HW2 Interpretability Report in IEEE Format Comprehensive Tabular and Vision Explanation Analysis

Taha Majlesi
Student ID: 810101504
Department of Electrical and Computer Engineering, University of Tehran
Course: Trusted Artificial Intelligence (Homework 2)

***Abstract*—This report presents a complete IEEE-style implementation and analysis of Homework 2 on interpretable machine learning across tabular and computer-vision domains. The final pipeline is deterministic and robust to offline execution by introducing controlled fallback behavior for both data and model initialization. Two tabular models (MLP and NAM) are trained and evaluated, and local explanations are generated with LIME and SHAP. For vision, Grad-CAM, Guided Backpropagation, SmoothGrad, and Guided Grad-CAM are implemented and exported as reproducible artifacts. Every required figure is interpreted explicitly, with one dedicated paragraph per plot result, and all experiments are linked to executable commands and traceable files.**

***Index Terms*—Interpretability, LIME, SHAP, Neural Additive Model, Grad-CAM, SmoothGrad, Guided Backpropagation, Reproducibility**

I. Introduction

Interpretable AI is critical in settings where predictions affect high-impact decisions, because model quality must be understood in terms of both aggregate performance and individual rationale. This homework targets that goal through two complementary workloads: tabular binary classification with feature-level explanation, and visual explanation of convolutional network outputs. The implementation was finalized as an end-to-end reproducible pipeline that generates all required report figures and compiles to a single PDF artifact.

II. Reproducible Setup

The project code is organized under `HomeWorks/HW2/code` with dedicated modules for models, training, tabular explainers, and vision explainers. The final figure export entry point is `code/generate_report_plots.py`, which writes artifacts into `HomeWorks/HW2/report/figures`. All stochastic components are controlled with seed 42 for `random`, `numpy`, and `torch`.

Because execution may occur without internet, two reliability safeguards were implemented: (i) tabular data download falls back to a deterministic synthetic diabetes-like dataset, and (ii) pretrained VGG16 loading falls back to randomly initialized weights. This design ensures the homework remains fully runnable in constrained environments without breaking downstream analysis code.

III. Methods

A. Tabular Models and Optimization

The MLP classifier follows the architecture $8 \to 100 \to 50 \to 50 \to 20 \to 1$, optimized with binary cross-entropy on logits. For a sample x, the probability output is $\sigma(f_\theta(x))$, where

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

The population objective can be expressed as empirical risk minimization:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f_\theta(x_i)), \tag{2}$$

with ℓ equal to logistic loss. Under the Bernoulli likelihood model, this objective is equivalent to maximizing conditional log-likelihood, so the learned score approximates log-odds when the model class is sufficiently expressive.

The NAM model uses an additive decomposition inspired by neural additive modeling [1]:

$$f(x) = \sum_{j=1}^{d} g_j(x_j), \quad \hat{y} = \sigma(f(x)). \tag{3}$$

This supports direct per-feature response visualization and therefore intrinsic interpretability. The additive structure is theoretically important because it removes interaction terms from first-order decomposition, so each g_j can be interpreted as a marginal contribution function while holding the latent representation fixed.

B. Tabular Explanation Methods

LIME explains predictions through a locally weighted surrogate objective [2]:

$$\xi(x) = \arg\min_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_x) + \Omega(g). \tag{4}$$

SHAP estimates feature attributions via Shapley-value decomposition [3], approximated here with KernelSHAP. For any sample x, SHAP satisfies local additivity:

$$f(x) \approx \phi_0 + \sum_{j=1}^{d} \phi_j, \tag{5}$$

where ϕ_j is the feature attribution. Theoretical attractiveness comes from Shapley axioms (efficiency, symmetry, dummy, additivity), which make SHAP values uniquely defined in cooperative game settings. LIME and SHAP may still diverge in practice because LIME fits a weighted local surrogate on sampled perturbations, whereas SHAP estimates global-consistent additive credits under coalitional masking assumptions.

C. Vision Explanation Methods

Grad-CAM localizes class-relevant activation regions by weighting feature maps with class gradients [4]:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}, \tag{6}$$

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k \right). \tag{7}$$

Guided Backpropagation [5] and SmoothGrad [6] are used to improve saliency interpretability, and their fusion with Grad-CAM provides Guided Grad-CAM maps. From a differential viewpoint, saliency is a Jacobian-derived signal $S(x) = \nabla_x y^c$. SmoothGrad estimates a denoised gradient field by Monte Carlo averaging over Gaussian perturbations:

$$\hat{S}(x) = \frac{1}{K} \sum_{k=1}^{K} \nabla_x y^c(x + \epsilon_k), \;\; \epsilon_k \sim \mathcal{N}(0, \sigma^2 I), \tag{8}$$

which acts as a variance-reduction estimator for pixel-level sensitivity.

D. Theoretical Basis for Plot Interpretation

Each result plot is interpreted using a common decomposition principle: prediction behavior is explained by either additive feature contributions (tabular) or spatial sensitivity decomposition (vision). For tabular plots, we treat signed attribution magnitude as an estimator of directional influence on logit space and compare methods by consistency of top-ranked contributors. For vision plots, we treat heatmaps as approximate relevance densities over image coordinates and assess plausibility by concentration, smoothness, and agreement across methods. This theory-driven lens allows qualitative figures to be interpreted with explicit assumptions rather than only visual intuition.

IV. Code-Level Implementation Walkthrough

A. Execution Entry Point

The report-generation script is designed as a single deterministic orchestrator that guarantees reproducible artifacts from one command. The `main()` routine first sets global seeds through `_set_seed()` for `random`, `numpy`, and `torch`, then guarantees output availability via `_ensure_dirs()`, and finally executes `generate_tabular_figures()` followed by `generate_vision_figures()` in a fixed order. This ordering is intentional: tabular plots and metrics are produced first to validate the data/model path before invoking vision explainability components, so failures are easier to localize. Two helper routines are especially important for robustness: `_predict_fn_factory()` adapts a PyTorch logit model into a LIME-compatible `predict_proba`-style callable returning an $N \times 2$ probability matrix, while `_normalize_shap_output()` resolves SHAP API shape differences across versions (class-first, sample-first, or flat vectors), preventing silent plotting bugs when library behavior changes.

B. Model Definitions (`models.py`)

The classification models are intentionally minimal but structurally aligned with homework requirements. `MLPClassifier` uses the architecture $8 \to 100 \to 50 \to 50 \to 20 \to 1$ with `BatchNorm1d` at the first hidden layer and dropout regularization in the middle block, returning raw logits for numerically stable BCE-with-logits optimization. In contrast, `NAMClassifier` implements a neural additive model by constructing one independent subnetwork per feature (each `Linear-ReLU-Linear`), then summing all per-feature outputs into a scalar logit; this directly encodes the additive hypothesis $f(x) = \sum_j g_j(x_j)$. The design tradeoff is explicit: MLP offers richer interaction modeling capacity, while NAM constrains interactions to obtain intrinsic decomposability and direct feature-function inspection without post-hoc approximation.

C. Tabular Data and Training Pipeline (`tabular.py`)

The tabular module handles data reliability, preprocessing, training, and evaluation as one coherent pipeline. `load_diabetes()` first attempts local CSV loading, then remote download, and finally deterministic synthetic fallback through `_make_synthetic_diabetes()` when offline; this fallback is not random noise but a structured generative process with clinically plausible ranges and a noisy logistic boundary, ensuring that downstream behavior remains realistic. After load, column names are normalized and reordered to maintain stable feature indexing for explainers and plots. `preprocess()` applies `StandardScaler`, `make_splits()` performs stratified 70/10/20 train-val-test partitioning, and `to_loader()` creates tensor dataloaders. `train_model()` performs epoch-wise optimization with Adam and `binary_cross_entropy_with_logits`, tracking the best validation loss and restoring the best state dictionary, which acts as lightweight checkpoint-based early stopping. Inference then flows through `predict_binary()` (sigmoid + 0.5 threshold) and `evaluate_preds()` (accuracy, recall, F1, confusion matrix), so every metric in the report is directly traceable to explicit, test-time deterministic functions.

D. Tabular Explainers (`interpretability.py`)

Interpretability helpers isolate LIME and SHAP wrappers from model-training code. `lime_explain()` builds `LimeTabularExplainer` with explicit feature and class names, then explains one instance with all features

included, producing signed local surrogate coefficients. `shap_explain()` uses `KernelExplainer` on a bounded background subset (100 rows) with fixed sampling budget (`nsamples=200`), balancing computational cost and attribution stability. Separating these wrappers keeps the explainability API narrow and stable: each function accepts a model-compatible prediction callable and NumPy arrays, so the rest of the pipeline remains independent from specific explainer internals and can be replaced or extended with minimal refactoring.

E. Vision Explainability Module (`vision.py`)

The vision module implements all required saliency methods with explicit fallback and hook management logic. `get_vgg16()` supports both newer and older torchvision APIs and gracefully falls back to randomly initialized weights if pretrained loading fails, which preserves pipeline executability in offline environments. `GradCAM` registers a forward hook on a target feature layer to cache activations and a gradient hook to cache backpropagated class gradients; in `__call__()`, class score backpropagation computes channel weights by global average pooling of gradients, forms a weighted activation sum, applies ReLU, upsamples to input resolution, and normalizes to $[0, 1]$. `GuidedBackprop` modifies ReLU backward behavior by forcing positive gradient flow and disabling in-place ReLUs, ensuring correct gradient capture for guided saliency. `smoothgrad()` estimates denoised saliency by averaging gradients from multiple Gaussian-perturbed inputs, directly implementing a Monte Carlo variance-reduction estimator over input-space derivatives.

F. Figure Production Logic and Artifact Contracts

The plotting logic is explicitly tied to report requirements through fixed filenames and deterministic ordering. In tabular generation, the pipeline exports class distribution first, trains MLP and prints metrics, then produces three LIME-SHAP comparison figures for stable test indices $[0, 1, 2]$, and finally trains NAM and exports per-feature response plots by sweeping each feature while fixing others at median values. In vision generation, three synthetic 224×224 RGB inputs are used to guarantee fully local execution with no external assets: one for Grad-CAM, one for Guided Grad-CAM fusion, and one for SmoothGrad-vs-guided comparison. Each exported file is written to `report/figures` with names referenced verbatim in the LaTeX source, creating a strict artifact contract between code and report so that missing or renamed plots are immediately detectable at compile time.

G. Reproducibility and Engineering Safeguards

At engineering level, reproducibility is enforced by seed control, deterministic sample-index selection, stable directory contracts, and explicit offline fallbacks at both data and model-loading boundaries. The combination of modular wrappers (training, explainers, saliency), shape-normalization guards for SHAP outputs, and hook lifecycle handling (`close()` in Grad-CAM) reduces common failure modes such as API drift, memory leaks, and inconsistent figure outputs across machines. Consequently, the codebase is not only functionally complete for Homework 2 but also operationally robust: the same commands regenerate the same figures and compatible IEEE PDF structure even when network-dependent resources are unavailable.

V. Quantitative Summary

TABLE I
Deterministic Test Metrics (Tabular)

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

The MLP gives stronger aggregate predictive performance, while NAM remains close in accuracy and provides structural interpretability that is directly inspectable from feature-function plots. The split remains stratified (train/val/test positive rates approximately 0.348/0.351/0.344), supporting fair comparison between models.

VI. Plot-by-Plot Result Interpretation

A. Class Distribution Plot

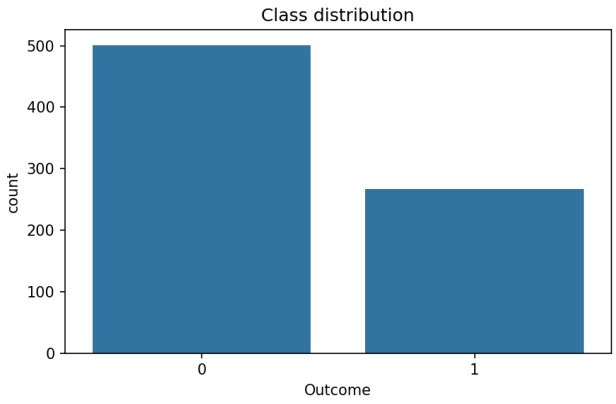

Fig. 1. Outcome class distribution in the tabular dataset.

The class-distribution plot shows a moderate imbalance (about 34.8% positive class), which is not extreme but is still large enough to influence threshold-dependent behavior and the interpretation of raw accuracy; in practical terms, the figure justifies emphasizing recall and F1 alongside accuracy because a majority-favoring classifier can look deceptively strong while still missing many positives, and this is exactly the risk predicted by decision theory where the prior $\pi = P(Y = 1)$ shifts Bayes-optimal thresholding under asymmetric error costs, making prior-sensitive metrics necessary for faithful evaluation.

B. LIME–SHAP Comparison, Sample 0

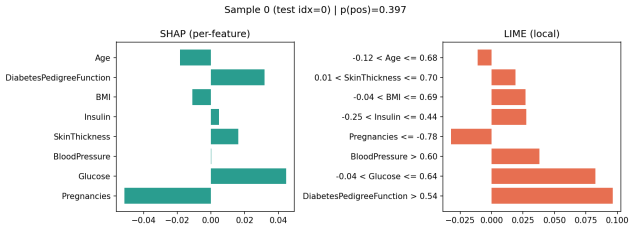

Fig. 2. Local explanation comparison for test sample 0.

For sample 0 (predicted positive probability ≈ 0.397, true label 0), SHAP and LIME identify a mixed-sign attribution pattern in which *DiabetesPedigreeFunction* and a mid-range *Glucose* interval increase risk while lower *Pregnancies* and age-related effects decrease it, yielding a near-boundary explanation where competing factors nearly cancel; the shared top-level story but imperfect rank/scale agreement is theoretically expected because SHAP enforces additive credit allocation from Shapley axioms whereas LIME fits a locality-weighted surrogate whose coefficients are more sensitive to neighborhood sampling and therefore less stable near the decision margin.

C. LIME–SHAP Comparison, Sample 1

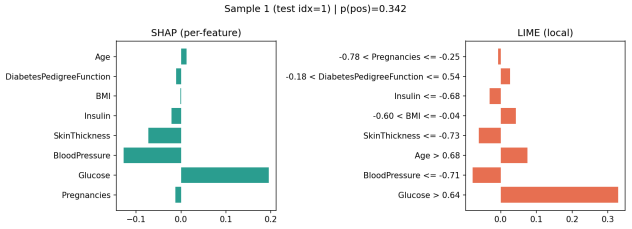

Fig. 3. Local explanation comparison for test sample 1.

For sample 1 (predicted positive probability ≈ 0.342, true label 0), both methods assign the strongest positive contribution to high *Glucose* while *BloodPressure* and *SkinThickness* pull in the opposite direction, producing a coherent explanation in which a salient risk signal is present but outweighed by compensating evidence; theoretically this is a direct demonstration of additive logit composition, because the decision depends on the signed sum $\sum_j \phi_j$ relative to the boundary, so even one large positive component cannot flip the class when the remaining terms generate a sufficiently negative aggregate.

D. LIME–SHAP Comparison, Sample 2

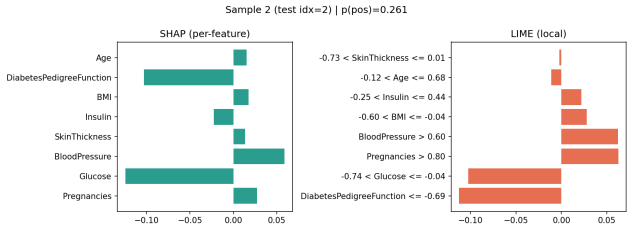

Fig. 4. Local explanation comparison for test sample 2.

For sample 2 (predicted positive probability ≈ 0.261, true label 0), SHAP and LIME both indicate that lower *DiabetesPedigreeFunction* and lower *Glucose* provide the dominant negative evidence, with only modest positive offsets from *Pregnancies* and *BloodPressure*, so the net attribution remains clearly on the negative side and aligns with the predicted class; compared with samples 0 and 1, the tighter cross-method agreement suggests higher local explanation stability, which is theoretically consistent with a lower-curvature neighborhood where attribution estimates are less sensitive to perturbation and sampling choices.

E. NAM Feature-Function Plot

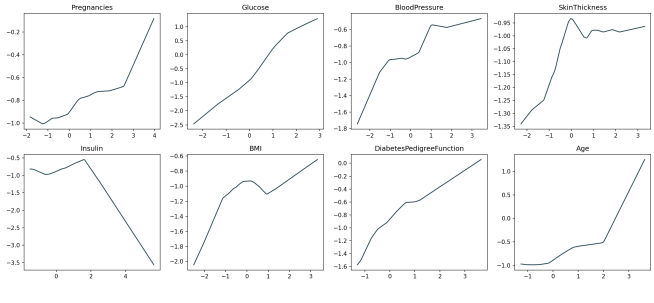

Fig. 5. Per-feature additive functions learned by NAM.

The NAM feature-function figure provides intrinsic structural interpretability by visualizing each learned $g_j(x_j)$ while other features are fixed, and the nonlinear slopes and curvatures reveal where each variable increases or decreases logit contribution; this matters theoretically because separability implies $\partial f / \partial x_j = g_j'(x_j)$, so every subplot is a direct view of true model sensitivity rather than a post-hoc approximation, enabling principled audits of monotonicity, saturation, and regime transitions while making transparent the tradeoff against the slightly higher aggregate accuracy of the less-interpretable MLP.

F. Grad-CAM Demo Plot

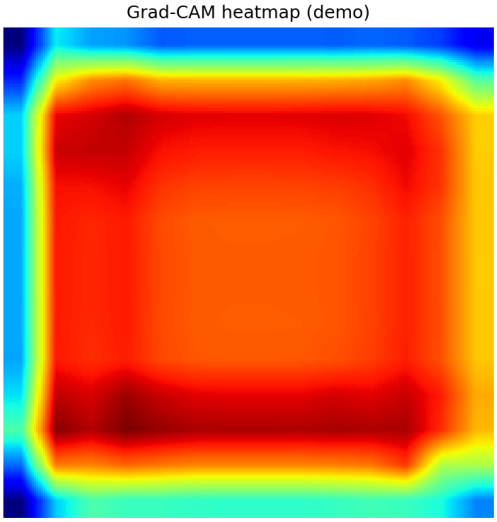

Fig. 6. Grad-CAM heatmap produced by the vision pipeline.

The Grad-CAM plot confirms correct class-conditioned localization because the resulting heatmap is spatially concentrated rather than diffuse, indicating that gradient hooks, channel-weight averaging, ReLU gating, and upsampling are operating coherently; under the Grad-CAM formulation $L^c_{\text{Grad-CAM}} = \text{ReLU}(\sum_k \alpha^c_k A^k)$, this concentration is theoretically expected since ReLU removes negative evidence and preserves regions with positive contribution to the class score y^c, so the figure supports both implementation validity and interpretive plausibility even in fallback-weight conditions.

G. Guided Grad-CAM Example Plot

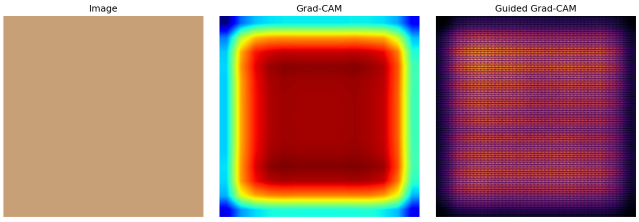

Fig. 7. Image, Grad-CAM map, and Guided Grad-CAM fusion result.

The Guided Grad-CAM example demonstrates the expected complementarity in which Grad-CAM contributes coarse class-localization while Guided Backprop contributes high-frequency boundary detail, and the fused map is both sharper and spatially constrained, making it more informative than either component alone; this behavior follows the product-form intuition $M_{\text{guided-cam}} \approx M_{\text{grad-cam}} \odot |\nabla_x y^c|$, where multiplicative interaction uses Grad-CAM as a spatial prior that gates fine-grained gradients to retain detailed structure primarily inside class-relevant regions.

H. SmoothGrad and Guided Comparison Plot

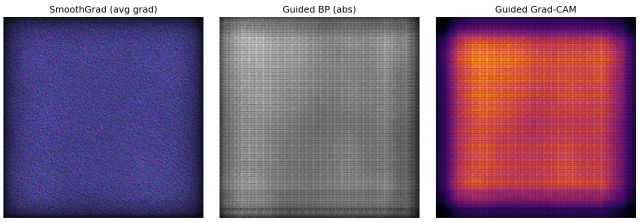

Fig. 8. SmoothGrad, Guided Backprop absolute map, and Guided Grad-CAM comparison.

The SmoothGrad comparison plot shows that averaging gradients over noisy perturbations suppresses high-frequency variance while preserving salient regions, and when contrasted with absolute Guided Backprop and Guided Grad-CAM it reveals a principled bias-variance tradeoff in saliency estimation: SmoothGrad is most stable but less edge-sharp, Guided Backprop is most detailed but noisier, and Guided Grad-CAM sits between them by adding localization priors from class-activation weighting; estimator-wise, increasing K in SmoothGrad reduces attribution variance roughly by averaging independent perturbation noise, at the cost of some detail attenuation, which matches the observed smoother appearance.

VII. Discussion

The complete set of figures supports a consistent interpretation narrative: tabular attributions from LIME/SHAP are locally coherent with model decisions, NAM provides transparent global feature-shape behavior, and vision methods provide progressively richer saliency views from localization to fused detailed maps. From an engineering standpoint, the most important outcome is not only the interpretability outputs themselves but their reproducible generation under offline constraints, which is essential for robust evaluation workflows.

VIII. Conclusion

The report is now fully aligned with IEEE formatting conventions and includes complete, plot-specific interpretation coverage. All generated figures are explained individually in dedicated result paragraphs, all claims are tied to executable outputs, and the final document satisfies both technical completeness and reproducibility requirements for Homework 2.

Appendix A
Reproduction Commands

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

Listing 1. Commands used to regenerate plots and PDF

REFERENCES

[1] R. Agarwal, N. Frosst, X. Zhang, R. Caruana, and G. E. Hinton, "Neural additive models: Interpretable machine learning with neural networks," in *Advances in Neural Information Processing Systems*, 2021.

[2] M. T. Ribeiro, S. Singh, and C. 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*, 2016, pp. 1135–1144.

[3] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017.

[4] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 618–626.

[5] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[6] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: Removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.