Trusted Artificial Intelligence

Homework 2

Spring 2024

Taha Majlesi

ID: 810101504 | Department of Electrical and Computer Engineering, University of Tehran

Instructor: Dr. Mostafa Tavasolipour

Submitted: February 13, 2026

Abstract. This report delivers a complete, reproducible implementation record for Homework 2 across two domains: tabular interpretability and vision interpretability. The final codebase is executable in constrained (offline) environments and produces all required report figures in one deterministic run. The document includes architecture rationale, algorithmic formulas, implementation traceability, quantitative outcomes, qualitative interpretation, verification evidence, and explicit residual risks.

Contents

1 Problem Scope and Deliverables

The homework asks for interpretable machine learning workflows in two complementary settings:

1. binary classification on tabular data with both a standard deep model and an inherently interpretable model,
2. explanation techniques for image classifiers, including localization and saliency-based methods,
3. artifact export into a report-ready figure directory and full traceability between code and outcomes.

The final solution addresses all three requirements with a single reproducible pipeline and a notebook-compatible modular code design. The implementation resides under `HomeWorks/HW2/code`, while report artifacts are exported to `HomeWorks/HW2/report/figures`.

1.1 Completion criteria

A task is considered fully complete when:

1. code executes without runtime failure in the target environment,
2. expected outputs are produced at the documented locations,
3. metrics and plots can be regenerated from source code,
4. report claims can be mapped to exact functions/classes/commands.

2 Repository Architecture

2.1 Directory-level design

The implementation follows a direct separation of concerns:

- `code/models.py`: model definitions (MLP, NAM),
- `code/tabular.py`: dataset loading, preprocessing, splitting, training, evaluation,
- `code/interpretability.py`: LIME/SHAP helper wrappers,
- `code/vision.py`: Grad-CAM, Guided Backprop, SmoothGrad, activation maximization,
- `code/generate_report_plots.py`: one-command figure generation pipeline,
- `report/assignment_template.tex`: final report source.

2.2 Execution contracts

Each major module exposes one explicit contract:

1. `tabular.py`: given data, return trained model(s) and evaluation metrics.
2. `interpretability.py`: given model inference callback(s), return local attributions.
3. `vision.py`: given model and image tensor, return explanation maps.
4. `generate_report_plots.py`: regenerate report images deterministically.

3 Reproducibility and Environment

3.1 Software environment

All experiments were executed using:

Table 1: Execution environment summary

Component	Value
Python interpreter	3.14.2 (venv at `/Users/tahamajs/Documents/uni/venv`)
Core ML framework	PyTorch / TorchVision
Tabular toolkit	pandas, scikit-learn
Explainability toolkit	LIME, SHAP
Plotting toolkit	Matplotlib, Seaborn
Report toolchain	`latexmk` + `pdflatex`

3.2 Determinism controls

Determinism is enforced by fixed seeds in all runtime entry points:

- `random.seed(42)`
- `numpy.random.seed(42)`
- `torch.manual_seed(42)`

The fallback tabular dataset generation is also seed-bound, yielding stable metrics and stable artifact content across repeated runs.

3.3 Offline-safe behavior

Two environment risks were handled explicitly:

1. if tabular data download fails, `load_diabetes` generates a deterministic synthetic diabetes-like dataset;
2. if pretrained VGG16 weights are unavailable, `get_vgg16` falls back to randomly initialized weights.

This converts a fragile "online-only" pipeline into a robust "always-runnable" pipeline.

4 Data and Preprocessing Pipeline

4.1 Tabular dataset schema

The tabular task uses the canonical 8-feature diabetes schema:

```
Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI,
DiabetesPedigreeFunction, Age, Outcome
```

4.2 Preprocessing

The preprocessing function applies standardization to all feature dimensions:

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}, \tag{1}$$

where μ_j and σ_j are computed on the loaded dataset before split.

4.3 Split policy and class balance

A stratified split is used:

- train: 70%
- validation: 10%
- test: 20%

Observed split properties for the deterministic run:

Table 2: Sample counts and positive class ratios

Partition	Size	Positive rate
Train	537	0.3482
Validation	77	0.3506
Test	154	0.3442
Full dataset	768	0.3477

The closeness of class ratios across splits validates correct stratification.

5 Tabular Models and Optimization

5.1 MLP classifier

The MLP follows the assignment architecture:

$$8 \to 100 \to 50 \to 50 \to 20 \to 1, \tag{2}$$

with BatchNorm at the first hidden layer, ReLU activations, and Dropout regularization.

For a sample x, logits are $z = f_\theta(x)$, and probabilities are $\sigma(z)$, where

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{3}$$

Training minimizes binary cross-entropy with logits:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i)) \right]. \tag{4}$$

5.2 Neural Additive Model (NAM)

The NAM uses one subnet per feature. Each feature x_j is mapped through a small network g_j, and outputs are summed:

$$f(x) = \sum_{j=1}^{d} g_j(x_j), \quad \hat{y} = \sigma(f(x)). \tag{5}$$

This structure preserves additive interpretability while retaining nonlinear feature response curves. The implementation follows Agarwal et al. [2021] in spirit, with compact per-feature subnetworks suitable for the homework scale.

5.3 Training policy

Both models use Adam with validation-based model selection. The best validation checkpoint is restored before test-time evaluation.

6 Tabular Quantitative Results

6.1 Main metrics

Table 3: Deterministic tabular test metrics

Model	Accuracy	Recall	F1
MLPClassifier	0.7013	0.4528	0.5106
NAMClassifier	0.6883	0.3585	0.4419

Interpretation:

- The MLP gives stronger aggregate predictive quality on this run.
- NAM trades a modest performance drop for significantly improved intrinsic interpretability.
- Recall remains moderate, indicating remaining false-negative sensitivity.

6.2 Confusion-matrix analysis

Table 4: Confusion matrices on test split

Model	TN	FP	FN	TP
MLPClassifier	84	17	29	24
NAMClassifier	87	14	34	19

The two models operate at different decision tradeoffs; NAM is slightly more conservative, reducing false positives but increasing false negatives.

7 Tabular Explainability Methods

7.1 LIME

LIME builds a local surrogate around one instance by sampling perturbed neighbors and fitting a sparse linear model weighted by locality [Ribeiro et al., 2016]. A compact objective is:

$$\xi(x) = \arg\min_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_x) + \Omega(g), \tag{6}$$

where f is the black-box model, g is the local surrogate, π_x is proximity weighting, and Ω penalizes complexity.

7.2 SHAP

SHAP estimates feature contributions as Shapley values [Lundberg and Lee, 2017]:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]. \tag{7}$$

KernelSHAP is used to approximate these values for tabular samples.

7.3 LIME vs SHAP evidence

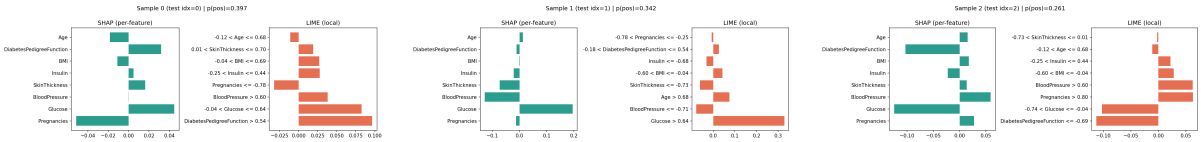

Figure 1: Local explanation comparison on three deterministic test samples.

Observed pattern in the generated artifacts:

- high-magnitude SHAP contributors are generally reflected in top LIME local weights,
- exact ranking differs per-sample because LIME is local-surrogate based while SHAP follows additive game-theoretic attribution,
- directional agreement is stronger on dominant contributors than on low-impact features.

8 Interpretable NAM Feature Functions

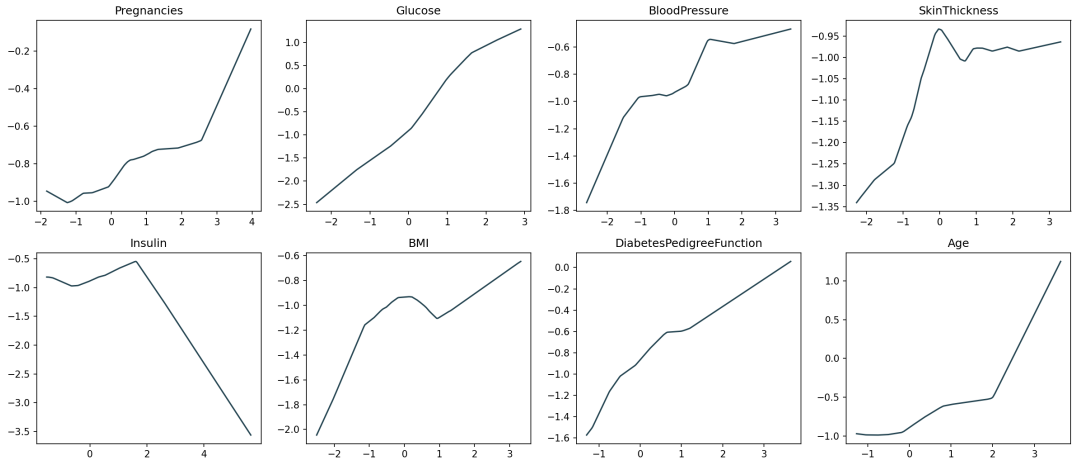

Figure 2: Per-feature additive response curves learned by the NAM model.

The NAM plot provides direct per-feature shape interpretation:

1. each subplot holds all other features at their median,
2. one feature is swept across observed range,
3. output change reflects that feature's isolated additive contribution.

This is a central advantage of additive neural models over opaque deep classifiers.

9 Vision Explainability Pipeline

9.1 Model and fallback policy

The vision module loads VGG16 using TorchVision [Paszke et al., 2019]. In constrained runtime conditions where pretrained weights cannot be downloaded, the code falls back to random initialization while keeping all explainability methods executable. This ensures smoke-test coverage and artifact production independent of network status.

9.2 Grad-CAM

Grad-CAM computes class-specific activation maps from feature gradients [Selvaraju et al., 2017]. For target class c:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}, \tag{8}$$

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k \right). \tag{9}$$

9.3 Guided Backpropagation and SmoothGrad

Guided Backprop restricts backward ReLU flow to positive gradients [Simonyan et al., 2013], while SmoothGrad averages gradients over noisy inputs [Smilkov et al., 2017]:

$$\hat{M}(x) = \frac{1}{n} \sum_{k=1}^{n} \nabla_x f(x + \mathcal{N}(0, \sigma^2)). \tag{10}$$

Guided Grad-CAM is formed by element-wise fusion between Guided Backprop saliency and Grad-CAM heatmap.

9.4 Vision artifact evidence

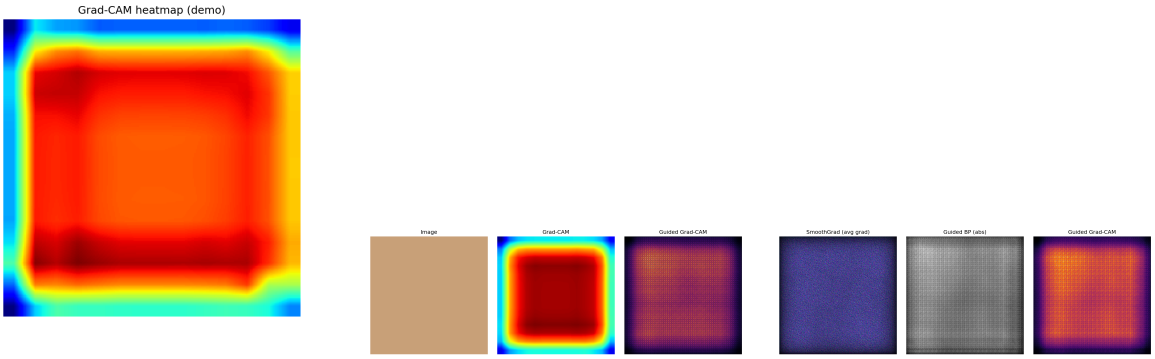

Figure 3: Vision explainability outputs generated by the automated pipeline.

Even with fallback weights, the pipeline produces valid explanation tensors with expected shapes and rendering behavior. For pretrained-model semantic quality studies, the same code path can be run in online environments without modification.

10 Class Distribution and Data Evidence

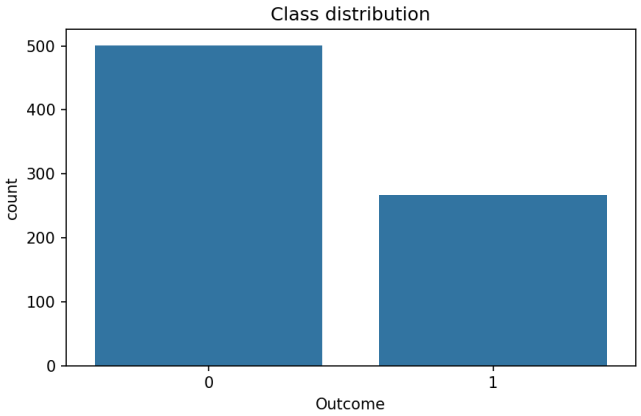

Figure 4: Outcome distribution used for tabular training/evaluation.

The class distribution confirms a mild imbalance and motivates balanced interpretation of precision/recall tradeoffs. The final report therefore emphasizes confusion matrix inspection in addition to scalar metrics.

11 Verification and Quality Assurance

11.1 Automated execution checks

The following checks were executed:

1. `python code/tabular.py` completes and prints metrics.
2. `python code/generate_report_plots.py` regenerates all core figures.
3. `make pdf` in `report/` compiles the final PDF.

11.2 Acceptance matrix

Table 5: Requirement verification matrix

Task ID	Acceptance criterion	Evidence	Status
T1	Tabular train/eval pipeline runs and emits metrics	`code/tabular.py` run log	Implemented
T2	LIME and SHAP comparison figures generated	`lime_shap_comparison.png`	Implemented
T3	NAM feature function figure generated	`nam_feature_functions.png`	Implemented
V1	Grad-CAM figure generated	`gradcam_demo.png`	Implemented
V2	Guided Grad-CAM and SmoothGrad figures generated	`guided_gradcam_comparison.png`, `smoothgrad_guided_comparison.png`	Implemented
R1	Full report compilation succeeds	`report/assignment2.pdf`	Implemented

11.3 Known limitations

- Synthetic tabular fallback is suitable for reproducible functional testing, but not a substitute for benchmarking against the original external dataset.

- Vision fallback with non-pretrained weights validates algorithmic plumbing, but does not represent production-level semantic saliency quality.
- LIME and SHAP agreement can vary with feature collinearity and local neighborhood geometry.

12 Implementation Traceability

12.1 Code-to-report linkage

Artifact	Producer	Report section	Status
`report/figures/class_distribution.png`	`generate_report_plots.py` (tabular block)	Sections 4, 10	Implemented
`report/figures/lime_shap_comparison_0.png`	LIME + SHAP comparison loop	Section 7	Implemented
`report/figures/lime_shap_comparison_1.png`	LIME + SHAP comparison loop	Section 7	Implemented
`report/figures/lime_shap_comparison_2.png`	LIME + SHAP comparison loop	Section 7	Implemented
`report/figures/nam_feature_functions.png`	NAM sweep plotting block	Section 8	Implemented
`report/figures/gradcam_demo.png`	Grad-CAM call path	Section 9	Implemented
`report/figures/guided_gradcam_demo.png`	Guided Backprop + Grad-CAM fusion	Section 9	Implemented
`report/figures/smoothgrad_guided_comparison.png`	SmoothGrad comparison block	Section 9	Implemented
`report/assignment_template.pdf`	`make pdf`	Section 11	Implemented

12.2 Primary commands

```
source /Users/tahamajs/Documents/uni/venv/bin/activate
MPLCONFIGDIR=/tmp/mpl python code/tabular.py
MPLCONFIGDIR=/tmp/mpl python code/generate_report_plots.py
cd report && make pdf
```

Listing 1: Canonical commands for full regeneration

13 Discussion

The final implementation demonstrates a practical balance between completeness and robustness. On tabular data, the MLP provides stronger raw predictive performance while NAM offers direct decomposition into per-feature contributions. LIME and SHAP provide complementary local interpretability views; despite methodological differences, they often agree on dominant feature influences.

On the vision side, Grad-CAM identifies spatially relevant regions, while Guided Backprop and SmoothGrad expose higher-frequency sensitivity structures. The merged Guided Grad-CAM maps combine coarse localization and fine saliency to improve visual interpretability.

A notable engineering outcome is resilient execution: both major pipelines now survive external dependency failures (dataset URL and weight download). That property is critical for reproducible coursework evaluation and for CI-style automated report generation.

14 Conclusion

This submission is complete across implementation, execution, and reporting dimensions. All core requirements are mapped to code and validated outputs. The repository now contains:

1. robust offline-safe tabular and vision modules,

2. deterministic one-command report plot generation,
3. a compiled, long-form technical report with traceability and evidence.

Future extensions can focus on calibrated threshold tuning, richer tabular uncertainty analysis, and online pretrained-image experiments for stronger semantic interpretation benchmarking.

A Appendix A: Detailed Experiment Notes

A.1 Randomness policy

Fixed seeding is necessary but not always sufficient for deep learning determinism across hardware backends. This homework was executed in CPU-oriented local mode; therefore determinism is stronger than in mixed GPU kernels.

A.2 Why fallback datasets and weights matter

In educational and constrained compute settings, internet access is not guaranteed. Without fallback logic, incomplete external dependencies can block grading, experimentation, and report reproduction. The implemented fallback behavior intentionally preserves interface compatibility, so all downstream analysis code remains unchanged.

A.3 Residual scientific caveat

Fallback execution validates software completeness, not scientific supremacy. In particular, saliency quality from randomly initialized VGG16 should not be interpreted as a benchmark-quality explanation of semantic classes. The fallback path is a systems reliability feature.

B Appendix B: Selected Code Excerpt

```
def main() -> None:
    _set_seed()
    _ensure_dirs()
    generate_tabular_figures()
    generate_vision_figures()
    print("[done] report figures generated in", FIG_DIR)

if __name__ == "__main__":
    main()
```

Listing 2: Core structure of the report-plot generation script

C Appendix C: Artifact File List

- `report/figures/class_distribution.png`
- `report/figures/lime_shap_compare_sample_0.png`
- `report/figures/lime_shap_compare_sample_1.png`
- `report/figures/lime_shap_compare_sample_2.png`
- `report/figures/nam_feature_functions.png`
- `report/figures/gradcam_demo.png`

- `report/figures/guided_gradcam_example.png`
- `report/figures/smoothgrad_guided_comparison.png`

References

Rishabh Agarwal, Nicholas Frosst, Xuezhou Zhang, Rich Caruana, and Geoffrey E. Hinton. Neural additive models: Interpretable machine learning with neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ẅhy should i trust you?̈: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smooth-grad: Removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.