The model is first pretrained on a source dataset and the checkpoint is stored, then fine-tuned (transfer mode) on the target dataset by re-loading this checkpoint and updating the column schema. During fine-tuning, TransTab reuses the pretrained encoder weights while adapting the classifier head to the target labels.

## 2.4 Results

For the first part, across the three benchmarks (Table 10), TabPFN is the strongest single model overall: it leads on *mfeat-fourier* for all classification metrics and slightly outperforms the tree baselines on *credit-g*. On *SARCOS*, TabPFN achieves the lowest MAE/RMSE, with CatBoost a close second.

Table 10. Comparison on three tabular datasets. Classification cells report *ROC-AUC / Accuracy / Macro-F1*; regression reports *MAE / RMSE*. Best per metric is **bold**; second-best is underlined.

| | mfeat-fourier (cls) | | | credit-g (cls) | | | SARCOS (reg) | |
|---|---|---|---|---|---|---|---|---|
| **Model** | ROC-AUC | Acc. | F1 | ROC-AUC | Acc. | F1 | MAE | RMSE |
| XGBoost | 0.983 | 0.840 | 0.839 | 0.777 | 0.762 | 0.697 | 1.532 | 2.218 |
| LightGBM | 0.981 | 0.829 | 0.828 | 0.697 | 0.718 | 0.629 | 1.398 | 2.066 |
| CatBoost | 0.984 | 0.841 | 0.842 | 0.782 | 0.760 | 0.686 | 1.339 | 1.964 |
| TabNet | 0.951 | 0.723 | 0.719 | 0.464 | 0.475 | 0.454 | 1.598 | 2.441 |
| TabPFN | **0.992** | **0.891** | **0.890** | **0.798** | **0.770** | **0.704** | **1.117** | **1.962** |

For the ensembel trask, on *credit-g*, stacking TabPFN with calibrated XGBoost raises ROC–AUC and accuracy but lowers macro–F1 (Table 11); this suggests complementary rank calibration between DL and trees, while class-wise balance remains sensitive to the decision threshold.

Table 11. Ensembling on *credit-g* (OOF). Metrics are ROC-AUC / Accuracy / Macro-F1. The stacking ensemble improves AUC and accuracy but slightly lowers macro-F1.

| Model | ROC-AUC | Acc. | Macro-F1 |
|---|---|---|---|
| XGBoost (calibrated, OOF) | 0.7798 | 0.769 | 0.6773 |
| TabPFN (OOF) | 0.7979 | 0.770 | 0.7041 |
| **Stacking [TabPFN + XGBcal]** | **0.7990** | **0.772** | 0.6927 |
| $\Delta$ vs XGB(cal) | **+0.0192** | **+0.003** | $-0.0154$ |
| $\Delta$ vs TabPFN | **+0.0012** | **+0.002** | $-0.0114$ |

For transfer learning, gains are positive for both pairs, larger when schemas are fully aligned. This pattern is consistent with feature compatibility aiding representation reuse.

Table 12. Transfer learning vs. baseline on *heart-statlog* $\rightarrow$ *heart-c*. $\Delta$ represents the improvement.

| Metric | Baseline (mean $\pm$ std) | Transfer (mean $\pm$ std) | $\Delta$ |
|---|---|---|---|
| Accuracy | $0.8039 \pm 0.0317$ | $0.8173 \pm 0.0492$ | **+0.0134** |
| Macro-F1 | $0.7826 \pm 0.0470$ | $0.7901 \pm 0.0633$ | **+0.0075** |
| ROC-AUC | $0.8931 \pm 0.0453$ | $0.9015 \pm 0.0370$ | **+0.0083** |

Table 13. Transfer learning vs. baseline on *monks-problems-1* $\rightarrow$ *monks-problems-2*. $\Delta$ represents the improvement.

| Metric | Baseline (mean $\pm$ std) | Transfer (mean $\pm$ std) | $\Delta$ |
|---|---|---|---|
| Accuracy | $0.7366 \pm 0.1324$ | $0.8033 \pm 0.0278$ | **+0.0667** |
| Macro-F1 | $0.7817 \pm 0.0777$ | $0.8273 \pm 0.0242$ | **+0.0456** |
| ROC-AUC | $0.8809 \pm 0.0278$ | $0.8832 \pm 0.0282$ | **+0.0022** |

## 2.5 Discussion

**Q1.** In these runs, **TabPFN** is consistently top across all three datasets, including mixed-type *credit-g* and large numeric *SARCOS*. **Tree-based models** still tend to dominate in many practical settings when (i) data are small/medium with many categoricals, (ii) careful CV regularization is applied, and (iii) deep models are under-tuned; in our results, however, TabPFN's prior/architecture gave it the edge.

**Q2.** On *credit-g*, stacking *TabPFN + calibrated XGBoost* improved ROC-AUC to 0.7990, which is **+0.0192** over XGBoost (0.7798) and **+0.0012** over TabPFN (0.7979). Accuracy also nudged up to 0.772 (**+0.003** vs. XGB; **+0.002** vs. TabPFN), while macro-F1 decreased to 0.6927 ($-0.0154$ vs. XGB; $-0.0114$ vs. TabPFN).

The ensemble lifts AUC and accuracy by combining complementary biases (additive partitions from trees + distributed representations from DL), but its decision threshold favors calibration/AUC more than balanced class-wise F1. If macro-F1 is critical, tune the stacking meta-learner and/or select class thresholds to directly optimize macro-F1 (or use cost-sensitive training).

**Q3.** Overall, pretraining helps as it can be seen from Tables 12 and 13. For the *heart-statlog* $\rightarrow$ *heart-c* pair , we see **+0.0134** Accuracy, **+0.0075** macro-F1, and **+0.0083** AUC. For the *monks-1* $\rightarrow$ *monks-2* pair, gains are larger on level-based metrics: **+0.0667** Accuracy and **+0.0456** macro-F1, with a modest AUC rise of **+0.0022**.

When source and target share feature semantics and column schemas (the *monks* pair), the pretrained encoder transfers decision structure and class separation more directly, boosting Accuracy/F1. On the other hand, with only partial alignment (the *heart* pair), improvements are smaller because column semantics and distributions do not match perfectly; nevertheless, shared low-level statistics still provide a useful prior. Using a slightly lower learning rate and more patient early stopping during transfer helps retain beneficial source features while adapting the prediction head to the target.