Comp 433 project

Zhaoyang Li
ID: 27838824

Abdelkader Habel
ID: 40209153

Andrzej Fedorowicz
ID: 40213607

December 18, 2023

1 Introduction

In the realm of image classification, the CIFAR-10 dataset stands as a benchmark for evaluating algorithmic advancements. However, a significant challenge arises when these datasets contain noisy labels, which can drastically impair the performance of learning models. This project delves into the CIFAR-10 dataset, specifically focusing on mitigating the detrimental effects of label noise. We explore two prevalent types of noise: symmetric, where labels are randomly flipped with a probability ϵ, and asymmetric, where certain classes are systematically mislabeled. Our objective is to develop an algorithm capable of robust learning despite the presence of label noise at various levels (10%, 30%, 50%, 80%, and 90%). This endeavor aims to not only enhance the model's accuracy but also to ensure its resilience in real-world scenarios where perfect labeling is often unattainable.

2 Methodology

2.1 Baseline Model: Convolutional Neural Network (CNN)

We initiated our study with a Convolutional Neural Network (CNN), optimized for the CIFAR-10 dataset. This baseline model included two convolutional layers followed by max-pooling and three fully connected layers. We incorporated Rectified Linear Unit (ReLU) activations and experimented with various hyperparameters to establish a strong foundation for further enhancements.

2.1.1 Model Architecture

Our baseline model was established using a standard Convolutional Neural Network (CNN), designed to cater to the CIFAR-10 dataset's image classification requirements. The model's structure is as follows:

- Convolutional Layer 1 with 6 output channels and a 5×5 kernel size
- MaxPooling Layer with a 2×2 window and stride
- Convolutional Layer 2 with 16 output channels and a 5×5 kernel size
- Three Fully Connected Layers reducing dimensions to ultimately classify into 10 classes (for CIFAR-10)

2.1.2 Forward Pass

In the forward pass, each convolutional layer applies ReLU (Rectified Linear Unit) as the activation function, post convolution and prior to pooling. The output of the last pooling layer is flattened before being passed through the fully connected layers, with ReLU activations in between, except for the final layer where raw scores are outputted.

2.1.3 Label Noise Implementation

To simulate a realistic training environment, we introduced label noise into the training data. Two types of noise were applied:

- Symmetric Label Noise: Implemented in the `apply_symmetric_noise` function, this noise randomly alters labels with a specified probability, `noise_rate`. For instance, with a noise rate of 10%, approximately 10% of the labels in the training data are randomly changed, disregarding their original class.
- Asymmetric Label Noise: Through `apply_asymmetric_noise`, we mimic more structured mislabeling, where specific classes are systematically altered to other classes based on a predefined mapping, again controlled by the `noise_rate`. This method reflects real-world scenarios where certain classes are more prone to mislabeling.

Both noise types were rigorously tested across varying noise levels (0%, 10%, 30%, 50%, 80%, 90%) to evaluate the robustness of our baseline model against label inaccuracies.

2.1.4 Training and Evaluation

The training process involved standard procedures like loss calculation using Cross-Entropy Loss, backpropagation, and weight updates using Stochastic Gradient Descent (SGD) with a learning rate of 0.001 and momentum of 0.9. We monitored the model's performance not only in terms of overall accuracy but also class-wise accuracy to gain insights into its behavior across different classes, especially under noisy conditions.

2.2 Enhancing CNN: Dropout, Data Augmentation, and Regularization

Our enhanced CNN model integrated dropout for regularization, data augmentation techniques (such as random flips and rotations), and L2 regularization. These modifications aimed to improve model generalization and robustness, especially in noisy environments.

2.2.1 Data Augmentation

Data augmentation plays a pivotal role in improving model generalization. We employed the following augmentation techniques using the `transforms` module from `torchvision`:

- Random horizontal flips, which mirror images along the vertical axis.
- Random rotations of up to 10 degrees, adding variability in the orientation of the images.
- Random resized crops to 32x32 pixels, with a scale range of 0.8 to 1.0, introducing scale invariance.
- Normalization of images with mean and standard deviation of 0.5 for each RGB channel.

These augmentations diversify the training dataset, enabling the model to learn from a wider variety of image representations.

2.2.2 Dropout

To further enhance the model's ability to generalize and to prevent overfitting, we integrated dropout layers into the architecture. Dropout, a form of regularization, randomly disables a fraction of neurons during training, which helps in breaking up happenstance patterns that are not representative of the general data distribution.

2.2.3 Regularization and Optimizer

L2 regularization was employed to penalize large weights, reducing the model's complexity and likelihood of overfitting. We experimented with two optimizers: SGD (Stochastic Gradient Descent) with a learning rate of 0.001 and momentum of 0.9, and Adam optimizer with the same learning rate. The Adam optimizer is known for its adaptive learning rate capabilities, which can lead to faster convergence.

In conclusion, these enhancements to the baseline CNN model were instrumental in improving its accuracy and robustness, particularly in the challenging scenario of training with noisy labels.

2.3 VGG-Inspired Model(Customized and VGG-16)

We explored VGG-like architectures for their depth and simplicity. Our customized VGG model and a VGG-16 implementation included multiple convolutional layers with max-pooling, augmented by similar data preprocessing and augmentation techniques as our enhanced CNN. This model can be found at the following Google Colab link: `https://colab.research.google.com/drive/1flq5Gs-BF03l3xGqUTbk5bW86C3CWv2q?usp=sharing`

2.3.1 Model Architecture

Our VGG-inspired model comprises several convolutional layers, each followed by a ReLU activation function, and then by max-pooling layers. The architecture is defined as a sequence of VGG blocks, each block containing a set of convolutional layers with an increasing number of output channels as the network deepens. Specifically, we utilized the following configuration:

- Convolutional blocks with an increasing number of filters: 64, 128, 256, and two blocks of 512 filters.
- Each convolutional layer employs a kernel size of 3×3 with padding set to 1, ensuring the spatial dimensions are preserved after convolution.
- Max pooling is performed after each convolutional block with a window of 2×2 and stride of 2, effectively halving the dimensions after each block.

2.3.2 Data Preprocessing and Augmentation

To enhance the model's ability to generalize and to mitigate overfitting, we applied the following preprocessing and data augmentation techniques:

- Resizing images to 40x40 pixels before cropping them back to 32x32 pixels. This random resizing and cropping introduce scale variability.

- Horizontal flips of images are performed randomly, introducing another level of augmentation.
- Normalization of images with predefined means and standard deviations for each RGB channel, aligning the data distribution for stable training.

2.3.3 VGG-16 Model Implementation

To further our exploration into advanced deep learning architectures, we implemented the VGG-16 model, renowned for its simplicity and effectiveness in image classification tasks. This model is characterized by its deep architecture of 16 layers, predominantly convolutional, making it significantly deeper than our previous models.

Model Architecture The core of the VGG-16 architecture comprises several convolutional blocks, each containing a varying number of convolutional layers followed by a max-pooling layer. The specific configuration we adopted is as follows:

- Convolutional Blocks: Each block contains 2 to 3 convolutional layers with kernel sizes of 3×3 and padding of 1. The number of output channels in these layers increases progressively through the network, starting from 64 in the first block to 512 in the final block.
- Max Pooling: A max pooling layer with a kernel size of 2×2 and stride of 2 follows each convolutional block, reducing the spatial dimensions by half after each block.

This arrangement of layers ensures a gradual and systematic reduction in spatial dimensions while increasing the depth of feature maps.

Fully Connected Layers and Output Post convolutional blocks, the network transitions to fully connected layers. A key aspect of our implementation includes dynamically calculating the size of the input to the first fully connected layer based on the output of the final pooling layer. This approach adapts the network to different input sizes. The sequence of fully connected layers is as follows:

- A flattening layer to convert the feature maps into a single vector.
- Three fully connected layers with 4096 neurons each, with ReLU activations and dropout with a rate of 0.5 after the first two fully connected layers.
- The final output layer consists of 10 neurons (for the CIFAR-10 dataset), each representing a class.

The VGG-16 model's deep architecture and extensive use of convolutional layers enable it to capture complex features from the CIFAR-10 dataset, making it an ideal candidate for our experiments in image classification under various noise conditions.

Through this exploration of a VGG-inspired model, we aimed to assess the adaptability and robustness of deeper and more complex architectures in handling the challenges posed by noisy datasets.

2.4 Experimenting with ResNet

The ResNet model, specifically ResNet-18, was adopted for its deep architecture and skip connections. This model was evaluated for its performance on CIFAR-10, leveraging data preprocessing, augmentation strategies, and Stochastic Gradient Descent (SGD) optimization.

ResNet Architecture We adopted the ResNet-18 architecture, a relatively lightweight variant in the ResNet family, making it suitable for our CIFAR-10 dataset application. This model was particularly chosen for its blend of depth and computational efficiency. It is customized to handle CIFAR-10's 3-channel (RGB) images and classify them into 10 distinct classes.

Data Preprocessing and Augmentation For effective training of ResNet, we applied various preprocessing steps, including image resizing, random cropping, horizontal flipping, and normalization. These transformations augmented the dataset and formatted the inputs to be compatible with the ResNet model.

Training and Evaluation The ResNet model was trained using Stochastic Gradient Descent (SGD) with momentum and weight decay, along with a learning rate scheduler to optimize the learning process. We assessed the model's performance using a validation dataset, ensuring that we could evaluate both its accuracy and its generalization capability.

Through the incorporation of ResNet into our study, we aimed to assess the capabilities of this deeper, more complex architecture in the context of image classification, contributing valuable insights into its performance and efficiency.

The ResNet model can be found at the following Colab link: `https://colab.research.google.com/drive/1vq1c7-o2nPdUiy_x3-kcBxsvZKrwUzxh?usp=sharing`

2.5 Co-Training with ResNet

We employed a co-training strategy using two parallel ResNet models. This approach allowed each model to learn and adapt independently, offering insights into the effectiveness of parallel model training in deep learning.

Architecture and Training Each ResNet model was configured with the standard ResNet-18 architecture, optimized for the CIFAR-10 dataset's 3-channel (RGB) images. The models were trained using Stochastic Gradient Descent (SGD) with momentum and weight decay. A learning rate scheduler was employed to adjust the learning rate during training, enhancing the training efficiency.

Co-Training Strategy The key aspect of this approach was the simultaneous training of two models, where each model's learning was informed by the performance of the other. This methodology aimed to exploit the diverse learning capabilities of each network, potentially leading to a more robust and comprehensive understanding of the dataset.

Evaluation The performance of each model was evaluated independently on a validation set. Metrics such as loss, accuracy, and training time were recorded to assess the effectiveness of the co-training approach. This method provided insights into the potential benefits of parallel model training in deep learning tasks.

This experimentation with co-training ResNet models allowed us to explore the dynamics of parallel model training and its impact on learning efficiency and accuracy in image classification tasks.

2.6 Label Smoothing

Label smoothing was implemented as a regularization technique to soften the hard target labels, promoting less confident predictions and potentially enhancing model generalization.

Implementation of Label Smoothing Label smoothing was applied to the training labels by adjusting their values slightly towards a uniform distribution. A smoothing factor was used to control the level of adjustment. The core idea behind this approach is to replace the hard target (1 for the true class and 0 for all others) with a softer target, where the true class has a slightly lower probability and the other classes a slightly higher probability than zero. This was achieved by the following steps:

- Each label was transformed into a one-hot encoded vector.
- The confidence level, set to $1 - \text{smoothing factor}$, was assigned to the true class.
- The remaining probability mass was evenly distributed among the other classes.

2.7 Warmup Strategy

In the initial phase of training, we employed a warmup strategy, which involves starting with a lower learning rate and gradually increasing it over time. This approach helps the model adapt more smoothly to the training data during the early stages, especially in the presence of label noise, thus reducing the risk of overfitting.

2.8 Custom Loss Function

To better handle noisy data, we experimented with custom loss functions. Specifically, we used a Weighted Cross-Entropy Loss, where samples with predictions inconsistent with their actual labels are assigned lower weights. This design assists the model in focusing on samples that are likely to have accurate labels, thereby improving overall learning efficiency and accuracy.

2.9 Co-Teaching Model

Another type of model explored was an implementation of the Co-Teaching training method proposed by Han et al. As discussed in the literature review, Co-Teaching involves training 2 separate models based on the same architecture at the same time. After each epoch, a percentage of the lowest loss instances from each model would be selected and shared with the other model to train on.

The original proposal used a CNN model that included 9 convolutional layers followed by batchnorm layers and a linear layer at the end for classification. In our model, we did not use the same setup as the paper, opting instead to try to use a more simplified model in order to reduce memory usage; memory usage was quite high considering the use of the entire training set on 2 separate models. Our model includes 3 sets of 2 convolutional layers that are followed by a max pooling and dropout layer. The convolutional layers increased in filters for each batch of 2 layers while the image size would remain the same between each layer thanks to padding. For each convolutional layer, our model uses the he_intializer for the kernels in order for the kernel to be able to be updated properly and to speed up training. Our model uses two linear layers at the end of the forward pass in order to perform classification using the softmax activation function. As an optimizer, we use the Adam optimizer with a learning rate of 0.001. Adam has been shown to be a more computationally effective optimizer than the traditional use of SGD which is why we chose to use it.[1] This model is inspired by models used in previous strategies, specifically the dropout strategy outlined in 2.2.2. We chose to run our training for 10 epochs for each noise level and type in order to mitigate the effects of overfitting on noisy data. As a preprocessing step, we normalize our data and implement one hot encoding for the labels.

We did not implement the entirety of the Co-Teaching method as described by Han et al. The reason for this is that during development, we realized that our lack of computational power, even using Colab, would cause the training to be excessively long and memory intensive. For that reason, we did not implement the weight sharing portion of Co-Teaching, opting to just sample the small loss instances in each model and train on those instances from the other model. As noisy labels are more likely to create high loss outputs, choosing to only train on low loss instances would help mitigate training on noisy data.

The Co-Teaching Model can be found at the following colab link: `https://colab.research.google.com/drive/1AF2A1qFX_VXnHFJrZLzLcdj_yKniwV7w?usp=sharing`

2.10 Validation Set and Callbacks Model

Another model we explored was a simpler adaptation of the model that we used in our previous Co-Teaching model. As such, the same preprocessing steps of normalization and one hot encoding were implemented. We wanted to explore both how different epoch callbacks and how incorporating a validation set would affect model performance on noisy labels. The validation set would help with choosing suitable hyperparameters and the callbacks we used such as early stopping and reducing the learning rate on the plateau would help combat overfitting.

This model can be found at the following colab link: `https://colab.research.google.com/drive/1SaLPprjLcNoBNtCn1vftUaKDDqe8WVDo?usp=sharing`

3 Experimental Setup

For almost all our models, we decided to use Keras and Tensorflow as our frameworks for development rather than Pytorch due to it being easier to develop with for our implementations. As we used the CIFAR-10 model, we used 50000 training images and 10000 testing images for all our models. For our callbacks and validation set model we chose 25% of the training set to be used for validation, resulting in 37500 images used for training, 12500 for validation and 10000 for testing.

We ran all our models on GPU initially, but had to switch to CPU at times due to limitations on Colab for GPU time. This caused our computation to slow down considerably, going from 30 seconds per epoch for our Co-Teaching using GPU to over 8 minutes when using CPU. This, along with the fact that we only had 12 gigabytes of system memory available, caused us to try to keep our models and testing as simple as possible while trying out complex solutions.

4 Empirical Results

In this section, we present the outcomes of our experiments, highlighting the performance of different models under varying levels of symmetric and asymmetric noise. The results, as shown in the table on the next page, demonstrate the effectiveness of each methodology in dealing with noisy data.

The co-teaching model was trained using two models 10 epochs. For every combination of noise level and accuracy type, the training accuracies increased through each epoch and the loss decreased. The validation losses and accuracies also similarly trended in their own expected directions, through the epochs. Upon termination of the training, the testing dataset with clean labels was used and accuracies measured. The 2 models were evaluated and it was found their accuracies are close to each other, within a point of percentage. Taking the average accuracies of model 1 and model 2 allows us to determine the accuracies across all combinations.

The first table, displayed on the next page, illustrates the accuracy of the models under varying levels of symmetric and asymmetric noise. The second table presents the average times per epoch for CPU and GPU, highlighting the computational demands of each model.

The results presented in Table 1 show the accuracy of the co-teaching model in the range of 50-60%. The lowest noise level had an accuracy of 52% under both symmetric and asymmetric noise. The highest noise level had the highest accuracy, with 61 and 59% for symmetric and asymmetric noise respectively. It appears higher noise levels yield higher accuracies, except for when the noise is at 80%. At that point, the models are unable to increase the accuracy from the previous levels. The training accuracies are consistently higher than the validation accuracies with training accuracies reaching the high 80s but the validation staying at no more than 50-60%.

The results in Table 1 show an expected trend downwards for the Callbacks model in terms of

Table 1: Experimental Results Across Different Noise Levels

Methodology	Symmetric Noise					Asymmetric Noise				
	10%	30%	50%	80%	90%	10%	30%	50%	80%	90%
Baseline	57.14%	-.0%	49.45%	-.0%	21.96%	56.13%	-.0%	50.39%	-.0%	41.95%
Enhanced CNN	58.18%	56.14%	50.17%	26.82%	26.81%	58.49%	55.28%	43.68%	41.28%	41.59%
VGG(customized)	77.85%	75.6%	66.6%	38.36%	-%	78.82%	76.25%	63.84%	50.50%	-%
VGG(torch)	70.0%	51.0%	25.0%	10.0%	-%	73.0%	69.0%	63.0%	74.0%	- -%
VGG16(torch)	68.0%	50.0%	20.0%	10.0%	-%	73.0%	68.0%	63.0%	72.0%	- -%
ResNet	76.0%	52.0%	32.0%	11.0%	10.0%	80.6%	72%	68%	79%	82%
Co- ResNet	-%	47.0%	31.0%	11.0%	-%	0.0%	62.0%	59.0%	70.0%	-%
Co-Teaching	52.0%	57.0%	57.0%	51.0%	61.0%	52.0%	55.0%	53.0%	53.0%	59%
Callbacks	76.06%	69.44%	58.90%	36.24%	9.81%	77.46%	74.56%	63.03%	48.74%	47.58%

Table 2: Average Time per Epoch for Different Methodologies

Methodology	Average Time per Epoch (CPU)	Average Time per Epoch (GPU)
Baseline	-	15 s
Enhanced CNN	-	22 s
VGG (customized)	45 s	-
VGG (torch)	-	15 s
VGG16 (torch)	-	15 s
ResNet	-	41 s
Co-Training ResNet	-	96 s
Co-Teaching	-	30 s
Callbacks	-	8 s

both noise types, with the symmetric noise accuracy decreasing from 76.06% with 10% noise to just 9.81% with 90% noise. The asymmetric noise decreases from 77.46% with 10% noise to 47.58% with 90% noise. Other than the really low accuracy for 90% noise in symmetric datasets, these values are pretty decent for a simple model. A 36.24% score for 80% symmetric noise when considering that only 20% of the training data has correct labels is 16.24% higher than if the model had overfit completely and predicted the test set with 20% accuracy. The asymmetric noise results are decent as well, especially with high noise data, with 90% asymmetric noise yielding an accuracy of 47.58%. For reference, 90% asymmetric equates to about 55% of the training data not being noise.

Note: The average times per epoch are for reference only. Each method utilized different computational resources, which may affect the reported times.

5 Discussion

The empirical results reveal significant insights into the behavior of different models under noise. Notably, the VGG (customized) model, implemented using Keras, shows superior performance in most scenarios compared to other models implemented with PyTorch. This could be attributed to the architectural differences or optimization techniques inherent to the frameworks. Interestingly, the VGG models and ResNet, known for their depth, exhibit a more robust response to higher noise levels, indicating their potential in complex real-world scenarios where label accuracy is not guaranteed.

The results highlight not only the variance in model accuracy under noise conditions but also significant differences in computational efficiency. The VGG (customized) model shows promising accuracy but demands longer processing time, especially on the CPU. The ResNet and Co-Training ResNet models, while offering competitive accuracy, exhibit higher computational times on the GPU, indicating a trade-off between accuracy and efficiency. This variance in time efficiency across different models underscores the importance of considering computational resources alongside model performance in practical applications.

In terms of symmetric noise, the customized VGG model exhibited the best performance in low noise cases, with it coming on top in the 10%, 30% and 60% noise levels. From then on, however, the co-teaching model has the highest accuracy with 51% and 61% for 80 and 90% noise levels, respectively. the co-teaching method did not yield the results expected. Han et al. obtained accuracies of 97 and 91% when the symmetric noise was 20 and 50% respectively [3]. Our results on the other hand are well below their values, which we were not able to reproduce. One reason for that could be that due to computational limitations, we did not implement the weight sharing part of the co-teaching method. As mentioned in the description of the models, we sampled the small loss instances in each model and trained on those instances from the other model. The discrepancy between their application of the method and ours may explain why our results are not satisfying. Another observation we make is that across noise levels, there is barely any difference between the accuracy in the symmetric and asymmetric cases. The only noise level where a difference between the two is slightly marked is at 50% with a 4 percent difference between the two types. Another observation that might be counterintuitive is that the highest accuracies are found at the highest noise level. It is not expected as higher noise levels typically degrade performance, thus the need to build noise resistant models. As we only incorporate the low loss example sharing portion of co-teaching, this could explain our relatively good results for high noise training data. Since the model is using only focusing on examples that yield low loss values, it could be distinguishing between noise and true labels well because of it.

6 Limitations and Challenges

This study faced several limitations, notably in computational resources. Despite utilizing Google Colab, we encountered significant constraints with CPU training speeds and instability in GPU training. These challenges were compounded by the financial impracticality of accessing stable, high-powered computing resources, limiting the scope and depth of model training and experimentation. Such resource limitations are a common hurdle in advanced machine learning tasks and underscore the need for more accessible and robust computational platforms in academic research.

Throughout this study, we encountered several limitations and challenges. One primary limitation was the varying performance of models across different noise levels, reflecting the inherent difficulty in training with noisy data. Furthermore, differences in the underlying frameworks (Keras vs. PyTorch) introduced additional variability in model performance. Technical challenges, such as computational constraints and model tuning, also played a significant role in the feasibility and effectiveness of different methodologies.

A key limitation observed in this study is the balancing act between computational efficiency and model accuracy. Models that excel in noise resilience, such as the VGG and ResNet, require substantial computational time, especially evident in GPU-intensive models like the Co-Training ResNet. This highlights the challenge of deploying such models in resource-constrained environments. Additionally, the variability in computational resources, as seen in the differences between CPU and GPU processing times, adds another layer of complexity to model selection and application.

7 Conclusion

The study conclusively demonstrates the varying degrees of resilience different models possess in the face of noisy data. While models like VGG (customized) and ResNet show promising robustness, there remains a notable gap in performance under high noise conditions. The Co-Teaching model seems to give the best performance in high noise symmetric conditions, though it fails in low noise ones. This underscores the need for continued research into more sophisticated algorithms capable of handling real-world data complexities. Overall, the findings of this study contribute valuable insights into the field of image classification and model robustness under label noise.

This study highlights the critical importance of model selection in the presence of noisy data, particularly distinguishing between symmetric and asymmetric noise types. Our findings suggest the necessity of choosing different models optimized for each noise type, reinforcing the concept that one size does not fit all in machine learning solutions. While models like VGG (customized) and Co-Teaching exhibited notable resilience, resource limitations and inherent model constraints present significant challenges in achieving optimal performance, particularly in high noise scenarios. Future research should focus on developing more resource-efficient algorithms and exploring novel approaches to model training that can better handle diverse and noisy datasets.

The study concludes that while advanced models like VGG, ResNet, and Co-Teaching demonstrate robustness against noisy data, this comes with a cost of increased computational time, particularly on GPUs. The results also suggest that model selection for symmetric versus asymmetric noise should be made carefully, considering both accuracy and computational efficiency. This research underscores the need for efficient yet accurate models in practical scenarios, where both noise resilience and computational resources are critical factors.

8 Literature Review

This review summarizes key works relevant to our project, particularly focusing on algorithms suitable for the CIFAR-10 dataset and strategies for addressing noisy label issues.

8.1 Classical Algorithms for CIFAR-10

Our study began with an exploration of several classic algorithms known for their effectiveness with the CIFAR-10 dataset. These included:

- AlexNet: A pioneering convolutional neural network that revolutionized image classification approaches.
- GoogleNet: Known for its inception module, offering a deep and efficient architecture.
- LeNet: One of the earliest convolutional neural networks, foundational in the field.
- Network in Network (NIN): Introduces micro neural networks to enhance model discriminability.
- ResNet: Features residual connections to enable the training of very deep networks.
- VGG: Characterized by its simplicity and depth, particularly in large-scale image recognition.

8.2 Approaches to Address Noisy Labels

Several techniques exist in the literature that aim to train models under noisy labels, one of them being co-teaching. It has been presented by Han et al. in their paper titled "Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels" [3]. Co-teaching involves the concurrent training of two neural networks, where each network identifies samples with small losses in each minibatch and treats them as important. Each model then shares these instances with the other network to change the other's parameters. This dynamic exchange can be seen as 2 models training each other to decrease their losses and improve their learning capabilities. The method is able to prevent error accumulation which is a quality of a robust model. Empirical results on CIFAR-10 with varying noise levels have shown that the performance of co-teaching is much better than other baselines, whether it be under high noise (45% in their study) or low noise (20%) [3]. Another method that is used in the field to work with noisy labels is label smoothing. Although it was not explored in this analysis, we find it important to discuss it for future works. Label smoothing as a regularizer as described by Lukasik et al. [6] is effective in improving the performance under label noise. It was linked to L2 regularization and was found to be effective when predicting both clean and noisy samples [6]. The researchers tested their idea on CIFAR-10, used ResNet models, and followed the experimental setup from Muller et al. [8], who also discuss label smoothing and regularization. Lukasik et al. got an accuracy of 84% when training their model with a ResNet-32 and a symmetric noise level of 20% [8].

We explored methodologies designed to handle noisy labels, such as the PGDF algorithm [10], ProMix [9], and studies on robust accuracy metrics [7]. Additionally, we reviewed significant works like the VGG networks [5].

- PGDF (Sample Prior Guided Robust Model Learning to Suppress Noisy Labels) by Wenkai Chen et al. [Code]. This method focuses on leveraging sample priors to enhance model robustness against label noise.
- ProMix (Combating Label Noise via Maximizing Clean Sample Utility) by Ruixuan Xiao et al. [Code]. This technique aims at maximizing the utility of clean samples in the presence of label noise.
- Research on 'Robustness of Accuracy Metric and its Inspirations in Learning with Noisy Labels' for cassava by Pengfei Chen et al. [Code], which examines the robustness of accuracy metrics in noisy label scenarios.
- The seminal paper on 'Very Deep Convolutional Networks for Large-Scale Image Recognition' by Karen Simonyan and Andrew Zisserman [Link].
- Studies on contrastive representations in noisy label learning by Yi Li et al. and other works by Ghosh et al., and Zhang and Sabuncu on robust loss functions.

Given the breadth of these studies, our final model selection was influenced by the practical considerations of resources and time, leading to the focused comparison of selected models and parameter adjustments as detailed in the report.

9 Links

My phase Report
GitHub Repository
ZL-10seeds-resnet model
ZL-10seeds-VGG-customized model
Validation and Callbacks Model
Co-Teaching Model

References

[1] Adam https://keras.io/api/optimizers/adam/

[2] G. James, D Witten, T.H., Tibshirani, R.: An introduction to statistical learning.

[3] Han B., Yao Q., Y.X.N.G.X.M.H.W.T.I.S.M.: Co-teaching: Robust training of deep neural networks with extremely noisy labels https://arxiv.org/pdf/1804.06872v3.pdf

[4] J. Friedman, T.H., Tibshirani, R.: The elements of statistical learning. Springer series in statistics

[5] Karen Simonyan, A.Z.: Very deep convolutional networks for large-scale image recognition https://arxiv.org/abs/1409.1556

[6] Lukasik M., Bhojanapalli S., M.A.K.S.: Does label smoothing mitigate label noise? https://proceedings.mlr.press/v119/lukasik20a/lukasik20a.pdf

[7] Pengfei Chen, Junjie Ye, G.C.J.Z.P.A.H.: Robustness of accuracy metric and its inspirations in learning with noisy labels. arXiv:2012.04193 https://github.com/chenpf1025/RobustnessAccuracy

[8] Rafael Müller, Simon Kornblith, G.H.: When does label smoothing help? https://papers.nips.cc/paper_files/paper/2019/file/f1748d6b0fd9d439f71450117eba2725-Paper.pdf

[9] Ruixuan Xiao, Yiwen Dong, H.W.L.F.R.W.G.C.J.Z.: Promix: Combating label noise via maximizing clean sample utility https://github.com/justherozen/promix

[10] Wenkai Chen, Chuang Zhu, Y.C.M.L.T.H.: Sample prior guided robust model learning to suppress noisy labels https://github.com/bupt-ai-cz/PGDF