FII ATNN 2024 - Project - Noisy CIFAR-100

Stefana Gheorghita
Computer Science Department
Alexandru Ioan Cuza University
Iasi, Romania

Maria-Antonia-Emanuela Pascu
Computer Science Department
Alexandru Ioan Cuza University
Iasi, Romania

***Abstract*—This study presents an extensive investigation into the impact of label noise on machine learning models using the CIFAR-100 dataset. By introducing noise into the labels, the project systematically evaluates the performance of multiple models under challenging conditions and explores innovative techniques to mitigate the effects of noisy data. A wide range of model configurations was tested, highlighting the robustness and adaptability of various approaches, including label correction algorithms, regularization techniques, and other strategies. The results provide valuable insights for potential practical applications, where noisy datasets are an unavoidable reality, emphasizing the importance of developing resilient models capable of extracting relevant information even from imperfect data.**

I. Introduction

In our era, where everything has become increasingly digitized and modernized, datasets have truly become the foundation of progress in the field of artificial intelligence and machine learning. The quality of the data we use plays a crucial role in training models, and the noise that is often present in the data can significantly impact their performance. The Noisy CIFAR-100 project aims to investigate and address the impact of mislabeled data (label noise) on machine learning algorithms, using one of the most well-known datasets in the field: CIFAR-100.

CIFAR-100 is a widely used dataset for performing image classification tasks, consisting of small-sized images organized into 100 distinct classes. Each class contains visually diverse images, providing a relatively challenging framework for the development and testing of various algorithms. However, in real-world applications, data are often contaminated by noise, either due to human errors during the labeling process or to certain technological limitations. This project aims to address the issue of noise within the CIFAR-100 dataset and also seeks to identify robust solutions to enhance the ability of models to generalize, even in the presence of such imperfect data.

In this project, a noise level of 40% in the CIFAR-100 labels was used to analyze the behavior of machine learning models under such conditions. In addition to performance analysis, the project explores innovative methods to mitigate the impact of noise, including the use of label correction algorithms, regularization techniques, and retraining strategies. The ultimate goal is to develop models that not only demonstrate robustness against noise but can also extract the most important insights even from imperfect data.

In this report, we have provided detailed documentation of the stages of the Noisy CIFAR-100 project, starting with the problem description and the methodology used, continuing with the experimental analysis, and concluding with findings and future perspectives. Essentially, we conducted an extensive study characterized by testing multiple models in highly diverse configurations. The project is not merely a theoretical exercise, but offers valuable contributions to both the academic community and the applied field, where noisy data is an unavoidable reality.

II. Dataset

In the context of working with images in the field of deep learning, the CIFAR datasets represent a standard and highly valuable resource for developing and evaluating models. They are used for image classification and have contributed significantly to technological advancements in this domain.

The current project involves using the Noisy CIFAR-100 dataset with 40% noise, a subset derived from CIFAR-100, with the distinction that the former introduces noise in the class labels to evaluate the robustness of algorithms in the presence of labeling errors. This type of data is essential in practice, as it serves to explore robust learning methods and to optimize model performance when noise appears in the input data.

Below is an overview of the CIFAR-10 and CIFAR-100 datasets, as well as their noisy versions, CIFAR-10N and CIFAR-100N.

CIFAR-10 is a dataset frequently used in machine learning and computer vision research. It contains a total of 60,000 color images of 32x32 pixels, divided into 10 distinct classes, each class containing 6,000 images. The dataset is divided into 50,000 images for training and 10,000 for testing. The classes in this dataset are: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*.

Similarly to the CIFAR-10 dataset, CIFAR-100 contains 60,000 color images of 32x32 pixels, but the difference lies in the fact that the latter is divided into 100 classes, each class containing 600 images. Each image has a ”fine” label (also known as the specific class), as well as a ”coarse” label (the superclass). The dataset is divided into 50,000 images for training and 10,000 for testing. Regarding the superclasses and classes, an example would be: the superclass ”aquatic mammals” may include classes such as beaver, dolphin, otter, seal, and whale, among others.

CIFAR-10N and CIFAR-100N are versions of the CIFAR-10 and CIFAR-100 datasets that include noisy labels inten-

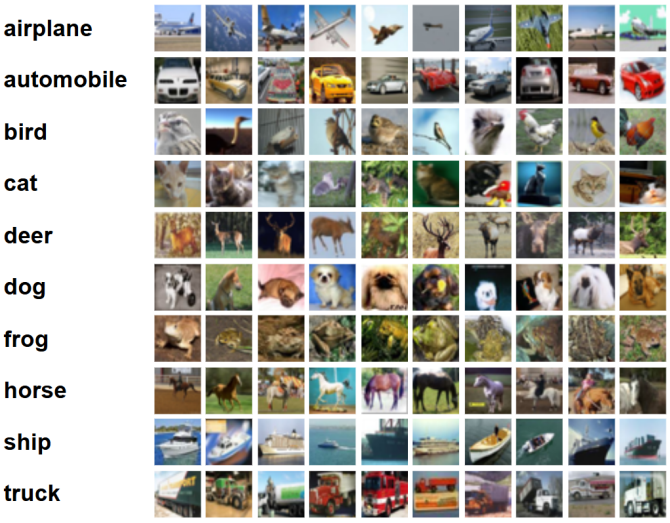

Fig. 1. CIFAR-10. Source of the image: here.

tionally added to study the robustness of machine learning algorithms to mislabeled data. Therefore, these datasets are used to evaluate the performance of the model under imperfect labeling conditions.

Fig. 2. Label Noise. Source of the image: here.

III. Related Work

This section provides valuable insights into existing methods proposed for working with the CIFAR-100N dataset, as well as strategies for handling noisy datasets in general.

A. Learning with Noisy Labels

Learning with noisy labels is a critical challenge in machine learning, as models trained on datasets with incorrect annotations can suffer from reduced performance and generalization capabilities. To address this issue, various strategies have been developed, including sample selection methods, robust loss functions, and label correction techniques. In the following, we will discuss some of these techniques.

1) Sample Selection Methods:: These approaches aim to identify and utilize clean data from noisy datasets. For instance, Zhang et al. [1] introduced an improved sample selection framework that combines an oversampling strategy with state-of-the-art methods to enhance model performance, particularly under high noise rates. However, the model does not achieve satisfactory results on the CIFAR-100N dataset. In [2], the authors propose a noise-tolerant expert model (ITEM) that integrates multiple experts to improve the selection of clean samples from noisy datasets. The method also introduces a mixed sampling strategy to address data imbalance. Their experiments on standard noisy datasets demonstrate that this approach effectively enhances model performance in high-noise scenarios, achieving promising results across several benchmarks. Similarly, Wang et al. [4] present a framework that combines Scalable Penalized Regression (SPR) with Knockoff filters to control the false-selection rate while identifying clean data, showing strong performance in noisy environments. These newly proposed solutions hold significant potential for further improvements and refinements.

2) Robust Loss Functions: Robust loss functions are a pivotal strategy in mitigating the adverse effects of noisy labels in machine learning models. These loss functions are crafted to reduce sensitivity to mislabeled data, enabling models to prioritize learning from correctly labeled instances. Notable contributions in this area include:

- **Mean Absolute Error (MAE) Loss**: Mean Absolute Error (MAE) is a metric that calculates the average magnitude of the absolute errors between the predicted and actual values. Ghosh et al. (2017) demonstrated that their proposed loss function is inherently robust to label noise in multiclass classification problems. Their theoretical analysis and empirical results indicate that MAE can effectively handle noisy labels without requiring modifications to the network architecture [6].
- **Normalized Loss Functions**: Ma et al. (2020) introduced a normalization technique applicable to any loss function, enhancing robustness to noisy labels. This method adjusts loss values to mitigate the impact of noise, leading to improved performance in deep learning models trained on corrupted datasets [7].
- **Active Negative Loss (ANL)**: Ye et al. (2024) proposed ANL, a framework that systematically addresses limitations in existing loss functions when dealing with noisy labels. Their approach enhances learning robustness by focusing on both active and passive loss components, demonstrating superior performance across various noise levels [8].
- **Equal Loss**: Cui et al. (2022) developed the Equal Loss function, which balances the gradient contributions of clean and noisy samples during training. This balance prevents the model from being disproportionately influenced by noisy labels, thereby enhancing generalization capabilities [9].
- **Asymmetric Loss Functions**: Zhou et al. (2021) introduced asymmetric loss functions tailored to handle various types of label noise. Their theoretical and empirical analyses show that these loss functions outperform symmetric counterparts, offering better noise tolerance and improved model performance [10].

All the papers mentioned above present interesting approaches for reducing the effects of noise using different loss functions. However, none of them have achieved satisfactory results on the CIFAR-100N dataset. Nevertheless, these loss functions have the potential to perform well when further improved and combined with appropriate models and techniques.

B. Advanced Strategies for Learning with Noisy Labels

1) Co-training: Co-training and co-learning strategies involve training multiple models simultaneously to handle noisy labels by leveraging their disagreements or consensus. These methods aim to mitigate the impact of label noise by allowing models to collaboratively identify and filter out noisy samples during training. Some notable implementations include Co-Teaching [11], Co-Teaching+ [12], and JoCoR [13]. However, none of these methods have proven to be robust in the case of 40% asymmetric noise.

2) Self-learning: Self-learning, also known as self-training, is a machine learning paradigm where models utilize their own predictions to iteratively improve performance. This approach is particularly effective in scenarios with noisy labels or limited labeled data, as it relies on pseudo-labels generated by the model to enhance training. Several notable self-learning methods have been proposed in the literature.

One prominent self-learning method is **Noisy Student Training**, introduced by Xie et al. [14]. In this approach, a pretrained teacher model generates pseudo-labels for unlabeled or noisy data. A student model is then trained using these pseudo-labels, along with strong data augmentation. This iterative process improves the student model, which can eventually outperform the teacher. Noisy Student Training has demonstrated state-of-the-art performance on benchmark datasets like ImageNet, showcasing its effectiveness in dealing with noisy and limited labels. This approach will be furthermore discussed in the following sections.

Another influential method is the **Mean Teacher** framework proposed by Tarvainen and Valpola [15]. This method leverages two models: a student and a teacher. The teacher model generates predictions that are averaged over time (using exponential moving averages of weights), serving as consistency targets for the student model. By enforcing consistency between the two models, Mean Teacher regularizes the learning process and is particularly effective in semi-supervised learning and scenarios with noisy labels.

Pseudo-Labeling, introduced by Lee [16], is another straightforward self-learning method. In this approach, a model generates pseudo-labels for unlabeled data based on its high-confidence predictions. These pseudo-labels are treated as ground truth during training. While simple and efficient, pseudo-labeling can propagate errors if the initial pseudo-labels are inaccurate, making it sensitive to noise levels and requiring careful thresholding of confidence scores.

Self-learning has also been extended to the realm of self-supervised contrastive learning. Methods like **SimCLR** [17] and **MoCo** [18] have demonstrated the power of learning robust representations without explicit labels. These methods rely on contrasting augmented views of the same sample (positive pairs) with different samples (negative pairs) to learn meaningful representations. While these methods do not explicitly address noisy labels, their learned representations can be fine-tuned for downstream tasks, making them versatile in handling noisy datasets.

Self-learning approaches have shown great promise in addressing noisy labels, as they can refine pseudo-labels over time and reduce reliance on explicit supervision. However, they also present challenges, such as the risk of error propagation from incorrect pseudo-labels and the need for careful regularization and thresholding. Nevertheless, methods like Noisy Student Training, Mean Teacher, and Pseudo-Labeling have paved the way for robust learning in noisy environments, making self-learning a vital component of modern machine learning research.

3) Early-Learning Regularization Methods: Early-learning regularization methods leverage the phenomenon that neural networks initially prioritize learning from clean samples before overfitting to noisy labels. By capitalizing on this early phase of training, these methods mitigate the impact of noisy labels and enhance model robustness.

Early-Learning Regularization (ELR): Liu et al. (2020) proposed ELR [20], which introduces a regularization term that penalizes deviations from predictions made during the early stages of training. This approach leverages the early-learning phenomenon, where models prioritize learning from clean samples before overfitting to noisy labels. By discouraging memorization of noisy labels, ELR encourages the model to focus on reliable data. It has demonstrated superior performance across multiple benchmarks, including CIFAR-10 and CIFAR-100, particularly under high noise scenarios.

Building upon ELR, ELR+ refines the regularization mechanism by incorporating more robust heuristics for identifying clean samples. This enhanced method is especially effective in addressing complex noise patterns, such as asymmetric or instance-dependent noise. ELR+ has achieved strong results on CIFAR-100 with significant noise levels, outperforming many existing approaches designed for noisy label learning.

4) ProMix: **ProMix: Combating Label Noise via Maximizing Clean Sample Utility** [3] introduces a novel framework designed to enhance learning from noisy datasets by maximizing the utility of clean samples. ProMix employs a matched high-confidence selection technique, which dynamically expands a base clean sample set by selecting examples with high confidence scores and predictions that match their given labels. To address the potential imbalance caused by this procedure, ProMix incorporates a semi-supervised learning (SSL) framework to train balanced and unbiased classifiers on both clean and noisy samples. Extensive experiments demonstrate that ProMix achieves state-of-the-art performance across multiple benchmarks, including an average improvement of 2.48% on the CIFAR-N dataset, showcasing its effectiveness in handling various noise levels and types. The implementation is publicly available, further promoting its applicability in noisy label learning tasks.

5) Contrast to Divide: Another approach that achieves good results is the Contrast to Divide (C2D) method, which aims to enhance the performance of Learning with Noisy Labels (LNL) models by pre-training them in a self-supervised manner. The best results are achieved by combining SimCLR [17] with DivideMix [19]. This resulting framework achieves

performance comparable to the ELR + approach on the CIFAR-100 dataset with 40% asymmetric noise.

IV. Proposed Method

In this section we will discuss the best method we constructed during our multiple experiments.

A. Technical Information

EfficientNet

EfficientNet is a family of convolutional neural networks (CNNs) that achieve state-of-the-art performance on image classification tasks while maintaining computational efficiency. It was introduced by Tan and Le in the paper *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. The key contributions of EfficientNet include:

- **Compound Scaling:** Instead of arbitrarily scaling depth, width, or resolution individually, EfficientNet introduces a systematic method called compound scaling. This method scales all three dimensions in a balanced way to optimize both accuracy and efficiency.
- **Baseline Architecture:** EfficientNet starts with a simple base model (EfficientNet B0) designed using a neural architecture search (NAS). This baseline is scaled up systematically to form larger models (B1 to B7).
- **Key Features:**
  - Uses Mobile Inverted Bottleneck Convolution (MBConv) layers for efficient computation.
  - Incorporates squeeze-and-excitation (SE) blocks to recalibrate feature maps and improve model expressiveness.
  - Optimized for FLOPs (floating-point operations per second), making it computationally efficient.

EfficientNet B0

EfficientNet B0 is the smallest and most computationally efficient model in the EfficientNet family. It serves as the baseline architecture from which other EfficientNet variants (B1 to B7) are scaled. Key details about EfficientNet B0 include:

- **Architecture:** Composed of MBConv layers with SE blocks and includes a total of 16 MBConv layers followed by a fully connected layer.
- **Parameters:**
  - Model Size: 5.3 million parameters.
  - Input Resolution: 224×224.
  - FLOPs: 390 million.
- **Performance:**
  - Accuracy: Achieves 77.1% top-1 accuracy on ImageNet.
  - Efficiency: Offers a better accuracy-to-computation trade-off compared to traditional architectures like ResNet and VGG.

Noisy Student Training

Noisy Student is a semi-supervised learning technique introduced by Xie et al. in the paper *Self-training with Noisy Student improves ImageNet classification*. It builds on the idea of self-training and improves model performance using unlabeled data. Key aspects include:

- **Training Process:**
  - A teacher model is trained on labeled data.
  - The teacher generates pseudo-labels for unlabeled data.
  - A student model is trained on the combination of labeled data and pseudo-labeled data, with additional noise added during training (e.g., data augmentation, dropout).
- **Key Features:** Incorporates noise into the student model to make it more robust to perturbations and uses data augmentation techniques such as RandAugment and stochastic depth.
- **Performance:** Achieves state-of-the-art results on ImageNet with an accuracy of 88.4% and improves performance on tasks such as object detection and segmentation.

EfficientNet B0 with Noisy Student

When combined with the Noisy Student technique, EfficientNet B0 achieves enhanced performance by leveraging unlabeled data and robust training strategies. Key observations include:

- **Improved Accuracy:** The pseudo-labeling and noise injection in Noisy Student help reduce overfitting and improve generalization, making it particularly effective for tasks with limited labeled data.
- **Applications:** Used for noisy datasets, semi-supervised learning tasks, and scenarios with class imbalances or limited training samples.
- **Benefits:** Combines the computational efficiency of EfficientNet B0 with the robustness of Noisy Student, leading to a scalable and adaptable model.

B. Implementation

In our implementation, we used the pretrained model `timm/tf_efficientnet_b0.ns_jft_in1k`. The performance of this model across multiple configurations is summarized in Table II, which provides detailed results for each experiment. Furthermore, the results are analyzed and compared in the subsequent section.

As observed through the experiments, the best score was achieved using this model. The pretrained weights, obtained from the Noisy Student training process, proved highly effective in leveraging additional unlabeled data to enhance model robustness and accuracy on the Noisy CIFAR-100 dataset.

Given the consistent performance of `EfficientNet B0 Noisy Student`, we also considered the possibility of using ensemble methods to further enhance results. By combining the predictions of multiple independently trained instances of the model, an ensemble approach could average out noise and variance in individual model predictions, potentially leading to more robust and accurate outcomes.

The prediction in the ensemble is made by combining the outputs of multiple models. For each batch of inputs,

all models independently predict class probabilities using the softmax function, which ensures the probabilities for each class sum to 1. These probabilities are accumulated and averaged across all models to produce the final ensemble prediction. The class with the highest averaged probability is selected as the predicted class using the argmax operation. This method reduces individual model biases and improves robustness, leveraging the strengths of multiple models for more accurate predictions.

We selected two configurations that performed well. For a detailed explanation of what these configurations represent, refer to the *Experiments* section, under the *EfficientNet* subsection.

- **Configuration 1:** (EfficientNet B0 Noisy Student, Adam*, RandAugment+CutMix+MixUp (Alpha=0.5, NumOps=1), CosineAnnealingWarmRestarts)
- **Configuration 2:** (EfficientNet B0 Noisy Student, Adam*, AutoAugment+CutMix+MixUp (Alpha=0.5), CosineAnnealingWarmRestarts)

Depending on the specific runs in which this ensemble method was applied, the results ranged between 75.14% and 75.50%. While this approach enhanced the accuracy on the test set, it also led to an increase in the test loss. This trade-off is likely due to the averaging of predictions, which improves overall correctness but may smooth out the model's confidence, affecting the loss calculation.

The ensemble method leverages the diversity among individual models to improve robustness and mitigate overfitting. However, the increased computational cost and higher inference time make it less practical for real-time applications or resource-constrained environments. Future experiments could explore weighted averaging, where models are assigned weights based on their individual performance, to strike a better balance between accuracy and loss.

V. Experiments

While aiming to achieve satisfactory results on the CIFAR-100N dataset, we experimented with various methods incrementally. These experiments encompassed diverse models and configurations, which we will describe in detail in the following section.

A. Initial Results

When we began working with this dataset, we experimented with configurations that had previously demonstrated good results on the original CIFAR-100 dataset. Starting from this foundation, we established a baseline model using the pretrained `hf_hub:grodino/resnet18_cifar10` model, the SGD optimizer with a learning rate of 0.001, and no scheduler. However, this approach achieved only 27.43% accuracy after 50 epochs.

Afterward, we conducted a grid search similar to the one performed for CIFAR-100 to visualize and gain insights into the relationships within the dataset.

As these experiments represent only an initial part of the work, we will briefly summarize the obtained results. All experiments were conducted using a batch size of 64 and employed an early stopping mechanism, defined by a patience of 5 epochs. Additionally, the scheduler used was *CosineAnnealingLR*, with *T_max = 100* and *eta_min = 1e-5*.

For the experiments in this section, we used two augmentation schemes, which we will now present for clarity and efficiency:

- **randaugment**:
  - CutMix
  - MixUp
  - RandAugment()
  - ToImage(),
  - ToDtype(torch.float32, scale=True),
  - Normalize((0.5,), (0.5,))
- **Combined Resize 2**:
  - CutMix
  - MixUp
  - RandomRotation(10),
  - RandomResizedCrop(32, scale=(0.9, 1.1)),
  - RandomHorizontalFlip(),
  - RandomAffine(degrees=0, shear=10),
  - RandomCrop(32, padding=3),
  - ToImage(),
  - ToDtype(torch.float32, scale=True),
  - Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))

Regarding the different values for normalization, the one used for *randaugment* is more general, while the second is more specific to the studied dataset. For now on, we will refer as *randaugment* to this augmentation scheme that includes CutMix and MixUp.

It can also be observed that both configurations use *CutMix* and *MixUp*. This choice was based on observations from experiments, which indicated that these two augmentation techniques are effective for the CIFAR-100 dataset. Both techniques were configured with *alpha = 1.0*, and the decision on which to apply was made randomly.

None of the configurations achieved satisfactory accuracy using the SGD optimizer. The following configurations, adhering to the previously mentioned parameters, were employed:

- with a *learning rate = 0.01*:
  We tested it using two models: *ResNet18* and *ResNet50*, using two augmentation schemes:
  - (ResNet18, randaugment) : 35.89% test accuracy;
  - (ResNet50, randaugment) : 33.24% test accuracy;
  - (ResNet18, Combined Resize 2) : 30.9% test accuracy;

The other experiments were conducted using the AdamW optimizer, as it proved to perform well with the CIFAR-100 dataset. For the dataset without noise, we achieved test accuracies of 80.61% and 81.88% using learning rates of 0.0005 and 0.001, respectively. Both experiments employed the *randaugment* data augmentation technique and a pretrained ResNet18 model with an additional upsampling layer to resize images to (224, 224). The upsampling layer made a significant

difference, as the model was pretrained on ImageNet, and this adjustment helped the model better capture the features of the images. Using the exact same configurations, we achieved accuracies of 67.6% and 66.19% on the CIFAR-100N dataset, respectively.

The effect of the upsampling layer is evident when training ResNet18 and ResNet50 on the CIFAR-100N dataset. Without the upsampling layer, both models achieved only around 48-49% accuracy. However, the addition of the upsampling layer significantly improved performance. Specifically, using *randaugment*, a ResNet50 model, and the AdamW optimizer with a learning rate of 0.001, the configuration achieved an accuracy of 70.55%.

In Figure 3 and Figure 4, a comparison of the accuracies and losses of some of the described methods is presented. It can be observed that CutMix contributes positively to accuracy, as its removal negatively impacts the results. Furthermore, most methods achieve their highest accuracy within the initial epochs, after which the accuracy slightly decreases and begins to plateau. Similarly, the losses also exhibit a tendency to plateau toward the end. Additionally, it can be observed that the run using SGD as the optimizer shows fluctuating losses, indicating a potential issue with its configuration.

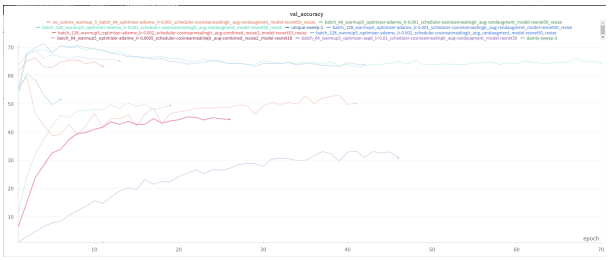

Fig. 3. Accuracy on the testing set

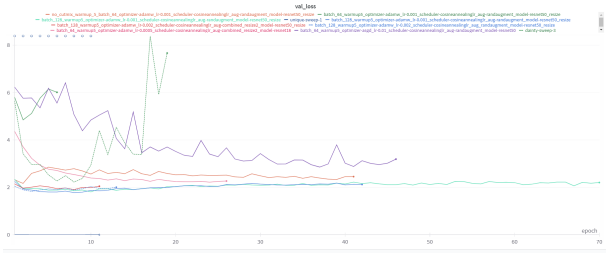

Fig. 4. Loss on the testing set

As discussed previously, adding an upsampling layer proved to be beneficial for the results obtained. However, this approach also introduced some drawbacks. These drawbacks are illustrated in the following plots, which compare GPU usage and power consumption between models with an added upsampling layer and those without it. Additionally, this difference extends to time consumption, as the larger images resulting from upsampling significantly increase training time.

All these experiments were run using the resources provided by Kaggle (P100 GPU).

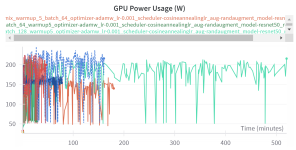

Fig. 5. GPU Power Consumption

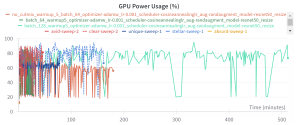

Fig. 6. GPU Usage Percent

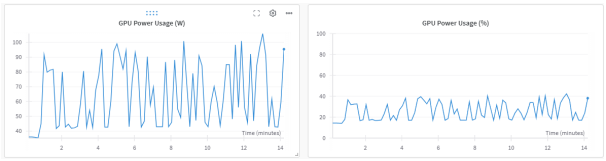

Fig. 7. GPU related metrics - ResNet18

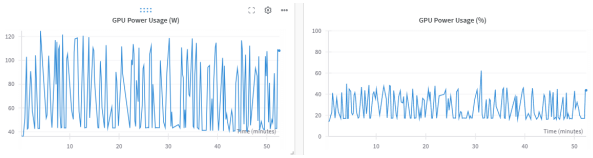

Fig. 8. GPU related metrics - ResNet50

B. BYOL

While searching for different methods to improve our results, we found the paper "Bootstrap your own latent-a new approach to self-supervised learning" [21], which introduces the BYOL model.

1) Theoretical Aspects: Bootstrap Your Own Latent (BYOL) is a self-supervised learning framework designed to learn effective representations without requiring labeled data. Unlike other contrastive learning methods such as SimCLR or MoCo, BYOL does not rely on negative samples or contrastive losses.

The principal key features of BYOL are the following:

- **No Negative Samples: BYOL avoids the need for explicit negative pairs, simplifying training.**
- **Two Networks:**
  - **Online Network:** Consists of an encoder, projector, and predictor that learns representations.
  - **Target Network:** A stable network updated via a momentum-based moving average of the online network.
- **Learning Objective:** The online network predicts the target network's features for augmented views of the same image.
- **No Labels Required:** BYOL is entirely self-supervised, learning from augmented data alone.

The training process can be summarized as follows:

1) **Augmented Views:** Two augmented versions of the same image are created using random transformations.
2) **Feature Extraction:** Both views are passed through the online and target networks to generate representations.

3) **Prediction:** The online network predicts the target network's representation for the corresponding view.
4) **Loss Function:** Minimize the distance between the online prediction and the target network representation.
5) **Network Updates:**
   - **Online Network:** Updated using backpropagation.
   - **Target Network:** Updated via exponential moving average of the online network's weights.

2) Implementation: As we observed that this model performed well on CIFAR-100, we decided to experiment with this approach as well. For this, we utilized the model from https://github.com/lucidrains/byol-pytorch. Following the instructions provided in the repository, we first installed the package using *pip install byol-pytorch*.

We decided to train the learner using the ResNet50 model. To evaluate its performance on CIFAR-100, we conducted two implementations: one using the standard image dimensions for CIFAR-100 (32x32) and the other by adding an upsampling layer. Constrained by memory and GPU usage, we used an upsampling layer to increase the input dimensions to 128x128. This adjustment aimed to strike a balance between leveraging the larger input size for improved feature extraction while accommodating our hardware limitations.

In addition, we attempted to adapt the ResNet50 model to the 32x32 image dimensions without upsampling. To achieve this, we modified the following two layers of the ResNet50 architecture:

```
model.conv1 = nn.Conv2d(3, 64, kernel_size=3,
    stride=1, padding=1, bias=False)
model.maxpool = nn.Identity()
```

The first modification, replacing the initial convolutional layer, was necessary because the original ResNet50 model's `conv1` layer was designed for larger image sizes, with a kernel size of 7 and a stride of 2. These parameters are unsuitable for the smaller CIFAR-100 images as they reduce the spatial resolution too aggressively. By changing the kernel size to 3 and the stride to 1, we ensured that the initial layer captures finer details from the smaller input dimensions without excessively downsampling the feature map.

The second modification, replacing the max-pooling layer with an identity layer, was made to preserve more spatial information in the early stages of the network. The original max-pooling operation, with a stride of 2, would further reduce the resolution of the feature map, which is already limited by the smaller input size of CIFAR-100. By substituting it with an identity layer, we maintained the spatial resolution, allowing the network to learn richer representations from the dataset.

Regarding the configurations used when training, we employed the following two configurations:

1) **Original BYOL:**
   - Model: ResNet50 without upsampling layer;
   - Optimizer: Adam, with lr = 3e-4;
   - Image size: 32x32;
   - Hidden layer: global_pool;
2) **BYOL Resized:**
   - Model: ResNet50 with upsampling layer;
   - Optimizer: AdamW, using lr=0.001 and weight_decay=0.0005;
   - Image size: 128x128;
   - Hidden layer: global_pool;

The optimizer used for the **Original BYOL** is the one provided in the repository, while the optimizer in the second configuration was chosen due to its effectiveness in other cases. The `global_pool` layer was used as the hidden layer to efficiently aggregate spatial features into compact global representations, simplifying downstream processing and ensuring robustness to spatial transformations. In both cases, we trained the models for 50 epochs.

3) Experiments: All experiments from this section were conducted using a system with 32 GB of RAM, an Nvidia GeForce RTX 3060 Ti GPU with 8 GB of VRAM and an AMD Ryzen 7 5800X processor.

Using the two configurations described above, we performed fine-tuning with different setups, as shown in Figures 9, 10, 11, and 12. These plots illustrate the evolution of accuracy and loss on the training and test sets. The augmentation schemes *randaugment* and *Combined Resize 2* refer to the two schemes described in the previous subsection. The term *basic* indicates that only CutMix and MixUp were used (with an alpha value of 0.8), along with normalization. The *Combined* scheme has the following structure:

- `CutMix`
- `MixUp`
- `RandomResizedCrop(32, scale=(0.8, 1.0))`
- `RandomHorizontalFlip()`, `v2.RandomRotation(15)`
- `ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1)`
- `RandAugment()`
- `ToImage()`
- `ToDtype(torch.float32, scale=True)`
- `Normalize((0.5, 0.5, 0.5), (0.25, 0.25, 0.25))`

The training loss plot shows a gradual decrease in loss across all configurations, indicating successful learning. Among the configurations, the setup using AdamW with a learning rate of 0.0005, CosineAnnealingLR, and the *basic* augmentation scheme achieves the lowest loss by the end of training, suggesting it is the most effective configuration in minimizing training loss. On the other hand, the configurations using *randaugment* show slower training progress, indicating a trade-off between regularization and faster convergence speed. Configurations with the *Combined Resize 2* augmentation scheme show mixed results, with those using a smaller learning rate (0.0005) exhibiting more stable performance in terms of both loss reduction and accuracy growth compared to those using a learning rate of 0.001. The configuration using the *Original BYOL* setup exhibits slower accuracy growth and

reaches the lowest final accuracy, suggesting it underperforms compared to the modified setups.

Fig. 9. BYOL - Accuracy on the test set

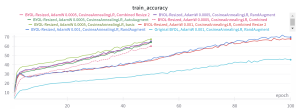

Fig. 10. BYOL - Accuracy on the train set

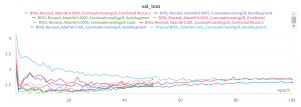

Fig. 11. BYOL - Loss on the test set

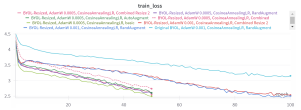

Fig. 12. BYOL - Loss on the train set

The validation loss plot reveals the generalization performance of the model on unseen data. All configurations show a significant decrease in loss during the initial epochs, followed by a stabilization phase. Among the configurations:

- The setup using AdamW with a learning rate of 0.0005, CosineAnnealingLR, and the *basic* augmentation scheme demonstrates consistently low validation loss, suggesting it generalizes well.
- Configurations using *Combined Resize 2* with AdamW and learning rates of 0.0005 and 0.001 show slightly higher and more oscillatory validation losses compared to the *basic* configuration.
- The *Original BYOL* configuration with AdamW (learning rate 0.001) and *randaugment* achieves a higher validation loss, indicating weaker generalization performance relative to the modified setups.

The results suggest that a smaller learning rate (0.0005) paired with simpler augmentation schemes achieves better stability and generalization.

The validation accuracy plot indicates the progression of accuracy on unseen data for the different configurations. Key observations include:

- The configuration using AdamW with a learning rate of 0.0005, CosineAnnealingLR, and the *basic* augmentation scheme achieves the highest validation accuracy, confirming its effectiveness for generalization.
- Configurations using *randaugment* (e.g., the *Original BYOL* setup) exhibit slower growth and lower final validation accuracy, reflecting the challenging nature of stronger augmentations.
- Configurations with *Combined Resize 2* show competitive validation accuracy but fail to outperform the *basic* scheme, likely due to the added complexity of the augmentation pipeline.

The validation results reinforce the trends observed in the training plots:

- The configuration with a learning rate of 0.0005, CosineAnnealingLR, and *basic* augmentation demonstrates the best balance between training and validation performance.
- Stronger augmentations such as *randaugment* introduce additional regularization, which may lead to slower accuracy growth and higher validation loss.
- Smaller learning rates consistently result in better stability and generalization.

Overall, these findings suggest that simpler augmentation schemes and fine-tuned learning rates are key to achieving both efficient learning and strong generalization performance on the test set.

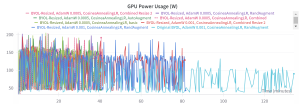

Fig. 13. GPU Power Consumption

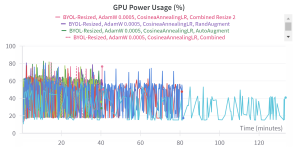

Fig. 14. GPU Usage

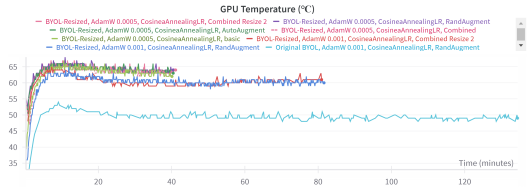

Fig. 15. GPU Temerature

The GPU temperatures plot (Figure 15) shows that all configurations stabilize after the first 15–20 minutes. Most configurations, including those using *Combined Resize 2* and *basic* augmentations, stabilize around 60–65°C. In contrast, the *Original BYOL, AdamW 0.001, CosineAnnealingLR, randaugment* configuration maintains a lower temperature of around 50–55°C, indicating less resource utilization.

Figure 14 reveals significant fluctuations across all configurations. The setups using *BYOL Resized*, such as those using *basic* augmentations or *Combined Resize 2*, consistently exhibit higher utilization, often peaking near 100%. The *Original BYOL* configuration shows lower and less consistent GPU usage, further reflecting lower computational demand.

The GPU power usage plot (Figure 13) demonstrates that configurations which use *BYOL Resized* consume more power, peaking at around 150–200 W. In contrast, the *Original BYOL, AdamW 0.001, CosineAnnealingLR, randaugment* configuration shows reduced power consumption, aligning with its lower GPU usage.

C. ELR+

1) Theoretical Aspects: As mentioned at the beginning of this paper, one approach that has demonstrated favorable results on the Noisy CIFAR-100 dataset is the ELR+ method, as presented in [20].

The ELR+ (Early-Learning Regularization Plus) method is designed to mitigate the negative effects of noisy labels during training. It introduces a regularization term that discourages the network from memorizing incorrect labels, leveraging the observation that neural networks tend to learn clean labels earlier in the training process. By combining this regularization with standard cross-entropy loss, ELR+ effectively balances early learning and robust generalization. This makes it particularly effective for datasets with label noise, such as Noisy CIFAR-100.

2) Implementation: The implementation of the ELR+ method in our work follows the theoretical foundation described in [20], incorporating key features that address the challenges posed by noisy labels.

Central to the implementation is the integration of a dual-model architecture, where two networks are trained simultaneously, providing mutual feedback to improve robustness against noisy labels. A key component is the regularization term introduced by the ELRLoss, which discourages memorization of noisy labels. The loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \cdot \mathcal{L}_{\text{reg}},$$

where $\mathcal{L}_{\text{CE}}$ represents the standard cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{y}_{ij},$$

and $\mathcal{L}_{\text{reg}}$ is the regularization term that penalizes memorization of noisy labels:

$$\mathcal{L}_{\text{reg}} = \frac{1}{N} \sum_{i=1}^{N} \log \left(1 - \sum_{j=1}^{C} \hat{y}_{ij} h_{ij} \right).$$

Here, $\hat{y}_{ij}$ is the softmax output probability for sample i and class j, h_{ij} represents the historical predictions for the sample, and λ controls the regularization strength.

To further enhance learning, data augmentation techniques such as MixUp and CutMix are employed to increase the diversity of training samples and encourage smoother decision boundaries. For Mixup, the augmented inputs and targets are generated as:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \quad \tilde{y} = \lambda y_i + (1 - \lambda) y_j,$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$. CutMix is also applied, where regions of one image are replaced with corresponding regions from another image, with a recalculated target:

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j, \quad \lambda = 1 - \frac{\text{area of the patch}}{\text{area of the image}}.$$

An exponential moving average (EMA) is maintained for both models during training, defined as:

$$\theta_{\text{EMA}} = \alpha \cdot \theta_{\text{EMA}} + (1 - \alpha) \cdot \theta,$$

where α controls the smoothing rate and θ_{EMA} and θ represent the parameters of the EMA model and the current model, respectively.

The implementation also incorporates efficient dataset handling and augmentation pipelines, ensuring scalability and reproducibility for noisy datasets like Noisy CIFAR-100.

Overall, the implementation is designed to combine the strengths of early-learning regularization, augmentation strategies, and stabilization techniques to achieve robust learning in the presence of noisy labels.

3) Experiments: All experiments from this section were conducted using a system with 32 GB of RAM, an Nvidia GeForce RTX 3060 Ti GPU with 8 GB of VRAM and an AMD Ryzen 7 5800X processor.

When conducting our experiments, we started with the configuration proposed in the paper and on the GitHub repository (https://github.com/shengliu66/ELR). Specifically, we used a PreActResNet18 model with MixUp augmentation, SGD as the optimizer (learning rate = 0.02, momentum = 0.9, weight_decay = 5e-4), and a MultiStepLR scheduler with a step at 200 epochs and gamma = 0.1. This configuration achieved a test accuracy of 49.15%. However, after 100 epochs, both the training and testing accuracy began to stagnate, prompting us to stop the process.

By changing the optimizer to AdamW, we achieved an improved accuracy of 51.82%. Furthermore, using the same AdamW optimizer and replacing the scheduler with CosineAnnealingWarmupRestart (with $T_0 = 10$, $T_{\text{mult}} = 2$, and $\eta_{\text{min}} = 1 \times 10^{-5}$), we achieved an accuracy of 50.13%. Notably, this configuration was significantly faster: the accuracy was reached after only 5 epochs, whereas the previous configurations required 22 epochs to achieve a similar level of performance. However, the configuration using CosineAnnealingWarmupRestart exhibited constant fluctuations in training accuracy.

We also conducted experiments using ResNet18 with an upsampling layer, testing configurations that resized inputs to either 224x224 or 128x128. Additionally, some configurations incorporated CutMix as a supplementary augmentation technique alongside randaugment. It was observed that the majority of experiments utilizing the model with the upsampling layer started with higher test accuracy, except for the configuration using CutMix, randaugment, and OneCycleLR, which did not demonstrate the same initial performance advantage. However, this method consistently improved its performance, while others gradually declined or stagnated.

Fig. 16. ELR+ - Accuracy on the train set

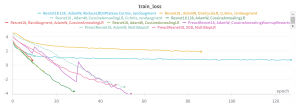

Fig. 17. ELR+ - Loss on the train set

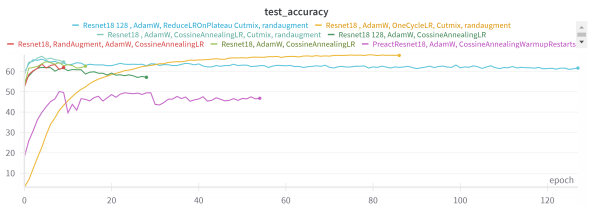

Fig. 18. ELR+ - Accuracy on the test set

D. EfficientNet

As presented in the previous section, the best approach was achieved using a pretrained EfficientNet B0 model trained with the Noisy Student technique. However, before reaching this level of performance, we conducted various experiments, which we will detail in the following sections.

All experiments from this section were conducted using a system with 32 GB of RAM, an Nvidia GeForce RTX 3060 Ti GPU with 8 GB of VRAM and an AMD Ryzen 7 5800X processor.

1) Simple EfficientNet: We initially experimented with different configurations using pretrained EfficientNet models. The results are summarized in Table I and will be discussed in detail below.

Table I summarizes the results of various configurations of EfficientNet models on the Noisy CIFAR-100 dataset. We note that the augmentation schemes used include CutMix and MixUp, unless explicitly stated otherwise. When CutMix and/or MixUp are not applied, it will be explicitly mentioned.

Below, we provide an analysis of the key factors affecting model performance, such as the choice of augmentation scheme, warmup strategy, and model size:

- **Effects of learning rate:** The choice of learning rate plays a crucial role in model performance. Increasing the learning rate to 0.01, as seen in the EfficientNet B0 and B3 configurations with randaugment, results in significantly lower accuracy (41.88% and 63.61%, respectively). This suggests that a higher learning rate may lead to unstable training or overfitting, particularly on noisy datasets.
- **Augmentation strategies**:
  - **randaugment:** This augmentation consistently improves accuracy. For example, EfficientNet B0 with Adam and randaugment achieves 65.16%, outperforming the Basic augmentation configuration (63.72%).
  - **Combined:** Introducing a combination of randaugment and other techniques, such as CutMix, leads to lower accuracy (56.89%) compared to using randaugment alone. This suggests that overly complex augmentations may not generalize well on noisy datasets.
- **Impact of warmup duration:** Introducing a warmup strategy improves training efficiency. For example, using a warmup period of 5 epochs with randaugment allows EfficientNet B0 to reach 63.69% accuracy in just 142.62 seconds, compared to 218.46 seconds with a warmup of 3 epochs. This demonstrates that warmup strategies can stabilize training and accelerate convergence.
- **Effects of model size:** Scaling up the model to EfficientNet B3 does not necessarily lead to significant improvements in accuracy. For instance, EfficientNet B3 with AdamW (0.01) and randaugment achieves an accuracy of 63.61%, which is comparable to EfficientNet B0 under similar conditions (65.16%) but at a slightly faster convergence time (171.99 seconds vs. 218.46 seconds). This suggests that increasing model complexity does not always yield better results, particularly on noisy datasets.

2) EfficientNet using Noisy Student: The models used in this study are pretrained on ImageNet, which requires input images to match the standard dimensions of 224×224. To ensure compatibility, we applied the resizing to 224×224 for all configurations, except for a specific case where the resizing to 128×128 was performed. This exception was made for a configuration run on Kaggle, where RAM limitations did not permit larger image sizes.

Where there is no value for alpha (the parameter for CutMix and MixUp), its value is 0.8.

For all configurations, MixUp and CutMix augmentations were applied unless explicitly mentioned otherwise (e.g., the entry with "randaugment (Without Mixup)"). In addition, the entries marked with an asterisk (*) indicate that a different normalization scheme was used. Instead of `v2.Normalize((0.5, 0.5, 0.5), (0.25, 0.25, 0.25))`, the normalization scheme `v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))` was applied, which is standard for ImageNet-pretrained models.

This alternative normalization is beneficial as it aligns with ImageNet statistics, ensuring better compatibility with the pretrained weights and potentially improving convergence. For the remaining configurations, the custom normalization tailored for the Noisy CIFAR-100 dataset was used to adjust to its data distribution.

The term *Self* in the table refers to a custom training loop in which pseudo-labeling is employed to enhance training performance. In this approach, the model itself is used to generate pseudo-labels for unlabeled or weakly labeled data during training. Only predictions with confidence above a specified threshold (e.g., 0.9) are used, and these pseudo-labeled samples are combined with the original training data in subsequent epochs. This technique should allow the model to leverage additional data effectively, improving robustness and performance, especially in noisy or semi-supervised learning scenarios. However, we did not succeed in achieving this goal.

Table II summarizes the results of different configurations for EfficientNet B0 Noisy Student on the Noisy CIFAR-100 dataset. The configurations vary by optimizer, learning rate, warmup strategy, scheduler, augmentation, and normalization. Below, we analyze the impact of these choices on model

TABLE I
ABLATION STUDY OF EFFICIENTNET CONFIGURATIONS ON NOISY CIFAR-100

Model	Optimizer	Augmentation	Warmup	Accuracy (%)	Time until best accuracy (s)
EfficientNet B0	Adam 0.001	Basic	3	63.72	698.53
EfficientNet B0	Adam 0.001	randaugment	3	65.16	218.46
EfficientNet B0	Adam 0.01	randaugment	3	41.88	494.90
EfficientNet B0	Adam 0.001	randaugment	5	63.69	142.62
EfficientNet B0	Adam 0.001	Combined	5	56.89	166.57
EfficientNet B3	Adam 0.01	randaugment	3	63.61	171.99

TABLE II
ABLATION STUDY ON EFFICIENTNET B0 NOISY STUDENT CONFIGURATIONS

Optimizer	Learning Rate	Warmup	Scheduler	Augmentation	Model	Accuracy (%)	Time to Best Accuracy (s)
Adam	0.0003	3	CosineAnnealingLR	randaugment (Without MixUp)	EfficientNet B0 Noisy Student	70.91	674.02
Adam	0.0003	3	ReduceLROnPlateau	randaugment	EfficientNet B0 Noisy Student	71.43	1081.98
Adam	0.0003	10	ReduceLROnPlateau	randaugment	EfficientNet B0 Noisy Student	72.78	1128.28
Adam	0.0003	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	73.35	1856.13
Adam	0.0001	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	74.18	1837.44
Adam	0.0001	5	CosineAnnealingWarmRestarts	randaugment (Alpha = 1)	EfficientNet B0 Noisy Student	73.81	1660.627
Adam*	0.0001	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	74.32	3318.19
Adam*	0.0001	5	CosineAnnealingWarmRestarts	randaugment (NumOps=1)	EfficientNet B0 Noisy Student	74.46	2589.06
Adam*	0.0001	5	CosineAnnealingWarmRestarts	randaugment (Alpha=0.5, NumOps=1)	EfficientNet B0 Noisy Student	74.73	1798.48
Adam*	0.0001	5	CosineAnnealingWarmRestarts	AutoAugment+CutMix+MixUp (Alpha=0.5)	EfficientNet B0 Noisy Student	74.27	2015.54
Adam*	0.0001	5	CosineAnnealingWarmRestarts	randaugment (NumOps=3)	EfficientNet B0 Noisy Student	73.10	1654.44
AdamSelf	0.0003	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	71.80	747.46
Adam (resize to 128)	0.0001	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	72.62	854.67
AdamSelf	0.0001	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	72.15	861.33
Adam	0.0001	5	CosineAnnealingWarmRestarts	randaugment	EfficientNet B0 Noisy Student	74.32	3318.19
AdamW	0.0001	5	CosineAnnealingWarmRestarts	randaugment (Alpha=0.5)	EfficientNet B0 Noisy Student	74.44	2318.03

performance, focusing on best accuracy and the time required to achieve it.

1) **Impact of Optimizer:** The best accuracy is achieved with Adam (74.73%) using a learning rate of 0.0001, a warmup of 5 epochs, and CosineAnnealingWarmRestarts with randaugment (Alpha=0.5). We can observed that the same configuration using AdamW achieved a slightly lower accuracy, but the difference being that small, it can be concluded that they perform similarly in this case.
2) **Effect of Learning Rate:** Lower learning rates generally achieve higher accuracy compared to higher learning rates. For instance:
   - Adam (0.0001) with randaugment and 5 warmup epochs achieves 74.18% accuracy.
   - Adam (0.0003) with the same settings achieves 73.35%.

   This suggests that smaller learning rates help stabilize training and prevent overfitting on noisy datasets.
3) **Warmup Strategy:** A warmup period of 5 epochs consistently improves accuracy across configurations. For example:
   - Adam (0.0003) with 3 warmup epochs and CosineAnnealingLR achieves 70.91%, while the same configuration with 5 warmup epochs and CosineAnnealingWarmRestarts achieves 73.35%.
   - Increasing the warmup to 10 epochs yields further improvements, as seen in Adam (0.0003) with ReduceLROnPlateau and randaugment, achieving 72.78%. However, longer warmup periods come at the cost of additional training time.
4) **Augmentation Schemes:** randaugment consistently outperforms AutoAugment in terms of accuracy:
   - Adam* (0.0001) with randaugment achieves 74.32%, while the same configuration with AutoAugment achieves 74.27%.

   Adjusting augmentation parameters impacts performance:
   - Using a smaller alpha value for CutMix and MixUp (0.5) achieves the highest accuracy of 74.73%, while increasing it reduces the accuracy.
   - Increasing the number of operations (NumOps=3) slightly reduces accuracy to 73.10%, likely due to overly complex augmentations.
5) **Normalization:** Configurations marked with an asterisk (*) use ImageNet normalization (`v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))`), which aligns with the pretrained weights of EfficientNet B0. This leads to slightly higher accuracies compared to the custom normalization for Noisy CIFAR-100 (`v2.Normalize((0.5, 0.5, 0.5), (0.25, 0.25, 0.25))`). For instance:
   - Adam* (0.0001) with ImageNet normalization and randaugment achieves 74.32%, while Adam (0.0001) with the custom normalization achieves 74.18%.
6) **Self Configuration:** The term *Self* refers to a custom training loop in which pseudo-labeling is employed to enhance training performance. In this approach, the model generates pseudo-labels for unlabeled or weakly labeled data based on its confident predictions (confidence threshold of 0.9). These pseudo-labeled samples are then combined with the original training data for

subsequent epochs. However, results for the Self configuration show mixed performance:

- AdamSelf (0.0003) achieves 71.80% accuracy, which is lower than most standard configurations.
- AdamSelf (0.0001) improves to 72.15% accuracy but still lags behind the best-performing configurations.

This suggests that pseudo-labeling benefits might be limited for this specific dataset.

7) **Resizing Impact:** For most configurations, input images are resized to 224×224, consistent with the requirements of ImageNet-pretrained models. One exception is the configuration run on Kaggle, where images were resized to 128×128 due to RAM limitations. This configuration (Adam, 0.0001, 128×128) achieves 72.62% accuracy, which is slightly lower than configurations with standard resizing. This highlights the importance of maintaining input dimensions aligned with the pretrained model's expectations.
8) **Scheduler Comparison:** CosineAnnealingWarmRestarts generally leads to better accuracy (up to 74.73%) but requires more training time. For example:
   - Adam (0.0001) with CosineAnnealingWarmRestarts achieves 74.18% accuracy in 1837.44 seconds.
   - Adam (0.0003) with ReduceLROnPlateau achieves 71.43% accuracy in 1081.98 seconds.

   ReduceLROnPlateau offers faster convergence but sacrifices performance. For the schedulers we used the following configurations:
   - ReduceLROnPlateau: patience = 5, factor = 0.1, mode = "max";
   - CosineAnnealingWarmRestarts: $T_0 = 10$, $T_{\text{mult}} = 2$, and $\eta_{\min} = 1 \times 10^{-5}$;
   - CosineAnnealingLR : $\eta_{\min} = 1 \times 10^{-5}$, $T_{\max} = 100$;

Fig. 19. Training Accuracy on EfficientNet

Fig. 20. Testing Accuracy on EfficientNet

When analyzing Figure 20, we observe distinct trends in the evolution of training accuracy. Configurations that fail to achieve high accuracy are those utilizing EfficientNet without the Noisy Student technique. However, there are two configurations using Noisy Student that exhibit interesting patterns of fluctuation. The reasons behind these fluctuations differ: one is attributed to a learning rate of 0.0003, which appears to be too high, while the other is caused by the use of a label smoothing value of 0.3. In contrast, all other configurations use a label smoothing value of 0.1, which has proven to be the most favorable choice.

E. Using Pretrained Models

In this study, we also explored the effect of using pretrained models to address the challenges posed by the Noisy CIFAR-100 dataset, which is characterized by a fairly high noise level of 40%. We selected a series of well-known architectures, each pretrained for various tasks, and adapted them for the classification of the 100 classes in this dataset. The investigated models include:

- ResNet18 that was implemented using the pretrained model `hf_hub:grodino/resnet18_cifar10`
- ResNet34 implemented using `resnet34` pretained model
- ResNet50 configured using `resnet50`
- Wide ResNet 101-2 that was implemented with `wide_resnet101_2` pretrained model
- ConvNeXt-Tiny configured with `convnext_tiny`
- EfficientNet-B0 using `efficientnet_b0`

This section investigates the performance of these models in the noisy dataset, evaluating their generalization ability and robustness to label noise. In addition, we analyzed the effects of configuring dropout in some of these architectures to enhance robustness and regularization capacity.

1) ResNet18: This is a so-called lightweight residual network architecture consisting of 18 layers, well known for its efficiency and performance in scenarios involving limited resources. The model used in this study was sourced from the Hugging Face Hub (`hf_hub:grodino/resnet18_cifar10`) and was initially trained on the CIFAR-10 dataset. CIFAR-10 contains 10 distinct classes with small images (32x32), making it an ideal starting point for adaptations to similar datasets such as CIFAR-100. Due to its compact size, this model is suitable for applications with computational constraints.

Using this pre-trained model, we conducted a total of 3 experiments. With respect to dataset processing, the same augmentation was used for all three experiments. For the training dataset, a combination of RandomHorizontalFlip, RandomCrop, and normalization was applied. For the testing dataset, only normalization was applied. Another common aspect of the three experiments is the batch size: 64 for training (with shuffling) and 500 for testing.

Among the three experiments, the one that produced the best final validation accuracy of 35.09% used an SGD optimizer with a learning rate of lr = 0.001 and support for fused optimization (fused = True). The loss function used was CrossEntropyLoss. Training was carried out over 150 epochs, and it is worth noting that for this experiment, dataset processing included caching with runtime transforms implemented.

The second-best experiment achieved a lower validation accuracy of 26.69%. For this experiment, training was carried out in 20 epochs using the AdamW optimizer with a learning

rate of lr = 0.001, a weight decay of weight_decay = 1e-4, and CrossEntropyLoss as the loss function. A simple caching method was used, without runtime transforms.

For the third experiment, which also used the pre-trained ResNet18 model, a validation accuracy of 6.79% was obtained. This experiment is a variation of the previously mentioned one, where caching with runtime transforms was implemented.

None of these three methods achieved good validation accuracy and also struggled to learn the training dataset, failing to reach good accuracy values. An exception is the second method, which surpassed a training accuracy of 92%. As shown in the graphs below, for the experiment that produced the best validation accuracy, there is a point where the accuracy begins to oscillate around a constant value. This indicates that the model approached its maximum generalization capacity. The second experiment reached overfitting relatively quickly, as evidenced by the sharp decline in validation accuracy.

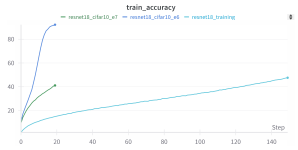

Fig. 21. ResNet18 - Train Accuracy

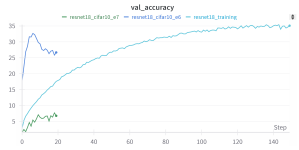

Fig. 22. ResNet18 - Validation Accuracy

Regarding the system's behavior during the execution of these three experiments, we have attached below some graphs showing the GPU utilization percentage as well as the CPU utilization percentage. GPU Utilization indicates the percentage of the GPU's effective usage for computations related to model training, and as can be observed, the experiment with the best score reached the highest GPU utilization, at 83%, while the other two methods remained below 60%. On the other hand, CPU Utilization monitors the percentage of CPU usage by the current process, which includes tasks such as data preprocessing, data transfer to the GPU, and layer coordination. As seen, all three experiments place minimal demands on CPU capacity, remaining below 40% for each of them.

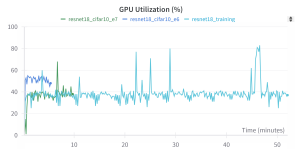

Fig. 23. GPU Utilization

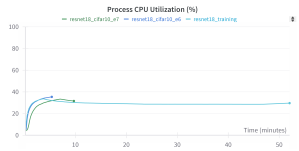

Fig. 24. CPU Utilization

2) ResNet34: It represents an extension of the ResNet18 architecture, with a higher number of layers, specifically 34, thus having greater learning capacity. The model used was pre-trained on ImageNet, which is a massive dataset containing millions of images grouped into 1,000 classes. This pretraining on ImageNet is designed to make the model a solid foundation for transfer learning on smaller datasets, such as CIFAR-100, helping capture complex image features.

Using the pretrained ResNet34 model, we conducted two experiments. For each of these experiments, the training dataset underwent augmentation: one simple augmentation involving RandomHorizontalFlip and RandomCrop, and an enhanced augmentation that added ColorJitter to the simple one. A common aspect of the two experiments is the batch size: 64 for training (with shuffling) and 500 for testing.

The experiment that achieved the highest validation accuracy, 39.25%, was trained for 20 epochs using the AdamW optimizer with a learning rate of lr = 0.001, a weight decay of weight_decay = 1e-4, and the simple augmentation method described earlier. The loss function used was CrossEntropyLoss, and the dataset caching method was simple, without implementing runtime transforms. As shown in the graphs below, this approach suffers from a serious overfitting problem.

In the second experiment, trained over 150 epochs, a validation accuracy of 21.82% was obtained. The enhanced augmentation method, including ColorJitter, was used, and runtime transforms were implemented for dataset caching. The optimizer used was SGD with a learning rate of lr = 0.01, a momentum of 0.9, and a weight decay of weight_decay = 5e-4. Additionally, a CosineAnnealingLR scheduler was applied, with CrossEntropyLoss as the loss function. For this experiment, an attempt was made to filter the noisy labels using KMeans clustering, with a total of 20 clusters and a threshold of 0.8. As seen in the graphs below, this approach is highly unstable, with accuracy values fluctuating significantly for both training and testing.

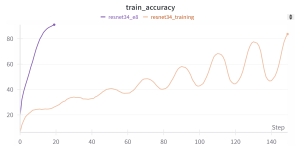

Fig. 25. ResNet34 - Train Accuracy

Fig. 26. ResNet34 - Validation Accuracy

Between the two experiments, the most unstable one also consumes the most resources, reaching a GPU Utilization value of 91%, while the CPU Utilization exceeds 40%.

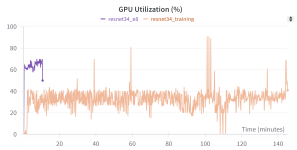

Fig. 27. GPU Utilization

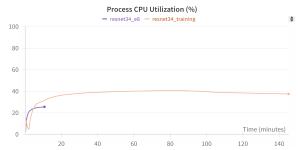

Fig. 28. CPU Utilization

3) ResNet50: This pretrained model, with its 50 layers, is among the most popular residual models due to its commitment to offering a balance between complexity and performance. This model was also pre-trained on ImageNet. In some experiments, it was configured with dropout to assess the impact of regularization on performance. Pre-training on ImageNet ensures a detailed understanding of visual features, which can make it quite effective for adaptation to the CIFAR-100 dataset.

Using the pretrained ResNet50 model, a total of 33 experiments were conducted, three of which I intend to highlight.

Among these, we want to present the experiment that achieved an accuracy of 45.29% delivered the best results. It utilized a complex set of augmentations: RandomHorizontalFlip, RandomCrop, and ColorJitter, followed by normalization for the training data, while only normalization was applied to the testing data. This ResNet50 model employs a dropout rate of 0.3 and a loss function, LabelSmoothingCrossEntropyLoss, with smoothing set to 0.1 to combat overfitting.

In addition, a MentorNet with four layers—one input layer and the others consisting of 512, 256, and 128 neurons, respectively—was used. The MentorNet was pretrained for 15 epochs using the Adam optimizer with a learning rate of lr = 1e-2, a weight decay of weight_decay = 5e-4, and BCEWithLogitsLoss as the loss function.

The optimizer for training was AdamW, with a learning rate of lr = 0.001, a weight decay of weight_decay = 5e-4, and a ReduceLROnPlateau scheduler to dynamically adjust the learning rate. Training was conducted over 50 epochs, with both training and testing data divided into batches of size 256.

With a result close to the one described earlier in terms of performance, the next experiment achieves an accuracy of 45.21% and uses similar augmentations, with the addition of a RandomRotation at an angle of 15 degrees, which can contribute to the diversity of the training data. The ResNet-50 model includes a dropout rate of 0.3, and the loss function used is CrossEntropyLoss with label smoothing set to 0.1.

An advanced augmentation, specifically mixup, is also applied to improve generalization. The optimizer used is AdamW with lr=0.001 and weight_decay=1e-4. The scheduler, CosineAnnealingLR, gradually adjusts the learning rate to maintain stability during training over 150 epochs. Early stopping is also applied to prevent overfitting.

However, the smaller batch size of 64 may contribute to a slight decrease in accuracy compared to the previous experiment.

For the final experiment, which achieved a validation accuracy score of 25.34%, the model was used without applying dropout, which can lead to overfitting. Additionally, simpler augmentations were employed, namely RandomHorizontalFlip and RandomCrop.

Furthermore, noisy examples were filtered with a threshold of 0.6, which significantly reduced the dataset size and may have removed relevant data. A batch size of 64 was selected for training, contributing to high variability during the training process, while a batch size of 500 was used for testing, with the training set also shuffled.

The optimizer used was AdamW with a learning rate of lr=0.001, a weight_decay of 1e-4, and the loss function employed was CrossEntropyLoss. The small number of 25 epochs also limited the model's ability to learn effectively.

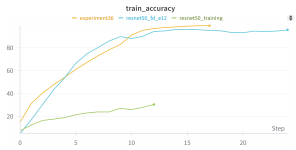

Fig. 29. ResNet50 - Train Accuracy

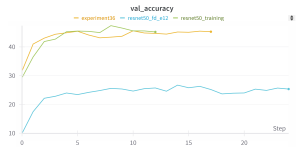

Fig. 30. ResNet50 - Validation Accuracy

4) Wide ResNet 101-2: This is an extended version of Wide ResNet 50-2, which has 101 layers. The model used was also pre-trained on ImageNet and aims to provide superior capacity due to the increased number of layers as well as the greater width. This architecture is designed to be ideal for working with complex datasets, such as Noisy CIFAR-100, where subtle details and label noise can negatively impact performance.

With this model, a single experiment was conducted, resulting in an accuracy value of 33.23% on the validation dataset. Data augmentation methods used included RandomHorizontalFlip and RandomCrop, and batch sizes of 64 for training and 500 for testing were allocated. No dropout layer was added to this model, which could lead to overfitting. The optimizer used was AdamW with a learning rate of lr = 0.001, a weight_decay of 1e-4, and the loss function applied was CrossEntropyLoss. As can be observed in the graph below, there is a tendency for overfitting to occur since, at a certain point, the validation accuracy values begin to fluctuate significantly.

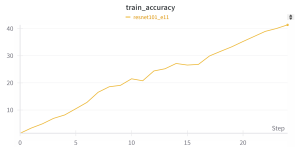

Fig. 31. ResNet101 - Train Accuracy

Fig. 32. ResNet101 - Validation Accuracy

5) ConvNeXt-Tiny: This is a modern model based on convolutions, combining the advantages of traditional CNN architectures with ideas inspired by transformer models. The model was pre-trained on ImageNet-1k. Its compact design makes it quite suitable for exploring noisy classification on CIFAR-100.

For this model, three experiments were conducted, with the best one achieving an accuracy of 55.39% after training for 6 epochs. The data augmentation methods used include RandomHorizontalFlip, RandomCrop, and ColorJitter. A large

batch size of 256 is used during training to provide stability in gradient updates and efficiently utilize the GPU. The data is shuffled before each epoch to prevent the model from learning the order of the samples, and a batch size of 256 is also used for testing. A dropout layer is added to the model to mitigate overfitting. The optimizer used is AdamW with a learning rate of lr = 0.0001, a weight_decay of 1e-4, and the loss function applied is LabelSmoothingCrossEntropyLoss with a smoothing parameter of 0.1. The ReduceLROnPlateau scheduler is used to automatically reduce the learning rate if the validation performance does not improve.

In the graphs below, a clear tendency for overfitting can be observed, as very high accuracy is achieved on the training set in just 6 epochs, while the accuracy on the validation dataset experiences a sharp drop

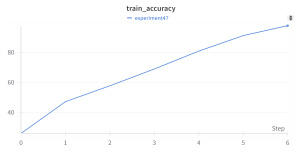

Fig. 33. ConvNeXt-Tiny - Train Accuracy

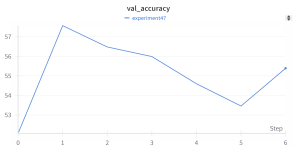

Fig. 34. ConvNeXt-Tiny - Validation Accuracy

As for the model's performance in terms of the system, it reaches a maximum GPU utilization of 93%, while GPU Memory Allocated remains below the 40% threshold.

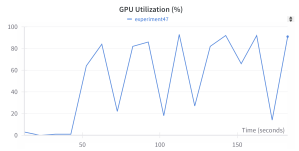

Fig. 35. GPU Utilization

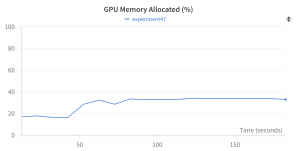

Fig. 36. GPU Memory Allocated

All the experiments described in the *Using Pretrained Models* section were run on a laptop equipped with 16 GB of RAM, an NVIDIA GeForce RTX 4050 GPU, and an AMD Ryzen 7 7435HS processor. These hardware specifications enabled the efficient execution of the implemented models and experiments, providing a balance between performance and available resources.

F. Custom Model

In this project, we also implemented a convolutional neural network (CNN) model for image classification on the CIFAR-100 dataset. The model includes some modern enhancements, such as Squeeze-and-Excitation (SEBlock) attention blocks, data augmentations, and advanced optimization techniques. In the following sections, we will detail the architectural features, preprocessing techniques, and training strategies employed.

1) Model Architecture: The convolutional network model consists of three main convolutional blocks, each followed by additional components aimed at regularization and performance improvement. Each convolutional block includes:

- A convolutional layer for extracting important features from the input images.
- Batch Normalization to accelerate training and reduce dependency on weight initialization.
- The ReLU activation function to introduce non-linearity.
- A Squeeze-and-Excitation (SEBlock) attention block, which allows the network to assign different importance to each channel based on the global context.
- A Max Pooling layer to reduce the spatial dimensions of the data.

After the three convolutional blocks, dense layers are used in the network for classification. The dimensions are flattened, and the extracted features are processed through two fully connected layers:

- The first fully connected layer has 512 neurons and uses dropout for regularization.
- The final layer produces predictions for the 100 classes in the dataset.

Thus, the model can be described as a combination of convolutional layers, SE blocks, and dense layers, all interspersed with regularization mechanisms such as Dropout and Batch Normalization.

For updating the model weights, the AdamW algorithm was used, which is considered more suitable for noisy datasets, with a learning rate of lr=1e-3, a weight_decay=1e-3, and CrossEntropyLoss as the loss function, adjusted by introducing label smoothing of 0.1. Additionally, the OneCycleLR scheduler was employed for dynamically adjusting the learning rate.

With this model, after training for 150 epochs, we achieved an accuracy of 37.58% on the validation dataset. The model is unstable, as can be observed from the attached graphs showing fluctuations in validation accuracy. One potential cause of this issue could be the training batch size of 64, which is relatively small, as small batches can lead to instability or insufficient learning. Other problems might stem from the insufficient complexity of the network architecture

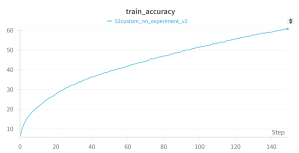

Fig. 37. CNN - Train Accuracy

Fig. 38. CNN - Validation Accuracy

The experiment described in the *Custom Model* section was run on a laptop equipped with 16 GB of RAM, an NVIDIA GeForce RTX 4050 GPU, and an AMD Ryzen 7 7435HS processor.

VI. Conclusion

In this study, we explored various strategies to enhance the performance of neural networks on the CIFAR-100N dataset, which incorporates asymmetric label noise. Our experiments demonstrated the effectiveness of combining advanced augmentation techniques such as RandAugment, CutMix, and MixUp with pretrained EfficientNet-B0 models. The integration of these techniques, alongside label smoothing and optimized hyperparameters, enabled us to achieve a peak validation accuracy of 74.73%. We were further able to optimize this value by using an ensemble method. Our findings also highlight the critical balance between regularization and augmentation intensity. While CutMix and MixUp with alpha=0.5 provided substantial benefits, increasing the alpha value degraded performance, emphasizing the importance of fine-tuning augmentation parameters to suit the characteristics of noisy datasets.

In summary, the methodologies presented in this paper establish a robust pipeline for training deep learning models on noisy datasets like CIFAR-100N. These results can guide future work in improving model generalization and robustness in the presence of label noise. Future research could explore additional noise-specific strategies and their synergy with the proposed pipeline to further enhance performance.

References

[1] Zhang Q, Zhu Y, Yang M, Jin G, Zhu Y, Lu Y, et al. (2024) An improved sample selection framework for learning with noisy labels. PLoS ONE 19(12): e0309841. https://doi.org/10.1371/journal.pone.0309841

[2] Wei, Qi, et al. "Debiased Sample Selection for Combating Noisy Labels." arXiv preprint arXiv:2401.13360 (2024).

[3] Wang, Haobo, Xiao, Ruixuan, Dong, Yiwen, Feng, Lei, Zhao, Junbo. (2022). ProMix: Combating Label Noise via Maximizing Clean Sample Utility. 10.48550/arXiv.2207.10276. .

[4] Wang, Yikai, Yanwei Fu, and Xinwei Sun. "Knockoffs-SPR: Clean Sample Selection in Learning With Noisy Labels." IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).

[5] Albert, Paul, Arazo, Eric, Krishna, Tarun, O'Connor, Noel, McGuinness, Kevin. (2023). Is your noise correction noisy? PLS: Robustness to label noise with two stage detection. 118-127. 10.1109/WACV56688.2023.00020.

[6] Ghosh, Aritra, Himanshu Kumar, and P. Shanti Sastry. "Robust loss functions under label noise for deep neural networks." Proceedings of the AAAI conference on artificial intelligence. Vol. 31. No. 1. 2017.

[7] Ma, Xingjun, et al. "Normalized loss functions for deep learning with noisy labels." International conference on machine learning. PMLR, 2020.

[8] Ye, Xichen, et al. "Active Negative Loss: A Robust Framework for Learning with Noisy Labels." arXiv preprint arXiv:2412.02373 (2024).

[9] L. Cui, H. Peng, Y. Li, C. Li and X. Xing, "Equal Loss: A Simple Loss Function for Noise Robust Learning," ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, Singapore, 2022, pp. 1950-1954, doi: 10.1109/ICASSP43922.2022.9747431.

[10] Zhou, Xiong, et al. "Asymmetric loss functions for learning with noisy labels." International conference on machine learning. PMLR, 2021.

[11] Han, Bo, et al. "Co-teaching: Robust training of deep neural networks with extremely noisy labels." Advances in neural information processing systems 31 (2018).

[12] Yu, Xingrui, et al. "How does disagreement help generalization against label corruption?." International conference on machine learning. PMLR, 2019.

[13] Wei, Hongxin, et al. "Combating noisy labels by agreement: A joint training method with co-regularization." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[14] Xie, Qizhe, et al. "Self-training with noisy student improves imagenet classification." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[15] Tarvainen, Antti, and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." Advances in neural information processing systems 30 (2017).

[16] Lee, Dong-Hyun. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks." Workshop on challenges in representation learning, ICML. Vol. 3. No. 2. 2013.

[17] Chen, Ting, et al. "A simple framework for contrastive learning of visual representations." International conference on machine learning. PMLR, 2020.

[18] He, Kaiming, et al. "Momentum contrast for unsupervised visual representation learning." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[19] Li, Junnan, Richard Socher, and Steven CH Hoi. "Dividemix: Learning with noisy labels as semi-supervised learning." arXiv preprint arXiv:2002.07394 (2020).

[20] Liu, Sheng, et al. "Early-learning regularization prevents memorization of noisy labels." Advances in neural information processing systems 33 (2020): 20331-20342.

[21] Grill, Jean-Bastien, et al. "Bootstrap your own latent-a new approach to self-supervised learning." Advances in neural information processing systems 33 (2020): 21271-21284.