# CoreML Report

Tanmay Joshi

March 2025

## 0.1 Main Task:

**Problem Statement**:Active-Passive Loss (APL) framework balances robustness and performance. Active losses (e.g., Cross-Entropy, Focal Loss, etc.) maximize the probability of the true class, while passive losses (e.g., Mean Absolute Error, Reverse Cross-Entropy) also minimize the probabilities of incorrect classes. Pairing the active and passive losses to address underfitting and enhance noise tolerance,making APL effective in noisy-label scenarios.
1. Data Preparation: Modify the CIFAR-10 dataset by introducing symmetric noise rate [0.2, 0.8] by randomly flipping the labels within each class to incorrect labels from other classes.
2. Normalized Losses: Train models on the noisy datasets using normalized loss functions, like Normalized Cross-Entropy (NCE) and Normalized Focal Loss (NFL). Record the performance metrics for these loss functions and compare their results with their vanilla counterparts (standard Cross-Entropy and Focal Loss). Generate plots to illustrate the effect of different noise rates on model performance and demonstrate the robustness of normalized losses over their vanilla alternatives.
3. APL Framework: Train models using the APL losses (you may use loss functions such as NCE,NFL, MAE, and RCE) and compare their performance with the previous results. Highlight how APL improves robustness to label noise while maintaining or enhancing model performance. Use plots to compare the performance of all methods under different noise rates.
4. BONUS: Repeat the experiments but with asymmetric noise in the CIFAR-10 dataset with [0.1, 0.4].

**Paper Used:https://arxiv.org/abs/2006.13554**

**Background:**
The paper - "Normalized Loss Functions for Deep Learning with Noisy Labels" shows by applying a simple normalization that: any loss can be made robust to noisy labels.To address this, the paper proposes a framework to build robust loss functions called Active Passive Loss (APL). APL combines two robust loss functions that mutually boost each other. APL losses contain- 1) "Active" loss, which only explicitly maximizes the probability of being in the labeled class,

and 2) "Passive" loss, which also explicitly minimizes the probabilities of being in other classes. Label Noise can be either symmetric or assymetirc.
For symmetric noise,

$$\eta_{j\to k} = \{\, 1-\eta, if k = j, \eta K - 1, if k \neq j.$$

For asymmetric noise, $\eta_{jk}$ is conditioned on both the true class $j$ and the mislabeled class $k$.

***Types of loss functions:***

- **Cross Entropy (CE):**

$$CE = \sum_{k=1}^{K} q(k|x) \log p(k|x)$$

  CE is widely used in classification tasks but is not robust to noisy labels.

- **Mean Absolute Error (MAE):**

$$MAE = \sum_{k=1}^{K} |p(k|x) - q(k|x)|$$

  MAE is provably robust to label noise.

- **Reverse Cross Entropy (RCE):**

$$RCE = \sum_{k=1}^{K} p(k|x) \log q(k|x)$$

  RCE is robust to label noise and can be combined with CE to form Symmetric Cross Entropy (SCE).

- **Focal Loss (FL):**

$$FL = \sum_{k=1}^{K} q(k|x)(1-p(k|x))^{\gamma} \log p(k|x)$$

  FL generalizes CE and helps with class imbalance but is not robust to noisy labels.

**2. Normalized Loss Functions**
A loss function $L$ is noise-tolerant under mild assumptions if:

$$\sum_{j=1}^{K} L(f(x), j) = C, \quad \forall x$$

Based on this, the normalized loss function is defined as:

$$L_{norm} = \frac{L(f(x), y)}{\sum_{j=1}^{K} L(f(x), j)}$$

This ensures $L_{norm} \in [0, 1]$.

**3. Normalized Loss Variants**

- **Normalized Cross Entropy (NCE):**

$$NCE = \frac{\sum_{k=1}^{K} q(k|x) \log p(k|x)}{\sum_{j=1}^{K} \sum_{k=1}^{K} q(y = j|x) \log p(k|x)} = \log \left( \frac{\sum_k p(k|x)}{p(y|x)} \right)$$

- **Normalized MAE (NMAE):**

$$NMAE = \frac{\sum_{k=1}^{K} |p(k|x) - q(k|x)|}{\sum_{j=1}^{K} \sum_{k=1}^{K} |p(k|x) - q(y = j|x)|} = \frac{1}{2(K-1)} MAE$$

- **Normalized RCE (NRCE):**

$$NRCE = \frac{\sum_{k=1}^{K} p(k|x) \log q(k|x)}{\sum_{j=1}^{K} \sum_{k=1}^{K} p(k|x) \log q(y = j|x)} = \frac{1}{A(K-1)} RCE$$

- **Normalized Focal Loss (NFL):**

$$NFL = \frac{\sum_{k=1}^{K} q(k|x)(1 - p(k|x))^{\gamma} \log p(k|x)}{\sum_{j=1}^{K} \sum_{k=1}^{K} q(y = j|x)(1 - p(k|x))^{\gamma} \log p(k|x)} = \log \left( \frac{\sum_k (1 - p(k|x))^{\gamma} p(k|x)}{(1 - p(y|x))^{\gamma} p(y|x)} \right)$$

- Normalized versions of robust loss functions like MAE and RCE are just scaled versions of the original forms, retaining their robustness.
- Normalizing non-robust losses like CE and FL yields new robust variants.
- This normalization framework can be applied to other loss functions as well.

**Theoretical Justification**:

***Lemma 1*** :In a multi-class classification problem, any normalized loss function $\mathcal{L}_{norm}$ is noise tolerant under symmetric (or uniform) label noise, if the noise rate $\eta < \frac{K-1}{K}$.

***Lemma 2:***In a multi-class classification problem, given $R(f^*) = 0$ and

$$0 \ \leq \ \mathcal{L}_{norm}(f(x), k) \ \leq \ \frac{1}{K-1} \quad \forall k,$$

any normalized loss function $\mathcal{L}_{norm}$ is noise tolerant under asymmetric (or class-conditional) label noise, if the class-wise noise rate $\eta_{jk} < 1 - \eta_y$. The authors

investigate the issue of *underfitting* in robust loss functions when training on noisy data. They conduct experiments on CIFAR-100 with 60% symmetric noise, using a ResNet-34. They compare standard loss functions (CE, FL) with their normalized versions (NCE, NFL), as well as originally robust losses (MAE, RCE) and their normalized variants (NMAE, NRCE). The results show that:

1. **Normalized CE (NCE) and normalized FL (NFL)** become more robust to label noise but *do not* lead to better final accuracy compared to their unnormalized counterparts.

2. **Originally robust losses (MAE, RCE)** suffer from severe underfitting, failing to converge under the given noisy setting.

3. The underfitting problem persists across different hyperparameter settings (learning rates, schedulers, weight decay, and training epochs).

The authors conclude that existing robust losses (NCE, NFL, MAE, RCE) can struggle with underfitting in high-noise scenarios and propose a new loss framework to address this limitation.

***APL Framework-*** [Active loss function] $\mathcal{L}_{Active}$ is an *active* loss function if for all $(x, y)$ in the dataset $D$,

$$\sum_{k \neq y} f(x)_k \;=\; 0.$$

[Passive loss function] $\mathcal{L}_{Passive}$ is a *passive* loss function if for all $(x, y)$ in the dataset $D$,

$$\sum_{k \neq y} f(x)_k \;\neq\; 0.$$

According to Definitions 0.1 and 0.1, **active losses** explicitly *maximize* the network's output probability at the class position specified by the label $y$, while ignoring (not explicitly penalizing) probabilities at other classes. For instance, in Cross-Entropy (CE), only the probability $q(k = y \mid x)$ is maximized, and the loss becomes zero once $q(k = y \mid x) = 1$.

By contrast, **passive losses** also explicitly *minimize* the probability at *one or more* other class positions in addition to maximizing the probability at $k = y$. For example, Mean Absolute Error (MAE) not only maximizes the probability at the correct class $y$ but also directly penalizes probabilities at other classes.

| **Loss Type** | **Examples** |
|---|---|
| Active | CE, NCE, FL, NFL |
| Passive | MAE, NMAE, RCE, NRCE |

Table 1: Examples of active and passive loss functions.

Inspired by the benefits of symmetric learning (Wang et al., 2019c) and complementary learning (Kim et al., 2019), we propose to combine a robust active

loss and a robust passive loss into an *Active Passive Loss* (APL) framework for both robust and sufficient learning. Formally, we define:

$$\mathcal{L}_{APL} = \mathcal{L}_{Active} + \mathcal{L}_{Passive},$$

where balancing parameters $\lambda_1 > 0$ and $\lambda_2 > 0$ (see Eq. (6)) adjust the contribution of each term.

An important requirement is that both loss terms must be robust. Hence, any nonrobust loss should be normalized (following Eq. (1)) to ensure that the resulting APL is noise tolerant. This robustness property is formalized in the following lemma.

If both $\mathcal{L}_{Active}$ and $\mathcal{L}_{Passive}$ are noise tolerant, then

$$\mathcal{L}_{APL} = \mathcal{L}_{Active} + \mathcal{L}_{Passive}$$

is noise tolerant.

Within the APL framework, the two loss terms optimize the same objective from two complementary directions (e.g., maximizing $p(k = y \mid x)$ and minimizing $p(k \neq y \mid x)$). In this paper, four possible combinations that adhere to the APL principle are considered:

1. NCE + MAE,
2. NCE + RCE,
3. NFL + MAE,
4. NFL + RCE.

For simplicity, the balancing parameters are omitted in the rest of the paper. According to our active/passive definitions, APL losses are considered as passive losses.

***Why APL can address underfitting?***

APL combines an active loss with a passive loss. By definition, the passive loss explicitly minimizes (at least one component of) the Q term discussed above so that it wont increase when py is fixed. This directly addresses the underfitting issue of a robust active loss. Therefore, APL losses can leverage both the robustness and the convergence advantages. Note that, by definition, passive loss has a broader scope than active loss. A single passive loss like MAE can be decomposed into an active term and a passive term, with the two terms already form an APL loss. With proper balancing between the two terms, the reformulated MAE can also be a powerful new loss. For example, a recent work has shown that a reweighted MAE can outperform CE.

***My Implementation of APL-*** The following experiments have been done on CIFAR-10 dataset with the following hyper-parameters.

Table 2: Summary of networks and training setups. All models use **SGD** with momentum = 0.9 and **cosine learning rate annealing**. Typical data augmentations include random width/height shift and horizontal flip.

| **Dataset** | **Network** | **Epochs** | **Weight Decay** | **Initial LR** | **Augmentations** |
|---|---|---|---|---|---|
| MNIST | 4-layer CNN | 50 | $1 \times 10^{-3}$ | 0.1 | Shift |
| CIFAR-10 | 8-layer CNN | 120 | $1 \times 10^{-4}$ | 0.1 | Shift + Flip |
| CIFAR-100 | ResNet-34 | 200 | $1 \times 10^{-4}$ | 0.01 | Shift + Flip |

Table 3: Parameter settings for baseline methods and APL. These values are chosen via grid search to match original papers or are set empirically as stated in the snippet.

| **Method** | **MNIST** | **CIFAR-10** | **CIFAR-100** |
|---|---|---|---|
| **GCE** | $p = 0.7$ | $p = 0.7$ | $p = 0.7$ |
| **SCE** | $A = -4,\ \alpha = 0.01,\ \beta = 1.0$ | $A = -4,\ \alpha = 0.1,\ \ \beta = 1.0$ | $A = -4,\ \alpha = 0.5,\ \ \beta = 1.0$ |
| **FL** | $\gamma = 0.5$ | $\gamma = 0.5$ | $\gamma = 0.5$ |
| **APL** | $\alpha = 1,\ \ \beta = 100$ | $\alpha = 5,\ \ \beta = 1$ | $\alpha = 50,\ \beta = 1$ |

num-epoches=15
batch-size=4
learning-rate=0.0001
Convnet connection-

***For symmetric noise creation, I have used 2 methods.***

1. The method given in the paper-

$$\eta_{j \to k} = \{ 1 - \eta, if k = j, \eta K - 1, if k \neq j.$$

For asymmetric noise, $\eta_{jk}$ is conditioned on both the true class $j$ and the mislabeled class $k$.

2. Traditional method(found on gpt)-The traditional method of adding symmetric label noise involves first fixing the exact fraction of samples to corrupt, based on the noise rate. Specifically, one randomly selects a subset of samples—equal in size to the product of the noise rate and the total number of samples—and designates those as "noisy." For each selected sample, its label is then flipped to one of the other K1 classes (uniformly at random). This ensures that no selected label remains correct,

and that each incorrect class is equally likely. In contrast to a probabilistic approach—where each sample has an independent chance of being flipped—the traditional method enforces an exact number of corrupted samples, offering a more controlled but less flexible way to introduce label noise.
Let $n$ be the total number of samples and $\eta$ be the noise rate. Define the set of indices selected for label corruption as

$$I \subset \{1, 2, \ldots, n\} \quad with \quad |I| = \eta n.$$

Then, for each sample $i$, the noisy label $y_i'$ is given by

$$y_i' = \{ y_i, if i \notin I, \pi(y_i), if i \in I,$$

where $\pi(y_i)$ is a random label drawn uniformly from the set

$$\{k \in \{1, \ldots, K\} \mid k \neq y_i\}.$$

This formulation indicates that exactly $\eta n$ samples have their label replaced by a uniformly chosen incorrect label, while the remaining labels remain unchanged.
***I have used the one given in the paper for all my tasks.***

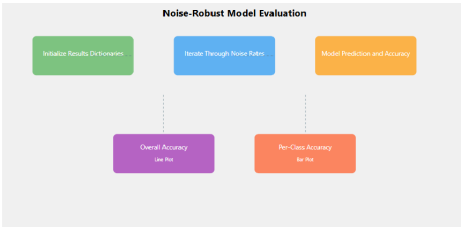
***My code for training remains similar in every snippet, but my loss function has changed in every snippet.***
***Loss functions used and created-***

1. Normalized Cross Entropy
2. Normalized Focal
3. Mean Absolute Error
4. Cross Entropy
5. Focal
6. NCE+MAE
7. NFL+MAE

All the loss functions have been designed using **numpy and some are imported from pytorch.** (I am not explicitly mentioning those code snippets again as the formulae have been given above and are similar to it)

***Evaluation and Results-*** The above is the flow of evaluation/train function of the code. In summary- Initialization Two dictionaries are created:

- `accuracy_results`: Stores overall accuracy for each noise rate
- `class_accuracy_results`: Stores per-class accuracy for each noise rate




Evaluation Process

- Uses `torch.no_grad()` to disable gradient calculation during inference
- Iterates through different noise rates
- Uses a pre-defined test loader and model

Accuracy Calculation For each noise rate:

- Tracks total samples and correct predictions
- Maintains separate counters for each class
- Compares model predictions with noisy labels
- Calculates overall and per-class accuracy percentages

Visualization Strategy

- Creates a two-subplot figure
  - First subplot: Line plot of overall accuracy vs. noise rate
  - Second subplot: Bar plot of accuracies per class versus noise rate
- Uses color-coded bars for different classes
- Adds grid lines and annotations for clarity

**Results for NCE loss-**

| Noise Rate | Overall Accuracy (%) |
|---|---|
| 0.4 | 9.59 |
| 0.5 | 10.16 |
| 0.6 | 9.29 |
| 0.7 | 9.96 |

1. **Overall Accuracy Around 10%**

   The overall accuracy across noise rates (roughly 9–10%) is close to random guessing for a 10-class dataset (which would be 10% if each class is equally likely).

2. **Class 5 Accuracy at 100%**

   Class 5 shows 100% accuracy. This means whenever the true label is class 5, the model correctly predicts class 5.

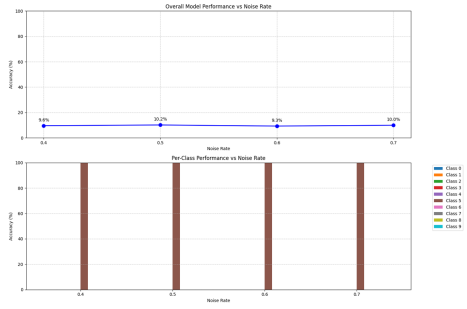



Table 4: Per-Class Accuracy (%) at Different Noise Rates

| Class | Noise 0.4 | Noise 0.5 | Noise 0.6 | Noise 0.7 |
|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 100.00 | 100.00 | 100.00 | 100.00 |
| 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9 | 0.00 | 0.00 | 0.00 | 0.00 |

3. **All Other Classes at 0%**

   Every other class (0–4, 6–9) has 0% accuracy, which indicates the model never predicts them.

Putting this together, the model seems to **always** predict class 5. Whenever the true class happens to be 5, it's correct; for any other class, it's wrong. Since class 5 likely constitutes about 10% of the data (assuming a balanced dataset), that yields an overall accuracy near 10%.

***For normalized focal loss-*** Numerical Results:

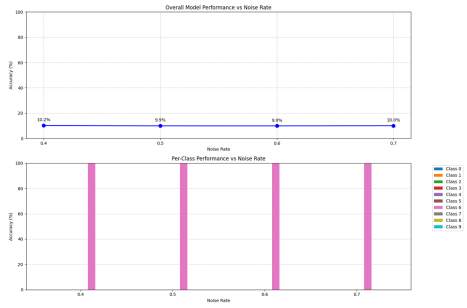


Table 5: Overall Accuracy at Different Noise Rates

| **Noise Rate** | **Overall Accuracy (%)** |
|---|---|
| 0.4 | 10.16 |
| 0.5 | 9.93 |
| 0.6 | 9.91 |
| 0.7 | 10.04 |

Table 6: Per-Class Accuracy (%) at Different Noise Rates

| **Class** | **Noise 0.4** | **Noise 0.5** | **Noise 0.6** | **Noise 0.7** |
|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 100.00 | 100.00 | 100.00 | 100.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9 | 0.00 | 0.00 | 0.00 | 0.00 |

These results indicate that the model is *always* predicting class **6**—and never predicting any other class. Here's why:

1. Overall Accuracy 10

   With a 10-class dataset (assuming it's roughly balanced), always predicting one single class yields an overall accuracy near 10%. The exact numbers (9.9–10.1%) match this expectation.

2. **Class 6 at 100% Accuracy**

   Whenever the true label happens to be class 6, the model is correct. Because the model *always* outputs class 6, it achieves perfect accuracy on that one class.

3. **All Other Classes at 0% Accuracy**

   For any sample whose true class is not 6, the model's prediction is incorrect—hence 0% accuracy for the other nine classes.

In other words, the model has *collapsed* to a trivial solution of predicting class 6 for every sample. This behavior often arises when the training procedure fails to learn discriminative features under heavy label noise or other training difficulties.

***Cross entropy loss-***

These results indicate that the model is *always* predicting class **8**—and never predicting any other class. Here's why:

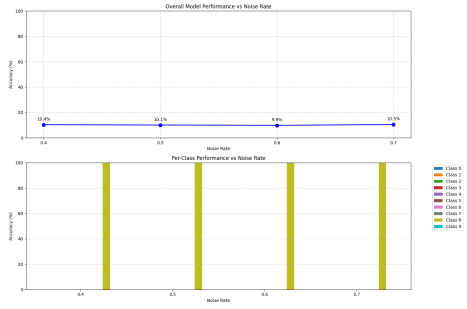



Table 7: Overall Accuracy at Different Noise Rates

| **Noise Rate** | **Overall Accuracy (%)** |
|---|---|
| 0.4 | 10.37 |
| 0.5 | 10.11 |
| 0.6 | 9.87 |
| 0.7 | 10.51 |

Table 8: Per-Class Accuracy (%) at Different Noise Rates

| **Class** | **Noise 0.4** | **Noise 0.5** | **Noise 0.6** | **Noise 0.7** |
|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 100.00 | 100.00 | 100.00 | 100.00 |
| 9 | 0.00 | 0.00 | 0.00 | 0.00 |

1. Overall Accuracy 10 With a 10-class dataset (assuming it's roughly balanced), always predicting one single class yields an overall accuracy near 10%. The exact numbers (9.9–10.1%) match this expectation.

2. **Class 8 at 100% Accuracy** Whenever the true label happens to be class 8, the model is correct. Because the model *always* outputs class 8, it achieves perfect accuracy on that one class.

3. **All Other Classes at 0% Accuracy** For any sample whose true class is not 6, the model's prediction is incorrect—hence 0% accuracy for the other nine classes.

In other words, the model has *collapsed* to a trivial solution of predicting class 6 for every sample. This behavior often arises when the training procedure

fails to learn discriminative features under heavy label noise or other training difficulties.

***Focal Loss-*** Numerical Results:

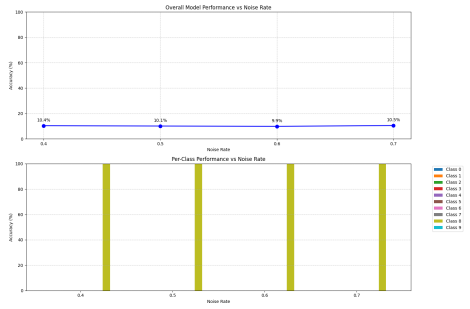



Table 9: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.4** | **Noise 0.5** | **Noise 0.6** | **Noise 0.7** |
|---|---|---|---|---|
| **Overall Accuracy** | 10.37 | 10.11 | 9.87 | 10.51 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

These results indicate that the model is *always* predicting class **8**—and never predicting any other class. Here's why:

1. Overall Accuracy 10 With a 10-class dataset (assuming it's roughly balanced), always predicting one single class yields an overall accuracy near 10%. The exact numbers (9.9–10.1%) match this expectation.

2. **Class 8 at 100% Accuracy** Whenever the true label happens to be class 8, the model is correct. Because the model *always* outputs class 8, it achieves perfect accuracy on that one class.

3. **All Other Classes at 0% Accuracy** For any sample whose true class is not 6, the model's prediction is incorrect—hence 0% accuracy for the other nine classes.

In other words, the model has *collapsed* to a trivial solution of predicting class 6 for every sample. This behavior often arises when the training procedure

fails to learn discriminative features under heavy label noise or other training difficulties. ***Using APL Framework-*** 1)NCE+MAE: The value of alpha and beta has been set to 1 as mentioned in the paper. Numerical Results:

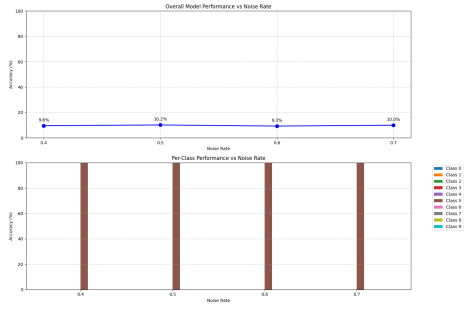



Table 10: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.4** | **Noise 0.5** | **Noise 0.6** | **Noise 0.7** |
|---|---|---|---|---|
| **Overall Accuracy** | 9.59 | 10.16 | 9.29 | 9.96 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

**Overall Accuracy Around 10%**

The overall accuracy across noise rates (roughly 9–10%) is close to random guessing for a 10-class dataset (which would be 10% if each class is equally likely).

**Class 5 Accuracy at 100%**

Class 5 shows 100% accuracy. This means whenever the true label is class 5, the model correctly predicts class 5.

**All Other Classes at 0%**

Every other class (0–4, 6–9) has 0% accuracy, which indicates the model never predicts them.

Putting this together, the model seems to **always** predict class 5. Whenever the true class happens to be 5, it's correct; for any other class, it's wrong. Since class 5 likely constitutes about 10% of the data (assuming a balanced dataset), that yields an overall accuracy near 10%.

***NFL + MAE :*** Numerical Results:

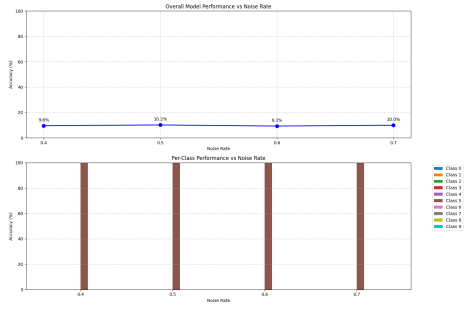



Table 11: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.4** | **Noise 0.5** | **Noise 0.6** | **Noise 0.7** |
|---|---|---|---|---|
| **Overall Accuracy** | 9.59 | 10.16 | 9.29 | 9.96 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

**Overall Accuracy Around 10%**

The overall accuracy across noise rates (roughly 9–10%) is close to random guessing for a 10-class dataset (which would be 10% if each class is equally likely).

**Class 5 Accuracy at 100%**

Class 5 shows 100% accuracy. This means whenever the true label is class 5, the model correctly predicts class 5.

**All Other Classes at 0%**

Every other class (0–4, 6–9) has 0% accuracy, which indicates the model never predicts them.

Putting this together, the model seems to **always** predict class 5. Whenever the true class happens to be 5, it's correct; for any other class, it's wrong. Since class 5 likely constitutes about 10% of the data (assuming a balanced dataset), that yields an overall accuracy near 10%.

***Conclusion-*** The overall accuracy lies near to 10 % with nearly one class showing food results while the others end up being 0. This maybe because-

1) ***Trained on less number of epochs***-The paper trains on nearly 100 epochs does giving it the desired results.Due to lack of computation power my model is only trained on 15 epoches, thus giving a bad result.

2)***Better hyper parameters/loss function selection***- According to me, if I would have tried with different hyper parameters and would have selected different loss functions, I would have got way better results.

## 0.2 Bonus:

***Adding asymmetric Noise:***

In this example, we add asymmetric noise to the CIFAR-10 dataset labels. Asymmetric noise means that instead of flipping the labels randomly, we flip them based on a predefined mapping.

### Noise Mapping

The mapping used is as follows:

- Airplane (0) → Bird (2)
- Automobile (1) → Truck (7)
- Bird (2) → Airplane (0)
- Cat (3) → Dog (5)
- Deer (4) → Horse (7)
- Dog (5) → Cat (3)
- Frog (6) → Deer (4)
- Horse (7) → Automobile (1)
- Ship (8) → Truck (9)
- Truck (9) → Ship (8)

### Implementation Details

For multiple noise levels, we do the following for each level:

1. Make a copy of the original labels.
2. Randomly select a subset of indices proportional to the noise level.
3. Replace the label at each selected index using the mapping.

## Summary

This function creates multiple versions of the dataset with varying noise levels by:

- Randomly selecting a fraction of the data for each noise level.
- Replacing their labels based on a fixed mapping.

This method simulates the effect of mislabeling in a structured (asymmetric) manner, which can be useful for testing the robustness of machine learning models.

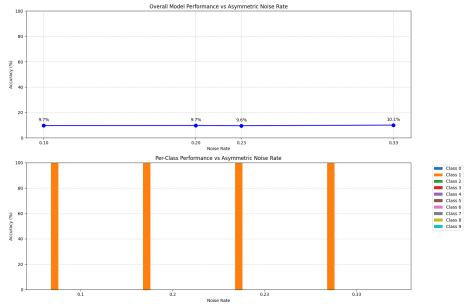
***Training and Evaluation-*** Training and evaluation remains the same where the model has been trained on 15 epochs with the same give hyper-parameters and the model uses all the same loss functions as above-

1. Normalized Cross Entropy
2. Normalized Focal
3. Mean Absolute Error
4. Cross Entropy
5. Focal
6. NCE+MAE
7. NFL+MAE

***Results-***
1)**For Cross Entropy Loss-**

Table 12: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.1** | **Noise 0.2** | **Noise 0.23** | **Noise 0.33** |
|---|---|---|---|---|
| **Overall Accuracy** | 9.68 | 9.74 | 9.56 | 10.07 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |




**Overall Accuracy:** The overall accuracy remains around 9.56%–10.07% across noise rates (0.1, 0.2, 0.23, 0.33). This is close to random guessing for a 10-class problem (i.e., approximately 10%).

**Per-Class Accuracy:** Only **Class 1** is correctly predicted with accuracy 100%, while all other classes (Classes 0, 2, 3, 4, 5, 6, 7, 8, and 9) have accuracy 0%.

***For Normalized Cross Entropy Loss-***

Numerical Results:

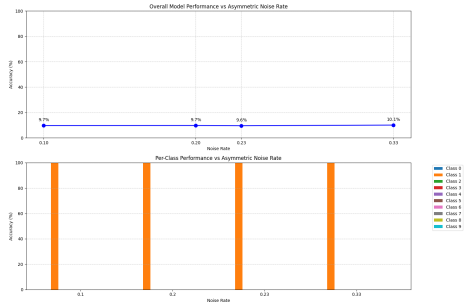



Table 13: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.1** | **Noise 0.2** | **Noise 0.23** | **Noise 0.33** |
|---|---|---|---|---|
| **Overall Accuracy** | 9.68 | 9.74 | 9.56 | 10.07 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

**Overall Accuracy:** The overall accuracy remains around 9.56%–10.07% across noise rates (0.1, 0.2, 0.23, 0.33). This is close to random guessing for a 10-class problem (i.e., approximately 10%).

**Per-Class Accuracy:** Only **Class 1** is correctly predicted with accuracy 100%, while all other classes (Classes 0, 2, 3, 4, 5, 6, 7, 8, and 9) have accuracy 0%.

***For Normalized Focal Loss-*** Numerical Results:

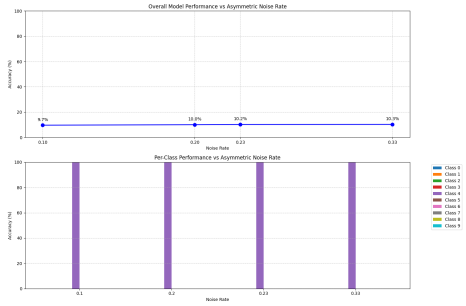



Table 14: Overall and Per-Class Accuracy (%) at Different Noise Rates

| Class / Overall | Noise 0.1 | Noise 0.2 | Noise 0.23 | Noise 0.33 |
|---|---|---|---|---|
| **Overall Accuracy** | 9.67 | 10.04 | 10.20 | 10.27 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

**Overall Accuracy ($\tilde{10}$%):**

The overall accuracy ranges from 9.67% to 10.27% across noise rates. For a 10-class classification task with balanced classes, random or degenerate prediction of a single class would yield an accuracy around 10%.

**Per-Class Accuracy:**

- **Class 4:** Achieves 100% accuracy at all noise rates, meaning every instance belonging to Class 4 is correctly predicted.
- **All Other Classes (0, 1, 2, 3, 5, 6, 7, 8, 9):** Have 0% accuracy, indicating that no instances from these classes are correctly classified.

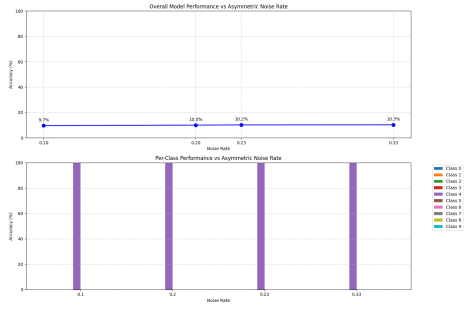



Table 15: Overall and Per-Class Accuracy (%) at Different Noise Rates

| Class / Overall | Noise 0.1 | Noise 0.2 | Noise 0.23 | Noise 0.33 |
|---|---|---|---|---|
| **Overall Accuracy** | 10.00 | 9.99 | 9.99 | 9.50 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

***Focal Loss:***

**Overall Accuracy:**

The overall accuracy is roughly 10% across noise rates (10.00%, 9.99%, 9.99%, and 9.50%), which is what one would expect if the model is always predicting one out of 10 classes in a balanced dataset.

**Per-Class Accuracy:**

- **Class 3:** Achieves 100% accuracy at every noise rate, meaning every instance that belongs to Class 3 is correctly classified.
- **All Other Classes (0, 1, 2, 4, 5, 6, 7, 8, 9):** Receive 0% accuracy, indicating that none of their instances are correctly predicted.

***For NCE+MAE-***

**Overall Accuracy:** The overall accuracy ranges from approximately 9.79% to 10.42% across the different noise rates. This is what you would expect if the model always predicted a single class in a balanced 10-class problem (around 10%).

**Per-Class Accuracy:**

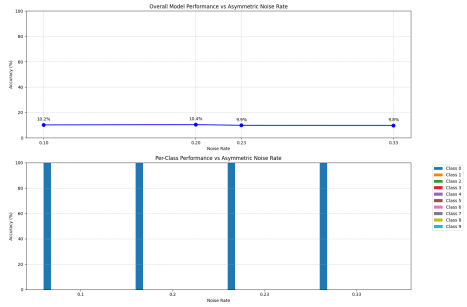



Table 16: Overall and Per-Class Accuracy (%) at Different Noise Rates

| **Class / Overall** | **Noise 0.1** | **Noise 0.2** | **Noise 0.23** | **Noise 0.33** |
|---|---|---|---|---|
| **Overall Accuracy** | 10.20 | 10.42 | 9.90 | 9.79 |
| Class 0 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

- **Class 0:** Achieves 100% accuracy at all noise rates, meaning that every instance belonging to Class 0 is correctly classified.
- **Classes 1, 2, 3, 4, 5, 6, 7, 8, and 9:** All have 0% accuracy, indicating that the model does not correctly classify any instance from these classes.

***For NFL+MAE***:

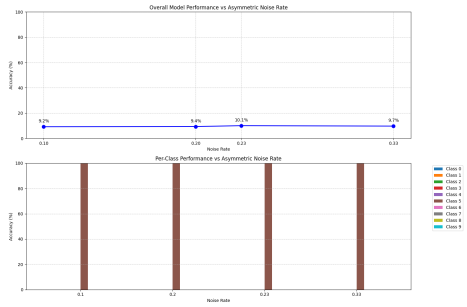


Table 17: Overall and Per-Class Accuracy (%) at Different Noise Rates

| Class / Overall | Noise 0.1 | Noise 0.2 | Noise 0.23 | Noise 0.33 |
|---|---|---|---|---|
| **Overall Accuracy** | 9.24 | 9.38 | 10.07 | 9.73 |
| Class 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 1 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 3 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 4 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 5 | 100.00 | 100.00 | 100.00 | 100.00 |
| Class 6 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 7 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 8 | 0.00 | 0.00 | 0.00 | 0.00 |
| Class 9 | 0.00 | 0.00 | 0.00 | 0.00 |

#### 0.2.1 Overall Accuracy

- The overall accuracy ranges between 9.24% and 10.07% across different noise rates.
- This overall performance is roughly what one would expect when the model predicts a single class in a balanced 10-class problem (about 10%).

#### 0.2.2 Per-Class Accuracy

- **Class 5:** Achieves 100% accuracy at all noise rates, meaning every instance from Class 5 is correctly predicted.
- **All Other Classes (0, 1, 2, 3, 4, 6, 7, 8, 9):** Receive 0% accuracy, indicating that no instance from these classes is correctly classified.

***Conclusions:***
1)All the loss functions give near to 10% accuracy. 2)A spike of 100% can be seen for a particular class for a particular loss function. 3)I feel better computations and fine-tuning of hyper-parameters can improve this scenario.

# 1 Note from my Side

: It was extremely fun to do this assignment.The assignment was an unique blend of numpy(for creating loss functions and labels) and pytorch(for training dataset).This was even more fun than the RL tasks I did.
I feel a bit disheartened. I could have got better results but due to lack of computation I could not run the code for more than 15 epoches. I genuinely would like to do this again with better computation, better fine-tuning of hyper-parameters and other factors. I am not blaming the environmental factors but

I think i did design a robust model and may have less errors in it.
Overall I enjoyed doing this assignment :)).