Final Documentation
MixMatch: A Holistic Approach to Semi-Supervised Learning

Team Name: ~
P Harshavardhan - 2021111003
Kapil Rajesh Kavitha - 2021101028

LINK FOR ALL WANDB PROJECTS:
harsha_cvit – Weights & Biases (wandb.ai)

Abstract

In this project, we implement MixMatch, a semi-supervised learning algorithm that unifies the current dominant approaches for semi-supervised learning. This approach involves guessing low-entropy labels for data-augmented unlabeled examples and mixes labeled and unlabeled data using the MixUp algorithm. We then test the performance of the model using datasets such as CIFAR-10, CIFAR-100, SVHN and Fashion-MNIST, using varying numbers of labeled examples. We also perform ablation studies to determine the impact of different parts of the algorithm on the performance of the entire algorithm.

Introduction

Semi-supervised learning (SSL) is a technique that aims to reduce the dependency on labeled data by enabling a model to benefit from unlabeled data. Numerous recent SSL methods incorporate a loss term computed on unlabeled data, promoting improved generalization of the model to unseen data. Some existing methods for this include:

1. Entropy minimization - Integrates data augmentation into semi-supervised learning by capitalizing on the concept that a classifier should maintain the same class distribution for an unlabeled instance even after undergoing augmentation. To elaborate, consistency regularization ensures that an unlabeled example x should receive the same classification as Augment(x), a modified version of itself.
2. Consistency regularization - Requires the classifier to output low-entropy predictions on unlabeled data such that the classifier's decision boundary does not pass through high-density regions of the marginal data distribution.
3. Generic/Traditional regularization - Imposes some constraint on a model to make it harder to memorize the training data and therefore hopefully make it generalize better to unseen data.

MixMatch

MixMatch is an approach which incorporates different elements from the techniques discussed above. MixMatch operates on a batch X comprising labeled examples with one-hot targets (indicating one of L potential labels) alongside an equally sized unlabeled batch U. MixMatch generates a processed batch of augmented labeled examples X' and a batch of augmented unlabeled examples with "guessed" labels, denoted as U'. Subsequently, both U' and X' are utilized in the computation of distinct loss terms for labeled and unlabeled instances.

Formally, the combined loss L for semi-supervised learning is defined by:

$$\begin{aligned}
\mathcal{X}', \mathcal{U}' &= \text{MixMatch}(\mathcal{X}, \mathcal{U}, T, K, \alpha) \\
\mathcal{L}_{\mathcal{X}} &= \frac{1}{|\mathcal{X}'|} \sum_{x,p \in \mathcal{X}'} \text{H}(p, \text{p}_{\text{model}}(y \mid x; \theta)) \\
\mathcal{L}_{\mathcal{U}} &= \frac{1}{L|\mathcal{U}'|} \sum_{u,q \in \mathcal{U}'} \|q - \text{p}_{\text{model}}(y \mid u; \theta)\|_2^2 \\
\mathcal{L} &= \mathcal{L}_{\mathcal{X}} + \lambda_{\mathcal{U}} \mathcal{L}_{\mathcal{U}}
\end{aligned}$$

where H(p, q) is the cross-entropy between distributions p and q, and T, K, α, and λ_u are other hyperparameters that will be described below.

The MixMatch algorithm employs different supporting techniques based on the earlier prevailing approaches:

Data Augmentation

Data augmentation is performed on both labeled and unlabeled data. For each element in a batch of labeled data, a transformed version of the element is generated, whereas for each element in a batch of unlabeled data, K augmentations are generated which are used to generate a “guessed label” for each unlabeled data point.

Label Guessing

This is the process through which the earlier mentioned “guessed label” is generated. For each unlabeled example in U, MixMatch produces a “guess” for the example’s label using the model’s predictions, which is later used in the unsupervised loss term. The “guess” is computed by taking the average of the model’s predicted class distributions across all K augmentations of a data point using the below formula:

$$\bar{q}_b = \frac{1}{K} \sum_{k=1}^{K} \mathrm{p}_{\mathrm{model}}(y \mid \hat{u}_{b,k}; \theta)$$

This is a commonly used technique in consistency regularization.

Inspired by the success of entropy minimization in SSL, an additional step is used for generating guessed labels, known as sharpening. Here, a sharpening function (sigmoid) is applied to the average prediction over augmentations, in order to reduce the entropy of the distribution. In practice, this is implemented by adjusting the *temperature* of the distribution.

$$\mathrm{Sharpen}(p, T)_i := p_i^{\frac{1}{T}} \Big/ \sum_{j=1}^{L} p_j^{\frac{1}{T}}$$

Here, 'p' is the average class prediction over augmentations and T is the temperature hyperparameter.

MixUp

MixUp is used for SSL, and both labeled examples and unlabeled examples are mixed with label guesses. A slightly modified version of MixUp is defined in order to be compatible with the separate loss terms.
For a pair of two examples with corresponding label probabilities (x1, p1) and (x2, p2), (x', p') is computed using:

$$\begin{aligned}
\lambda &\sim \mathrm{Beta}(\alpha, \alpha) \\
\lambda' &= \max(\lambda, 1 - \lambda) \\
x' &= \lambda' x_1 + (1 - \lambda') x_2 \\
p' &= \lambda' p_1 + (1 - \lambda') p_2
\end{aligned}$$

where α is a hyperparameter. The second equation helps preserve the order of the batch in order to appropriately compute the individual loss components - x' is closer to x1 than x2.
To apply MixUp:

$$\begin{aligned}
\hat{\mathcal{X}} &= \big((\hat{x}_b, p_b); b \in (1, \ldots, B)\big) \\
\hat{\mathcal{U}} &= \big((\hat{u}_{b,k}, q_b); b \in (1, \ldots, B), k \in (1, \ldots, K)\big)
\end{aligned}$$

All the augmented labeled examples with their labels and all unlabeled examples with their guessed labels are collected. Then they are combined and the result is shuffled to form W, which will be the data source for MixUp.

To summarize, MixMatch transforms X into X', a collection of labeled examples which have had data augmentation and MixUp (potentially mixed with an unlabeled example) applied. Similarly, U is transformed into U', a collection of multiple augmentations of each unlabeled example with corresponding label guesses.

Algorithm 1 MixMatch takes a batch of labeled data $\mathcal{X}$ and a batch of unlabeled data $\mathcal{U}$ and produces a collection $\mathcal{X}'$ (resp. $\mathcal{U}'$) of processed labeled examples (resp. unlabeled with guessed labels).

Input: Batch of labeled examples and their one-hot labels $\mathcal{X} = \big((x_b, p_b); b \in (1, \dots, B)\big)$, batch of unlabeled examples $\mathcal{U} = \big(u_b; b \in (1, \dots, B)\big)$, sharpening temperature T, number of augmentations K, Beta distribution parameter α for MixUp.
for $b = 1$ **to** B **do**
  $\hat{x}_b = \text{Augment}(x_b)$ *// Apply data augmentation to* x_b
  for $k = 1$ **to** K **do**
    $\hat{u}_{b,k} = \text{Augment}(u_b)$ *// Apply* k^{th} *round of data augmentation to* u_b
  end for
  $\bar{q}_b = \frac{1}{K} \sum_k \text{p}_{\text{model}}(y \mid \hat{u}_{b,k}; \theta)$ *// Compute average predictions across all augmentations of* u_b
  $q_b = \text{Sharpen}(\bar{q}_b, T)$ *// Apply temperature sharpening to the average prediction (see eq. (7))*
end for
$\hat{\mathcal{X}} = \big((\hat{x}_b, p_b); b \in (1, \dots, B)\big)$ *// Augmented labeled examples and their labels*
$\hat{\mathcal{U}} = \big((\hat{u}_{b,k}, q_b); b \in (1, \dots, B), k \in (1, \dots, K)\big)$ *// Augmented unlabeled examples, guessed labels*
$\mathcal{W} = \text{Shuffle}\big(\text{Concat}(\hat{\mathcal{X}}, \hat{\mathcal{U}})\big)$ *// Combine and shuffle labeled and unlabeled data*
$\mathcal{X}' = \big(\text{MixUp}(\hat{\mathcal{X}}_i, \mathcal{W}_i); i \in (1, \dots, |\hat{\mathcal{X}}|)\big)$ *// Apply* MixUp *to labeled data and entries from* $\mathcal{W}$
$\mathcal{U}' = \big(\text{MixUp}(\hat{\mathcal{U}}_i, \mathcal{W}_{i+|\hat{\mathcal{X}}|}); i \in (1, \dots, |\hat{\mathcal{U}}|)\big)$ *// Apply* MixUp *to unlabeled data and the rest of* $\mathcal{W}$
return $\mathcal{X}', \mathcal{U}'$

Implementation Details

Data Augmentations

The following data augmentations were implemented:

- Random Pad and Crop: takes an input image 'x', applies random padding of 4 pixels to all sides of the image, then randomly crops it to the specified output size.
- Random Horizontal Flip: performs a horizontal flip on the input image with a 50% probability.
- Gaussian Noise: takes an input image x and adds Gaussian noise to it. Random samples from a standard normal distribution for each channel, and these samples are multiplied by 0.15 to control the intensity of the noise.
- Resize Image: Resizes the image to (3, 32, 32) which is an acceptable size for the model.

Model used:
The model used was a WideResNet model with depth 28. The model was obtained from the implementation used by Google.
Documentation: [Wide ResNet | PyTorch]

Optimizer

The Adam optimizer was used for the WideResNet model, as it has the ability to decay the weights - which has been mentioned in the Google implementation.

Exponential Moving Average (EMA) Optimizer

An EMA optimizer was also implemented as part of our code. Another model was created on which EMA was applied.
It is a technique used to smooth out noisy data and highlight trends over time, and is often applied to the parameters of a neural network during training. The purpose is to maintain a moving average of the model's parameters to provide a more stable and less noisy estimate. It creates a more stable version of the model, and makes the model more robust.

Hyperparameters:
- Temperature (T) : 0.5
- Number of augmentations for unlabeled data (K) : 2
- Unsupervised loss weight (λu) : 0.75
- Weight decay : 0.0008

Tuning

Tuning was performed on the number of labeled samples in each dataset, and the model performance was noted for different number of labeled samples.

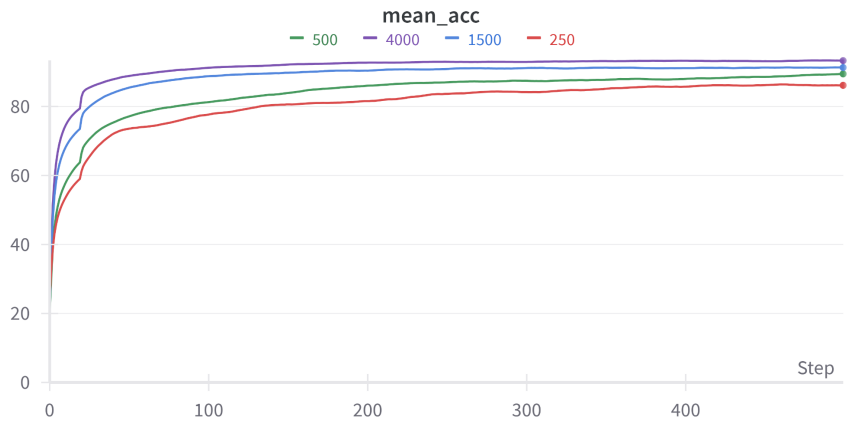
Logging of these runs was done in WandB.

Experimentation
Datasets in paper
- **CIFAR10**
- **CIFAR100**
- **SVHN**
- **STL10 (extra and no extra)**

Additional datasets
- **Fashion-MNIST**

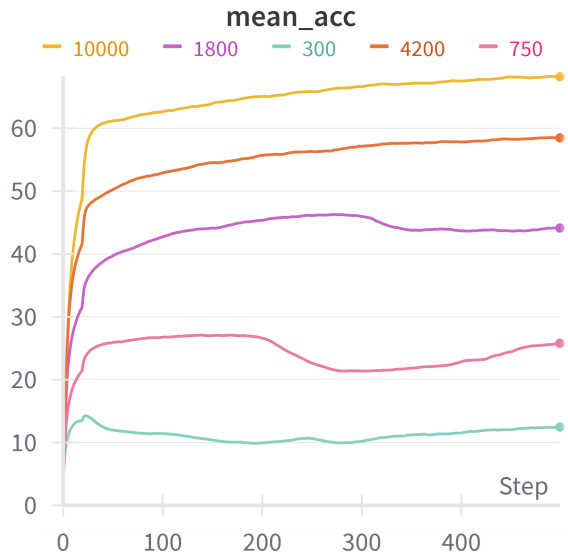
- **MNIST**

CIFAR10



CIFAR-10 was run for 4 values of number of labeled data points (250, 500, 1500, 4000)
We observe that the performance of the model increases as the number of labeled examples increases for the given data, but the model performs sufficiently well for all 4 values.

CIFAR100

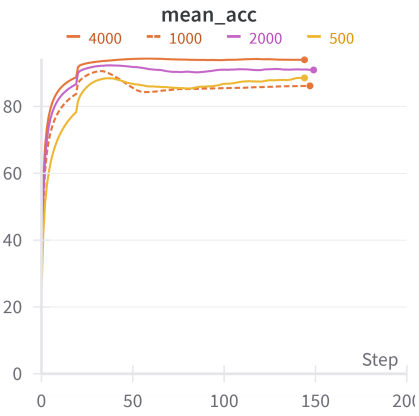


CIFAR-100 was run for 5 values of labeled data (300, 750, 1800, 4200, 10000). The best performance was observed for the models with 4200 and 10000 labels. Due to the higher possibility of incorrect labels which would affect the performance of a semi-supervised model, the model's performance was not substantially improved by the MixMatch approach for 300 labels.

SVHN

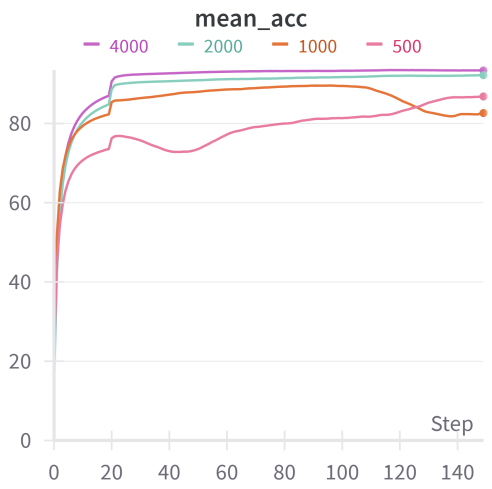
SVHN has two training sets: train and extra. Here we compare the performance of the models when the inputs are from just the 'train' set or are from 'train' + 'extra'.
Additionally, due to the tendency of the model to overfit quicker to this dataset, we use learning rate=0.0005 for this dataset. We also use lambda-u = 250 as recommended by the paper

No 'extra' set



All 4 runs showed good performances, with accuracies ranging from ~85% to ~93% for the four of them. The best performance was shown when 4000 labeled data points are passed to the model.

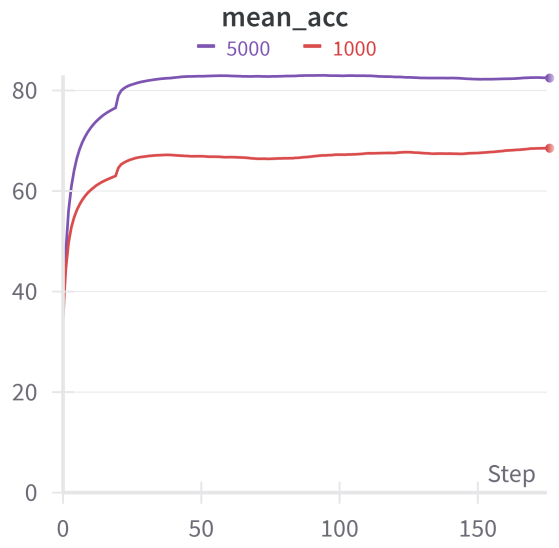
With 'extra' set



With extra set, due to the much larger number of unlabeled samples, we modify the model input parameters: we set the mixing ratio (alpha) as 0.25 and lower weight decay to 0.000002 to account for the larger amount of available data.

Here the performance of the model when 2000 and 4000 labeled inputs are passed stands out even further, as its performance is substantially better than the model when lesser number of labeled inputs are passed.

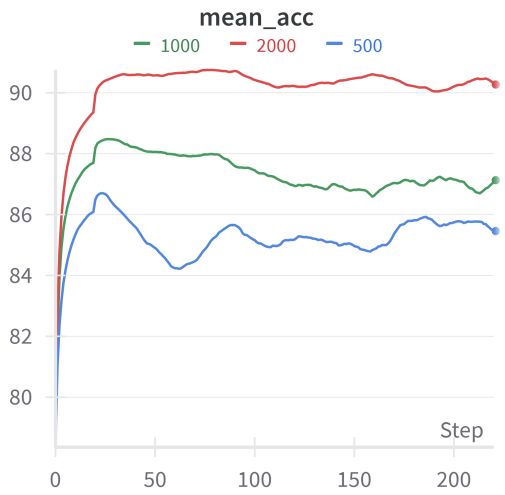
STL10



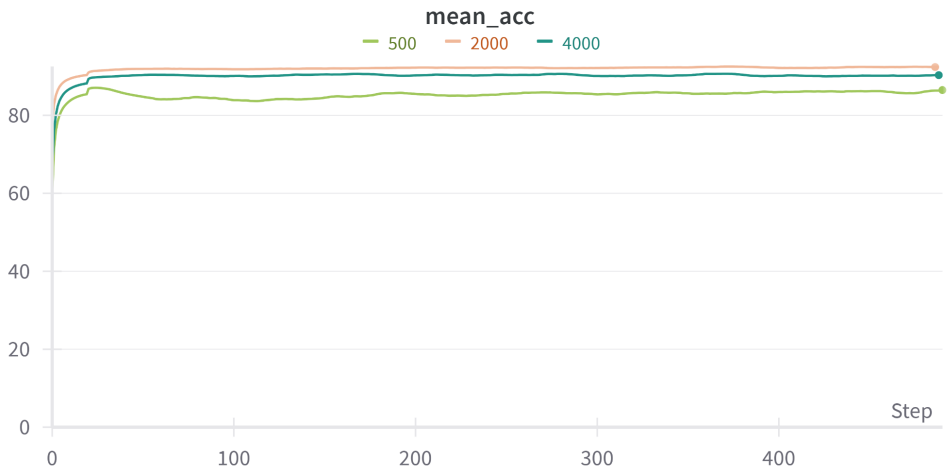
The performance of our MixMatch implementation on the STL-10 dataset falls short of the performance of the Google paper. This is because of the need to resize the 96x96 input images to 32x32 so that they fit in our model, due to the inability of being able to compute with a larger number of parameters in our WideResNet model.

Additional datasets:

MNIST



FASHION-MNIST
Report: https://api.wandb.ai/links/harsha_cvit/qyz66lmi



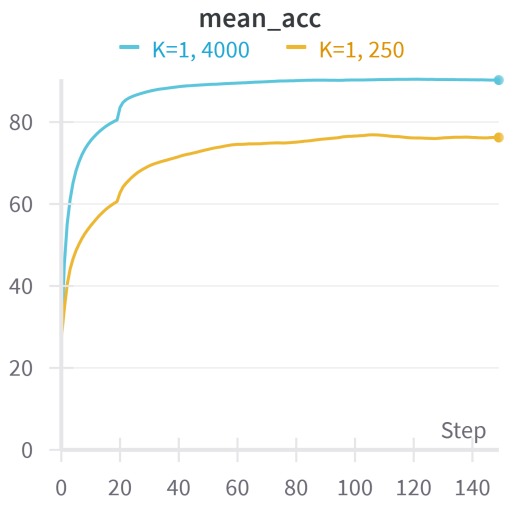
The MixMatch approach shows promising results on the MNIST and the Fashion-MNIST dataset, being able to hit over 90% accuracy when at least 1000 labeled inputs are passed.

Ablation Studies

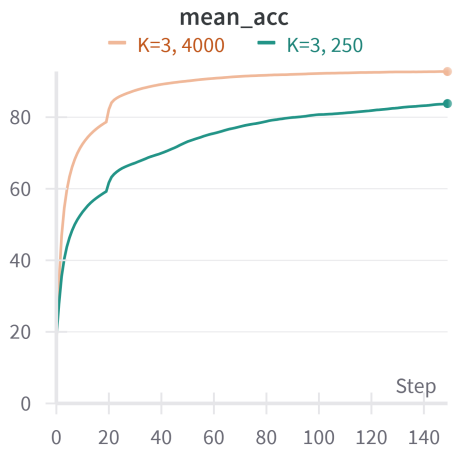
Report: https://api.wandb.ai/links/harsha_cvit/ypegh2i1

In this section, we aim to study the impact of the different parts of the MixMatch algorithm on its success. All of these experiments were run on the CIFAR-10 dataset.

Number of augmentations (k) = 1

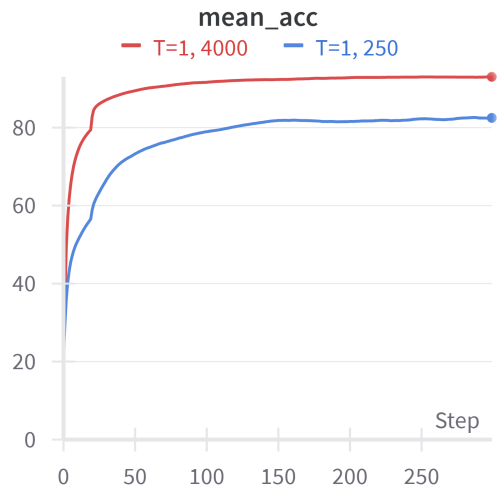


Number of augmentations (k) = 3



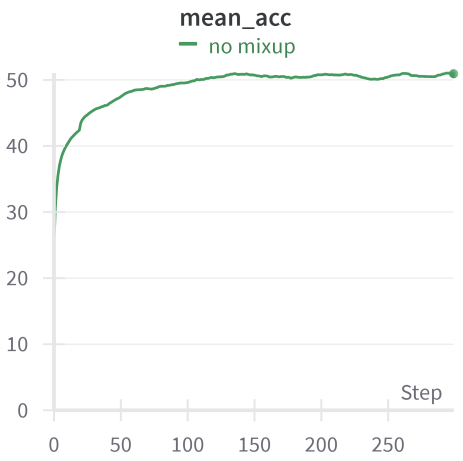
The model performance for K=1 and K=3 appears to be a bit worse (substantially so for K=1) for 250 labels, and the difference in model performance reduces with increase in the number of labels (there is only a ~3% reduction in accuracy between K=1 and K=2, and it is even lesser for K=3)

Temperature (T) = 1



The impact of T=1 appears to be more significant when the number of labels is lesser, and it becomes less significant on the increase in the number of labels, as the model has more amount of reliable information to learn from.

MixMatch with no MixUp



This experiment was performed on 250 labels. The impact of no MixUp on the MixMatch algorithm is very significant as there is a drop in accuracy of above 30%, as the training process may have become much less stable.

Comparison

In the cases of CIFAR10, CIFAR100 and SVHN, our implementation of MixMatch performs comparatively well to the Google implementation, with a ~4-5% difference in error rates.
Only the STL-10 dataset saw a dip in performance. This is due to usage of WideResNet models with different depths, and the need to resize images in the STL-10 dataset to fit such a model.

The MNIST and Fashion-MNIST models seemed to display performances relatively close to the benchmark scores, indicating the strength of the approach.

Issues faced

- Lack of compute and relatively high running time, which also was the reason for not being able to perform hyperparameter tuning significantly.
- The lack of compute gave us an issue with not being able to run models with more than 1.4M parameters, so we couldn't train the CIFAR-100 dataset on a larger model or efficiently train the STL-10 dataset.