MixMatch: A Holistic Approach to Semi-Supervised Learning

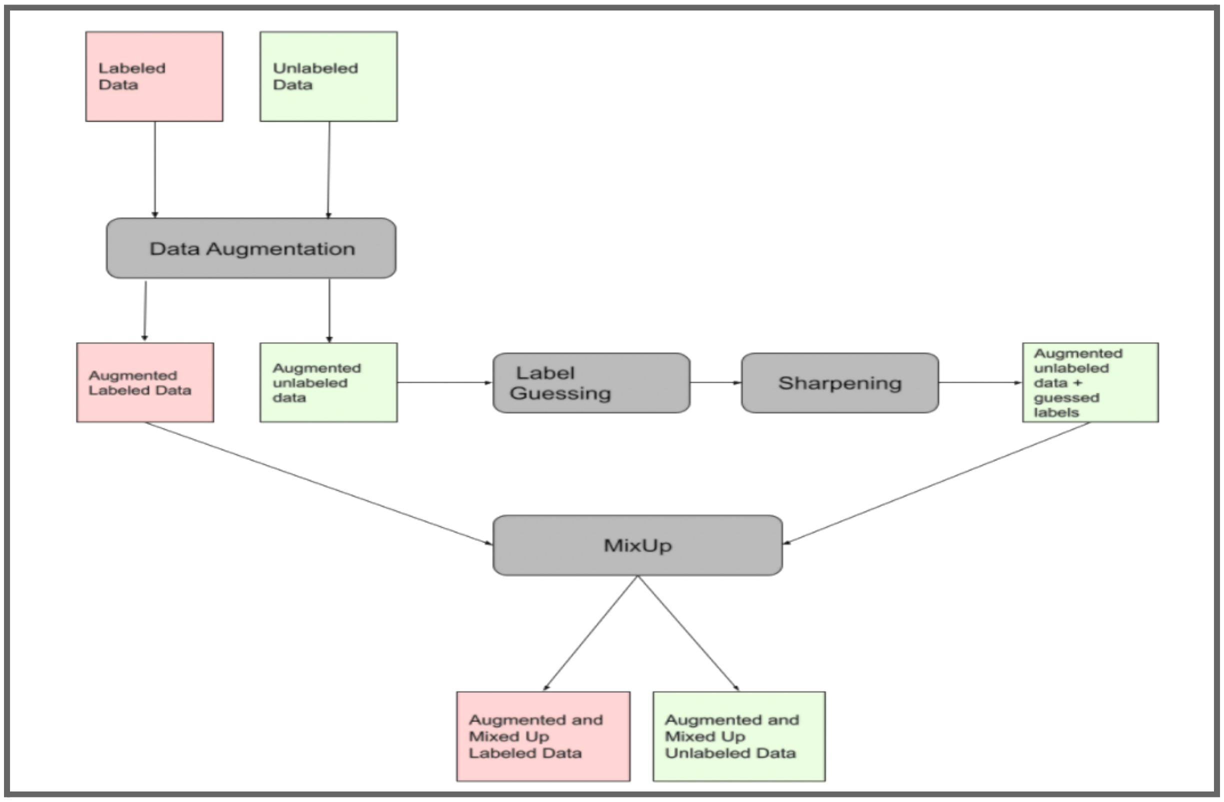
Name: Lamia Salsabil
UIN: 01181847
Email: lsals002@odu.edu

Goal: To implement the algorithm "MixMatch" proposed in the "MixMatch: A Holistic Approach to Semi-Supervised Learning" paper and make possible and reasonable improvements over it.

Library Requirements:
Python 3.6+
PyTorch 1.0(older versions are not compatible with this code)
tensorboardX
progress
matplotlib
numpy

Hardware Requirements:
GPU

MixMatch Architecture:



Pseudo Code:

- Given a batch of n labeled samples with their labels = X and
- A batch of m unlabeled samples = U
- Data Augmentation is performed on labeled and unlabeled sample and the result is
    - Augment (X) = X’
    - Augment (U) = U’ (Data augmentation performed on unlabeled samples for k number of times)

- Label Guessing
    - In this step, average label prediction, q is computed across all the augmentations of unlabeled data
- Sharpening
    - Temperature sharpening is applied on the average prediction, q which results in a batch of unlabeled data with their “guessed label”, U_b

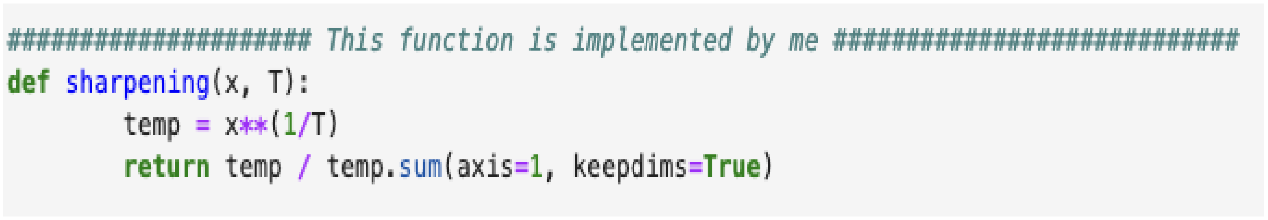
- The augmented labeled data X’ and unlabeled data U_b are shuffled and concatenated together which results in, W
    - W = Shuffle(Concat(X’, U_b))
- MixUp
    - MixUp algorithm is applied to the labeled data and n number of samples from W
    - MixUp algorithm applied to the unlabeled data the rest of entries from W

MixMatch Implementation:

- **Data Augmentation**
    - In the paper, the authors performed random horizontal flip and random crop on image. In my code I have applied gaussian blur along with performing random horizontal flip and random crop on image. I have written three classes
        - RandomPadandCrop - To crop images randomly
        - RandomFlip - To flip images randomly
        - GaussianNoise - To apply gaussian blur on images to reduce image noise and detail.

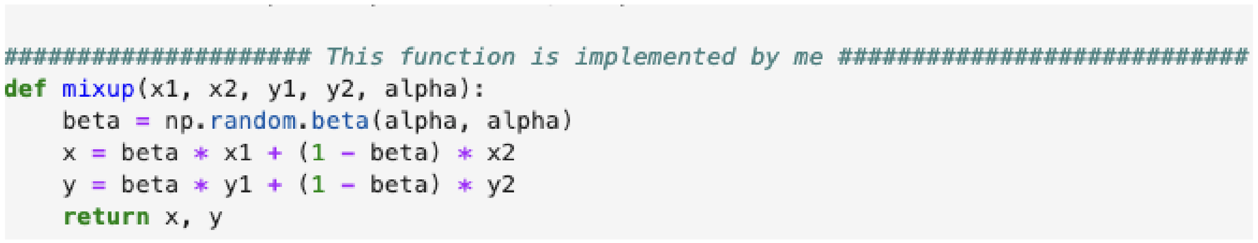
- **Label Guessing**
    - For label guessing, the authors have used the “Wide ResNet-28” model in the paper. In my implementation, I have used both “Wide ResNet-28” and “Wide ResNet-40” models on unlabeled data.

- **Sharpening**
    - In my code, I have written a sharpening function to implement temperature sharpening on the computed average predictions of the unlabeled data.
    - In the sharpening function, T is a hyperparameter and in my code I have used the value of T = 0.5



```
##################### This function is implemented by me ############################
def sharpening(x, T):
        temp = x**(1/T)
        return temp / temp.sum(axis=1, keepdims=True)
```

- **MixUp**
    - I concatenated and shuffled the labeled and unlabeled data before applying MixUp. After that, I applied my "mixup" function to them.



```
##################### This function is implemented by me ############################
def mixup(x1, x2, y1, y2, alpha):
    beta = np.random.beta(alpha, alpha)
    x = beta * x1 + (1 - beta) * x2
    y = beta * y1 + (1 - beta) * y2
    return x, y
```

Dataset: The authors have used following image datasets in their paper for training and evaluating their algorithm:

- CIFAR-10
- CIFAR-100
- SVHN
- SVHN+Extra
- STL-10

In my code, I have used the CIFAR-10 dataset for training and evaluating my implementation.

CIFAR-10 Dataset: The CIFAR-10 dataset consists of 60000 32x32 colour images divided into 10 classes, with 6000 images per class. It contains 50000 training images and 10000 test images. I have used the “torchvision.datasets.CIFAR10” class to use the CIFAR-10 dataset.
CIFAR-10 dataset classes are - airplane, automobile, bird, deer, cat, dog, frog, horse, ship, truck.

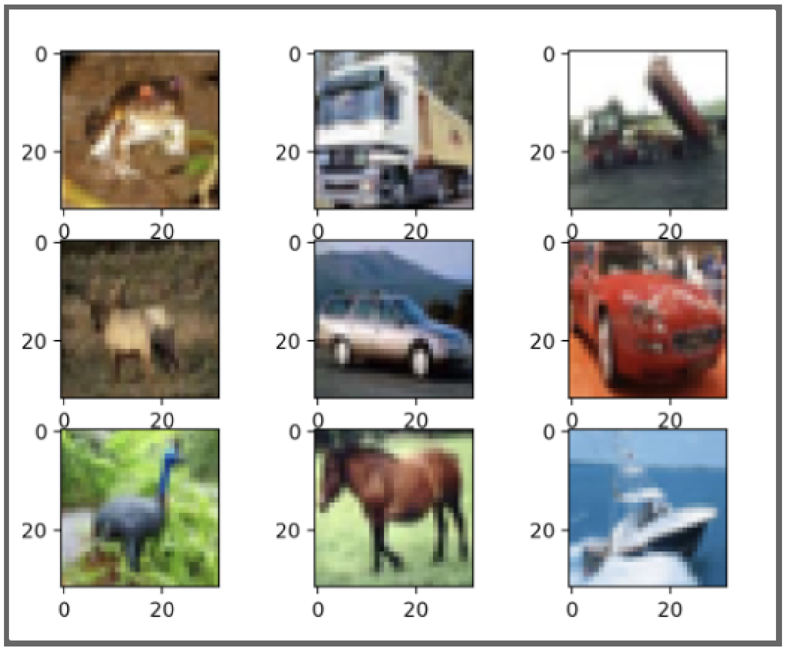


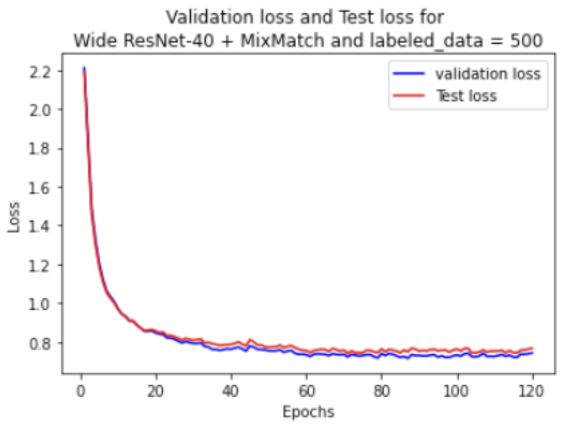
Figure: Plot of a subset of images of CIFAR-10

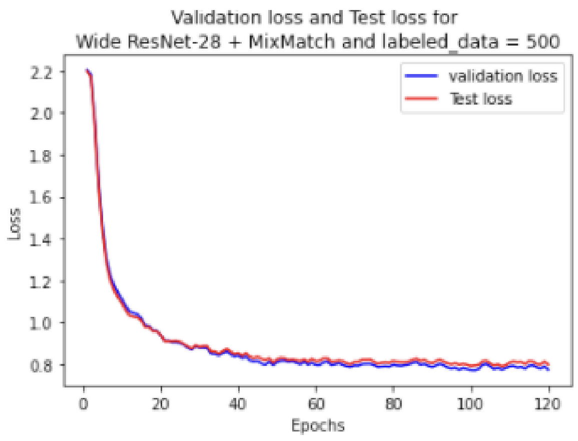
Training with CIFAR-10: After applying the MixMatch algorithm on the CIFAR-10 dataset, I have used both “Wide ResNet-28” and “Wide ResNet-40” for training and testing. In the paper, the authors have used only the “Wide ResNet-28” model for training and testing. I have trained my model with two different sample size of labeled data:

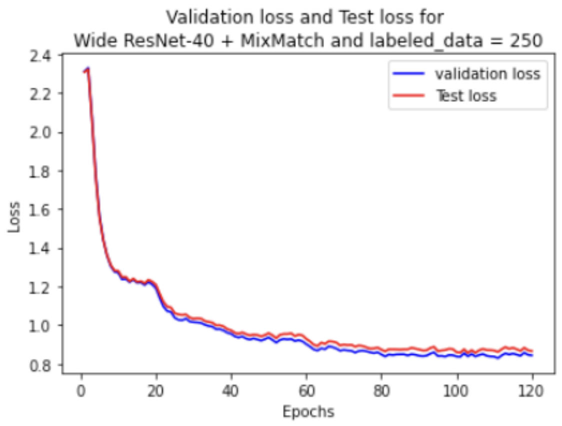
- Labeled = 250, Unlabeled = 44750, Validation = 5000, Test = 10000
- Labeled = 500, Unlabeled = 44500, Validation = 5000, Test = 10000
- **Hyperparameters:**
    - Epochs = 120
    - Batch size = 100
    - Learning rate = 0.01
    - alpha, α = 0.75
    - lambda-u, λ_u = 75
    - T = 0.5
    - Widening factor of “Wide ResNet” models = 2
    - Number of round of data augmentation to unlabeled data, k =2
- For training the model with CIFAR-10 data, I have used the GPU of ODU HPC. For Epochs = 120, it took approximately 14 hours to train the model and get the test results.

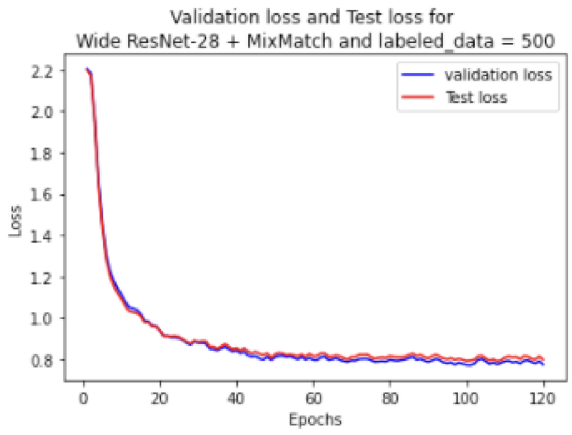
Evaluation:

Model	**Test accuracy for labeled sample = 250**	**Test accuracy for labeled sample = 500**
Wide ResNet-28 + MixMatch	73.42	78.58
Wide ResNet-40 + MixMatch	76.7	80.14
Author’s implementation (Wide ResNet-28 + MixMatch)	88.92 ± 0.87	90.35 ± 0.94









From the “Loss vs Epoch” graph, we can see that test and validation loss is decreasing steadily with increasing the number of epochs. From all the four “Loss vs Epoch” graphs, we see that validation loss is lower than test loss. The first “Loss vs Epoch” graph shows that the test and validation loss is lower for the “Wide ResNet-40 + MixMatch” model and labeled data size = 500 than for all three other models. From the comparison shown in the table, we reach to a conclusion that my implemented “Mixmatch+ Wide ResNet-40” model achieves a better test accuracy than “Mixmatch+ Wide ResNet-28” model. My implementation achieved the best accuracy of 80.14 for labeled data size = 500 and “Mixmatch+ Wide ResNet-40” model.

Conclusion:

I have learnt many things while working on this project. I got to learn about data augmentation, consistency regularization and entropy minimization techniques. I had little knowledge about image datasets at the beginning of the project. But over time, I got to learn more about many image datasets - CIFAR-10, CIFAR-100, SVHN, STL-10, USPS, Fashion-MNIST and Places365. Initially, my plan was to work with the CIFAR-10 and Fashion-MNIST dataset. But, the Fashion-MNIST dataset is in grayscale and when I tried to work with the Fashion-MNIST dataset, I was getting errors. Because of the time constraints, I couldn’t proceed further with the “Fashion-MNIST” dataset. In future, I wish to work with “Fashion-MNIST” dataset and also evaluate my implementation for other domains other than image benchmarks.

References:

1. Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., & Raffel, C. (2019). Mixmatch: A holistic approach to semi-supervised learning. arXiv preprint arXiv:1905.02249.
2. https://festinais.medium.com/mixmatch-a-holistic-approach-to-semi-supervised-learning-1480b56f96b7
3. Source code for MixMatch - https://github.com/YU1ut/MixMatch-pytorch
4. Official source code for MixMatch - https://github.com/google-research/mixmatch
5. https://blog.roboflow.com/why-and-how-to-implement-random-crop-data-augmentation/
6. Blog on data augmentation - https://neptune.ai/blog/data-augmentation-in-python
7. A Comprehensive Guide to the DataLoader Class and Abstractions in PyTorch - https://blog.paperspace.com/dataloaders-abstractions-pytorch/
8. Pytorch https://pytorch.org/get-started/locally/
9. CIFAR-10 dataset - https://pytorch.org/vision/stable/datasets.html#cifar
10. Grandvalet, Y., & Bengio, Y. (2005). Semi-supervised learning by entropy minimization. CAP, 367, 281-296.