Assignment 2

Tutors: Yu Yao, Zhuo Huang, Jieru Liao
Group members:
Yuyang Cai, 530709729, ycai7924
Jialing Cai, 530338998, jcaj0620
Zhen Yang, 530549354, zyan0430
Mingyue Ma, 530400789, mima0482

Abstract

This study evaluates the impact of using transition matrices to improve the robustness of LeNet and ResNet models against label noise on Fashion-MNIST and CIFAR datasets. Transition matrices help mitigate the negative effects of label noise by accounting for potential mislabeling in training data, thereby enhancing model generalization and stability. We observed that the integration of transition matrices reduced test loss and prevented overfitting, particularly in noisy datasets. ResNet demonstrated higher accuracy but showed greater variability in performance, while LeNet displayed consistent stability. This research highlights the efficacy of noise-robust training strategies in deep learning applications, with an emphasis on the balance between model complexity and robustness.

1 Introduction

1.1 Problem Statement

In classification tasks, obtaining completely accurate and reliable labels for training data is often a significant challenge. Label noise, which refers to errors in the provided labels, is a common issue that arises from various sources, including insufficient information, human error, and subjective interpretation. Despite the prevalence of label noise, traditional classifiers typically assume that the training data is perfectly labeled, leading to potential problems when dealing with noisy labels.

1.2 Significance and Applications

Label noise can manifest in different forms, such as noise completely at random (NCAR), noise at random (NAR), and noise not at random (NNAR). NCAR occurs independently of both the true class and the feature values. In contrast, NAR depends on the true label, indicating that certain courses may be more prone to mislabeling [1]. NNAR, the most general form, occurs when the mislabeling depends on the feature values, such as errors near the classification boundaries. Each type of noise poses distinct challenges, potentially affecting the training process and reducing model accuracy.

The consequences of label noise are manifold: it can significantly deteriorate the prediction accuracy of classifiers, increase the number of training instances required, lead to more complex and overfitting models, and distort observed class frequencies [1]. Such adverse effects highlight the importance of effectively addressing label noise in machine learning models. Many studies have investigated methods to handle label noise, ranging from noise-robust models to data-cleansing techniques and noise-tolerant learning algorithms [2].

Effectively learning in the presence of label noise is imperative to improve model reliability, especially in high-stakes applications such as medical diagnosis or financial predictions. This research explores different types of label noise and reviews state-of-the-art approaches for mitigating its negative impact on classification tasks.

1.3 Overview of Methods

In this project, we focused on addressing the challenge of learning with label noise, a common issue in machine learning where training dataset labels are corrupted or incorrect. To mitigate the effects of label noise, we implemented a forward correction approach, using LeNet and ResNet as classifiers to apply this correction method effectively.

The forward correction method involved incorporating a known noise transition matrix to adjust the loss function. This matrix provided prior information about the probability of label corruption, enabling the models to correct the learning process and reduce the negative impact of noisy labels. By leveraging this correction mechanism, our classifiers could learn the true underlying patterns even in the presence of label noise [3].

We estimated the transition matrix for the CIFAR dataset to capture the noise rate of the labels. Using the estimated matrix, we trained both the LeNet and ResNet models, applying the forward correction method to correct the noisy labels and subsequently conducting training and validation based on these corrected labels. This process allowed our models to adapt to noisy data and maintain strong performance on a clean test set.

To enhance model robustness and reduce overfitting, we manually tuned the hyperparameters for both models. For each parameter, we selected three possible values and iteratively chose the best value as the starting point for the next round of tuning. This method allowed us to refine the models step by step and achieve optimal performance.

To ensure the robustness of our results, we ran each optimal configuration 10 times. This repeated execution helped confirm that our results were consistent and not due to random variation. Through this careful parameter tuning and repeated testing, we were able to achieve reliable and stable model performance, even in the presence of label noise.

2 Previous Work

2.1 Overview of Label Noise Methods

Classification with noisy labels has been a significant area of research, focusing on mitigating the impact of label corruption on model performance. Angluin and Laird [4] introduced the Random Classification Noise (RCN) model, which led to various noise adaptation techniques.

Natarajan et al. [5] proposed methods to adapt surrogate loss functions for classification under asymmetric label noise using unbiased estimators and label-dependent costs. Importance reweighting has also been applied to traditional models, adjusting the contributions of each training sample to mitigate label noise. Our work builds on this reweighting strategy, leveraging LeNet and ResNet architectures.

Recent advancements in deep learning have focused on noise-aware models, often employing the EM algorithm for noise rate estimation or kernel density estimation for consistency. Architectural modifications, such as dropout and batch normalization, have also proven effective against label noise. Our approach combines importance weighting with robust methods like CNN to enhance model resilience.

2.2 Advantages and Disadvantages of Traditional Method

This section discusses the advantages and disadvantages of the proposed methods and the importance of reweighting techniques for handling noisy labels[6][7].

2.2.1 Advantages

- **Versatility Across Architectures:** The reweighting approach is compatible with both LeNet and ResNet models, allowing for the application of various surrogate loss functions tailored to each architecture. This adaptability ensures that our methods remain effective across diverse noise conditions and model complexities.
- **Noise Robustness in Different Models:** By leveraging known noise rates or estimated noise transition matrices, both LeNet and ResNet can better mitigate the effects of label noise. LeNet's simpler architecture is less prone to overfitting noisy labels, while ResNet's deep architecture with residual connections enables it to learn complex patterns even in the presence of noise.
- **Consistency Assurance:** The importance reweighting strategy ensures consistency for both LeNet and ResNet models, meaning they converge to optimal solutions as the sample size increases, even with noisy labels. This property makes our approach reliable in scenarios where accurate labels are scarce or unavailable.
- **Flexibility in Noise Handling:** Our methods accommodate both symmetric and asymmetric label noise, making them suitable for real-world datasets where noise distribution is not uniform. This flexibility broadens the applicability of LeNet and ResNet models in handling various types of label noise effectively.

2.2.2 Disadvantages

- **Complexity in Noise Rate Estimation:** Estimating noise rates accurately is challenging and can be computationally expensive. Incorrect estimations can lead to suboptimal performance, especially for complex models like ResNet, which are sensitive to errors in noise handling due to their deep architectures.
- **Dependency on Noise Assumptions:** The effectiveness of our methods relies on accurate knowledge or estimation of noise rates. Inaccurate noise assumptions can significantly degrade the performance of both LeNet and ResNet models. Additionally, noise-specific estimation methods may struggle with high-dimensional data, leading to further inaccuracies.
- **Increased Training Complexity:** Incorporating importance weights and adjusting loss functions for noise introduces additional computational overhead during training. This added complexity can be problematic when dealing with large datasets or computationally intensive models like ResNet, potentially resulting in longer training times compared to non-corrected baselines.
- **Limited Effectiveness at High Noise Levels:** While our methods are effective under moderate noise, their performance tends to degrade when noise levels are very high. Both LeNet and ResNet may struggle to distinguish true signals from overwhelming noise, leading to reduced classification accuracy and reliability.

2.3 T-Revision

Recently, T-Revision [8] was proposed by Xia et al. as a novel approach for refining the transition matrix during training to handle label noise more effectively. This method is one of the latest advancements in the field and addresses issues of numerical instability and high computational complexity associated with direct inversion of the transition matrix. T-Revision builds upon the concept of risk-consistent estimators but introduces an iterative adjustment mechanism, enhancing the robustness and consistency of the learning process. The method consists of three main stages:

Stage 1: Initial Estimation of the Transition Matrix
In the first stage, the transition matrix is estimated using samples that approximate anchor points:

$$P(\bar{Y} = j \mid X = x) = \sum_{k=1}^{C} T_{kj} P(Y = k \mid X = x) = T_{ij}, \tag{1}$$

where $P(\bar{Y} = j \mid X = x)$ is the probability of observed label $\bar{Y}$ given input X, $P(Y = k \mid X = x)$ is the probability of the true label Y, C is the number of classes, and T_{ij} is the probability of label $Y = i$ being misclassified as $\bar{Y} = j$.

Stage 2: Refinement with a Correction Term
A correction term ΔT is introduced:

$$T' = T + \Delta T, \tag{2}$$

where T is the initial matrix, ΔT is the learned correction, and T' is the refined matrix.

Stage 3: Joint Learning of Classifier and Transition Matrix
The empirical risk is defined as:

$$\bar{R}_{n,w}(T, f) = \frac{1}{n} \sum_{i=1}^{n} \frac{g_{Y_i}(X_i)}{(T^T g)_{\bar{Y}_i}(X_i)} \ell(f(X_i), \bar{Y}_i), \tag{3}$$

where n is the sample size, $g_{Y_i}(X_i)$ is the posterior probability of the true label, $(T^T g)_{\bar{Y}_i}(X_i)$ is the weighted probability for $\bar{Y}_i$, and $\ell(f(X_i), \bar{Y}_i)$ is the loss function.

2.4 Advantages and Disadvantages of T-Revision

Advantages:

- **Dynamic Adjustment:** The method allows for continuous updates to T, making it adaptable to different noise conditions.
- **Improved Robustness:** The iterative refinement process enhances the learning stability and helps align the model more closely with clean data.

Disadvantages:

- **Implementation Complexity:** Learning and tuning ΔT add significant complexity to the training process.
- **Training Instability:** In practical experiments, the random nature of ΔT sometimes caused negative loss values, disrupting the training stability and performance.
- **Computational Overhead:** The iterative updates require additional computation, making the method resource-intensive, especially for large datasets.

2.5 Comparison with Traditional Methods

Compared to traditional label noise estimators, which often rely on directly inverting T and suffer from numerical instability, T-Revision avoids inversion and adapts during training. While forward correction methods rely on a fixed or pre-estimated T, T-Revision's iterative updates allow it to potentially better align with clean data over time. However, this advantage comes at the cost of increased complexity and potential training instability.

2.6 Conclusion and Practical Limitations

Despite its theoretical advantages, we chose not to use T-Revision in our primary approach. During our attempts to replicate the method, we observed that the training process was often unstable, with the correction term causing the loss to become negative at times. This behavior led to suboptimal model performance, making T-Revision less effective for our purposes. Consequently, we opted for more stable, well-understood methods that offered reliable performance.

3 Methods

3.1 Preprocess

In preprocessing, normalization was applied to the image data to facilitate faster and more stable model training. By rescaling pixel values to a range of $[-1, 1]$ with a mean and standard deviation of 0.5, the input data is standardized, helping to prevent issues such as vanishing or exploding gradients. This normalization step ensures consistent input distributions, allowing the model to converge more efficiently and effectively during training. In addition to the normalization step, a preprocessing transformation was applied to convert the image data to three channels. This transformation prepares the datasets, FashionMNIST0.3 and FashionMNIST0.6, for compatibility with models that require three-channel input, such as ResNet. By expanding the single-channel grayscale images into three identical channels, we ensure that the input data aligns with model architecture expectations without altering the original pixel information. This step is essential for maintaining compatibility across various deep learning models and contributes to more effective feature extraction during training.

3.2 LeNet5

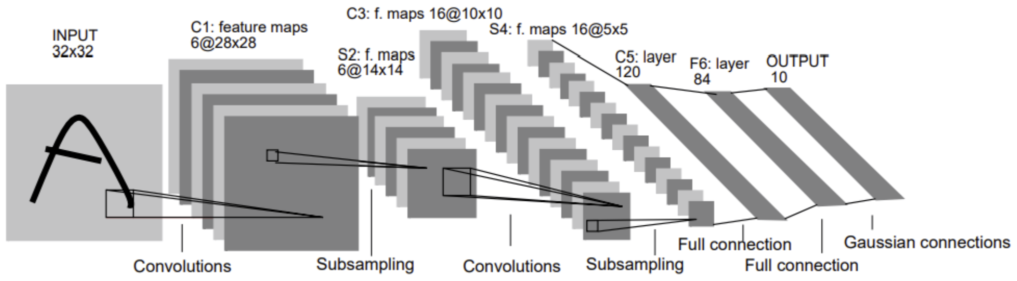


Figure 1: Architecture of LeNet a Convolutional Neural Network here for digits recognition Each plane is a feature map ie a set of units whose weights are constrained to be identical

Formulation: LeNet-5 is a Convolutional Neural Network (CNN) designed for character recognition, using multiple layers to progressively learn and extract hierarchical features from raw images. This network operates on 32×32 pixel grayscale images, allowing sufficient space to capture essential patterns while remaining efficient in computation. The architecture follows a sequential combination of convolution, subsampling (pooling), and fully connected layers, leading to a final output layer [9].

- **Input Layer (32x32 pixels):**
  The input is a grayscale image normalized to a 32×32 pixel size. Normalization is performed to maintain a standard input format, which helps in capturing consistent features across varying character sizes and positions.
- **First Convolutional Layer (C1):**
  The first layer applies six convolutional filters (kernels) of size 5×5 across the input, resulting in six feature maps of size 28×28. The convolution operation can be mathematically represented as:

$$C1(x, y) = \sum_{i=0}^{4} \sum_{j=0}^{4} I(x+i, y+j) \cdot K_{C1}(i, j) + b_{C1} \tag{4}$$

  where I is the input image, K_{C1} is the kernel (weights) for layer C1, and b_{C1} is the bias term. This operation allows the layer to detect simple patterns, like edges or corners.
- **First Subsampling Layer (S2):**
  After convolution, subsampling (or pooling) reduces the spatial resolution of each feature map to 14×14 by taking the average or maximum within non-overlapping 2×2 regions.

This step makes the representation robust to small distortions and shifts. The subsampling operation for each position can be written as:

$$S2(x, y) = \frac{1}{4} \sum_{i=0}^{1} \sum_{j=0}^{1} C1(2x + i, 2y + j) \tag{5}$$

The weights and biases for this layer are shared, reducing the number of parameters and enhancing the network's generalization capability.

- **Second Convolutional Layer (C3):**
  This layer consists of 16 feature maps of size 10×10, created by convolving different combinations of feature maps from the previous layer (S2). Each unit in a C3 feature map is connected to subsets of S2 feature maps, enabling it to learn more complex and distinctive patterns. The convolution in C3 can be expressed similarly to C1, but with a broader range of feature map combinations.
- **Second Subsampling Layer (S4):**
  Similar to S2, S4 reduces the feature maps from 10×10 to 5×5 through subsampling, further decreasing spatial dimensions while retaining crucial pattern information. This step helps in forming translation-invariant features by reducing positional sensitivity.
- **Third Convolutional Layer (C5):**
  Layer C5 consists of 120 feature maps, each connected to all 16 feature maps from S4 with a 5×5 kernel, resulting in a single unit per feature map (i.e., 120 units in total). Unlike earlier convolutional layers, C5 connects fully to S4, as its 5×5 receptive fields cover the entire 5×5 feature maps from S4. The output from C5 is represented as:

  $$C5(u) = \sum_{x=1}^{5} \sum_{y=1}^{5} S4(x, y) \cdot K_{C5}(x, y) + b_{C5} \tag{6}$$

  This layer captures global patterns across the entire image.
- **Fully Connected Layers (F6 and Output Layer):**
  The F6 layer contains 84 units fully connected to the 120 units in C5, followed by an output layer with 10 units representing the classification scores for each character class. The final output layer uses a Gaussian RBF (Radial Basis Function) to compute the Euclidean distance between input and target vectors, allowing the network to penalize misclassifications while encouraging accurate predictions.

Cost function: LeNet-5 employs a loss function based on the Minimum Mean Squared Error (MSE) between the network's output and the correct class labels. This cost function measures the network's error by comparing the output of the radial basis function (RBF) units in the output layer to the desired target values. RBF units in LeNet-5 are used as outputs to compute the Euclidean distances between feature vectors and pre-defined class templates, allowing the network to penalize incorrect classifications while reinforcing correct ones.

The MSE is defined as:

$$E(W) = \sum_{p=1}^{P} \left(y_{D_p} - \hat{y}_{D_p}\right)^2 \tag{7}$$

where:

- $E(W)$ is the loss over all training examples,
- P is the number of samples,
- y_{D_p} is the output for the correct class of the input pattern,
- $\hat{y}_{D_p}$ is the predicted class output for the input pattern.

To improve the model's discrimination power, the network also employs Maximum a Posteriori (MAP), a criterion enhancing class distinction by incorporating competition among classes. This competitive adjustment helps prevent "collapse" scenarios (where outputs become undifferentiated) by maximizing the likelihood of the correct class while minimizing the likelihood of incorrect classes.

3.3 Residual Neural Network (ResNet18)

ResNet is a deep convolutional neural network proposed by Kaiming He and his team in 2015. This network was specifically designed to address the challenges associated with training very deep networks, such as vanishing and exploding gradients. The architecture consists of 18 layers, including convolutional layers, pooling layers, and fully connected layers. By incorporating residual learning, the network can significantly increase the depth and efficiency of training while maintaining high-quality performance[10].

Residual Block One of the most significant innovations in ResNet is the introduction of residual learning, which is implemented through residual blocks within the network. Before the development of residual blocks, machine learning specialists had already recognized that increasing the depth of neural networks could greatly improve model performance. However, as the depth increased, problems such as vanishing and exploding gradients often occurred during backpropagation, making it difficult for the model to learn effectively[11].

To overcome these issues, ResNet introduces skip connections, also known as shortcut connections, which effectively bypass one or more layers, allowing a direct connection between the input and the output. This technique helps stabilize gradient flow during training, ensuring that the network remains trainable even as it becomes deeper[12].

The motivation behind the residual block is to simplify the learning process for deep neural networks. Instead of forcing each layer of the network to learn a complete mapping from inputs to outputs, the network learns the residual mapping, which is the difference between the desired output and the initial input. This approach greatly simplifies the learning task, allowing each layer to focus on making incremental changes necessary to reach the desired output. By reducing the complexity of what each layer must learn, the residual block makes deep networks more efficient and easier to train. The formula of this approach can be written as[13]:

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{8}$$

where x is the input to the residual block, $\mathcal{F}(x, \{W_i\})$ is the residual mapping learned by the block that represents the difference between the output and the input, and y is the final output of the residual block.

Structure and Architecture of ResNet18 In ResNet18, the network is built using multiple residual blocks. Each residual block is designed with two convolutional layers, followed by Batch Normalization and ReLU activation. Batch normalization normalizes the input to each layer, reducing bias and helping the model maintain consistent gradient values, which ultimately accelerates the training process. The ReLU activation function introduces non-linearity, allowing the model to learn complex features more effectively[10]. The architecture of ResNet18 is composed of five main stages:

- **Stage 1**: A 7x7 convolutional layer with 64 filters, followed by a 3x3 max pooling layer.
- **Stage 2**: 2 residual blocks, each containing 2 convolutional layers with 64 filters.
- **Stage 3**: 2 residual blocks, each containing 2 convolutional layers with 128 filters.
- **Stage 4**: 2 residual blocks, each containing 2 convolutional layers with 256 filters.
- **Stage 5**: 2 residual blocks, each containing 2 convolutional layers with 512 filters.

After passing through all five stages, the network features a global average pooling layer that replaces the fully connected layers typically used in traditional architectures. This reduces the risk of overfitting by significantly lowering the number of parameters in the model. The output from the global average pooling layer is then passed to a fully connected layer to produce the final output classification[10].
The introduction of ResNet made efficient training of very deep neural networks possible, addressing key issues such as gradient vanishing and gradient explosion. Unlike traditional deep neural

networks, which often suffer from degraded performance as their depth increases, ResNet's residual connections ensure that deeper networks can be trained effectively without negatively impacting model convergence and performance. This allows the addition of more layers while maintaining or improving the accuracy of the network[10].

3.4 Forward Correction

Forward Correction is a method used in label-noise learning to ensure that the training process remains reliable even with noisy data. The main idea of forward correction is to adjust the transition matrix to better approximate the true transition matrix, allowing the model to learn effectively from noisy data. This adjustment works by closely approximating the expected risk for clean data. However, traditional forward correction approaches rely on the use of the inverse of the transition matrix, which can negatively impact classification performance and poses challenges as not all matrices are invertible[8].

Core Concept
The objective of forward correction is to modify the training procedure so that the model can achieve performance similar to training on a clean dataset, even when learning from noisy data. Let X represent the feature space and Y represent the true labels. Due to label noise, the observed labels are denoted as $\bar{Y}$. The noisy class posterior probability is $P(\bar{Y} \mid X = x)$, while the clean class posterior probability is $P(Y \mid X = x)$[8].

The relationship between the clean and noisy posterior probabilities can be expressed using the noise transition matrix T:

$$P(Y \mid X = x) = (T^{\top})^{-1} P(\bar{Y} \mid X = x) \tag{9}$$

This equation shows that the clean label posterior can be inferred from the noisy label posterior by inverting the transposed transition matrix. However, directly inverting T can lead to numerical instability, especially if the matrix is poorly conditioned[8].

To address this, forward correction adjusts the risk estimation by applying an importance reweighting technique. The expected risk with respect to the clean data can be expressed as:

$$R(f) = \mathbb{E}_{(X,Y)\sim D}[\ell(f(X), Y)] = \mathbb{E}_{(X,\bar{Y})\sim \bar{D}}[\tilde{\ell}(f(X), \bar{Y})], \tag{10}$$

where f is the classifier, $\ell(f(X), Y)$ is the loss function for clean data, and $\tilde{\ell}(f(X), \bar{Y})$ is the adjusted loss for noisy data. D is the distribution of clean data, and $\bar{D}$ is the distribution of noisy data. Since clean labels Y are not accessible, they are replaced by noisy labels $\bar{Y}$ during training[8].

3.5 Transition Matrix Estimation

A method for estimating the transition matrix in datasets with label noise is presented, leveraging a model's predicted probabilities at its optimal validation performance. By extracting these probabilities, high-confidence samples are identified—instances where the model assigns a probability exceeding a predefined threshold δ to a particular class. Assuming these high-confidence predictions correspond to the true labels, the distribution of their observed (noisy) labels is statistically analyzed to estimate the transition probabilities from true labels to observed labels.

Specifically, the transition probability T_{ij} from true class i to observed class j is estimated using[8]:

$$T_{ij} = \frac{N_{ij}}{\sum_k N_{ik}},$$

where N_{ij} is the number of high-confidence samples predicted as class i but observed as class j, and $\sum_k N_{ik}$ is the total number of high-confidence samples predicted as class i[14].

This estimation allows for the construction of the transition matrix T, effectively capturing the label noise patterns within the dataset. By integrating the estimated transition matrix into the training process, the learning algorithm is adjusted to account for label noise, thereby enhancing the model's robustness and improving its performance in noisy environments[14].

The approach requires no additional manual annotation or prior knowledge about the noise distribution, making it practical for real-world applications where label noise is prevalent. Experimental results demonstrate that this method effectively models label noise and significantly improves classification performance compared to baseline methods[14].

3.6 Evaluation Metric

To assess the performance of each classifier, we use the top-1 accuracy metric, which is defined as the proportion of correctly classified examples in the test set. Formally, top-1 accuracy is given by:

$$\text{Top-1 Accuracy} = \frac{\text{Number of correctly classified examples}}{\text{Total number of test examples}} \times 100\%. \tag{11}$$

This metric evaluates the model's ability to identify the most probable class label and is a standard measure of classification performance.

To ensure a robust and unbiased evaluation, each classifier is trained at least 10 times using different training and validation splits. These splits are generated through random sampling to account for variability in model performance due to data partitioning. The final evaluation involves reporting both the mean and standard deviation of the top-1 accuracy across these multiple runs. This approach provides insights into the model's consistency and generalization ability, allowing us to assess not only the average performance but also the stability of the classifier under varying conditions.

4 FashionMNIST dataset

Estimated Transition Matrix on FashionMINIST03

The provided true transition matrix for the **FashionMINIST03 dataset** is:

$$\begin{bmatrix} 0.7000 & 0.3000 & 0.0000 & 0.0000 \\ 0.0000 & 0.7000 & 0.3000 & 0.0000 \\ 0.0000 & 0.0000 & 0.7000 & 0.3000 \\ 0.3000 & 0.0000 & 0.0000 & 0.7000 \end{bmatrix}$$

The estimated transition matrix generated using LeNet for the **FashionMINIST03 dataset** is:

$$\begin{bmatrix} 7.5311 \times 10^{-1} & 2.0919 \times 10^{-1} & 2.0378 \times 10^{-3} & 3.5662 \times 10^{-2} \\ 2.3307 \times 10^{-4} & 6.6382 \times 10^{-1} & 3.3581 \times 10^{-1} & 1.3554 \times 10^{-4} \\ 1.0062 \times 10^{-2} & 6.8239 \times 10^{-3} & 7.4295 \times 10^{-1} & 2.4017 \times 10^{-1} \\ 3.4345 \times 10^{-1} & 7.1665 \times 10^{-3} & 2.1649 \times 10^{-3} & 6.4722 \times 10^{-1} \end{bmatrix}$$

The estimated transition matrix generated using ResNet for the **FashionMINIST03 dataset** is:

$$\begin{bmatrix} 8.3838 \times 10^{-1} & 1.6105 \times 10^{-1} & 4.6483 \times 10^{-4} & 1.0962 \times 10^{-4} \\ 1.9440 \times 10^{-4} & 7.7473 \times 10^{-1} & 2.2463 \times 10^{-1} & 4.4676 \times 10^{-4} \\ 4.3864 \times 10^{-3} & 2.7865 \times 10^{-3} & 8.9112 \times 10^{-1} & 1.0170 \times 10^{-1} \\ 2.5957 \times 10^{-2} & 4.7102 \times 10^{-4} & 2.9587 \times 10^{-5} & 9.7354 \times 10^{-1} \end{bmatrix}$$

For both models (LeNet and ResNet), using the estimated transition matrix yielded competitive results close to the accuracies achieved using the true matrix.For LeNet_03, the model using the estimated matrix achieved a mean test accuracy of 95.03%, which is 0.43% lower than using the true matrix (95.46%). The accuracy reduction here is minimal, suggesting that the estimation closely approximated the true noise distribution. In the case of ResNet_03, the mean test accuracy was 94.76% with the estimated matrix, slightly higher than the 94.67% achieved with the true matrix. This marginal difference implies that the estimated transition matrix was quite effective for noise-robust learning in this instance.And the standard deviation using the estimated matrix was slightly

Model	Transition Matrix	Mean Test Accuracy (%)	STD of Test Accuracy (%)
LeNet_03	Estimated	95.03	0.14
	True	95.46	0.14
ResNet_03	Estimated	94.76	0.46
	True	94.67	0.49

Table 1: Performance comparison between estimated and true transition matrices for LeNet_03 and ResNet_03.

lower (0.46%) compared to using the true matrix (0.49%). Although this reduction is small, it indicates improved consistency in model performance when using the estimated matrix.

The estimated transition matrix successfully captured the general noise patterns in the dataset for both models. This led to consistent model performance with minimal variations in test accuracy.The slight drop in mean test accuracy for LeNet when using the estimated matrix shows that the approximation of noise patterns is reasonably accurate but not perfect. On the other hand, ResNet performed marginally better with the estimated matrix, indicating that it may have benefited

Estimated Transition Matrix on FashionMINIST06

The provided true transition matrix for the **FashionMINIST06 dataset** is:

$$\begin{bmatrix} 0.4000 & 0.2000 & 0.2000 & 0.2000 \\ 0.2000 & 0.4000 & 0.2000 & 0.2000 \\ 0.2000 & 0.2000 & 0.4000 & 0.2000 \\ 0.2000 & 0.2000 & 0.2000 & 0.4000 \end{bmatrix}$$

The estimated transition matrix generated using LeNet for the **FashionMINIST06 dataset** is:

$$\begin{bmatrix} 0.44887977 & 0.15679256 & 0.1897802 & 0.20454747 \\ 0.22160741 & 0.41430054 & 0.17465689 & 0.18943517 \\ 0.22087129 & 0.19078401 & 0.38260282 & 0.20574187 \\ 0.21808888 & 0.15406125 & 0.21171238 & 0.41613749 \end{bmatrix}$$

The estimated transition matrix generated using ResNet for the **FashionMINIST03 dataset** is:

$$\begin{bmatrix} 9.99810158 \times 10^{-1} & 1.15036397 \times 10^{-4} & 5.56557169 \times 10^{-5} & 1.91500348 \times 10^{-5} \\ 3.41617518 \times 10^{-6} & 9.99875397 \times 10^{-1} & 1.81283210 \times 10^{-5} & 1.03058914 \times 10^{-4} \\ 7.09591202 \times 10^{-5} & 3.09953837 \times 10^{-6} & 9.99925653 \times 10^{-1} & 2.88827016 \times 10^{-7} \\ 6.96933115 \times 10^{-6} & 2.54745634 \times 10^{-6} & 3.46373911 \times 10^{-5} & 9.99955846 \times 10^{-1} \end{bmatrix}$$

When using the estimated transition matrix for `ResNet_06`, there is a significant decrease in av-

Model	Transition Matrix	Mean Test Accuracy (%)	STD of Test Accuracy (%)
ResNet_06	Estimated	89.70	2.00
	True	90.86	0.85
LeNet_06	Estimated	90.67	0.96
	True	90.63	0.82

Table 2: Performance comparison between estimated and true transition matrices for ResNet_06 and LeNet_06 on Dataset 06.

erage accuracy, and the difference value between each training session increases significantly. This indicates that the training is not very stable, as ResNet is more sensitive to inaccuracies in transition matrix estimation. At the same time, `LeNet_06` exhibits stronger resilience to inaccuracies in the estimation matrix. The small difference in average accuracy and standard deviation obtained during each training indicates that `LeNet_06` can effectively utilize the estimation matrix without significantly reducing performance or stability.

Table 10 shows that although both models achieve good accuracy and standard deviation in the estimated transition matrix, `LeNet_06` demonstrates better consistency and stability in handling noise approximations. `ResNet_06` shows significant fluctuations, and stable and accurate training results can be achieved by improving estimation techniques or replacing noise-robust strategies in the future.

4.1 Hypermeter Tuning

For each of the models (LeNet_03, resNet_03, LeNet_06, resNet_06), we conducted hyperparameter tuning to determine the optimal settings.his involved systematically testing different combinations of key hyperparameters, such as the optimizer, learning rate, batch size, and the number of epochs.The summary of the best combinations for each model is as follows:

Model	**Optimizer**	**Learning Rate**	**Batch Size**	**Number of Epochs**
LeNet_03	Adam	0.001	64	15
resNet_03	Adam	0.001	64	5
LeNet_06	Adam	0.001	64	5
resNet_06	Adam	0.001	64	15

Table 3: Training Parameters Summary

According to Table 1, in the selection of optimizers, **Adam** can achieve faster convergence and better stability. Its performance across different models outperformed that of SGD. The range we selected for the learning rate is $\{0.1, 0.01, 0.001, 0.0001\}$. A learning rate of **0.001** is the most balanced choice among all models. A higher learning rate can lead to unstable training and overfitting problems, while a lower learning rate can result in slower training speed and no significant improvement in accuracy.

For the batch size, we chose $\{32, 64, 128\}$ and ultimately obtained **64** as the optimal balance point between training efficiency and accuracy. Smaller batch sizes exhibit noisy gradients, while larger batch sizes increase the risk of overfitting. For the number of epochs, we chose $\{5, 10, 15\}$ and the shorter training duration (e.g., 5 epochs) was effective for smaller models like `resNet_03` and `LeNet_06`, but larger models required longer training durations (e.g., 15 epochs) to fully capture the data's complexity.

4.2 Comparison Experiment

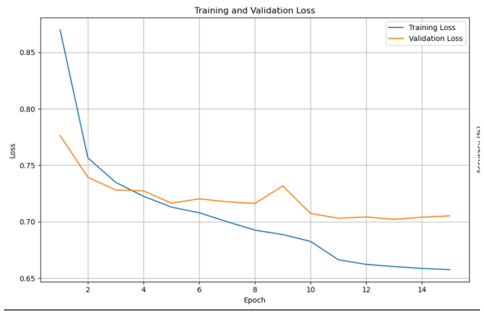
LeNet(FashionMNIST03)

The LeNet model on the Fashion-MNIST 0.3 dataset was evaluated with and without the use of a transition matrix for noise-robust training. The comparison focuses on how the transition matrix impacts the training, validation, and test performance of the model.

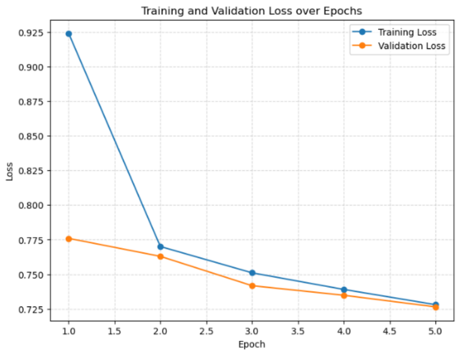
We applied forward correction and directly used the model for classification. The accuracy achieved was 95.17%, indicating an improvement in the model's generalization capabilities. The training and validation losses in the provided graphs demonstrate a consistent decrease in training loss, with the validation loss stabilizing after a few epochs. The training accuracy steadily increases, while the validation accuracy remains relatively consistent, with minor fluctuations. This pattern indicates that the transition matrix helps mitigate overfitting by learning to be robust against noisy labels.

The use of the transition matrix effectively improves the model's robustness against label noise by accounting for potential mislabeling in the training dataset. The transition matrix aids in regularizing the model's predictions, leading to A reduction in test loss (from 0.5131 to 0.4441) indicates better generalization and stability in the model's predictions. As observed in the graphs[figure 2], the transition matrix helped in stabilizing the validation loss, preventing drastic fluctuations that might indicate overfitting.

Incorporating a transition matrix for noise-robust training proved to be beneficial for the LeNet model on the Fashion-MNIST 0.3 dataset. It not only resulted in a significant reduction in test loss but also led to improved accuracy. The transition matrix helped the model effectively learn under the



(a) Forward



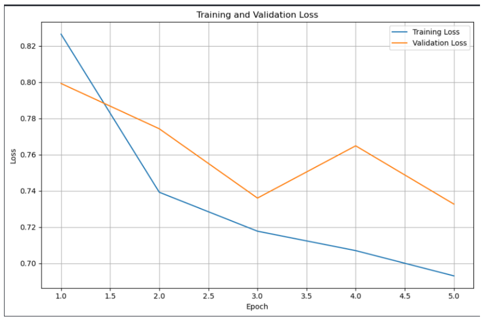
(b) Base

Figure 2: Comparison of Loss Curves: Forward Correction vs. Direct Classification for LeNet on Fashion-MNIST 0.3.

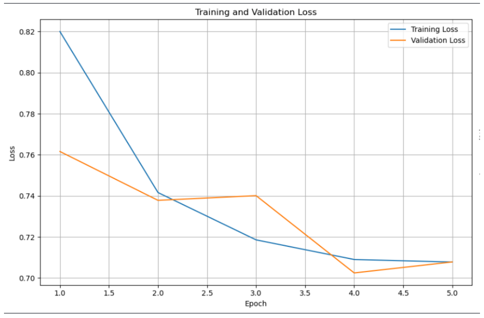
presence of label noise, enhancing its robustness and generalization ability. This highlights the importance of noise-robust strategies in training deep learning models on datasets prone to mislabeling or noisy annotations.

ResNet(FashionMNIST03)

The ResNet model on the Fashion-MNIST 0.3 dataset was evaluated with and without the use of a transition matrix for noise-robust training.



(a) Forward



(b) Base

Figure 3: Comparison of Loss Curves: Forward Correction vs. Direct Classification for Resnet on Fashion-MNIST 0.3.

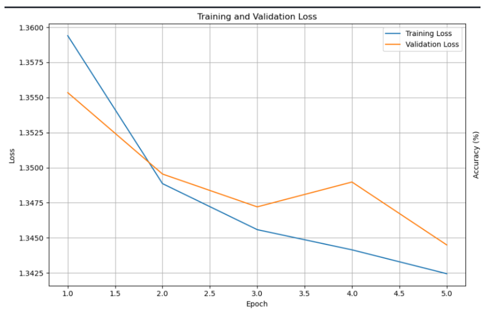
The accuracy achieved was 95.17%, indicating an improvement in the model's generalization capabilities.The training and validation losses in the provided graphs demonstrate a consistent decrease in training loss, with the validation loss stabilizing after a few epochs. The training accuracy steadily increases, while the validation accuracy remains relatively consistent, with minor fluctuations. This pattern indicates that the transition matrix helps mitigate overfitting by learning to be robust against noisy labels.

The test accuracy was 92.80%, showing a lower generalization ability compared to the results achieved with the transition matrix.the model appears to overfit the training data. The higher test loss and lower accuracy indicate that the model could not adequately handle noisy labels, leading to poorer generalization on unseen data.

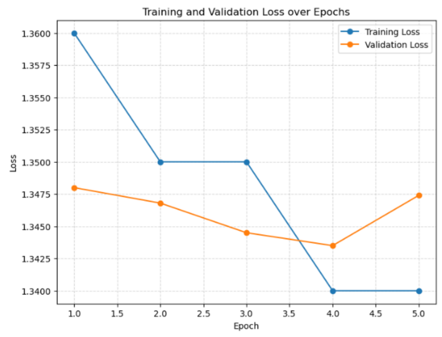
Incorporating a transition matrix for noise-robust training proved to be beneficial for the ResNet model on the Fashion-MNIST 0.3 dataset. It not only resulted in a significant reduction in test loss but also led to improved accuracy. The transition matrix helped the model effectively learn under the presence of label noise, enhancing its robustness and generalization ability. This highlights the importance of noise-robust strategies in training deep learning models on datasets prone to mislabeling or noisy annotations.

LeNet(FashionMNIST06)

The LeNet model on the Fashion-MNIST 0.6 dataset was evaluated with and without the use of a transition matrix for noise-robust training. The comparison focuses on how the transition matrix impacts the training, validation, and test performance of the model.



(a) Forward



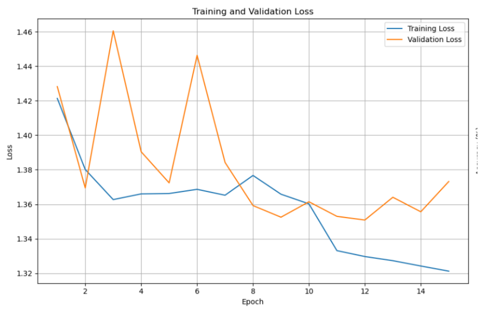
(b) Base

Figure 4: Comparison of Loss Curves: Forward Correction vs. Direct Classification for Lenet on Fashion-MNIST 0.6.

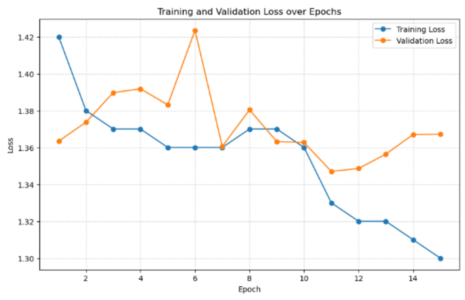
The accuracy achieved was 90.50%, The training loss consistently decreased over the epochs, while the validation loss also demonstrated a decreasing trend, indicating good convergence. The training accuracy improved steadily, while the validation accuracy showed some fluctuations, indicating that the model was learning under noisy conditions.

The test accuracy achieved was 89.78%, which is lower than when using the transition matrix.The model without the transition matrix showed signs of overfitting, as indicated by the higher test loss. The training and validation accuracies diverged, indicating that the model had difficulty handling the label noise without additional robustness strategies.

ResNet(FashionMNIST06)



(a) Forward



(b) Base

Figure 5: Comparison of Loss Curves: Forward Correction vs. Direct Classification for Resnet on Fashion-MNIST 0.6.

For Resnet on Fashion-MNIST 0.6 The model reached an accuracy of 86.00%, indicating a better capability to generalize to unseen data.The training loss decreased consistently, while the validation loss showed some oscillations, reflecting the challenges posed by label noise. Despite the fluctuations, the use of the transition matrix stabilized the model's training process and mitigated the effects of noise on validation performance.

The model achieved an accuracy of 81.90% without the transition matrix.The model without the transition matrix demonstrated significant overfitting. The validation accuracy failed to match the training accuracy, and the higher test loss suggests that the model struggled with noisy labels, resulting in degraded generalization performance.

4.3 Top-1 accuracy metric

The table below presents the top-1 accuracy standard deviation of test accuracy for each model, based on a minimum of 10 training and validation runs with random sampling:

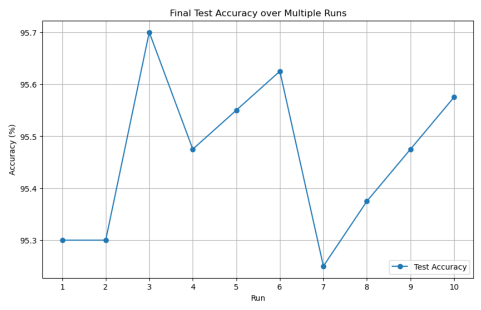
Model	Top-1 Accuracy (%)	STD of Test Accuracy (%)
LeNet_03	95.46	0.14
ResNet_03	94.67	0.49
LeNet_06	90.63	0.82
ResNet_06	90.86	0.85

Table 4: FashionMINIST Classifier Evaluation Summary

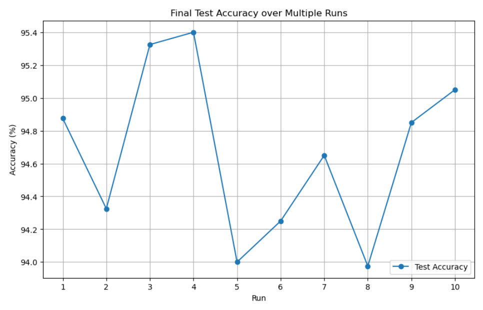
For the 03 dataset, Lenet achieved a mean accuracy of 95.70%, outperforming ResNet in the same dataset, indicating that it correctly classified most of the test examples. The low standard deviation of 0.14 also suggests that the performance of the model is consistent across different runs. Although ResNet's mean accuracy of 95.33% is strong, it is slightly less stable compared to Lenet.

For the 06 dataset, Lenet's Top-1 accuracy is 90.63% with a standard deviation of 0.82, showing that the model's performance varies more, indicating greater sensitivity to different training and validation sets. ResNet performs slightly better than Lenet, with a Top-1 accuracy of 90.86%, but a standard deviation of 0.85 still indicates that the results of the model vary across different runs.

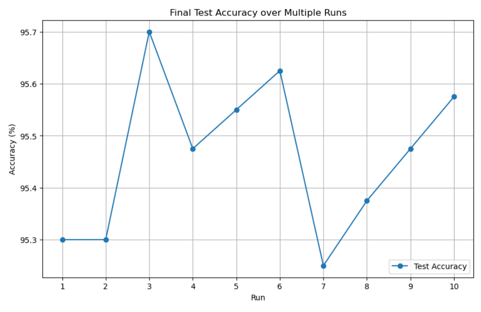
Overall, Lenet and ResNet showed the best results on different datasets, with Lenet demonstrating good stability on the 03 dataset and ResNet showing higher variability on the 06 dataset.



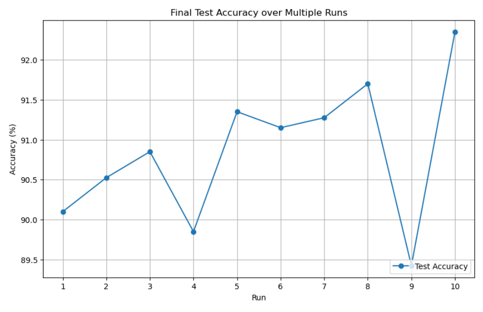
(a) LeNet_03 Performance



(b) ResNet_03 Performance



(c) LeNet_06 Performance



(d) ResNet_06 Performance

Figure 6: 10 times Accuracy

This plot (a) shows the test accuracy of the LeNet_03 model over 10 different runs. The model demonstrated a high level of stability with minimal variation in accuracy, showing reliable and consistent learning behavior. It showed the most stable performance, with minimal fluctuation in test accuracy across multiple runs. This plot (b) shows the test accuracy of the ResNet_03 model across 10 different runs. The ResNet_03 model shows more variability compared to LeNet_03, indicating greater sensitivity to the training set variations or random initialization. This plot shows the test accuracy of the LeNet_06 model over 10 different runs. This plot (c) shows the test accuracy of the LeNet_06 model over 10 different runs. The model demonstrates a relatively stable trend after the initial decline in accuracy. The plot (d) indicates the test accuracy of the ResNet_06 model over 10 runs. The plot shows sharp dips and recoveries, highlighting significant sensitivity in some training runs. Both LeNet_06 and ResNet_06 demonstrated more pronounced fluctuations in accuracy, indicating the increased difficulty in handling noise as the models' complexities increased.

5 CIFAR dataset

5.1 Estimated Transition Matrix

The estimated transition matrix generated using LeNet for the **CIFAR dataset** is:

$$\begin{bmatrix} 0.74994078 & 0.1864813 & 0.05166683 & 0.01191709 \\ 0.01745607 & 0.99079986 & 0.07098568 & 0.00175838 \\ 0.0575697 & 0.0476007 & 0.74604145 & 0.14878815 \\ 0.07944772 & 0.01662176 & 0.2180014 & 0.68542912 \end{bmatrix}$$

The estimated transition matrix generated using LeNet on the CIFAR dataset reveals the noise distribution for each class. The diagonal elements are relatively large, indicating that for most samples, the predicted label matches the true label with high probability. For instance, the probability of a true class 0 sample being correctly observed as class 0 is approximately 74.99%. However, there are significant off-diagonal elements, such as the probability of a true class 2 sample being mislabeled as class 3, which is 14.88%. These off-diagonal values indicate the presence of label noise, which varies across different classes. Overall, LeNet struggles with higher noise levels in certain classes, suggesting limited ability to capture distinct features for all categories.

The estimated transition matrix generated using ResNet for the **CIFAR dataset** is:

$$\begin{bmatrix} 8.23900316\text{e-}01 & 1.62174667\text{e-}01 & 1.31734168\text{e-}02 & 7.51600461\text{e-}04 \\ 5.98188110\text{e-}02 & 8.88095559\text{e-}01 & 5.78787821\text{e-}02 & 1.40680228\text{e-}03 \\ 6.94950437\text{e-}02 & 1.25053465\text{e-}03 & 8.16127944\text{e-}01 & 1.13126477\text{e-}01 \\ 9.77456563\text{e-}02 & 3.77538751\text{e-}03 & 9.43807133\text{e-}02 & 8.04098243\text{e-}01 \end{bmatrix}$$

The estimated transition matrix generated using ResNet on the CIFAR dataset demonstrates a higher robustness to label noise compared to LeNet. The diagonal elements of the matrix are generally higher, such as the probability of correctly labeling a true class 1 sample as class 1, which reaches 88.81%. This suggests that ResNet is more effective in retaining label consistency. While there are still noticeable off-diagonal elements—such as the probability of a true class 2 sample being mislabeled as class 3 (11.31%)—the overall noise levels appear lower than in the LeNet matrix. This indicates that ResNet has a stronger ability to differentiate among classes, thus reducing label noise and enhancing prediction accuracy.

The comparison indicates that ResNet handles label noise more effectively than LeNet, with higher diagonal values showing better label retention and reduced misclassification. ResNet's robustness is likely due to its ability to capture more complex features, maintaining higher label consistency.

5.2 Hypermeter Tuning

In this section, a systematic exploration of the hyperparameter optimization for the LeNet and ResNet models on the CIFAR dataset. Effective hyperparameter tuning is essential for achieving optimal model performance, as it significantly influences the convergence behavior and generalization ability of neural networks. The key hyperpaprmeters including: optimizer, learning rate, batch size, and number of epochs. Each hyperparameter was adjusted independently while keeping other settings constant to isolate its specific impact on model performance. The goal was to identify the configuration that yields the highest accuracy across both models. The results are summarized in Tables 2, 3, 4 and 5, followed by a detailed analysis of each parameter's effect on model accuracy.

Optimizer

Classifier	Num Epochs	Batch Size	Optimizer	Learning Rate	Accuracy (%)
LeNet	5	64	SGD	0.01	74.95
LeNet	5	64	Adam	0.001	75.85
ResNet	5	64	SGD	0.01	75.22
ResNet	5	64	Adam	0.001	80.97

Table 5: Hyperparameter Tuning on Optimizer (CIFAR)

Results indicate that the Adam optimizer generally outperforms SGD across both models, LeNet and ResNet. For instance, with LeNet, the accuracy improved from 74.95% with SGD to 75.85% with Adam. Similarly, for ResNet, Adam achieved an accuracy of 80.97% compared to 75.22% with SGD. This suggests that the adaptive learning rate of Adam provides an advantage over the fixed learning rate of SGD for this dataset and training setup.

A learning rate of 0.01 is used for SGD, while 0.001 is used for Adam, as these are commonly effective learning rates for each optimizer.

Learning Rate

Classifier	Num Epochs	Batch Size	Optimizer	Learning Rate	Accuracy (%)
LeNet	5	64	Adam	0.0005	74.30
LeNet	5	64	Adam	0.001	75.85
LeNet	5	64	Adam	0.01	60.27
ResNet	5	64	Adam	0.0005	80.12
ResNet	5	64	Adam	0.0001	77.12
ResNet	5	64	Adam	0.001	80.97

Table 6: Hyperparameter Tuning on Learning Rate (CIFAR)

For LeNet, a learning rate of 0.001 performed the best, achieving an accuracy of 75.85%. A smaller learning rate of 0.0005 led to a slight drop in accuracy to 74.30%, while a larger rate of 0.01 significantly degraded performance, yielding an accuracy of only 60.27ResNet also demonstrated a similar pattern, with the best accuracy (80.97%) at a learning rate of 0.001. Learning rates that were too high caused unstable training, whereas very low values slowed convergence without significant accuracy gains.

Batch Size

Classifier	Num Epochs	Batch Size	Optimizer	Learning Rate	Accuracy (%)
LeNet	5	32	Adam	0.001	75.35
LeNet	5	64	Adam	0.001	75.85
LeNet	5	128	Adam	0.001	75.95
ResNet	5	32	Adam	0.001	79.00
ResNet	5	64	Adam	0.001	80.97
ResNet	5	128	Adam	0.001	79.60

Table 7: Hyperparameter Tuning on Batch Size (CIFAR)

For LeNet, a batch size of 64 yielded the best results, with an accuracy of 75.85%. Increasing the batch size to 128 slightly reduced accuracy to 75.0%, while reducing it to 32 also led to suboptimal performance.

For ResNet, a batch size of 64 again provided the highest accuracy (80.97%). Both smaller and larger batch sizes (32 and 128) resulted in slight drops in performance, suggesting that 64 strikes the best balance between training stability and speed.

Number of Epochs

Classifier	Num Epochs	Batch Size	Optimizer	Learning Rate	Accuracy (%)
LeNet	5	128	Adam	0.001	75.95
LeNet	10	128	Adam	0.001	77.30
LeNet	15	128	Adam	0.001	76.80
ResNet	5	64	Adam	0.001	80.97
ResNet	10	64	Adam	0.001	81.05
ResNet	15	64	Adam	0.001	77.15

Table 8: Hyperparameter Tuning on Number of Epochs (CIFAR)

For LeNet, increasing the number of epochs from 5 to 15 led to a steady increase in accuracy, from 75.95% to 76.80%. This indicates that LeNet benefited from additional training iterations.

In the case of ResNet, however, the accuracy peaked at 81.05% with 10 epochs, but decreased to 77.15% when further increasing to 15 epochs. This suggests overfitting occurred with additional training, where the model started to learn noise rather than useful patterns.

5.3 Comparison Experiment

This comparison experiment aims to evaluate the impact of using forward correction versus the base classification mode on model performance.The base mode refers to the standard classification approach, where the model directly performs classification without applying any correction methods. The forward mode involves using forward correction, a technique that uses a transition matrix to correct for noisy labels before performing classification.

The results indicate that the effect of forward correction depends on the model architecture. ResNet appears to benefit more from this correction technique compared to LeNet, highlighting the importance of selecting appropriate noise correction strategies based on the model and dataset characteristics.

Model	Epochs	Batch Size	Optimizer	Learning Rate	Mode	Accuracy (%)
LeNet	10	128	Adam	0.001	forward	77.15
					base	76.00
ResNet	10	128	Adam	0.001	forward	80.40
					base	79.38

Table 9: Base and Best Model Performance Comparison (CIFAR)

For LeNet, the forward correction mode achieved an accuracy of 77.15%, which is slightly higher than the base mode accuracy of 76.00%. This suggests that forward correction provides a marginal benefit for LeNet under these conditions, potentially overcoming minor label noise but without a significant improvement over the base model.

For ResNet, the forward correction mode achieved an accuracy of 80.40%, which is slightly higher than the base mode accuracy of 79.38%. This indicates that forward correction can enhance the classification performance of ResNet, likely by mitigating the negative effects of noisy labels during training more effectively compared to the simpler LeNet model.

LeNet

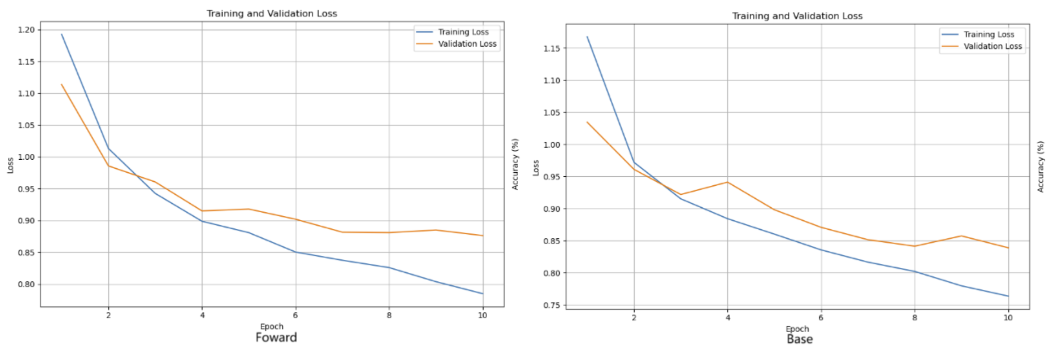


Figure 7: Comparison of Loss Curves: Forward Correction vs. Direct Classification for LeNet on Noisy Labels

The provided plots illustrate the training and validation loss curves for the LeNet model under two different conditions: the left plot shows the loss when applying Forward Correction to handle label noise, while the right plot shows the loss without any noise correction (i.e., direct classification on the original dataset). The Forward Correction method results in a more stable and consistent reduction in both training and validation loss, indicating improved robustness and better generalization performance. In contrast, the model trained without noise correction exhibits higher fluctuations in validation loss, particularly towards the latter epochs, suggesting susceptibility to overfitting and difficulty in handling noisy labels effectively. This comparison highlights the benefit of Forward Correction in reducing the negative impact of label noise and improving overall model stability.

ResNet

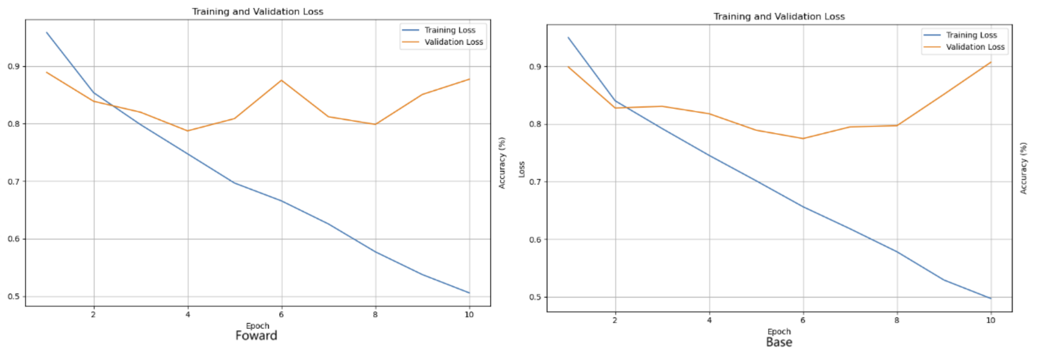


Figure 8: Comparison of Loss Curves: Forward Correction vs. Direct Classification for ResNet on Noisy Labels

The provided plots illustrate the training and validation loss curves for the ResNet model under two conditions: the left plot applies Forward Correction for label noise, whereas the right plot shows the model trained directly on the original dataset without any noise correction. The Forward Correction approach results in a consistent decrease in training loss but exhibits fluctuating validation loss, indicating challenges in fully mitigating label noise effects during validation. In contrast, the direct classification approach leads to a steadily decreasing training loss, but the validation loss increases consistently after the initial epochs, suggesting significant overfitting due to label noise. Overall, Forward Correction provides some stability compared to the baseline, but the effectiveness in reducing noise impact on validation data remains limited.

5.4 Top-1 accuracy metric

To address the variability inherent in training deep learning models, both LeNet and ResNet were trained and tested 10 times, and the results were averaged. This approach aims to provide a more reliable estimate of the model performance and reduce the impact of randomness.

The Mean Test Accuracy represents the average accuracy across all 10 runs.

Model	Mean Test Accuracy (%)	STD of Test Accuracy (%)
LeNet	75.68	0.65
ResNet	79.41	1.31

Table 10: CIFAR Classifier Evaluation Summary (Training for 10 times)

For LeNet, the mean accuracy over 10 runs was 75.68%, with a standard deviation (STD) of 0.65%. This low standard deviation indicates consistent performance across multiple runs, suggesting the model is stable.

For ResNet, the mean accuracy was 79.41%, with a higher standard deviation of 1.31%. The slightly larger standard deviation compared to LeNet suggests that ResNet has more variability across different runs, which might be due to its higher complexity and sensitivity to initialization.

These results indicate that ResNet generally outperforms LeNet on the CIFAR dataset in terms of mean accuracy. However, the higher variability also suggests that careful tuning and possibly more advanced training strategies may be necessary to consistently achieve optimal results with ResNet.

The following line plots illustrate the test accuracy over 10 runs for both LeNet and ResNet models. These plots provide a visual representation of the variability in model performance during repeated training sessions. The line plot for LeNet(Figure 7) shows moderate fluctuations in accuracy across

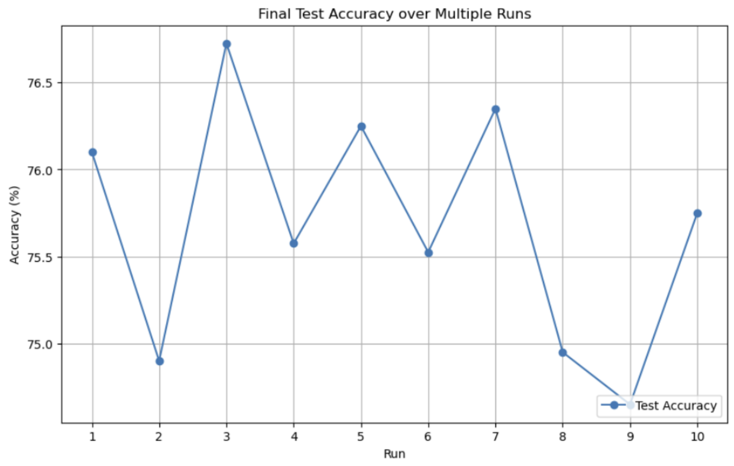


Figure 9: Final Test Accuracy over Multiple Runs - LeNet(CIFAR)

the 10 runs, with accuracy values ranging between 75.0% and 76.72%. This relatively consistent performance aligns with the low standard deviation observed, confirming the stability of LeNet's training results. The line plot for ResNet(Figure 8) shows more pronounced variability, with accuracy ranging from 77.0% to 81.0%. This greater fluctuation is also reflected in the standard deviation of 1.31%, suggesting that ResNet's training process is more sensitive to initialization and other random factors, despite its generally higher accuracy compared to LeNet.

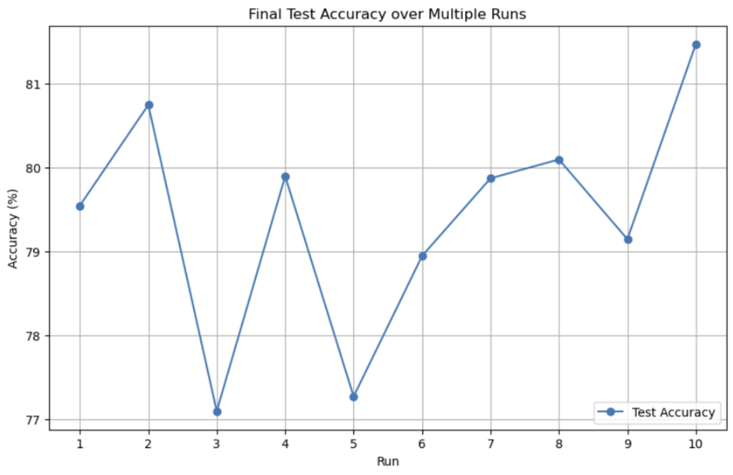


Figure 10: Final Test Accuracy over Multiple Runs - ResNet(CIFAR)

6 Conclusion

6.1 Summary of Methods and Results

In this study, we explored the integration of transition matrices in deep learning models to enhance robustness to label noise, especially important for real-world data with significant noise. Both LeNet and ResNet architectures benefited from this approach, each showing unique strengths: while ResNet achieved higher accuracy, it was also more sensitive to noise, indicating a trade-off between complexity and stability. On the other hand, LeNet, with its simpler architecture, demonstrated consistent performance across different noise levels, offering a more stable option for environments where noise cannot be easily controlled.

These results highlight that model selection for noise-prone data depends heavily on the specific goals—whether to prioritize accuracy or consistency. This study underscores the importance of noise-robust training techniques, which are crucial not only for research but also for applications in real-world settings where data accuracy may be unreliable. Overall, we confirm that transition matrices provide an effective pathway for enhancing model performance under noisy conditions, offering insights into balancing model complexity and robustness.

6.2 Future Work

Future research could focus on adaptive transition matrix estimation to help models adjust to dynamic noise patterns, suitable for fluctuating environments like industry datasets. Additionally, exploring hybrid approaches combining transition matrices with regularization may improve stability in complex models like ResNet, balancing accuracy and robustness.

Examining how noise levels affect model performance could guide best practices for noise handling. Testing newer architectures, like transformer-based models, may further enhance noise resilience, particularly in low-reliability labeling contexts.

This project has enriched our understanding of noise robustness, equipping us with essential skills for real-world data science, where data quality varies. On a broader scale, these findings can support fields like medical diagnostics and environmental monitoring, contributing to reliable and ethical AI that performs well under imperfect conditions. The techniques and insights gained here provide a solid foundation for us to make impactful contributions in AI and beyond.

7 Appendix

References

[1] B. Frénay, A. Kabán *et al.*, "A comprehensive introduction to label noise." in *ESANN*. Citeseer, 2014.

[2] B. Han, Q. Yao, T. Liu, G. Niu, I. W. Tsang, J. T. Kwok, and M. Sugiyama, "A survey of label-noise representation learning: Past, present and future," *arXiv preprint arXiv:2011.04406*, 2020.

[3] T. Liu and D. Tao, "Classification with noisy labels by importance reweighting," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 38, no. 3, pp. 447–461, 2015.

[4] D. Angluin and P. Laird, "Learning from noisy examples," *Machine learning*, vol. 2, pp. 343–370, 1988.

[5] A. K. Menon, B. Van Rooyen, and N. Natarajan, "Learning from binary labels with instance-dependent corruption," *arXiv preprint arXiv:1605.00751*, 2016.

[6] V. Cherkassky and Y. Ma, "Practical selection of svm parameters and noise estimation for svm regression," *Neural networks*, vol. 17, no. 1, pp. 113–126, 2004.

[7] T. Joachims, "Making large-scale svm learning practical," Technical report, Tech. Rep., 1998.

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020, accessed: 2024-11-02. [Online]. Available: https://arxiv.org/pdf/1906.00189

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, accessed: 2024-11-02. [Online]. Available: https://arxiv.org/pdf/1512.03385

[11] GitCode, "Resnet——cnn(pytorch)," 2020, accessed: 2024-11-02. [Online]. Available: https://blog.csdn.net/weixin_44023658/article/details/105843701

[12] M. Li, "resnet," 2020, accessed: 2024-11-02. [Online]. Available: https://zhuanlan.zhihu.com/p/31852747

[13] jiangxiao xiao, "cnnresnet," https://zhuanlan.zhihu.com/p/31852747, 2022, accessed: 2024-11-02. [Online]. Available: https://zhuanlan.zhihu.com/p/31852747

[14] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization," 2020, accessed: 2024-11-02. [Online]. Available: https://arxiv.org/pdf/2011.04406

A Instructions for Running the Code

Our methods are implemented using PyTorch and can be run on systems equipped with an NVIDIA GPU, such as the NVIDIA RTX 3060. To ensure compatibility and optimal performance, please follow the instructions below for setting up your environment.

Environment and Dependencies

- **Python** version: 3.7 to 3.10
- **PyTorch** version: 1.10 or higher
- **CUDA** version: 11.3 or 11.6
- **cuDNN**: Version 8.x compatible with your CUDA installation
- **Anaconda3** (recommended for environment management)

A.1 Usage

Running the Jupyter Notebook or Google Colab

1. Launch Jupyter Notebook or google Colab
2. Open `COMP5328_A2_Final_Version.ipynb` and run the cells sequentially.

B Contribution

Each members contribute the equal amount of project.