Fix(Match)ing Up Unlabeled Data: CS 7643

Vimal Kumar Venugopal, John Dugan, Andrew Price, Caleb Goertel,
Georgia Institute of Technology
vvenugopal32@gatech.edu jdugan3@gatech.edu aprice63@gatech.edu cgoertel3@gatech.edu

Abstract

Learning from unlabeled data is an incredibly valuable skill in deep learning, due to the abundance of unlabeled data when compared to labeled data. As state of the art supervised models push high quality datasets to their absolute limit, it's clear that the data available is growing exponentially faster than high quality labels can be applied. Semi-supervised learning is the leveraging of labeled datasets to bootstrap near-labeled-quality learning out of unlabeled datasets. This project explores recent semi-supervised learning techniques that have quickly iterated on themselves over recent years. We implemented a version of FixMatch [8] and ran it through a multitude of experiments to determine how it is affected by different configurations, and why. Following are the implementation details, experimentation procedures, and comparisons to baselines as well as analysis and interpretation.

1. Introduction

Large, rich datasets are essential to the latest breakthroughs in the field of Deep Learning. High-quality datasets take an enormous amount of human and technological resources to acquire, as the curation, verification, and manual classification process is quite involved. In order to overcome these challenges at scale, new approaches have emerged to remove the amount of full supervision needed. Several techniques in the Unsupervised learning family of algorithms are able to cluster groups of data into separate classes, for example, but robust classification can be tricky at large scales. Weakly-supervised methods attempt to strike a balance between these two extremes. Semi-supervised learning is a specific form of weak supervision which involves taking a supervised labeled fraction of the overall dataset along with the remaining portion of unlabeled data to train on. We explore some of the latest techniques in semi-supervised learning on image data to understand how they work and what the tradeoffs are across a variety of experiments.

Over the last few years, several groundbreaking achievements have been achieved in the realm of semi-supervised learning. MixMatch [2] introduced new methods of classifying the unlabeled images by using stochastic techniques to augment the source image to use as a classifier prediction and unified loss term. ReMixMatch [1] was a followup work that added new techniques for distributing and predicting the unlabeled data, as well as an overall more efficient use of data. FixMatch [8], a major area of focus for this paper, further builds upon insights from ReMixMatch and introduces a new dual augmentation flow for the unlabeled examples. There are still several limitations with the state of the art approaches however. The size of the datasets and compute resources required to train the models are substantial, and are not easily accessible on consumer-grade hardware.

Since we are working with a limited hardware budget, we hope to explore the areas that have the greatest balance between data and compute requirements when compared to the final testing accuracy. For example, if we can find the point of diminishing returns in the training process, this can help further researchers focus their efforts on the most impactful areas of the model training, hyperparameter values, data requirements, etc. Additionally, we would like to see image-based semi-supervised learning techniques become more accessible on consumer-grade hardware as few people have access to the substantial compute resources utilized in the FixMatch paper for example.

For our final experimentation we used the popular image dataset CIFAR-10 [7] that is commonly used for research. CIFAR-10 has 10 classes of 32x32 color images, with 50,000 images in the training set and 10,000 in the test set. The classes can be summarized as vehicular (airplane, automobile, ship, truck) and animal (bird, cat, deer, dog, frog, horse) and there are no overlapping classes for any of the images. Since humans are excluded, there should not be any privacy concerns, discrimination, or other confidential data included.

2. Approach

Our approach was to adapt existing Pytorch implementations of FixMatch [5], integrate the key component

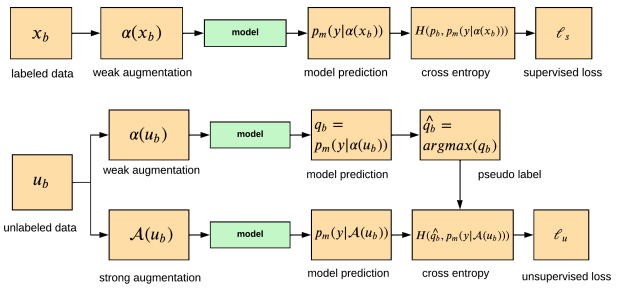

Figure 1. Diagram of FixMatch structure.

$\mathcal{X} = \{(x_b, p_b) : b \in (1, \ldots, B)\}$ – *batch of B labeled examples*
$\mathcal{U} = \{u_b : b \in (1, \ldots, \mu B)\}$ – *batch of μB unlabeled examples*
$\alpha(\mathcal{X})$ = *weakly augmented batch of B labeled examples*
$\mathcal{A}(\mathcal{U})$ = *strongly augmented batch of μB unlabeled examples*
$\alpha(\mathcal{U})$ = *weakly augmented batch of μB unlabeled examples*
Each batch = B weakly augm labeled examples + μB weakly augm unlabeled examples + μB strongly augm unlabeled examples

In our implementation, B = 64 & μ = 7, Therefore :
Each batch = 64 *weakly augm labeled examples* + 448 *weakly augm unlabeled examples* + 448 *strongly augm unlabeled examples*
Total Batch Size = 960

Figure 2. Batch size explanation

(MixUp) of MixMatch [6] and to combine them into a modular system for hot-swapping different techniques. No optimization or tuning was involved as the same hyperparameters from the FixMatch paper was reused. Our focus was to reproduce the results of Fixmatch using CIFAR10 [7] data & to run ablation experiments on the essential components of FixMatch. The structure for FixMatch is shown in Figure 1 which generates supervised and unsupervised loss. Implementation approach of each block will be explained below.

2.1. Labeled & Unlabeled Data

The CIFAR10 database is split into labeled x_b and unlabeled examples u_b. For experimentation, different amounts of labeled data (70, 250, 1000, 2500, 4000 etc), uniformly spread across all classes was selected i.e., $400\ examples\ per\ class\ X\ 10 classes\ = 4000\ labeled\ examples$. This uniform selection is required to allow the model to train evenly across all labeled classes and not overfit to the majority classes. Both the labeled & unlabeled data (after augmentation) are fed into the model in batches. B represents batch size of labeled data, while μB represents batch size of unlabeled data where μ is an unlabeled data ratio hyperparameter. Figure 2 shows the calculation of batch size per epoch, the model receives weakly augmented labeled data, weakly augmented unlabeled data & strongly augmented unlabeled data, all within the same batch.

2.2. Data Augmentation

FixMatch uses weak augmentation & strong augmentation to perturb images. This is based on the Semi-Supervised Learning (SSL) technique called Consistency Regularization which assumes that the model's prediction should not change when fed augmented versions of the same image.

Weak Augmentation: The weak augmentation $\alpha(.)$ is done on images with a random crop (shift) followed by a random horizontal flip. Both these image augmentations are stochastic with 50% probability. Weak augmentation is used on both labeled data before class prediction and unlabeled data before pseudo label generation. Weak augmentation is required so that our model doesn't overfit to training data.

Strong Augmentation: For strong augmentation $\mathcal{A}(.)$, RandAugment was implemented with a maximum distortion magnitude of 10. RandAugment randomly applies image augmentations from a predefined list of operations. Strong augmentation is used only on the unlabeled images in order to obtain perturbed images which are then used to calculate cross entropy from the pseudo labels. Thus Strong augmentation through RandAugment allows us to implement consistency regularization and determine unsupervised loss ℓ_u. Our implementation is different from FixMatch paper which utilizes both RandAugment & CTAugment techniques for Strong Augmentation. RandAugment & CTAugment are similar as they use the same collection of transformations, however, the distortion parameter for RandAugment is sampled from a predefined range whereas in CTAugment the distortion parameter is learned through training. RandAugment was fairly straight forward to implement as it requires no learned parameter. CTAugment was difficult to implement as the FixMatch paper did not clarify how the weights of the magnitude bins were to be updated, thus it was left out.

2.3. WideResNet28-2 Model

The implementation of WideResNet28-2 model with 1.47M parameters for CIFAR-10 was used as in the Fix Match paper [8]. Basic understanding of the model's shape and operations came from the original Wide ResNet paper [9]. Its structure is based around basic blocks of two 2D Convolutional layers with BatchNorm and ReLU (leaky ReLU in [8]). These blocks are then repeated four times in a row, which represents one block for a certain number of feature channels. Three of these larger repeated blocks are strung together, with expanding channel width each time,

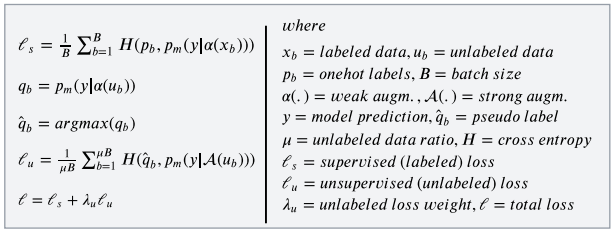

Figure 3. Loss Function Formula

hence the "Wide" Resnet. The 28 specification within 28-2 refers to the total number of convolutional layers, and the 2 represents a multiplicative widening factor denoting the width over the original design. These wider and shallower networks can achieve comparable results faster to thinner/deeper counterparts [9].

Our implementation uses ReLU instead of the official implementation's leaky ReLU. This is because the paper was lacking detail on leaky ReLU and our attempts to implement leaky ReLU in Pytorch lead to modeling errors. Stochastic gradient descent (SGD) optimizer with Nestrov momentum was implemented for training the model. The learning rate was decayed with a cosine function $\eta cos(\frac{7\pi k}{16K})$ where η is the learning rate, K is the total timestep and k is the current timestep. Simple weight decay was implemented by adding L2 penalty of all weights to model losss. The accuracy values were generated from the EMA (Exponential moving average) of the model parameters.

2.4. Loss Function

FixMatch makes use of both labeled data & prior knowledge of unlabeled data in the training process through the two cross entropy loss terms: supervised loss ℓ_s and unsupervised loss ℓ_u. Figure 3 shows the formulas for both the loss functions. The loss functions were implemented as the FixMatch paper [8].

Supervised Loss: ℓ_s is just the regular cross-entropy loss on weakly augmented labeled examples. The weak augmentation on labeled data is to avoid model overfitting.

Unsupervised Loss: Due to the semi-supervised setting, the supervised loss term alone is not sufficient to train the model parameters as the number of labeled examples will be less. Therefore in order to make use of the unlabeled data, we estimate the unsupervised loss through the following technique. First the weakly augmented unlabeled data $\alpha(u_b)$ is sent through the same model that was trained with labeled data to obtain predictions. From these predictions, pseudo labels are generated by keeping only predictions $>$ threshold τ. The threshold τ is needed to make the pseudo labels reliable (i.e., quality over quantity). The loss of the unlabeled data is computed by taking the cross entropy of the pseudo labels & strongly augmented unlabeled data.

Total Loss: The total loss minimized by the WideResNet28-2 model will be $\ell_s + \lambda_u \ell_u$ where λ_u is denotes the relative weight of unlabeled loss. Unlike ReMixMatch [1] where λ_u is annealed, for FixMatch, the λ_u is set to constant 1 as the threshold in pseudo labeling has similar effect.

3. Experiments and Results

The experiments where run on a single V100/A100 GPU from Google Colab with each experiment taking an average of 5 hours. Due to limited computational complexity, we executed the experiments on a 40% reduced CIFAR10 database. We attempted to recreate as many of the experiments from the original paper [8] as possible to test reproducibility when dealing with limited data due to computational constraints.

3.1. Quantity of Labeled Data

An important trade-off in FixMatch is the the decision of how much labeled data to use in training. Labeled data is much more expensive than unlabeled, but gives improved performance. The goal is to find an amount of labeled data that provides high accuracy performance at a low cost. Figure 4 shows the test accuracy curves for varied amounts of labeled data across 200 epochs of FixMatch. As the number of labeled data included increases, the testing accuracy improves, however there are diminishing returns with only small improvements in accuracy ocurring between 1000 labeled data and subsequently larger amounts. Because of this, downstream ablation experiments will only use 1000 labeled datapoints for training. In addition, Figure 4 shows that epochs 100-200 provide little improvement to our model's test accuracy. Due to limitations in time and using consumer hardware, downstream experiments will only use 100 epochs. When cutting off the 1000 label dataset training at 100 epochs, the model has a high testing accuracy of 94.18%. The final additional consideration made for improved computational performance was to utilize a reduced version of the CIFAR-10 dataset, consisting of 40% of the total data points. Figure 5 shows that only using 40% of the data does lower testing accuracy, since it has less variety of data to train on, but the resulting accuracy curves are the same. Therefore, we theorize that using the smaller 40% reduced dataset can be used a proxy for the larger dataset in our experiments and will allow us to reach the same conclusions with less computational run-time.

3.2. Strong Augmentation Ablation

Our implementation for Strong Augmentation consists of RandAugment [3] followed by CutOut [4] while the FixMatch paper adds CTAugment too. RandAugment [3] is

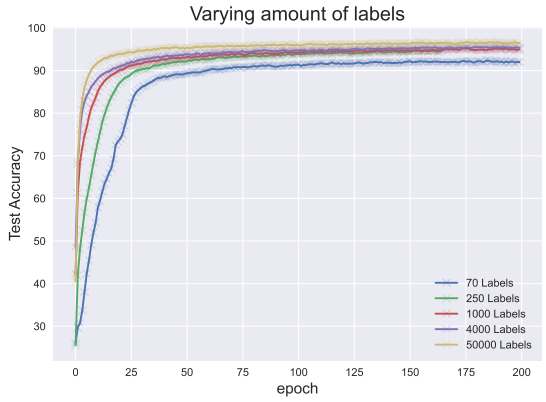

Figure 4. Number of labels Ablation: Higher numbers of labels result in better accuracy

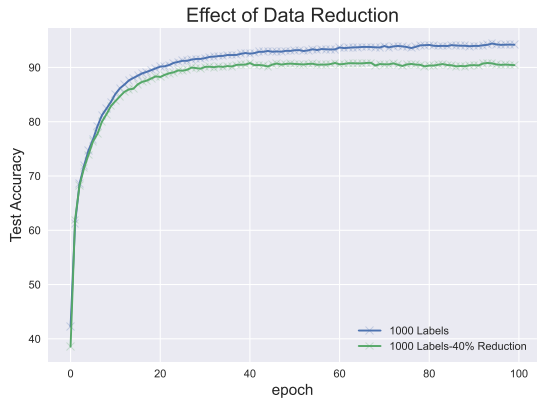

Figure 5. Comparison of Full CIFAR-10 and 40% Reduced Version

a parameter-free procedure which selects from K transformationss with 1/K uniform probability. Cutout [4] randomly masks out square regions of input during training. Both RandAugment & Cutout are regularization techniques which augments image to make it more difficult to learn while also encouraging the model to learn general patterns. Three ablation experiments were run for Strong Augmentation - 1) Keeping CutOut alone 2) Keeping RandAugment alone & 3) Regular baseline consisting of both RandAugment & Cutout. Figure 6 shows that both Cutout & RandAugment are required to obtain optimal performance and removal of either results in significant increase in error rate. This could be because Cutout & RandAugment are distinct augmentation strategies and each strategy allows the model to learn unique patterns from the unlabeled dataset.

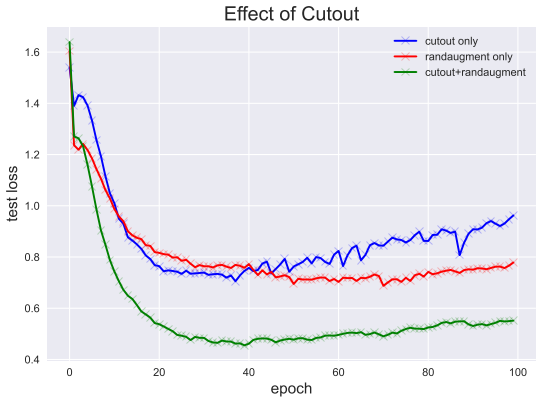

Figure 6. Strong Augmentation Ablation: Using CutOut & RandAugment separately caused a significant increase in loss

3.3. Pseudo-label Threshold

The pseudo-label threshold τ is used to determine the level at which the weakly-augmented image's predicted output is used as the pseudo-label for the strongly-augmented image's cross-entropy loss calculation. If the threshold is too low then weakly-augmented predictions could potentially be mis-classified compared to the ground truth classifications and a threshold value too high could prevent the correct weakly-augmented predictions from being utilized in the loss function. Figure 7 illustrates a range of τ values and the testing accuracy over time. As mentioned in the FixMatch paper, a τ of 0.95 appears to be the optimal balance of leniency vs confidence. Our 1.0 τ experiment performed substantially worse than the FixMatch result however. Since we are only using 40% of the available image data we likely ran into a scenario where we did not have enough weakly augmented images that were confidently predicted at the 1.0 level, and as a result our overall learning was reduced.

3.4. Pseudo-label Sharpening

In models that utilize softmax, the temperature hyperparameter controls the confidence in the predictions. At lower temperature levels, the softmax function approaches a one-hot encoding, causing the sharpness of the model's prediction confidence to increase. In FixMatch, the temperature helps control the confidence of the pseudo-labels and is used in conjunction with the threshold parameter above. By pushing the probabilities to favor the highest scoring label, the training loss is reduced, which could impair the model's ability to accurately learn from the data. This can be seen as a form of overfitting. Because of this, it is hypothesized that higher temperature values will have improved performance. Too much sharpening could also be problematic if the resulting predictions are not confident enough to meet

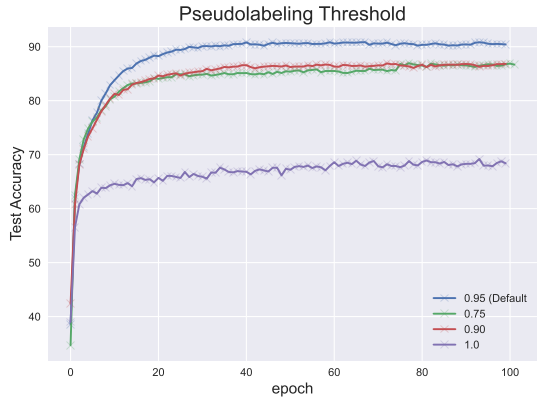

Figure 7. Threshold Ablation: The highest predicted class probability is above the threshold

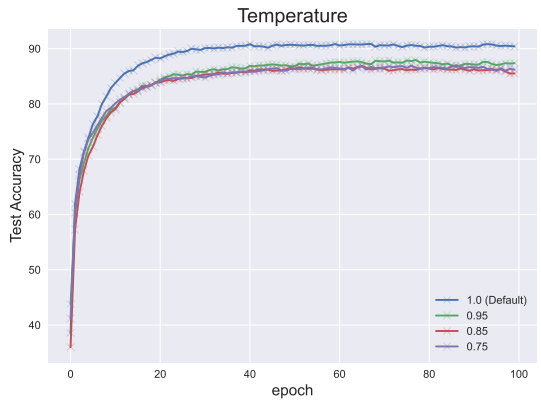

Figure 8. Temperature Ablation: Effect of Adjusting Softmax Temperature on Model Accuracy

the threshold, preventing that data from being used to calculate the loss function. Figure 8 shows that as the temperature value approaches 1.0, which FixMatch uses as their optimal value, the test accuracy improves. There is a substantial drop-off in accuracy between 0.85 and 0.75, showing that sharpening the model's predictions too much can be detrimental to it's performance on training data.

3.5. MixUp Augmentation Adjustment

An extension target for out re-implementation was the hybridization of the FixMatch model with previous/contemporaryh algorithm pieces to see if a combination could exceed the performance of its parts. The FixMatch algorithm is an iteration on the MixMatch [2] algorithm, but does not make use of the MixUp [10] subprocess, which mixed both labeled and unlabeled examples with the guesses from the unlabeled examples. By adding this procedure in after the augmentation step, we hoped to add robustness, as MixUp is known to regularize the training process by teaching linear mixtures of true examples [10]. The MixUp algorithm introduced two new hyperparameters, alpha and Temperature, but following the implementation in MixMatch, these were set at initial values and not altered, as they showed no improvement from tuning in that instance [2].

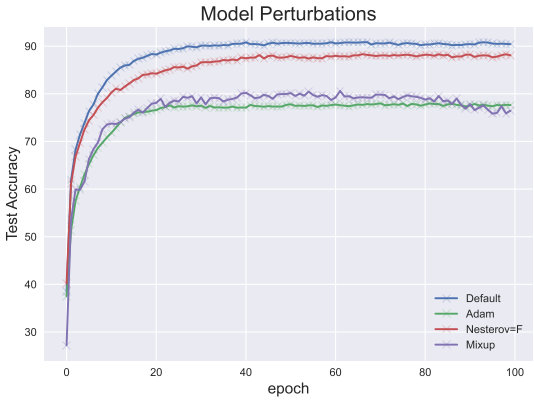

Figure 9. Additional Model Alterations

When implemented, however, the MixUp enhanced FixMatch performed far worse than baseline, as seen in Figure 9. On top of its poor peak performance, its learning curve was heavily unstable, and showed signs of overfitting when test accuracy began to decrease towards the end of training. The overfitting implies that the regularization was not powerful enough, so on revisiting additional hyperparameter tuning would have been useful in this newer environment. Additionally, locking augmentation types to the same ones used in MixMatch research could have yielded better results due to the structure of the augmentations matching the structure of the MixUp algorithm more closely.

3.6. Barely Supervised Learning

To understand the effects of only using a single labeled example for each class, we experimented with a single randomly selected image that will be used as the representation for the entire category. We had to run this experiment with a reduced batch size since the amount of labeled data was so small, and consequently this dramatically increased the training time. The FixMatch authors mention extreme variability in accuracy of their barely supervised runs from 48% - 85%, but our results were not even that successful. Figure 4 illustrates the stark difference between a single-label training run to only with only seven labels per class. The single label training only has minuscule improvement across epochs, which could be due to the reduced dataset size, poor random sampling of the single

Figure 10. Barely supervised ablation: Training with only 1 labeled example per class

image used, threshold value for the pseudo-label, or other hyperparameter values.

3.7. Additional Adjustments

One of the primary goals of our re-implementation was the ability to change some of the decisions made by the FixMatch authors and see how they affect model performance. The default implementation of FixMatch utilizes an SGD optimizer with Nesterov momentum to find the optimal parameters and solution. In machine learning, momentum accelerates the speed at which our solution approaches the optimal solution, which is particularly important because it speeds up the movement around regions with flat gradients, where models without momentum can get stuck. Because of this, it was hypothesized that removing the Nesterov momentum would result in a worse performing model. Figure 9 shows that removing Nesterov momentum, while keeping everything else the same, results in a decrease of testing accuracy from 90.425% to 88.075%, matching our original hypothesis. The next model alteration we tested was changing from an SGD optimizer to using Adam, which computes individual adaptive learning rates for each parameter. The FixMatch authors claim they tested both SGD and Adam before deciding on SGD, but we wanted to see how big of a difference this decision made. Figure 9 shows that switching the optimizer from SGD to Adam resulted in almost a 13% decrease in test accuracy. However, the default SGD model had a training error of 0.103, while the Adam model had a training error of 0.024. This mismatch in performance between the training and testing datasets points towards the Adam optimizer overfitting the training data and being unable to generalize the findings to previously unseen points. Adam's individual adaptive learning rates per parameter could be the source of the model overfitting that we see in our experiment.

4. Conclusions / Reflections

4.1. Hardware and data challenges

We quickly found that it was infeasible to perform any of the full experiments on consumer-grade hardware. Even with the small CIFAR10 dataset, training times were estimated on the order of weeks just to replicate a single experiment on our personal GPUs (RTX 3060/3070ti). Although we wanted to explore other datasets like CIFAR-100, SVHN, and STL-10, we decided to focus on the CIFAR10 dataset throughout experimentation in the interest of time and cost. Even with the reduced CIFAR10 dataset, training on the cloud A100 GPUs still took 4-12 hours per experimental run, and as a result any errors or configuration issues set the project back substantially.

4.2. Experimental Challenges

In addition to the general computational challenges, we also had several research avenues that did not ultimately pan out. The Fixmatch paper utilised an Exponential Moving Average (EMA) technique to smooth the model's parameters during training. Unfortunately, this required a fair amount of restructuring for our training and testing procedures and was ultimately deprioritized in favor of other experiments. ReMixMatch was another research avenue that got removed from the final results; we wanted to test out the data efficiency claims of the ReMixMatch algorithm vs the FixMatch implementation but the scope of the implementation was too great to be included in this paper.

4.3. Project Reflection

Our initial project goals were much grander than we ended up executing in the end, with many more experiments, model variations, and datasets implemented. Despite being heavily hampered by performance constraints, platform struggles, and implementation difficulties, we were still able to execute our re-implementation of FixMatch and conduct many relevant experiments to observe trends similar to the original paper. What's more is that we were able to turn a negative (being forced to operate on a small subset of the dataset for performance reason) into an opportunity to explore how this model functions in lower performance areas.

5. Work Division

Team contributions can be found in Table 1.

Student Name	Contributed Aspects	Details
Vimal Kumar Venugopal	1) Implemented CIFAR10 data preprocessing 2) Implemented data augmentations 3) Implemented WideResNet28-2 model 4) Implemented Loss Function 5) Setup Strong Augment Ablation Experiment	1) see Sect 2.1 for description and dataset_cifar.py in code 2) see Sect 2.2 for description and randaugment.py in code 3) see Sect 2.3 for description and wide_resnet.py in code 4) see Sect 2.4 for description and training.py in code 5) see Sect 3.2 for description and Experiments\Cutout
John Dugan	1) Initial project setup + notebook support 2) Researched existing FixMatch approaches 3) Pseudo-labeling threshold experimentation 4) Barely supervised experiment 5) Introduction and Technical Considerations sections	1) Initial repository, project, and Jupyter notebook setup 2) Analyzed existing implementations of FixMatch, assessed training time and scope 3) see Sect 3.3 4) see Sect 3.6 5) see Sect 1, Sec 4.1-4.2
Andrew Price	1) Assisted in getting existing FixMatch implementation running 2) Amount of Labeled Dataset Experiment 3) Varying Temperature Experiment 4) Ran varying Optimizer Experiments	1) Getting the existing implementation to run on our GPUs and package versions 2) see Sect.3.1 3) see Sect 3.4 4) see Sect 3.7
Caleb Goertel	1) Implemented of MixUp Augmentation 2) Ran MixUp Augmentation Experiment 3) Implemented MixMatch Algorithm 4) Researched WideResNet Architecture 5) Organized Deliverables	1) see Sect 3.5 for description and models/mixup.py in code 2) see Sect 3.5 for writeup, Fig. 9 for results 3) Unused in final experiments, implemented alongside FixMatch before deciding on which to focus 4) Determined type of network used, analyzed architecture for modification and adaptation to environment 5) Submitted and organized project proposal, report, and code submissions

Table 1. Contributions of team members.

References

[1] David Berthelot, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *CoRR*, abs/1911.09785, 2019. 1, 3

[2] David Berthelot, Nicholas Carlini, Ian J. Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *CoRR*, abs/1905.02249, 2019. 1, 5

[3] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019. 3

[4] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 3, 4

[5] Github. Pytorch implementaton of fixmatch. https://github.com/kekmodel/FixMatch-pytorch. 1

[6] Github. Pytorch implementaton of mixmatch. https://github.com/YU1ut/MixMatch-pytorch. 2

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 1, 2

[8] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *CoRR*, abs/2001.07685, 2020. 1, 2, 3

[9] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. 2016. 2, 3

[10] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017. 5