Phase-wise Development and Optimization of a Non-Neural Specialist Architecture for MNIST Classification

ML ENDSEM PROJECT REPORT

Abstract

This project explores the development of a high-performing non-neural architecture for the MNIST handwritten digit classification task. Through a three-phase design process—rapid prototyping with `scikit-learn`, complete self-implementation in Python, and final time optimization—we demonstrate that traditional models, when systematically engineered, can achieve competitive accuracy within strict runtime constraints.

1 Phase 1: Architectural Exploration using Scikit-Learn

1.1 Objective

I used the models from scikit learn to test out my ideas and to build my architecture. This allowed me to explore different ideas wihtout the compromise of time. This was also in hope of setting a benchmark of what can be achieved with raw python code.

The first phase focused on rapid prototyping and experimentation to discover an optimal non-neural architecture capable of rivaling neural baselines. Using `scikit-learn`, I iteratively identified the strengths and limitations of classical algorithms under various feature representations.

1.2 Identified Challenges

- **Lack of spatial awareness:** Pixel independence led to frequent misclassifications between similar digits such as (3, 5) and (7, 9).
- **Feature redundancy:** The raw 784-dimensional pixel vectors contained high correlations, leading to inefficient learning.
- **Uncalibrated probabilities:** Classifiers like Random Forests and XGBoost provided poorly scaled confidence values.
- **Digit-specific ambiguity:** Certain digits (3, 7, 9) consistently exhibited lower F1-scores.

1.3 Proposed Architecture

We developed a hierarchical ensemble combining both feature and model-level integration:

1. **Feature Compression:** PCA retaining 95% variance reduced input dimensionality to ∼150 features.
2. **Base Learners:** Logistic Regression, KNN ($k = 5$), and Random Forest (300 trees).
3. **Meta-Learner:** XGBoost as the final estimator in a StackingClassifier.
4. **Specialists:** Class-specific one-vs-rest and pairwise Logistic Regression classifiers for digits 3, 7, 9, (3,5), and (7,9).

1.4 Feature Evolution

Through experimentation, the following features were introduced incrementally to battle the inability of spacial awareness of classical ML, by using some heuristics:

- **PCA:** Dimensionality reduction capturing global variance.
- **HOG:** Captures local orientation histograms; +1.2% F1 improvement.
- **Directional Energy:** Four-bin gradient magnitude features; +0.4% F1.
- **Zonal Features:** 4×4 grid intensity sums + row/column stats; +0.6% F1.
- **Skeleton Features:** Topological structure points (endpoints, holes, curvature); +0.3% F1.

1.5 Phase 1 Results

Model	Description	Accuracy (%)	Weighted F1
XGBoost only	PCA + softmax	96.1	0.961
Stacking (RF+KNN+LR→XGB)	PCA only	96.16	0.9616
+ Skeleton Features	PCA + Skeleton	95.16	0.951
+ HOG Features	PCA + HOG	97.24	0.972
+ Zonal Features	PCA + HOG + Zonal	97.56	0.9756
+ Directional + Specialists	Full	**97.40**	**0.9740**

Table 1: Phase 1 experimental results.

The architecture stabilized at 97.4–97.6% weighted F1, forming the foundation for later self-implemented phases.

2 Phase 2: Self-Implemented Architecture

2.1 Objective

Phase 2 involved replacing all `scikit-learn` modules with self-written Python implementations while preserving the logic and architecture from Phase 1.

2.2 Implemented Models

- **LogisticRegressionCustom:** Softmax gradient descent with L2 regularization.
- **KNNCustom:** Vectorized distance computation using Euclidean similarity.
- **RandomForestFast:** Bootstrap aggregation with Gini impurity and randomized splits.
- **GradientBoostingFast:** Residual fitting with shallow trees as weak learners.
- **SigmoidCalibrator:** Platt scaling on meta-model probabilities.
- **StackingCV:** Custom cross-validation stacking using out-of-fold predictions.

2.3 Performance Comparison

Phase	Accuracy (%)	Weighted F1	Runtime (s)
Phase 1 (scikit-learn)	97.4	0.974	200
Phase 2 (Python)	94.8	0.948	720

Table 2: Performance comparison between library-based and self-coded implementations.

The 2.6% accuracy drop was expected due to the lack of C-level optimizations in Python and the heavier loop-based tree traversals.

3 Phase 3: Runtime Optimization under 5 Minutes

3.1 Objective

To reduce the total training time to under 300 seconds (5 minutes) while maintaining a minimum of 94.5% weighted F1. No parallelization or pre-trained parameter caching was permitted.

3.2 Optimization Techniques

Type	Technique	Description
Caching	Precomputed PCA + HOG + Directional + Zonal	Skipped redundant trans
Memory Layout	Contiguous NumPy arrays	Removed implicit copies
Normalization Reuse	Cached normalized matrices for KNN	Avoided recomputation e
Calibration	Single-step global calibration	Prevented redundant per

Table 3: Optimizations introduced in Phase 3.

3.3 Phase 3 Results

Model Version	Accuracy (%)	Weighted F1	Training Time (s)
Scikit-learn Reference	97.4	0.974	200
Pure Python (Phase 2)	94.8	0.948	720
Optimized Python (Phase 3)	**94.6**	**0.945**	**277**

Table 4: Final comparative results across all phases.

3.4 Hybrid Evaluation

To validate that performance was limited by compute rather than architecture, a hybrid test was conducted where the optimized self-implemented features were combined with scikit-learn estimators. The results confirmed the architecture's full potential under efficient computation:

Accuracy: 97.39%, Weighted F1: 0.9739, Runtime: 200.75 s

This established that the slight performance loss in Phase 3 was purely computational and not due to architectural degradation.

3.5 Digit-wise Report (Phase 3)

Digit	Precision	Recall	F1-Score
0	0.960	0.972	0.966
1	0.962	0.982	0.972
2	0.947	0.944	0.946
3	0.938	0.941	0.939
4	0.944	0.971	0.957
5	0.955	0.929	0.942
6	0.976	0.976	0.976
7	0.946	0.935	0.940
8	0.896	0.885	0.891
9	0.930	0.915	0.923

Table 5: Digit-wise evaluation metrics for Phase 3.

4 Conclusion

The developed Stacked Specialist Architecture achieves a competitive **94.5% weighted F1** within a strict **5-minute training limit**, without any neural networks or parallelization. The progression from Phase 1 to Phase 3 demonstrates that:

- Systematic feature engineering (PCA + HOG + Directional + Zonal) provides substantial representational power.
- Specialist correction layers effectively resolve class-specific confusion.

- Compute-efficient caching and memory optimizations bridge most of the performance gap between Python and C-accelerated implementations.

Final Outcome:

> "Training completed in 277 seconds with 94.56% accuracy and 0.945 weighted F1. The architecture retains interpretability, modularity, and computational fairness."