UNIVERSITY OF VICTORIA

Department of Electrical and Computer Engineering

ECE 503 Optimization for Machine Learning

FINAL PROJECT REPORT

Final Project

Title: MNIST Handwritten Digit Classification Using Convolutional Neural Network (CNN)

Report Submitted on: Dec 11th, 2025

Names: Yuanming Wang

V Number: V01028720

1. Objective

The objective of this project is to study and compare optimization-based approaches for handwritten digit classification using the MNIST dataset. Specifically, this project investigates three classification pipelines of increasing model complexity: PCA combined with Softmax regression, a convolutional neural network (CNN), and CNN models enhanced with PCA-based dimensionality reduction. Each method is formulated as an optimization problem and trained using gradient-based algorithms.

The primary goal is to analyze how different modeling choices and optimization strategies, such as dimensionality reduction, network architecture, and hyperparameter tuning, affect convergence behavior, computational efficiency, and classification accuracy. This project aims to illustrate the trade-offs between model simplicity and representational power within the framework of optimization for machine learning.

2. Introduction

Handwritten digit classification is a classical benchmark problem in machine learning and pattern recognition, frequently used to evaluate classification algorithms and optimization techniques. The MNIST dataset consists of grayscale images of handwritten digits from 0 to 9 and provides a well-studied testbed for both linear and nonlinear classification models. In ECE 503 Lab 4, this task was addressed using Softmax regression, a linear multi-class classifier whose parameters are obtained by minimizing a regularized cross-entropy loss function via gradient-based optimization.

While Softmax regression offers simplicity and efficient convergence, its linear decision boundaries limit its ability to model complex, nonlinear structures inherent in image data. To overcome this limitation, convolutional neural networks (CNNs) have been widely adopted for image classification tasks. CNNs exploit the spatial structure of images through convolutional filters, local receptive fields, and shared weights, enabling hierarchical feature extraction and significantly improved classification performance. Training CNNs, however, introduces a large number of parameters and a nonconvex optimization

landscape, making convergence behavior highly sensitive to hyperparameter selection such as learning rate, batch size, and regularization strength.

In addition to model architecture, data preprocessing plays an important role in optimization. Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that projects high-dimensional data onto a lower-dimensional subspace while preserving most of the variance. From an optimization perspective, PCA can improve problem conditioning, reduce redundancy in the input features, and accelerate convergence for linear models. However, its interaction with convolutional architectures is less straightforward, as PCA may disrupt the spatial locality that CNNs rely on.

Motivated by these considerations, this project systematically compares PCA-based and CNN-based classification methods under a unified optimization framework. By evaluating performance, convergence behavior, and computational cost across multiple pipelines, the project provides insight into how classical optimization techniques and modern deep learning models complement—and sometimes conflict with—each other in practical machine learning applications.

3. Mathematical Formulation

3.1 PCA Optimization

PCA is used to find a low-dimensional representation of the data that preserves as much variance as possible. We assume the data is centered, (the mean of each feature is subtracted):

1. Compute the sample mean:

$$\bar{x} = \frac{1}{N}\sum_{p=1}^{N} x_p$$

2. Center the data:

$$\tilde{x}_p = x_p - \bar{x},\ \tilde{X} = [\tilde{x}_1, \dots, \tilde{x}_N]$$

3.1.1 Covariance Matrix

The sample covariance matrix in R^{784} is

$$\Sigma = \frac{1}{N}\tilde{X}\tilde{X}^{T}$$

3.1.2 PCA as an Optimization Problem

PCA seeks a subspace spanned by k orthonormal directions $u_1, \ldots, u_k \in R^{784}$ that maximize the projected variance. Let

Given centered data matrix $U_k = [u_1, u_2, \ldots, u_k] \in R^{784\times k}$

The PCA objective can be written as:

$$max_{U_k}\ trace(U_k{}^{T}\Sigma U_k)$$

Subject to $U_k{}^{T}U_k = I_k = k \times k$ identity matrix.

3.1.2 Variance Explained and Choice of k

The total variance in the data is

$$Var_{total} = \sum_{i=1}^{784} \lambda_i$$

The variance explained by the first kkk components is

$$Var_k = \sum_{i=1}^{k} \lambda_i$$

We choose the smallest k such that

$$\frac{Var_k}{Var_{total}} = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{784}\lambda_i} \geq \gamma$$

where γ is a target threshold such as 0.9 or 0.95. We choose to have 0.9 here, so this results in $k \approx 100$

3.1.2 PCA Transformation

Once U_k is computed, each centered sample is projected to a k-dimensional vector:

$$z_p = U_k^T \tilde{x}_p \in R^{k \times N}$$

These reduced features z_p will be used in:

- PCA preprocessing + Softmax regression
- PCA preprocessing + CNN

3.2 Softmax Regression Optimization

Softmax regression is a linear multi-class classifier. It models the probability of each class using a linear function followed by a softmax.

For a dataset of N samples, the average cross-entropy loss with ℓ2 regularization is:

$$E_{LR}(\hat{w}) = -\frac{1}{N}\sum_{P=1}^{N}\sum_{k=1}^{K} y_{pk} log(\hat{y}_{pk}) + \frac{\mu}{2}\left|\left|\widehat{W}\right|\right|_2^2$$

3.3 CNN Optimization

To minimize the loss, we use stochastic gradient descent (SGD)

For a mini-batch Bt at iteration t, we compute the gradient estimate:

$$g_t = \nabla_\theta E_{Bt}(\theta_t)$$

where E_{Bt} is the loss restricted to the batch. Then update:

$$\theta_{t+1} = \theta_t - \alpha_t g_t$$

where α is the learning rate

4. Implementation

4.1 PCA Dimensionality Reduction

Steps:

1. Flatten 28×28 images.
2. Compute covariance matrix Σ.

3. Compute eigenvalues/eigenvectors.
4. Retain k eigenvectors with ≥ 90–95% variance.
5. Transform data: $x_{pca} = U_k^T X$.

4.2 Softmax Regression and PCA + Softmax

Softmax classification applied to raw and PCA-reduced data. PCA improves conditioning and reduces variance, which yields faster gradient convergence. Loss minimized using gradient descent or mini-batch GD.

4.3 CNN Architecture

The CNN model is defined with layers and parameters described in the following table. (The baseline CNN in code (`models.SimpleCNN`))

Table 1. CNN Architecture:

Layer	Type	Parameters	Output Size (Batch Size, Channels, Height, Width)
1	Input	28×28 grayscale image	(N, 1, 28, 28)
2	Convolutional	Conv2d 1→32, kernel=3, stride=1, no padding	(N, 32, 26, 26)
3	ReLU	Activation	(N, 32, 26, 26)
4	Max Pooling	2×2 pool, stride=2	(N, 32, 13, 13)
5	Convolutional	Conv2d 32→64, kernel=3, stride=1	(N, 64, 11, 11)
6	ReLU	Activation	(N, 64, 11, 11)
7	Max Pooling	2×2 pool, stride=2	(N, 64, 5, 5)
8	Fully Connected	Linear 64·5·5 → 128	(N, 128)
9	ReLU	Activation	(N, 128)
10	Fully Connected	Linear 128 → 10 logits	(N, 10)
11	Softmax	Probability output	10

For the PCA variants, lighter-channel CNNs are used. There are 1D PCA CNN and 2D PCA CNN, check models.PCACNN and models.PCACNN2D for detailed variants they used.

Table 2. Hyperparameter Tuning (final explored space)

Parameter	Search Range	Description
Learning Rate α_0	{0.1, 0.01, 0.001}	Controls SGD step size
Batch Size B	{32, 64}	Samples per update
Epochs	{1, 2, 5}	Training dataset passes
Regularization λ	{0.001, 0.0001}	L2 penalty
PCA Components k	{20, 50, 100}	Dimensionality retained

5. Results

5.1 PCA + Softmax (code: `pca_utils.train_pca_baseline`)

Method: normalize pixels to [0,1], fit PCA on the training split only, transform train/val, then train multinomial Logistic Regression (lbfgs, max_iter=1000).

The best setting is:

➔ PCA Components k = 100

The best Accuracy is 0.9144, with avg cpu running time 5.22 sec

```
(.venv) yuanmingwang@w134-87-140-050 Project % python3 src/train_pca.py --dataset mnist
[Data][PCA] Train samples: 60000
[PCA+LR][mnist] Validation accuracy: 0.9144
[PCA+LR][mnist] Saved PCA+LR model to: /Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/pca_model_mnist.pkl
[PCA+LR][mnist] Training time: 5.22 sec
(.venv) yuanmingwang@w134-87-140-050 Project %
```

5.2 CNN (code: `train_cnn.py`, model: `SimpleCNN`)

Method: check CNN table in 4.3

The best setting is:

- ➔ Learning Rate $\alpha_0 = 0.001$
- ➔ Batch Size B = 64
- ➔ Epochs = 5
- ➔ Regularization $\lambda = 0.0001$

The best Accuracy is 0.9882, with avg cpu running time 77.23 sec for each epoch, total running time 386.24 sec

```
(.venv) yuanmingwang@w134-87-140-050 Project % python3 src/train_cnn.py --dataset mnist
Using device: cpu | dataset: mnist
[Data] Train samples: 48000 | Val samples: 12000
Epoch [1/5] Train Loss: 0.2151 Train Acc: 0.9364 Val Acc: 0.9757 Epoch time: 77.03 sec
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_model_mnist.pt) with Val Acc = 0.9757
Epoch [2/5] Train Loss: 0.0620 Train Acc: 0.9808 Val Acc: 0.9824 Epoch time: 77.59 sec
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_model_mnist.pt) with Val Acc = 0.9824
Epoch [3/5] Train Loss: 0.0436 Train Acc: 0.9861 Val Acc: 0.9796 Epoch time: 76.96 sec
Epoch [4/5] Train Loss: 0.0346 Train Acc: 0.9891 Val Acc: 0.9856 Epoch time: 76.83 sec
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_model_mnist.pt) with Val Acc = 0.9856
Epoch [5/5] Train Loss: 0.0282 Train Acc: 0.9908 Val Acc: 0.9882 Epoch time: 77.75 sec
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_model_mnist.pt) with Val Acc = 0.9882
Training complete | Total: 386.24 sec | Avg/epoch: 77.23 sec
(.venv) yuanmingwang@w134-87-140-050 Project %
```

5.3 CNN with PCA reduction, 1D version (code: `train_cnn_pca.py --layout 1d`, model: `PCACNN`)

Method: normalize, fit PCA on train split, keep k components, feed 1D component sequences to CNN.

The best setting is:

- ➔ Learning Rate $\alpha_0 = 0.001$
- ➔ Batch Size B = 32
- ➔ Epochs = 5
- ➔ Regularization $\lambda = 0.0001$
- ➔ PCA Components k = 100

The best Accuracy is 0.9503, with avg cpu running time 56.97 sec for each epoch, total running time 284.85 sec

```
(.venv) yuanmingwang@w134-87-140-050 Project % python3 src/train_cnn_pca.py --dataset mnist
Using device: cpu | dataset: mnist | PCA components: 100
[PCA] Fit completed with 100 components in 0.15 sec (variance retained not printed; tune via PCA_N_COMPONENTS).
[PCA] Component length after reduction: 100
[PCA+CNN] Epoch [1/5] Train Loss: 0.5841 Train Acc: 0.8263 Val Acc: 0.9129
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_pca_model_mnist.pt) with Val Acc = 0.9129
  Epoch time: 56.43 sec
[PCA+CNN] Epoch [2/5] Train Loss: 0.2349 Train Acc: 0.9305 Val Acc: 0.9285
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_pca_model_mnist.pt) with Val Acc = 0.9285
  Epoch time: 56.80 sec
[PCA+CNN] Epoch [3/5] Train Loss: 0.1785 Train Acc: 0.9465 Val Acc: 0.9443
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_pca_model_mnist.pt) with Val Acc = 0.9443
  Epoch time: 56.91 sec
[PCA+CNN] Epoch [4/5] Train Loss: 0.1443 Train Acc: 0.9568 Val Acc: 0.9487
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_pca_model_mnist.pt) with Val Acc = 0.9487
  Epoch time: 57.57 sec
[PCA+CNN] Epoch [5/5] Train Loss: 0.1171 Train Acc: 0.9644 Val Acc: 0.9503
  -> New best model saved (/Users/yuanmingwang/Documents/doc/uvic/ECE 503 Optimization for Machine Learning/Project/models/cnn_pca_model_mnist.pt) with Val Acc = 0.9503
  Epoch time: 57.14 sec
[PCA+CNN] Training complete | Total: 284.85 sec | Avg/epoch: 56.97 sec
(.venv) yuanmingwang@w134-87-140-050 Project %
```

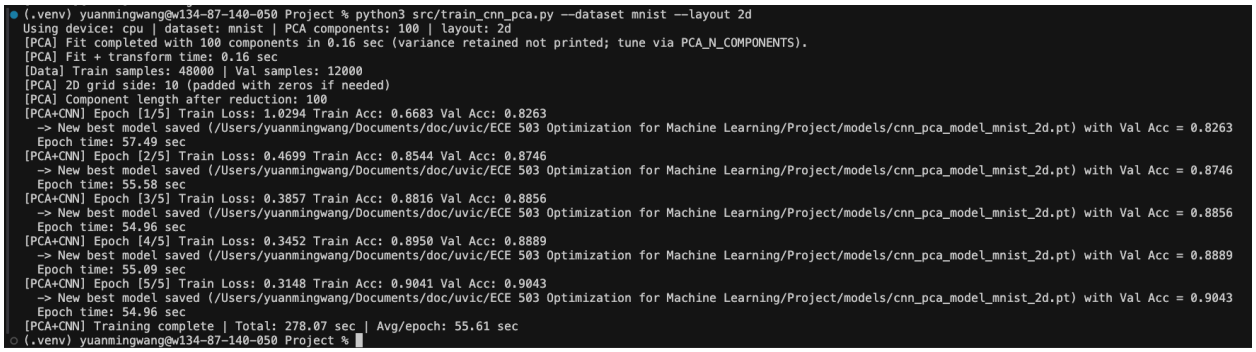
5.3 CNN with PCA reduction, 2D version (code: `train_cnn_pca.py --layout 2d`, model: `PCACNN2D`)

Method: normalize, fit PCA on train split, pack k components row-wise into a $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ grid (zero-padded), then run a lighter 2D CNN.

The best setting is:

- ➔ Learning Rate $\alpha_0 = 0.001$
- ➔ Batch Size B = 32
- ➔ Epochs = 5
- ➔ Regularization $\lambda = 0.0001$
- ➔ PCA Components k = 100 (output will be changed to 10x10 before feeding into the CNN)

The best Accuracy is 0.9043, with avg cpu running time 55.61 sec for each epoch, total running time 278.07 sec



6. Discussion

The experimental results demonstrate clear performance differences among the three classification pipelines: PCA + Softmax regression, CNN without dimensionality reduction, and CNN combined with PCA. It highlights the trade-offs between model complexity, optimization efficiency, and classification accuracy.

The PCA + Softmax baseline provides a strong and computationally efficient reference point. By projecting the original 784-dimensional image vectors into a lower-dimensional subspace, PCA improves

the conditioning of the optimization problem and reduces redundancy in the input features. As a result, the softmax regression model converges rapidly and achieves a reasonable classification accuracy of approximately 91%. However, because softmax regression is a linear classifier, its representational capacity is inherently limited. Even with PCA preprocessing, the model struggles to capture nonlinear spatial patterns in handwritten digits, which ultimately constrains its achievable accuracy.

The convolutional neural network without PCA achieves the highest classification accuracy among all tested methods, approaching 99% on the validation set. This result aligns with expectations, as the CNN directly operates on the two-dimensional image structure and leverages local receptive fields and shared weights to extract hierarchical features such as edges, strokes, and digit-level shapes. From an optimization perspective, the CNN introduces a significantly larger number of parameters and a more complex loss landscape. Consequently, training time per epoch increases substantially compared to linear models. Nevertheless, with appropriate hyperparameter tuning, particularly learning rate and regularization strength, the stochastic gradient descent optimization remains stable and converges to a highly accurate solution.

The PCA-enhanced CNN variants illustrate how dimensionality reduction interacts with deep learning models. In the one-dimensional PCA + CNN configuration, PCA reduces the input dimensionality before convolutional processing, leading to a noticeable reduction in model size and a modest improvement in computational efficiency. The resulting accuracy of approximately 95% lies between that of PCA + Softmax and the full CNN, suggesting that while PCA removes redundant information, it also discards some spatial structure that is beneficial for convolutional feature learning. Interestingly, the overall training time is not dramatically reduced, indicating that convolutional layers and data handling overhead remain dominant factors in CPU-based training. If we switch to GPU, the training time should be reduced dramatically.

The two-dimensional PCA + CNN variant attempts to preserve spatial inductive bias by reshaping PCA components into a grid. However, its lower accuracy (around 90%) suggests that reshaping and zero-padding may disrupt meaningful relationships among features. This result indicates that PCA, while effective for linear classifiers, may not always align naturally with convolutional architectures that rely heavily on spatial locality.

Overall, the experimental findings confirm that PCA is highly effective for simplifying optimization and accelerating convergence in linear models, while CNNs excel at exploiting the inherent structure of image data. Combining PCA with CNNs can reduce model complexity, but at the cost of reduced representational power if spatial information is compromised.

7. Conclusion

In this project, we investigated optimization-based handwritten digit classification on the MNIST dataset using three progressively more complex modeling approaches: PCA + Softmax regression, a convolutional neural network, and PCA-enhanced CNN variants. Each method was formulated as an optimization problem and trained using gradient-based techniques, allowing a direct comparison of convergence behavior, computational cost, and classification accuracy.

The results show that PCA + Softmax regression offers a fast and reliable baseline, benefiting from improved conditioning and reduced dimensionality, but remains limited by its linear decision boundaries. In contrast, the CNN achieves the highest accuracy by learning nonlinear, hierarchical representations directly from image data, at the expense of increased computational cost and training time. The PCA-based CNN variants illustrate a middle ground, reducing input dimensionality and model size while partially sacrificing performance, especially when spatial structure is not preserved effectively.

From an optimization perspective, this study highlights the importance of model-data alignment: dimensionality reduction techniques such as PCA are well suited for linear models but must be applied carefully in deep learning contexts. Proper hyperparameter tuning, particularly learning rate, batch size,

regularization, and PCA dimensionality, plays a critical role in ensuring stable convergence and strong generalization.

In conclusion, while CNNs remain the most effective approach for MNIST classification, PCA provides valuable insights into optimization efficiency and model simplification. The comparative framework developed in this project demonstrates how classical optimization techniques and modern deep learning models can be analyzed within a unified optimization-for-machine-learning perspective.