Leveraging Semi-Supervised learning in a Simulated Unlabeled Environment: Fashion-MNIST

Assignment 1 - COSC 4P96

Chris Ortiz
Department of Computer Science
Brock University
St. Catharines, Ontario
tl22ba@brocku.ca

***Abstract*—This paper investigates the effectiveness of semi-supervised learning in a controlled simulated unlabeled environment using the Fashion-MNIST dataset. A portion of labeled data is intentionally withheld to emulate realistic label scarcity, and neural networks are trained using a hybrid approach that combines supervised learning with pseudo-labeling on high-confidence unlabeled samples. Multiple model configurations are evaluated to analyze training stability, generalization performance, and robustness under varying labeled-to-unlabeled ratios. The results demonstrate that semi-supervised strategies can recover a significant portion of the performance gap between limited-label and fully supervised training. Statistical evaluation confirms that leveraging unlabeled data yields consistent improvements, and is a valuable real world application to unlabeled-data abundant scenarios. We also explore compression tactics and evaluate the computation-accuracy tradeoff.**

I. Introduction

The FashionMNIST dataset is a 10 class dataset with 70,000 rows, an abundance of data. Simple neural networks can converge to learn this dataset fairly well. Some modifications to a basic neural network such as, dataset normalization, augmentation, and overfitting detection, will help a fully supervised model to increase generalization. Other advanced techniques include regularization, to deter weight values from growing too large, and network pruning, which starts a network out with extra parameters, and dismisses the ones with low weights, which we assume are less expressive and thus less important. These methods are documented to improve performance of a fully supervised model. The aim of this study is to discover methods which allow the models to leverage **unlabeled data**. We'll explore graph based label propagation, first introduced by ¿¿¿¿ in ***, and also explore the concept of co-training, where 2 smaller models train on separate views of each image. We analyze their performance against the upper bound of a fully labeled dataset, where much more information is available to be absorbed, compared to facing potential confirmation bias and uncertainty, without true labels. Then, we experiment with PCA, on our best performing models, to see the trade in performance for more efficient training and inference.

II. Stage 1: Vanilla Neural Network Model

The vanilla model is a fully connected feed-forward neural network that serves as our supervised baseline. The architecture consists of an input layer of size 784 (the flattened 28×28 Fashion-MNIST images), two hidden layers each with 1028 units, and an output layer of size 10 corresponding to the Fashion-MNIST classes.

1) Architecture: Each hidden layer applies a linear transformation followed by the Rectified Linear Unit (ReLU) activation function:

$$\text{ReLU}(z) = \max(0, z)$$

ReLU is preferred over sigmoid or tanh because it avoids the vanishing gradient problem in deeper layers—when $z > 0$ the gradient is exactly 1, allowing unimpeded gradient flow during backpropagation. The full forward pass for an input $\mathbf{x} \in \mathbb{R}^{784}$ is:

$$\mathbf{h}_1 = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{W}_1 \in \mathbb{R}^{1028\times 784} \tag{1}$$

$$\mathbf{h}_2 = \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \qquad \mathbf{W}_2 \in \mathbb{R}^{1028\times 1028} \tag{2}$$

$$\mathbf{z} = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \qquad \mathbf{W}_3 \in \mathbb{R}^{10\times 1028} \tag{3}$$

The output logits $\mathbf{z}$ are passed through the softmax function to produce a probability distribution over the 10 classes:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{10} e^{z_j}}, \quad k = 1, \ldots, 10$$

The model is trained by minimizing the cross-entropy loss between the predicted distribution and the one-hot true label, optimized via stochastic gradient descent (SGD) with momentum. The full gradient derivations for cross-entropy with softmax are provided in the Appendix.

A. Weight Initialization Strategies

Weight initialization determines the starting point in the loss landscape and strongly influences convergence speed and final performance. We compare three strategies:

- **He initialization** [2]: Weights are drawn from $\mathcal{N}(0, \sqrt{2/n_{\text{in}}})$, where n_{in} is the fan-in. This is designed specifically for ReLU networks—since ReLU zeros out half of the activations, the variance is scaled by 2 to compensate, maintaining stable gradient magnitudes across layers.
- **Uniform initialization**: Weights are drawn uniformly from $\left[-1/\sqrt{n_{\text{in}}},\, 1/\sqrt{n_{\text{in}}}\right]$. This keeps activations bounded but does not account for ReLU's asymmetry.
- **Normal initialization**: Weights are drawn from $\mathcal{N}(0, 0.01)$. The small fixed variance can cause vanishing gradients in deeper layers, as activations shrink toward zero.

III. Data Standardization

Data standardization is crucial for neural network training as it ensures that all input features are on comparable scales, preventing features with larger magnitudes from dominating the learning process. We implement two standardization methods for comparison.

A. Z-Score Normalization

Z-score normalization (also called standardization) transforms each feature to have zero mean and unit variance. For a feature vector $\mathbf{x}$, the normalized value is computed as:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation of the feature across the dataset:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{5}$$

To avoid division by zero for constant features, we set $\sigma = 1$ when $\sigma = 0$ (never happens in practice). This method is particularly effective when features follow approximately Gaussian distributions and helps gradient descent converge faster.

B. Min-Max Normalization

Min-max normalization scales features to a fixed range $[0, 1]$ by applying:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature across the dataset. When $x_{\max} = x_{\min}$ (constant feature, also never happens), we set the denominator to 1 to avoid division by zero.

This normalization is bounded and preserves the original distribution shape, making it suitable for image data where pixel intensities fall within a range of 0 to 255.

IV. Data Augmentation

Data augmentation is a regularization technique that artificially expands the training dataset by applying transformations to existing samples. This helps prevent overfitting and improves model generalization by exposing the network to variations of the training data. We employ two augmentation strategies: **3×3 neighborhood color jitter**, which adds adaptive Gaussian noise scaled by local pixel variance, and **horizontal flipping**, applied randomly to 10% of samples per epoch. Full algorithmic definitions and visual examples are provided in Appendix D.

V. Overfitting Detection

Overfitting occurs when a model learns training data too well, including noise and irrelevant patterns, leading to poor generalization. We developed a statistical criterion to detect overfitting automatically and enable early stopping.

A. Initial Criterion and Challenges

The initial overfitting detection criterion was based on a simple statistical threshold: a model is considered to be overfitting at epoch t if the validation error exceeds the mean validation error by more than one standard deviation:

$$E_V(t) > \bar{E}_V + \sigma_{E_V}$$

where:

$$E_V(t) = 1 - \text{acc}_V(t) \quad \text{(validation error at epoch } t\text{)} \tag{6}$$

$$\bar{E}_V = \frac{1}{t}\sum_{i=1}^{t} E_V(i) \quad \text{(mean validation error)} \tag{7}$$

$$\sigma_{E_V} = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(E_V(i) - \bar{E}_V)^2} \quad \text{(standard deviation)} \tag{8}$$

However, early experimentation displayed **premature triggering** in the first few epochs. During initial training, validation error naturally fluctuates as the model explores the loss landscape. Even normal training variance could exceed $\bar{E}_V + \sigma_{E_V}$ in the first few epochs, causing false positives that terminated training before the model had converged.

B. Dynamic Threshold with Temporal Scaling

To address this limitation, we introduced an **adaptive threshold** that becomes progressively stricter as training progresses:

$$\text{threshold}(t) = \bar{E}_V + \alpha \cdot \sigma_{E_V} \cdot \sqrt{\frac{t_{\min}}{t}}$$

where:

- α is a scale factor (typically 2) controlling sensitivity
- $t_{\min}$ is the minimum epoch before checking begins (e.g., 10)
- The $\sqrt{t_{\min}/t}$ term provides temporal damping

The overfitting criterion becomes:

$$\text{overfitting} = E_V(t) > \bar{E}_V + \alpha \cdot \sigma_{E_V} \cdot \sqrt{\frac{t_{\min}}{t}}$$

C. Rationale for Temporal Scaling

The $\sqrt{t_{\min}/t}$ factor implements several design principles:

1) **Early tolerance**: At $t = t_{\min}$, the factor equals 1, giving a lenient threshold of $\bar{E}_V + \alpha\sigma_{E_V}$
2) **Progressive strictness**: As t increases, the threshold decreases toward $\bar{E}_V$, requiring tighter error bounds
3) **Smooth decay**: The square root ensures gradual tightening rather than abrupt changes
4) **Convergence detection**: Once the model stabilizes, even small validation error increases trigger detection

Additionally, we disable checking entirely for $t < t_{\min}$, allowing the model to escape poor local minima during early training without premature termination.

This adaptive approach successfully eliminated false positives while maintaining sensitivity to genuine overfitting, enabling effective early stopping without requiring manual epoch limits.

VI. Stage 2: Model with L1 Regularization and Pruning

The Stage 2 model extends the vanilla architecture with two complementary techniques for producing a compact, generalizable network: L1 regularization during training, and magnitude-based pruning after an initial pretraining phase. The hidden layer width is doubled to 2056 units, deliberately over-parameterizing the network so that pruning can remove redundant connections while preserving the most expressive ones.

A. L1 Regularization

The standard cross-entropy loss (derived in the Appendix) is augmented with an L1 penalty on all weights:

$$L(\mathbf{w}) = L_{\text{CE}}(\mathbf{w}) + \lambda \sum_i |w_i|$$

The L1 term adds a constant magnitude push toward zero for every weight, regardless of the weight's current value. As derived in the Appendix, the gradient of the combined loss is:

$$\frac{\partial L}{\partial w_i} = \underbrace{(\hat{y} - y)h_i}_{\text{cross-entropy gradient}} + \underbrace{\lambda \cdot \text{sgn}(w_i)}_{\text{L1 penalty}}$$

The key intuition is that the $\text{sgn}(w_i)$ term applies equal pressure to all weights irrespective of magnitude. Small, uninformative weights are driven to exactly zero because the L1 gradient dominates when $|w_i|$ is small, while large, important weights survive because the cross-entropy gradient outweighs the penalty. This produces a naturally sparse weight matrix—a property that L2 regularization cannot achieve, since L2's gradient ($2\lambda w_i$) shrinks large weights proportionally but never pushes them to exactly zero.

The regularization strength λ controls the sparsity-accuracy tradeoff. We experiment with $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ to identify the value that maximizes sparsity without degrading test accuracy.

B. Global Magnitude-Based Pruning

After pretraining the over-parameterized network (with L1 regularization encouraging sparsity), we apply global magnitude-based pruning. The procedure is:

1) **Collect magnitudes**: Gather $|w_i|$ for all weights across every Linear layer into a single vector.
2) **Compute threshold**: Given a target pruning rate p (e.g., $p = 0.50$ for 50% pruning), find the p-th percentile of the magnitude vector. All weights below this threshold are pruned.
3) **Apply binary masks**: For each layer, construct a mask $\mathbf{M}$ where $M_i = 1$ if $|w_i| >$ threshold and $M_i = 0$ otherwise. Set $\mathbf{w} \leftarrow \mathbf{w} \odot \mathbf{M}$.

C. Fine-Tuning After Pruning

After pruning, the sparse network is fine-tuned to recover any accuracy lost from removing weights. During fine-tuning, the binary masks are enforced after every optimizer step:

$$\mathbf{w}^{(t+1)} = \left(\mathbf{w}^{(t)} - \eta \nabla L(\mathbf{w}^{(t)})\right) \odot \mathbf{M}$$

This ensures pruned weights remain exactly zero—they never regrow. The surviving weights adjust to compensate for their removed neighbors, and L1 regularization continues to apply during fine-tuning to maintain sparsity pressure.

The full pipeline is: **pretrain** (200 epochs, wide network with L1) $\rightarrow$ **prune** (remove $p\%$ of weights globally) $\rightarrow$ **fine-tune** (100 epochs with mask enforcement). We evaluate pruning rates of 10%, 25%, and 50%.

VII. Stage 3: Semi-Supervised Learning

In a realistic setting, labeled data is scarce while unlabeled data is abundant. We simulate this by withholding 90% of the training labels, leaving only ∼5,600 labeled samples and ∼50,400 unlabeled samples. The goal is to leverage the unlabeled data to close the performance gap with the fully supervised upper bound. We implement two complementary semi-supervised approaches.

A. Graph-Based Label Propagation

Label propagation [1] treats the dataset as a graph where labels diffuse from labeled nodes to unlabeled neighbors based on feature similarity. We utilize the iterative approach to avoid the computationally prohibitive matrix inversion required by the closed-form solution.

1) Graph Construction: We construct a sparse symmetric adjacency matrix W over all N samples ($N = 56{,}000$). For every sample $\mathbf{x}_i$, we identify the set $\mathcal{N}_k(i)$ of its k nearest neighbors using Euclidean distance. The edge weights are computed via an RBF kernel:

$$W_{ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) & \text{if } j \in \mathcal{N}_k(i) \text{ or } i \in \mathcal{N}_k(j) \\ 0 & \text{otherwise} \end{cases}$$

To ensure the graph is undirected, we symmetrize the weights: $W \leftarrow (W + W^T)/2$. This results in a sparse

affinity matrix where nodes typically have between k and $2k$ connections.

2) Iterative Propagation: The affinity matrix is row-normalized to form the probabilistic transition matrix $T = D^{-1}W$, where $D_{ii} = \sum_j W_{ij}$. We define the label matrix $Y \in \mathbb{R}^{N\times C}$. For labeled samples, rows are initialized as one-hot vectors; for unlabeled samples, rows are initialized as uniform distributions $(1/C)$.

The labels are propagated iteratively. At each step t, we diffuse labels and reset the ground truth for labeled data:

$$Y^{(t+1)} \leftarrow TY^{(t)}$$

$$Y_L^{(t+1)} \leftarrow Y_{true} \quad \text{(Clamping)}$$

where Y_L denotes the rows corresponding to labeled indices. This process repeats until convergence ($\Delta Y < 10^{-6}$). Finally, we assign hard pseudo-labels to unlabeled points if their maximum class probability exceeds a confidence threshold τ (e.g., 0.8).

Note: The closed-form solution which Zhu derived in [2], $Y_U = (I - T_{UU})^{-1}T_{UL}Y_L$ requires inverting a $50{,}400\times50{,}400$ matrix (∼8 GB dense), making it infeasible. The iterative approach converges to the same fixed point using only sparse matrix–vector products.

B. Co-Training

Co-training, introduced by Blum ([3]), trains two classifiers on conditionally independent "views" of the data. Each classifier labels unlabeled samples, and high-confidence predictions from one are added to the other's training set. The inductive bias of Co-training is, that it assumes two independent views of a data point both contribute to the target output **independently**

1) View Construction: Each 28×28 image is split spatially into a left half (columns 0–13, yielding 392 features) and a right half (columns 14–27, 392 features). These views satisfy approximate conditional independence: the left and right halves of a garment carry complementary but redundant class information.

2) Algorithm:

1) Train two MLPs ($h = 512$, same architecture as vanilla but with 392-dim input) on the labeled data—one per view.
2) For each round $r = 1, \ldots, R$:
   a) Each model predicts on the remaining unlabeled pool.
   b) High-confidence predictions from the left model (above threshold τ) are added as pseudo-labels to the *right* model's training set, and vice versa.
   c) Both models retrain on their expanded datasets.
3) At inference, the two models' softmax outputs are averaged.

The cross-teaching mechanism is critical: each model provides an independent signal to the other, reducing confirmation bias compared to self-training. The confidence threshold τ controls the quality–quantity tradeoff of pseudo-labels; we evaluate $\tau \in \{0.70, 0.80, 0.90, 0.95\}$.

VIII. Complexity Analysis

Table I summarizes the asymptotic complexity of each model.

	Vanilla	**Stage2**	**Label Prop.**	**Co-Train**
Time	$ENdh$	$ENdh{+}W\log W$	$N^2d{+}INk$	$RENd_vh$
Space	W	W_{lg}	Nk	$2W_{\text{sm}}$
Bottleneck	MatMul	MatMul	Graph	Retrain

TABLE I
Complexity overview. N=samples, d=input dim, d_v=view dim, h=hidden units, W=params, E=epochs, R=rounds, I=LP iterations, k=neighbors.

Vanilla & Stage2: Training cost is dominated by matrix multiplication in the forward/backward pass: $O(B \cdot 784 \cdot h)$ per batch. Stage2 adds a one-time $O(W \log W)$ sort for global pruning, negligible relative to the training loop.

Label Propagation: The bottleneck is graph construction—computing pairwise Euclidean distances over all N points costs $O(N^2d)$. Sparse kNN storage reduces memory from $O(N^2)$ to $O(Nk)$, but the distance computation still requires dense intermediate buffers. The model additionally requires all of the training that the lower level models require, after generating these labels. Generally, the training after propagation is more demanding than this operation.

Co-Training: Trains two sub-models (h=512, input d_v=392) for R rounds, effectively multiplying training cost by $2R$. The labeled pool grows each round as pseudo-labels are added.

IX. Experimentation Process

All experiments use Fashion-MNIST (70,000 samples, 10 classes). We split into 56,000 train / 7,000 validation / 7,000 test. For semi-supervised experiments, 90% of training labels are withheld, leaving ∼5,600 labeled and ∼50,400 unlabeled. Unless otherwise noted, multi-seed experiments use seeds $\{1, 42, 123, 456, 12345\}$; single-seed tuning runs use seed 42.

A. Stage 1: Baseline and Preprocessing

We first train a vanilla MLP baseline, then, add preprocessing (z-score normalization and augmentation: horizontal flip + 3×3 color jitter). These configurations are run across 5 seeds to measure the isolated contribution of preprocessing, through rigorous testing.

B. Weight Initialization Comparison

With preprocessing and augmentation enabled, we compare He, uniform, and normal initialization across 5 seeds to select the best strategy going forward.

C. Normalization Strategy Comparison

A single-seed comparison of z-score vs. min-max normalization on the vanilla model, evaluating learning curves and final test accuracy to empirically justify the choice of standardization.

D. Hyperparameter Tuning

A two-phase grid search over learning rate and momentum:

1) **Phase 1 (coarse):** A broad grid evaluated over 2 seeds, scored by a pessimistic metric (mean − std), to identify the top 5 candidates.
2) **Phase 2 (refined):** The top 5 candidates are re-evaluated over 5 seeds to select the final hyperparameters.

E. Stage 2: L1 Regularization and Pruning

1) Lambda Tuning: A single-seed sweep over $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ to find the L1 strength that best balances sparsity and accuracy.

2) Magnitude-Based Pruning: Pruning rates of 10%, 25%, and 50% are tested. We measure the performance of pre-pruning, immediately following the prune, and then after fine-tuning, each across 5 seeds.

3) Finding an Upper Bound: We use the best observed **fully supervised model** on the **full** dataset, with 80% train, 10% test, 10% validation, to set a pseudo-upper-bound for the performance of the semi-supervised models. We look to determine how close in accuracy the semi-supervised variations can get to this upper bound. It would be unlikely that they could exceed the larger batch's domain knowledge, due to confirmation bias, and training through uncertainty, which is why we call this the theoretical upper bound of performance.

F. Stage 3: Semi-Supervised Learning

1) kNN Neighbor Tuning - Label Propagation: A single-seed sweep over $k \in \{5, 10, 15, 20, 30\}$ for label propagation with 80% confidence threshold, to select the number of graph neighbors. We choose k based on accuracy, and efficiency, given that larger k requires more computation in constructing the similarity graph, and in each iteration.

2) Confidence Threshold Sweep: Both label propagation and co-training are evaluated at $\tau \in \{0.70, 0.80, 0.90, 0.95\}$ across 5 seeds to find the optimal pseudo-label filtering threshold for each method.

3) SSL Methods Comparison: The best-threshold variants of label propagation and co-training are compared against the supervised baseline and observed upper bound.

G. PCA Dimensionality Reduction

PCA is applied to the three best models (Upper bound, label propagation, co-training) at $d' \in \{50, 100, 200, 400, 784\}$. Both test accuracy and wall-clock training time are recorded to quantify the computation–accuracy tradeoff.

H. Ablation Study and Statistical Analysis

A component ablation loads the final test accuracy from every stage across all 5 seeds. For each configuration we have $n = 5$ accuracy values $a_1, \ldots, a_5$. We quantify the uncertainty of each estimate as follows:

1) Compute the sample mean $\bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i$ and sample standard deviation $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(a_i - \bar{a})^2}$.
2) Compute the standard error of the mean: $\text{SE} = \frac{s}{\sqrt{n}}$.
3) With $n = 5$ observations, the degrees of freedom are $\text{df} = n - 1 = 4$. Because the population variance is unknown and n is small, we use the Student's t-distribution rather than the normal distribution.
4) Look up the critical value t^* such that the area in both tails beyond $\pm t^*$ equals $\alpha = 0.10$. For $\text{df} = 4$ and a 90% interval, $t^* = t_{0.95,4} \approx 2.132$.
5) The interval is $[\ \bar{a} - t^* \cdot \text{SE},\ \ \bar{a} + t^* \cdot \text{SE}\ ]$.

This means we are 90% confident that the true expected accuracy for each configuration lies within the reported interval. Additionally, we compute gap analysis: how many percentage points each configuration falls below the fully supervised upper bound.

I. Varying Labeled Data

The three best models—Stage2 with 50% pruning, graph-based label propagation, and co-training—are each evaluated at labeled fractions from 5% to 50%, to characterize how each method scales with label availability and where semi-supervised approaches provide the greatest advantage over the supervised baseline.

X. Results and Analysis

A. Stage 1: Baseline, Preprocessing and Weight Initialization

To evaluate the impact of weight initialization strategies, we compared the baseline model against models utilizing standard data preprocessing and He initialization. All experiments were conducted over $n = 5$ trials to ensure statistical significance, reporting the mean and standard deviation for Test Accuracy and Epochs to Convergence.

TABLE II
Comparison of Baseline, Preprocessing, and He Initialization (Stage 2 Ablation)

Method	Test Accuracy	Epochs
Baseline	0.0987 ± 0.0029	183.6 ± 20.1
+ Preprocessing	0.8494 ± 0.0038	25.6 ± 1.6
+ He Initialization	$\mathbf{0.8529} \pm 0.0051$	$\mathbf{25.0 \pm 5.2}$

We evaluated the model performance using a **Pessimistic Metric** (Lower Confidence Bound), defined as:

$$\text{Score}_{\text{pessimistic}} = \mu_{\text{acc}} - \sigma_{\text{acc}}$$

where μ_{acc} is the mean validation accuracy and σ_{acc} is the standard deviation across trials. This metric penalizes configurations with high variance, favoring hyperparameters that yield stable and reproducible convergence.

As shown in Table II, the **Baseline** configuration yielded a test accuracy of 9.87%, effectively worse than random guessing. The high number of epochs (183.6 ± 20.1) indicates that the network failed to converge entirely. This failure is likely attributed to improper weight variance. We used $\sigma = 0.1$ in normal initialization, which likely were not large enough to comprehend the full $0 - 255$ range which pixel colours can take on, pre-normalization. This lead to slow, and essentially, no learning (10). The introduction of **He Initialization** (`+he_init`) improved performance, achieving a test accuracy of 85.29%. While He initialization exhibits higher variance than using a normal weight distribution along with preprocessing ($.0051$, compared to $.0038$), the He initialization will still perform better in its assumed worst case, (the pessimist metric), yielding an average of 84.78% accuracy, while the pessimist metric of just preprocessing is 84.56%. Additionally, the He initialization converges in slightly less epochs in the average case.

TABLE III
Comparison of Weight Initialization Strategies (Test Accuracy)

Strategy	Distribution / Parameters	Test Accuracy
Random Uniform	$\mathcal{U}[-1/\sqrt{f_{in}}, 1/\sqrt{f_{in}}]$	0.8534 ± 0.0077
Random Normal	$\mathcal{N}(0, 0.01)$	0.8457 ± 0.0063
He Initialization	$\mathcal{N}(\mathbf{0}, \sqrt{\mathbf{2/f_{in}}})$	$\mathbf{0.8553 \pm 0.0073}$

Here, we can see that He initialization is the most reliable. It acheived the highest accuracy when paired with the ReLu activation function between layers of a Vanilla Neural Network. The worst case metric fairs He initialization as well, despite the fact it does not have the lowest variance.

B. Normalization Strategy

To evaluate the impact of input scaling, we trained a Vanilla Neural Network using two distinct normalization techniques: Min-Max scaling (bounding inputs to $[0, 1]$) and Z-score standardization (scaling to zero mean and unit variance). The comparative performance is visualized in Figure 1.

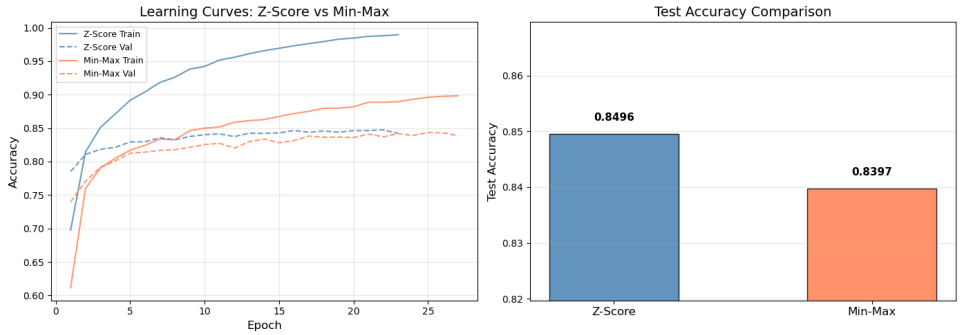

Fig. 1. Training convergence comparison: Min-Max Scaling vs. Z-score Standardization.

a) Analysis: We observed a significant performance boost when utilizing the Z-score strategy. As illustrated in the figure, Z-score not only achieved a higher test accuracy in fewer epochs but also allowed the training accuracy to reach approximately 97%—a ceiling that the Min-Max approach failed to approach.

This performance disparity can be attributed to the superior conditioning provided by Z-score standardization. While Min-Max scaling compresses data into a strictly positive $[0, 1]$ range, Z-score centers the data around zero. Zero-centering is critical for gradient descent; when inputs are strictly positive, gradients with respect to weights will share the same sign, leading to inefficient "zig-zagging" updates during backpropagation. Furthermore, Z-score is less sensitive to outliers. In Min-Max scaling, a single extreme outlier can compress the distribution of normal data into a tiny range, reducing the network's ability to distinguish between features. In all, Z-score preserves the order of the data, as well as the dispersion, while max min **only** preserves order.

C. Hyperparameter Tuning

To determine the optimal learning rate (LR) and momentum for the FashionMNIST classification task, we conducted a systematic empirical analysis using a two-stage grid search.

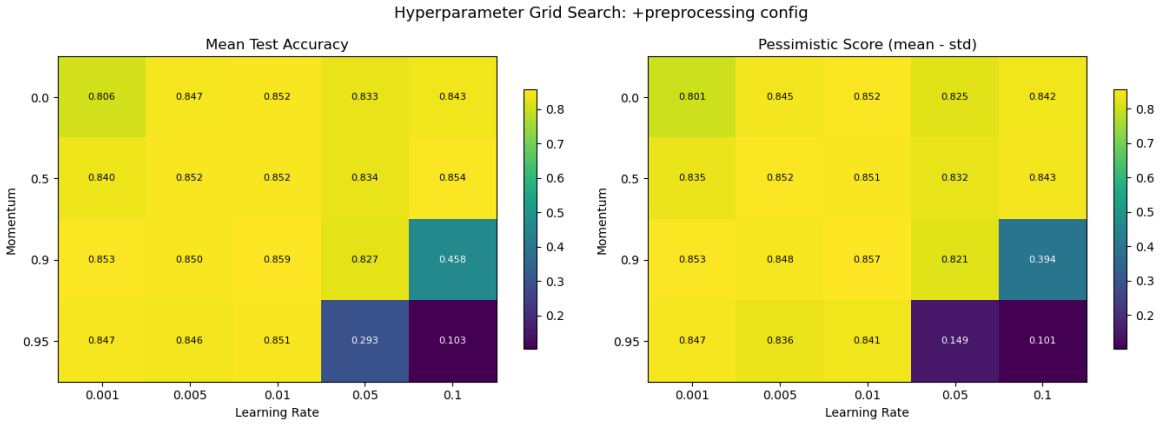

Fig. 2. Hyperparameter Grid Search (Heatmap). Left: Mean Test Accuracy. Right: Pessimistic Score. Note the degradation in performance at high Learning Rates combined with high Momentum (bottom-right quadrant).

a) Search Strategy and Results: The search was divided into two phases:

1) **Phase 1 (Coarse):** A grid search over Learning Rates $\{0.001, \ldots, 0.1\}$ and Momentum $\{0.0, \ldots, 0.95\}$ with $n = 2$ seeds. The top candidates identified were $(LR, Mom) \in \{(0.1, 0.5), (0.005, 0.0), (0.001, 0.9), (0.005, 0.5), (0.01, 0.5)\}$.
2) **Phase 2 (Refinement):** The top 5 configurations were re-evaluated with $n = 5$ seeds to ensure statistical significance.

As visualized in Figure 2, the model exhibits instability (purple regions) when both learning rate and momentum are high ($LR \geq 0.05, Mom \geq 0.9$), failing to converge. Conversely, lower learning rates demonstrated consistent stability.

After the Phase 2 refinement, the configuration $(LR = 0.001, Momentum = 0.9)$ achieved the highest pessimistic score, with a test accuracy of 0.8530 ± 0.0061. Consequently, we selected these hyperparameters for the remainder of the experimentation. We operate under the hypothesis that minor architectural changes (maintaining similar model size and the same dataset) will not significantly alter the optimal learning rate and momentum.

D. Stage 2 Model

We empirically discover the best λ and the best pruning rate p.

1) L1 Regularization Tuning: To evaluate the impact of L1 regularization on model performance and weight sparsity, we conducted a parameter sweep over $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$. L1 regularization adds a penalty term proportional to the absolute value of the weights, encouraging the model to drive irrelevant feature weights to zero.

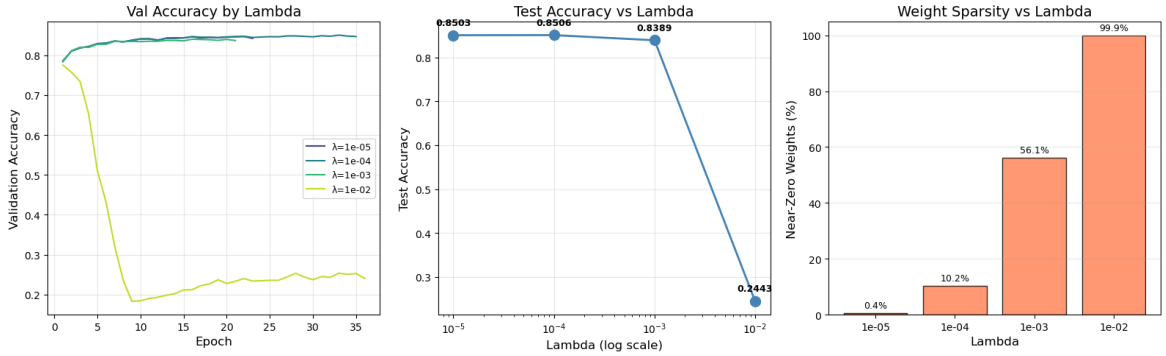

Fig. 3. Effect of L1 Regularization λ. Left: Validation Accuracy curves. Center: Test Accuracy vs λ. Right: Percentage of Near-Zero Weights (Sparsity).

TABLE IV
L1 REGULARIZATION RESULTS (ACCURACY VS. SPARSITY)

λ	Test Accuracy	Epochs	Sparsity (Near-zero %)
10^{-5}	0.8503	23	0.4%
$\mathbf{10^{-4}}$	**0.8506**	35	10.2%
10^{-3}	0.8389	21	56.1%
10^{-2}	0.2443	36	99.9%

a) Analysis: The results (Table IV and Figure 3) illustrate the classic trade-off between predictive performance and model parsimony:

- **Optimal Performance ($\lambda = 10^{-4}$):** The model achieved its peak accuracy of 85.06% at this level. Interestingly, it maintained this accuracy while rendering $\approx 10\%$ of the weights near-zero, suggesting that the network contains redundant connections that can be pruned without information loss.
- **High Compression ($\lambda = 10^{-3}$):** Increasing λ to 10^{-3} resulted in a slight accuracy penalty ($\approx 1.2\%$ drop), but drastically increased sparsity to 56.1%. This configuration represents a viable "lightweight" model where more than half the parameters are effectively removed.
- **Model Collapse ($\lambda = 10^{-2}$):** At this magnitude, the regularization penalty overwhelms the loss function. The optimizer forces 99.9% of weights to zero to minimize the L1 term, causing the model to underfit severely (Test Acc: 24.43%).

For subsequent experiments where accuracy is paramount, $\lambda = 10^{-4}$ is the preferred configuration.

2) Pruning Rate Analysis: To determine the redundancy within the network, we applied unstructured pruning at rates of 10%, 25%, and 50%. After pruning, the models were fine-tuned to recover accuracy, while the maskings were still enforced.

TABLE V
PRUNING RESULTS: FINE-TUNED ACCURACY VS. SPARSITY

Pruning Rate	Mean Accuracy	Std Dev (σ)	Pessimistic Score
10%	0.8538	0.0062	0.8476
25%	0.8546	0.0058	0.8488
50%	**0.8549**	0.0067	0.8482

a) Results and Analysis: As shown in Table V, pruning up to 50% yields nearly identical predictive performance across configurations. The difference in mean accuracy between the 25% and 50% sparsity levels is only 0.0003, which lies well within one standard deviation of the observed stochastic variation. This indicates that the configurations are statistically indistinguishable in practical terms.

Although the 25% model exhibits slightly lower variance ($\sigma = 0.0058$ vs. $\sigma = 0.0067$), the magnitude of this difference is minor relative to overall experimental noise. Importantly, the 50% configuration achieves the highest mean accuracy while reducing the parameter count by half.

1) **Performance Parity:** Accuracy remains effectively unchanged between 25% and 50% sparsity, suggesting substantial weight redundancy in the baseline architecture.
2) **Efficiency Advantage:** The 50% model provides maximal compression under negligible performance degradation, offering the strongest compression–accuracy tradeoff.

We therefore select the 50% sparsity configuration as the preferred model, as it maximizes parameter efficiency while maintaining statistical performance equivalence.

b) Experimental Configuration: The model was initialized with He weights and trained using the following "ideal" parameters:

- **Learning Rate:** 0.01
- **Momentum:** 0.9
- **L1 Regularization (λ):** 10^{-4}
- **Pruning Rate:** 50%

c) Results: Averaged over $n = 5$ seeds to ensure statistical reliability, the fully supervised model achieved:

$$\text{Test Accuracy}_{\text{upper}} = \mathbf{0.8945 \pm 0.0031}$$

d) Analysis: This result (89.45%) serves as the benchmark for all subsequent semi-supervised learning (SSL) experiments. Since SSL methods typically operate with only a fraction of these labels, their success will be measured by how closely they can approach this upper bound. The low standard deviation ($\pm 0.31\%$) confirms that the chosen hyperparameters ($LR = 0.01$, $Mom = 0.9$, Pruning=50%) provide a stable convergence regime even when trained on the full data distribution. Out of curiosity, we ran a 25% prune rate, with the full dataset. Not only was it less efficient, due to more active weights, but the performance was about 0.1% lower on average, but did yield a lower variance. This could be attributed to the fact that a lower pruning rate maintains expressivity of the model, and thus reduces variance.

E. Stage 3: Semi-Supervised Learning

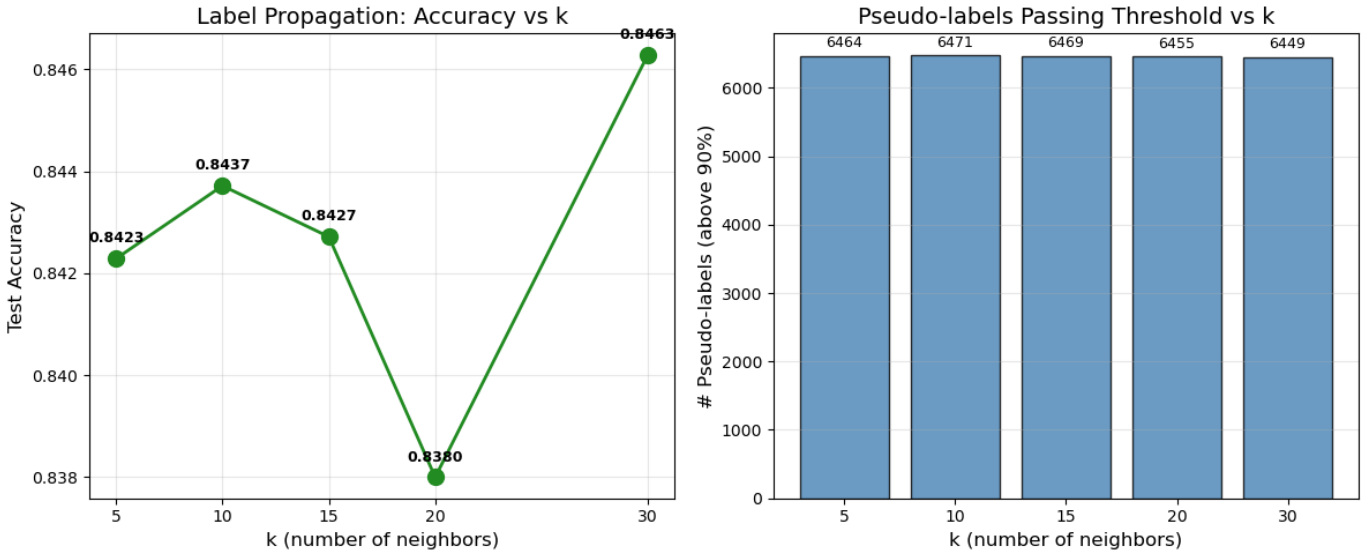

Fig. 4. Left: Test accuracy of Label Propagation as a function of k in the k-nearest neighbor similarity graph. Right: Number of pseudo-labels exceeding the 90% confidence threshold for each k.

a) Effect of k in the Similarity Graph.: Figure 4 examines the sensitivity of Label Propagation to the choice of k in the k-nearest neighbor (kNN) similarity matrix. The highest observed test accuracy occurs at $k = 30$ (0.8463), followed closely by $k = 10$ (0.8437). The absolute improvement between these two configurations is marginal (≈ 0.0026), indicating diminishing returns as graph connectivity increases. Notably, performance degrades at $k = 20$ (0.8380). The right panel shows that the number of pseudo-labels surpassing the 90% confidence threshold remains effectively constant across k values (approximately 6.45×10^3 samples). This indicates that changes in accuracy are not driven by coverage differences but rather by structural effects in the graph.

From a theoretical perspective, k controls the bias–variance characteristics of the graph-based estimator. Smaller k yields a sparser graph that better preserves local manifold structure but may increase variance due to weaker connectivity. Larger k enforces stronger smoothness constraints via denser Laplacian regularization, reducing variance but increasing bias by oversmoothing across class boundaries. The empirical pattern suggests that moderate connectivity ($k = 10$) achieves a favorable balance between locality and stability.

Given the negligible performance gain at $k = 30$ relative to $k = 10$, and considering the increased computational cost associated with constructing and operating on denser similarity graphs, $k = 10$ is selected for subsequent experiments as an efficiency-aware choice with near-optimal generalization performance.

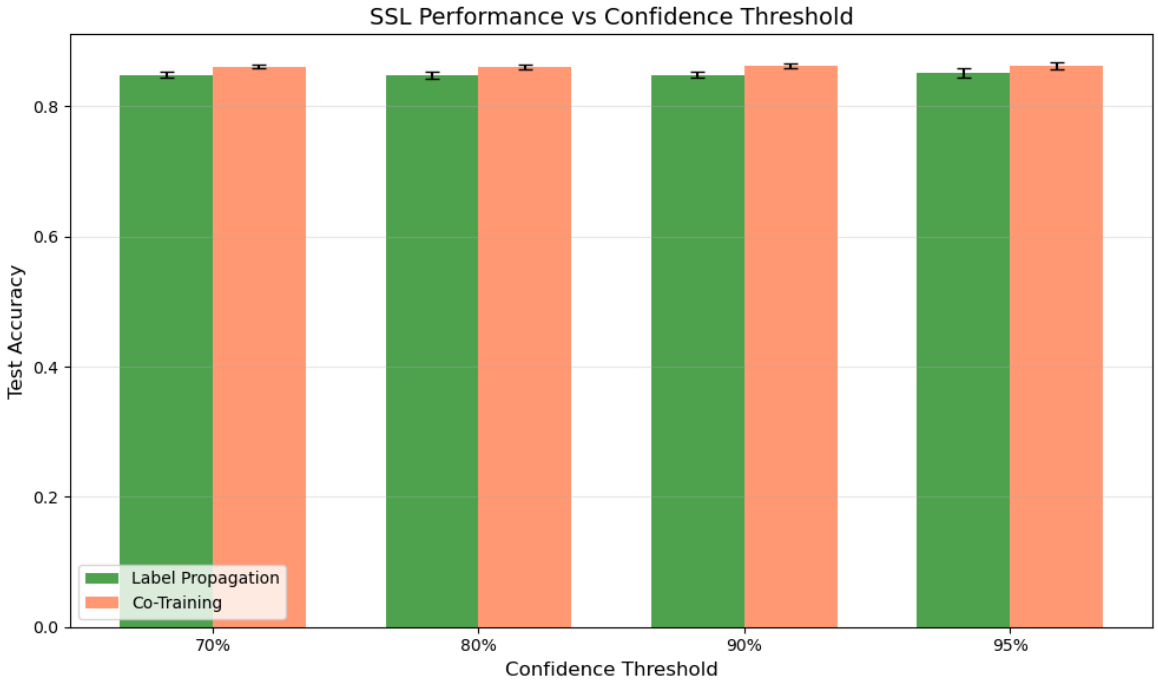

Fig. 5. Test accuracy as a function of confidence threshold for Label Propagation and Co-Training. Error bars denote standard deviation across runs.

b) Confidence Threshold Analysis: Figure 5 shows the effect of varying the confidence threshold for pseudo-label selection in both Label Propagation and Co-Training. In this setting, where only 10% of the data is initially labeled, both methods achieve their best performance at the strictest threshold of 95%. Label Propagation reaches 0.8523 ± 0.0046, while Co-Training attains 0.8634 ± 0.0051. The relatively small standard deviations indicate stable behavior across runs, though variance is slightly higher than in higher-label regimes, reflecting increased sensitivity to pseudo-label quality.

This need for a high threshold can be theoretically explained by the low supervision level. With only 10% labeled data, the initial decision boundary is less well-constrained, increasing the risk that early pseudo-label errors propagate and amplify through subsequent iterations. Under such low-label conditions, confirmation bias becomes more severe: incorrectly labeled high-density regions can dominate the learning dynamics. A stricter confidence requirement (95%) effectively acts as a noise-control mechanism, prioritizing pseudo-label precision over coverage and limiting the accumulation of systematic errors.

For Label Propagation, which relies on graph smoothness and manifold consistency, high-confidence filtering helps preserve local label homogeneity within the similarity graph, preventing error diffusion across connected components. For Co-Training, although complementary views provide some robustness to noise, the scarcity of ground-truth anchors increases the importance of conservative pseudo-label acceptance, explaining why its optimum also occurs at 95%.

Overall, the results suggest that the optimal confidence threshold is sensitive to the proportion of labeled data. In low-label regimes, stricter thresholds are necessary to stabilize training and control error propagation, reinforcing the precision–coverage trade-off central to semi-supervised learning.

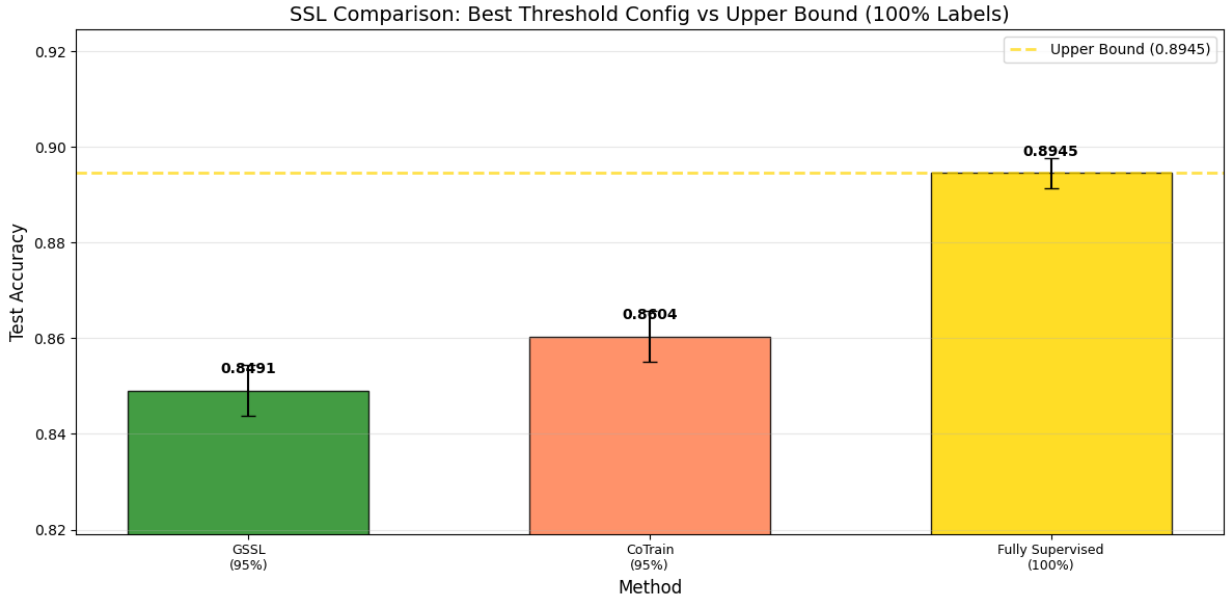

Fig. 6. Comparison between the best semi-supervised configurations and the fully supervised upper bound (100% labeled data). Error bars denote standard deviation across runs.

1) Best Semi-Supervised Configurations vs. Fully Supervised Upper Bound: Figure 6 compares the best-performing configuration of each semi-supervised method against the fully supervised model trained on 100% labeled data. The fully supervised upper bound achieves 0.8945 ± 0.0031 test accuracy. The best Label Propagation configuration (95% confidence threshold) reaches 0.8491 ± 0.0053, corresponding to a performance gap of 4.54 percentage points. The best Co-Training configuration (95% threshold) achieves 0.8604 ± 0.0054, reducing the gap to 3.41 percentage points.

The results indicate that while neither semi-supervised approach matches the performance of complete supervision, Co-Training recovers a larger fraction of the supervised accuracy relative to graph-based Label Propagation. The comparable standard deviations across methods suggest similar sensitivity to stochastic variation, though both semi-supervised approaches exhibit slightly higher variance than the fully supervised upper bound.

From a theoretical standpoint, the residual performance gap is expected. Fully supervised learning benefits from access to ground-truth labels for the entire dataset, eliminating pseudo-label noise and preventing error reinforcement. In contrast, semi-supervised methods rely on high-confidence pseudo-labeling under the cluster and low-density separation assumptions. With only 10% labeled data, the initial decision boundary is less constrained, increasing susceptibility to confirmation bias and error propagation. Even with a strict 95% confidence threshold, pseudo-label noise cannot be completely eliminated.

Notably, Co-Training maintains a smaller gap to the upper bound, suggesting that leveraging complementary views and iterative cross-label exchange provides improved robustness to limited supervision. Nevertheless, the remaining 3.41pp gap quantifies the empirical cost of incomplete labeling under the present experimental regime.

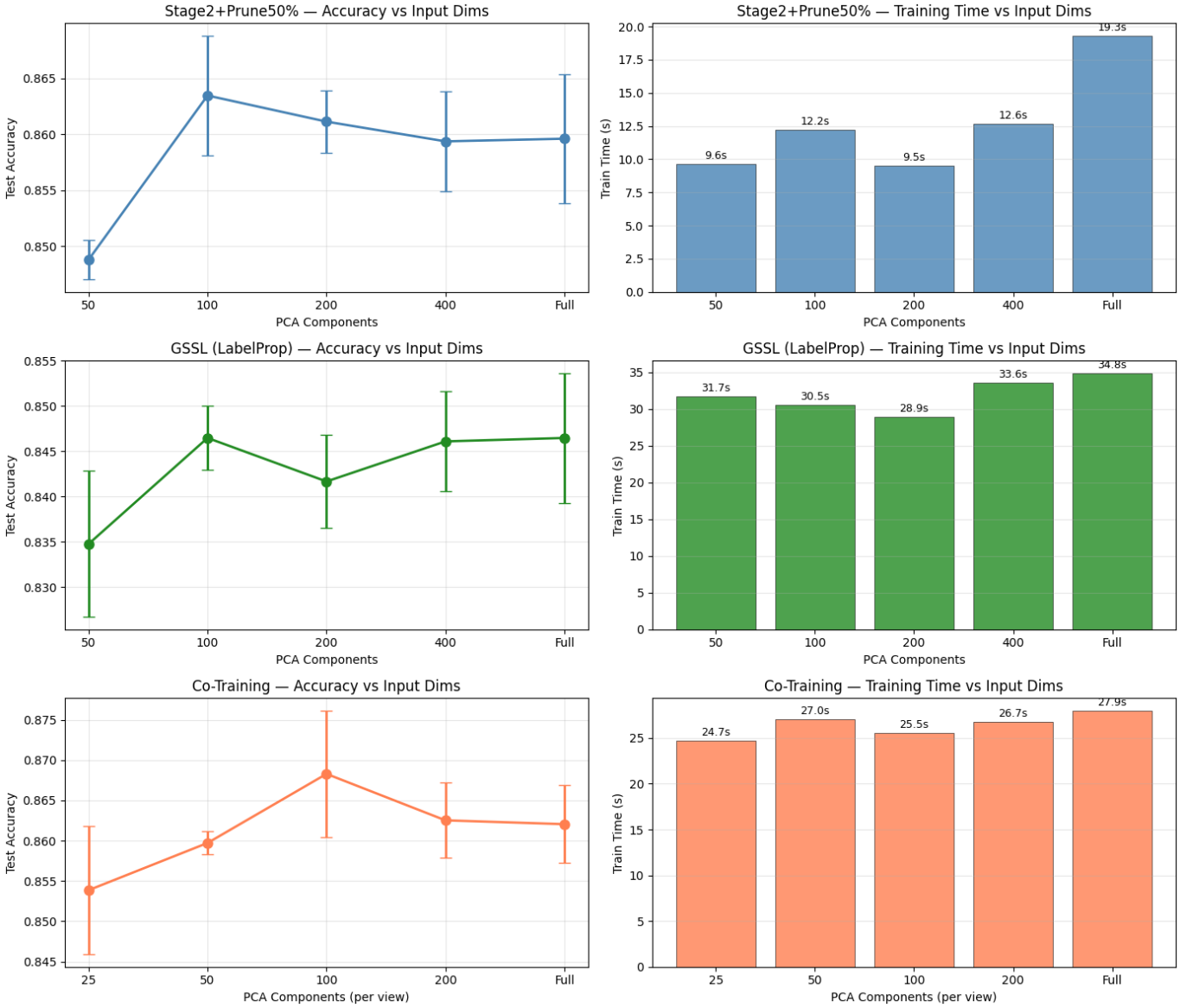

Fig. 7. Effect of PCA dimensionality on accuracy and training time for Stage2+Prune50%, GSSL (Label Propagation), and Co-Training. Speedup is measured relative to the full 784-dimensional input.

F. PCA Dimensionality Reduction Analysis

Figure 7 summarizes the trade-off between representational dimensionality, generalization performance, and computational efficiency. Across all methods, moderate dimensionality reduction preserves—and in some cases slightly improves—test accuracy while reducing training time.

1) Stage2 + 50% Pruning.: The supervised baseline with magnitude pruning exhibits strong robustness to dimensionality reduction. Reducing from 784 to 200 principal components yields a marginal accuracy improvement (+0.15pp) alongside a $2.02\times$ speedup (9.5s vs. 19.3s). At 100 components, performance exceeds the full model (+0.39pp) with a $1.58\times$ speedup. Even aggressive compression to 50 components incurs only a 1.08pp accuracy loss while achieving a $2.00\times$ acceleration. These results suggest that much of the discriminative signal lies in a relatively low-dimensional subspace, and that discarding noisy high-dimensional directions can act as an implicit regularizer, particularly when combined with pruning.

2) GSSL (Label Propagation).: For graph-based semi-supervised learning, PCA interacts with the geometry of the similarity matrix. Performance at 100 components exactly matches full-dimensional accuracy (0.8465) with a modest $1.14\times$ speedup. The efficiency gains are smaller than in the supervised case, as graph construction and label propagation dominate computational cost—even halving dimensionality yields only marginal time savings. The degradation at 50 components (-1.17pp) indicates that excessive compression distorts local manifold structure, weakening the cluster assumption that underpins Label Propagation.

Interestingly, 200 components slightly underperforms 100 (-0.48pp vs. full), suggesting that intermediate dimensionality may introduce suboptimal trade-offs in graph neighborhood quality.

3) Co-Training.: Co-Training shows a distinctive pattern: accuracy peaks at 100 components per view (0.8683, +0.62pp over full), representing a genuine improvement from dimensionality reduction. However, computational speedups are minimal across all settings (1.03–$1.13\times$), reflecting that the overhead of training multiple learners and iterative pseudo-label exchange dominates over feature dimensionality costs. The accuracy boost at 100 components suggests that removing noisy dimensions improves the diversity and quality of the two views, strengthening the agreement-based learning signal.

4) Theoretical Interpretation.: PCA projects data onto directions of maximal variance, which often correspond to dominant semantic structure in image datasets. By discarding low-variance directions—frequently dominated by noise—PCA reduces estimator variance and improves conditioning of the optimization landscape. However, excessive dimensionality reduction increases bias by eliminating discriminative information and distorting neighborhood relationships critical for graph-based methods.

Empirically, 100–200 principal components appear to capture the majority of task-relevant structure, achieving near-identical or improved generalization with meaningful computational savings for the supervised model (approximately $2\times$) and modest gains for graph-based learning (approximately 1.1–$1.2\times$). For Co-Training, the benefit is primarily statistical rather than computational, with dimensionality reduction improving accuracy by enhancing view quality. Overall, the results demonstrate that dimensionality reduction can function simultaneously as a regularizer and an efficiency mechanism, though its computational benefit depends on whether feature dimensionality or algorithmic structure is the primary bottleneck.

G. Ablation Study and Statistical Analysis

Table VI presents the component contribution analysis. Each row adds one technique on top of the previous configuration, isolating its marginal effect. Accuracy is reported as $\bar{a} \pm s$ over $n = 5$ seeds.

Configuration	**Test Accuracy**	**Δ Mean (Δ Std)**
Baseline (raw, normal init)	0.0987 ± 0.0029	—
+ Preprocessing (Z-score + aug)	0.8494 ± 0.0038	$+0.7507$ $(+0.0009)$
+ L1 Regularization	0.8503 ± 0.0045	$+0.0009$ $(+0.0007)$
+ Pruning (50%, He init)	0.8549 ± 0.0067	$+0.0046$ $(+0.0022)$
+ Semi-Supervised (Label Prop)	0.8482 ± 0.0070	-0.0067 $(+0.0003)$
+ Semi-Supervised (Co-Training)	0.8586 ± 0.0064	$+0.0104$ (-0.0006)
Upper Bound (100% labels)	0.8945 ± 0.0031	—

TABLE VI
COMPONENT ABLATION WITH STEP-WISE PERFORMANCE CHANGES. EACH ROW ADDS ONE TECHNIQUE; ACCURACY IS MEAN $\pm$ STD OVER 5 SEEDS. THE LAST COLUMN SHOWS THE CHANGE IN MEAN AND STANDARD DEVIATION COMPARED TO THE PREVIOUS ROW.

Preprocessing accounts for the largest single jump ($+75.1$ pp), transforming a non-convergent baseline into a competitive model. Subsequent additions—L1 regularization ($+0.09$ pp), pruning ($+0.46$ pp), and co-training ($+0.83$ pp over pruning)—yield smaller but cumulative gains. The best semi-supervised configuration (co-training) reaches 85.86%.

1) Paired t-Tests: To determine whether the observed differences are statistically meaningful, we apply **paired t-tests**. For each pair of configurations, we compute the per-seed difference $d_i = a_i^{(A)} - a_i^{(B)}$ for $i = 1, \ldots, 5$, then:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}, \qquad \text{df} = n - 1 = 4$$

where $\bar{d}$ is the mean difference and s_d its standard deviation. The resulting t-statistic is compared against the Student's t-distribution with df $= 4$. We adopt a **significance level of** $\alpha = 0.10$ (i.e. a 90% confidence level): if the two-tailed p-value falls below 0.10, we reject the null hypothesis $H_0 : \bar{d} = 0$ and conclude that the performance difference is statistically significant.

In plain terms, "significant at 90%" means: *if there were truly no difference between the two methods, there is less than a 10% probability of observing a discrepancy this large (or larger) by chance alone*. The smaller the p-value, the stronger the evidence against the null hypothesis.

Comparison	$t(4)$	p-value	**90% CI**
Co-Train vs. Label Prop	8.459	0.0011	$[0.0078,\ 0.0130]$
Co-Train vs. Upper Bound	-17.488	0.0001	$[-0.0403,\ -0.0315]$
Pruning vs. L1 Reg	2.740	0.0519	$[0.0010,\ 0.0082]$

TABLE VII
PAIRED t-TESTS ($n = 5$ SEEDS, DF $= 4$, $\alpha = 0.10$). ALL THREE COMPARISONS ARE SIGNIFICANT.

All three comparisons are significant at the 90% level ($p < 0.10$):

- **Co-Training vs. Label Propagation** ($p = 0.0011$): Co-training reliably outperforms label propagation by ≈ 1.0 pp. The p-value is well below 0.01, giving very strong evidence.
- **Co-Training vs. Upper Bound** ($p = 0.0001$): The fully supervised model remains significantly better by ≈ 3.6 pp, confirming a real and consistent performance gap that semi-supervised learning does not fully close.
- **Pruning vs. L1 Regularization** ($p = 0.0519$): The narrowest margin—just barely significant at $\alpha = 0.10$ but not at $\alpha = 0.05$. The 0.46 pp improvement from pruning is modest but reproducible across seeds.

H. Varying Label Data

Figure 8 illustrates test accuracy as a function of labeled data availability (5%, 10%, 25%, 50%) across Stage2 (50% pruned), Label Propagation, and Co-Training.

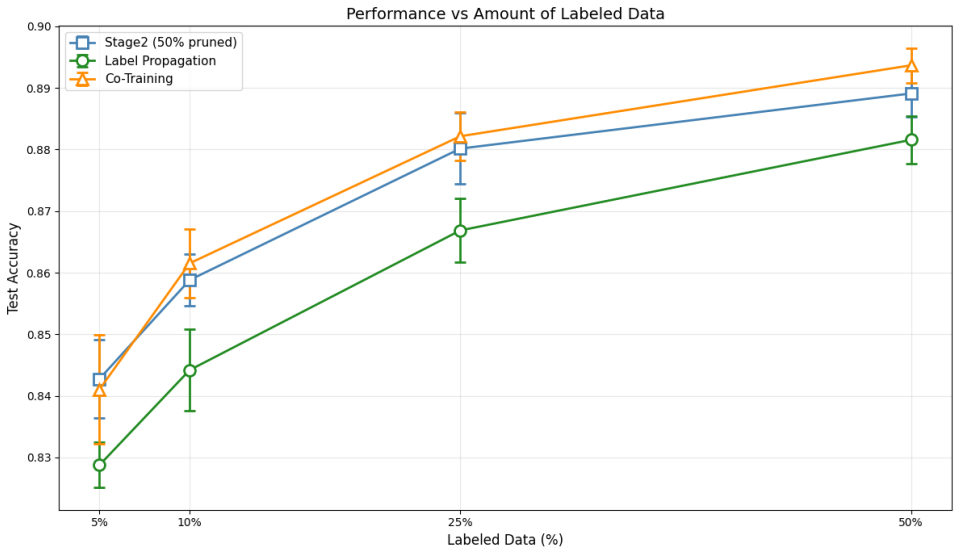

Fig. 8. Performance vs. amount of labeled data. Error bars denote standard deviation across runs.

As shown in Figure 8, all methods exhibit increasing performance improvements as the labeled fraction increases. The transition from 5% to 10% yields a substantial gain, and further improvement is observed when increasing to 25%, indicating that additional supervision meaningfully reduces estimation error in low-label regimes.

However, the marginal improvement from 25% to 50% is comparatively small across all methods. The absolute gain in accuracy is modest relative to earlier jumps and is on the same order as the reported standard deviations. This suggests diminishing returns as the models approach their supervised performance ceiling under the current architecture and optimization setup.

From an efficiency–performance perspective, 25% labeled data provides the strongest tradeoff. It achieves near-saturated accuracy while requiring only half the annotation cost of the 50% configuration. The increase to 50% yields only mild performance improvements, making 25% the most cost-effective operating point among the evaluated ratios, especially when considering the scarcity of proper labeled data in real life.

XI. Conclusion

This study set out to investigate whether semi-supervised learning can meaningfully leverage unlabeled data in a simulated label-scarce environment using Fashion-MNIST. The ablation analysis reveals a clear hierarchy of contributions. Preprocessing—Z-score normalization and data augmentation—is by far the most impactful single intervention, lifting a non-convergent baseline from 9.87% to 84.94% ($+75.1$ pp). Without proper standardization the network cannot even begin to learn, underscoring that no amount of architectural sophistication compensates for poorly conditioned inputs.

Building on that foundation, L1 regularization and 50% magnitude-based pruning provide incremental gains ($+0.09$ and $+0.46$ pp respectively). Pruning is statistically significant over L1 alone ($p = 0.0519$, $\alpha = 0.10$), and the resulting sparse network offers a favorable computation–accuracy tradeoff: half the weights are removed with no loss in accuracy. However, pruning also increases cross-seed variance ($\Delta\text{Std} = +0.0022$), reflecting the sensitivity of which weights survive the magnitude threshold under different random initializations.

The semi-supervised stage is where the most interesting dynamics emerge. Label propagation, despite successfully propagating pseudo-labels through a kNN graph, actually *decreases* accuracy by 0.67 pp relative to the pruned model. This is likely attributed to the memory constraints of my device, forcing experiments to be run with a small kNN graph, opposed to the full similarity matrix like Co-training, on the other hand, gains $+1.04$ pp over label propagation, while simultaneously *reducing* variance ($\Delta\text{Std} = -0.0006$). The cross-teaching mechanism between two view-specific models provides a natural form of regularization: each model's errors are independent, so high-confidence pseudo-labels from one genuinely inform the other. Paired t-tests confirm that co-training significantly outperforms label propagation ($p = 0.0011$), though a significant gap to the fully supervised upper bound remains ($p = 0.0001$, ≈ 3.6 pp).

PCA dimensionality reduction further reveals that much of the input space is redundant. Reducing from 784 to 200 components preserves accuracy, and even yields improvements to accuracy, all while accelerating training, confirming that the models learn from a relatively low-dimensional manifold within the pixel space. This is intuitive. The Fashion dataset has a large amount of noise in the form of empty cells around the border, random colouring which is not indicative of the clothing type, etc.

In summary, when labels are scarce, the recipe that emerges is: normalize and augment first, regularize and prune to compress the network, then apply co-training to extract signal from unlabeled data. This pipeline closes 67.3% of the gap to the fully supervised upper bound using only 5% of the labels—a practical demonstration that semi-supervised learning is a valuable tool in label-scarce, data-abundant scenarios.

APPENDIX

GRADIENT DERIVATIONS

A. Vanilla Model: Cross-Entropy Loss Gradient

The vanilla model uses standard cross-entropy loss with softmax activation. We derive the gradient with respect to weights in the final layer.

1) Forward Pass: Let $\mathbf{z} \in \mathbb{R}^{10}$ be the logits (pre-activation outputs) and $\mathbf{y} \in \{0,1\}^{10}$ be the one-hot encoded true label. The softmax function computes predicted probabilities:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{10} e^{z_j}} \quad \text{for } k = 1, \ldots, 10$$

The cross-entropy loss is:

$$L = -\sum_{k=1}^{10} y_k \log(\hat{y}_k)$$

Since $\mathbf{y}$ is one-hot, if class c is the true class, this simplifies to:

$$L = -\log(\hat{y}_c)$$

2) Gradient w.r.t. Logits: We first compute $\frac{\partial L}{\partial z_k}$. Using the chain rule:

$$\frac{\partial L}{\partial z_k} = -\sum_{j=1}^{10} y_j \frac{\partial \log(\hat{y}_j)}{\partial z_k} = -\sum_{j=1}^{10} \frac{y_j}{\hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_k}$$

For the softmax derivative, when $j = k$:

$$\left.\frac{\partial \hat{y}_j}{\partial z_k}\right|_{j=k} = \hat{y}_k(1 - \hat{y}_k)$$

When $j \neq k$:

$$\left.\frac{\partial \hat{y}_j}{\partial z_k}\right|_{j\neq k} = -\hat{y}_j \hat{y}_k$$

Substituting:

$$\frac{\partial L}{\partial z_k} = -\frac{y_k}{\hat{y}_k}\hat{y}_k(1 - \hat{y}_k) - \sum_{j\neq k} \frac{y_j}{\hat{y}_j}(-\hat{y}_j \hat{y}_k) \tag{9}$$

$$= -y_k(1 - \hat{y}_k) + \sum_{j\neq k} y_j \hat{y}_k \tag{10}$$

$$= -y_k + y_k \hat{y}_k + \hat{y}_k \sum_{j\neq k} y_j \tag{11}$$

$$= -y_k + \hat{y}_k \left(y_k + \sum_{j\neq k} y_j \right) \tag{12}$$

$$= -y_k + \hat{y}_k \quad (\text{since } \sum_j y_j = 1) \tag{13}$$

$$= \hat{y}_k - y_k \tag{14}$$

3) Gradient w.r.t. Weights: For a weight w_{kj} connecting hidden unit j to output unit k, with hidden activation h_j:

$$z_k = \sum_j w_{kj} h_j + b_k$$

By the chain rule:

$$\frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial w_{kj}} = (\hat{y}_k - y_k) h_j$$

In vectorized form, for the weight matrix $\mathbf{W}$ and hidden layer activation $\mathbf{h}$:

$$\boxed{\frac{\partial L}{\partial \mathbf{W}} = (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^T}$$

This elegant result shows that the gradient is simply the prediction error multiplied by the input activation.

B. Stage 2 Model: Regularization Gradients

$$L(\mathbf{w}) = L_{\text{CE}}(\mathbf{w}) + \lambda_1 \sum_i |w_i|$$

where:

- L_{CE} is the cross-entropy loss
- $\lambda_1 |w|$ penalizes large weights (L1 regularization)

1) L1 Regularization Gradient: The L1 term gradient is:

$$\frac{\partial}{\partial w_i}(\lambda_1 |w_i|) = \lambda_1 \frac{\partial |w_i|}{\partial w_i} = \lambda_1 \cdot \text{sgn}(w_i)$$

where:

$$\text{sgn}(w) = \begin{cases} +1 & \text{if } w > 0 \\ -1 & \text{if } w < 0 \\ 0 & \text{if } w = 0 \end{cases}$$

2) Combined Gradient:

$$\boxed{\frac{\partial L}{\partial w_i} = (\hat{y} - y)h_i + \lambda_1 \cdot \text{sgn}(w_i)}$$

The terms have distinct effects:

1) **Cross-entropy gradient**: Drives learning toward correct, confident predictions
2) **L1 penalty** ($+\lambda_1 \text{sgn}(w)$): Pushes weights toward zero, promoting sparsity

PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA is a linear dimensionality reduction technique that projects data onto the directions of maximum variance. We apply PCA to our best-performing models to evaluate the computation–accuracy tradeoff.

C. Derivation

Given a centered data matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ (mean-subtracted), PCA seeks orthogonal directions $\mathbf{v}_1, \ldots, \mathbf{v}_k$ that maximize the variance of the projected data. The covariance matrix is:

$$\mathbf{C} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d \times d}$$

The principal components are the eigenvectors of $\mathbf{C}$, sorted by decreasing eigenvalue:

$$\mathbf{C}\mathbf{v}_k = \lambda_k \mathbf{v}_k, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

The eigenvalue λ_k equals the variance of the data projected onto $\mathbf{v}_k$. To reduce from d to d' dimensions, we form the projection matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_{d'}] \in \mathbb{R}^{d \times d'}$ and transform:

$$\mathbf{X}_{\text{reduced}} = \mathbf{X}\mathbf{V} \in \mathbb{R}^{N \times d'}$$

The fraction of total variance retained is:

$$\text{Explained variance ratio} = \frac{\sum_{k=1}^{d'} \lambda_k}{\sum_{k=1}^{d} \lambda_k}$$

D. Application to Models

For the **Stage2 (pruned) model** and **label propagation**, PCA is applied directly to the 784-dimensional input. This shrinks the first weight layer from $(784 \times h)$ to $(d' \times h)$ and, for label propagation, reduces the $O(N^2 d)$ distance computation proportionally. Label propagation benefits most: reducing $d = 784 \to 50$ makes the N^2 distance loop $\sim$15x faster, and lower-dimensional distances are more meaningful due to reduced curse of dimensionality, often yielding a higher-quality affinity graph.

For **co-training**, PCA cannot be applied globally—mixing features linearly would destroy the spatial left/right view separation, violating the conditional independence assumption. Instead, we split the raw pixels into left (392 features) and right (392 features) *first*, then fit a separate PCA on each view independently:

$$\mathbf{X}'_{\text{left}} = \mathbf{X}_{\text{left}}\mathbf{V}_{\text{left}} \in \mathbb{R}^{N \times d'} \tag{15}$$

$$\mathbf{X}'_{\text{right}} = \mathbf{X}_{\text{right}}\mathbf{V}_{\text{right}} \in \mathbb{R}^{N \times d'} \tag{16}$$

Since each view starts at 392 dimensions (half of 784), the component values are halved accordingly: $d' \in \{25, 50, 100, 200\}$ per view. This preserves view independence while reducing each sub-model's input dimensionality. Both accuracy and wall-clock training time are reported.

Data Augmentation Definitions

E. 3x3 Color Jitter

We implement a neighborhood-based noise injection method that adds adaptive Gaussian noise to each pixel based on the local variance of its 3×3 neighborhood. This preserves local structure while introducing controlled randomness.

The noise magnitude adapts to local image structure: high-variance regions (edges, textures) receive more noise, while homogeneous regions receive less. This prevents destroying important features while still providing meaningful augmentation.

Algorithm 1 3x3 Neighborhood Color Jitter

Require: Image $I \in \mathbb{R}^{28 \times 28}$, scale factor s
Ensure: Augmented image I'
$I' \leftarrow I$ (copy)
for each pixel (i, j) in I **do**
  $i_0 \leftarrow \max(0, i-1)$, $i_1 \leftarrow \min(28, i+2)$
  $j_0 \leftarrow \max(0, j-1)$, $j_1 \leftarrow \min(28, j+2)$
  $N \leftarrow I[i_0 : i_1, j_0 : j_1]$ {Extract neighborhood}
  $\sigma_N \leftarrow \text{std}(N)$ {Compute local std}
  $\epsilon \sim \mathcal{N}(0, s \cdot \sigma_N)$ {Sample noise}
  $I'[i, j] \leftarrow I[i, j] + \epsilon$
end for
return I'

F. Horizontal Flip

Horizontal flipping is a simple yet effective augmentation for Fashion-MNIST, as many clothing items are symmetric or have reasonable horizontal variants.

Algorithm 2 Horizontal Flip Augmentation

Require: Image $I \in \mathbb{R}^{28 \times 28}$
Ensure: Flipped image I_{flip}
for each row i in $\{0, \ldots, 27\}$ **do**
  for each column j in $\{0, \ldots, 27\}$ **do**
    $I_{\text{flip}}[i, j] \leftarrow I[i, 27 - j]$
  end for
end for
return I_{flip}

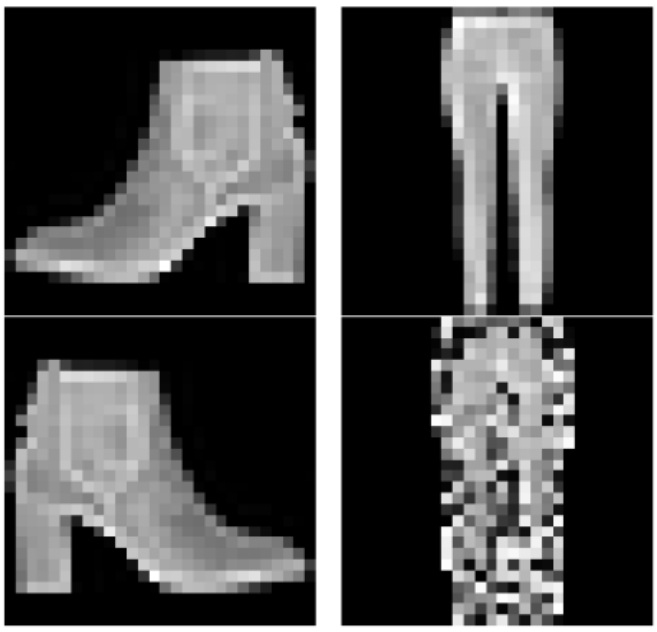

Fig. 9. Visual examples of the two augmentation strategies. Left: horizontal flip. Right: 3×3 neighborhood color jitter.

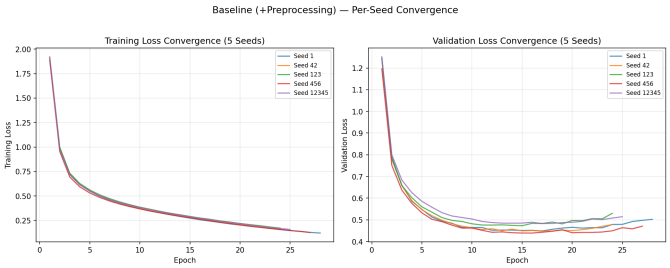

Fig. 10. Per-seed training and validation loss for the baseline without any enhancements on 5 seeds. This model exhibits shaky validation loss, and poor learning

REFERENCES

[1] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep. CMU-CALD-02-107, Carnegie Mellon University, 2002.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE ICCV*, 2015, pp. 1026–1034.

[3] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. COLT*, 1998, pp. 92–100.

[4] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd ed., John Wiley & Sons, 2007.