Computer Vision SSL Final Project (Fall 2019)

Yunhao Li
New York University
yl6220@nyu.edu

Tin Luu
New York University
tbl245@nyu.edu

Fan Xie
New York University
fx349@nyu.edu

Abstract

In the recent decade, Deep Learning grows to be more and more essential in many fields, especially image classification. However, the extremely high cost for supervised labeling makes it sometimes unpractical. Therefore, people are focusing on self-supervised learning. In this paper, we re-implement a self-supervised learning[15] model that achieves a top-1 classification accuracy of 88.6125% on the STL-10[2] data set. Our model choice is adapted from the 2019 MixMatch[1] model.

1. Introduction

In recent years, Deep Learning grows to be more and more essential in many fields, especially Computer Vision. As one of the most representative application, image classification shows the power of Deep Learning. However, the supervised Deep Learning method requires a lot of data labeled by human, which is extremely expensive and hard to collect, making it sometimes not really practical.

Compared to the available labeled data, the amount that is unlabeled is often free and hugely more abundant. This fact is often overshadowed by the past-decade successes in image classification where achievements have mainly occurred with supervised models, and hence in tandem with the availability and accessibility of labeled data. These successes are limited in that it only scales with the amount of labeled data which does not scale, and leaves out an enormous amount of unlabeled data that necessitates the need to build computer vision models that can take advantage of a small amount of labeled data and can generalize its performance to the abundant amount of unlabeled data.

Self-supervised learning, in recent years, have shown huge potential in taking advantage of the huge pool of unlabeled data. Models of this type of learning find supervision signal from within the input unlabeled data and uses it as the target, allowing for self-labeling of input data and the use of loss functions and model architectures that are standard in supervised learning models. Some of the methods for getting supervision signals for unlabeled input and learning latent representations are distortion (rotations), patches, colorization , jig saw [4, 17, 11]. These techniques can be grouped into 3 categories: entropy minimization[5], which encourages the model to output confident predictions on unlabeled data; consistency regularization[12] which encourages the model to produce the same output distribution when its inputs are perturbed; and generic regularization[6], which encourages the model to generalize well and avoid overfitting the training data.

MixMatch combines all the three efficient self-supervised paradigms described above and achieved state-of-the-art results in self-supervised learning tasks. According to the paper, experiment results on CIFAR-10 show that MixMatch is able to reduce the testing error to 11.08% with only 250 label images and with 4000 label images the testing error is further reduced to 6.24%. The best model before MixMatch is MR-GAN[9], which achieved 14.45% testing error and the performance of MixMatch is far more better than previous models.

2. Related Work

A variety of approaches have been proposed for self-supervised learning. Ulicny et al. propose Harmonic Networks[14] which introduce a harmonic block that uses Discrete Cosine Transform (DCT) filters in CNNs to improve accuracy and computationally efficient. Ji et al. propose a new clustering method called Invariant Information Clustering[7] to discover clusters that accurately match semantic classes. Invariant Information Clustering is capable of learning a representation that preserves the common features between two images while discarding instance-specific details. It learns a neural network classifier from scratch with only unlabelled data samples. Swersky et al. propose Multi-Task Bayesian Optimization[13] which transfer the knowledge gained from previous optimizations to new tasks in order to find optimal hyperparameter settings more efficiently. Zhao et al. present "stacked what-where auto-encoders" (SWWAE)[18] which integrates discriminative and generative pathways. Each pooling layer of SWWAE produces two sets of outputs: the "what" features

are fed to the next layer, and its complementary "where" features are fed to the corresponding layer in the generative decoder.

3. Main method

3.1. Model Description

We re-implemented the MixMatch model. The whole procedure can be separated in the following important parts:

1. **Label guessing**
   Every time we get a labeled image and an unlabeled image from the dataset. The unlabeled data will be transformed multiple times. There is a parameter K, determine the number of the transformed times. In our project we choose $K = 2$. So we get two transformed unlabeled images.
   Then we run the model on these two images, and we get two predict labels. Then we compute the average value of the two labels, and take that result as the guessing label.
   The other parameter T is used to sharpen the data.
   $Sharpen(p, T) = p_i^{\frac{1}{T}} / \sum_{j=1}^{L} p_j^{\frac{1}{T}}$

2. **concatenate**
   Then we need to concatenate the $X^{'}$ $U^{'}$. In practice, we concatenate $X^{'}$ with U_1 and U_2 into W at dimension 0. The U_i is the transformed date of unlabeled image U.

3. **Mixup**
   First we get a random value λ from a Beta distribution. Then we compute $\lambda^{'} = max(\lambda, 1 - \lambda)$. Let's call the images as x_1 and x_2, the labels as p_1, p_2.
   Then we can compute:
   $x^{'} = \lambda^{'} x_1 + (1 - \lambda^{'}) x_2$
   $p^{'} = \lambda^{'} p_1 + (1 - \lambda^{'}) p_2$

4. **Mixmatch**
   The Mixmatch is the sum up of the procedures above.

   (a) Get a labeled image and an unlabeled image, apply data augmentation to the labeled data. Apply data augmentation to the unlabeled data K times respectively. Assume labeled data is X, augmented X is $X^{'}$ label is p; unlabeled augmented data is $Ui(i = 1, 2, \ldots)$

   (b) Run label guessing on the K augmented unlabeled data, get a prediction q.

   (c) Concatenate $X^{'}$ with Ui to get W, then shuffle it.

   (d) Mixup $X_i^{'}$ with W_i to get $\hat{X}$; Mixup $U_i^{'}$ with $W_{i+|X|}$ to get $\hat{U}$;

5. **Loss function** After running Mixmatch, we already have new $X^{'}$ and $p^{'}$. Then we calculate the loss. Suppose H is cross entropy loss function.
   Then we compute 2 parts of loss separately.
   $L_X = \frac{1}{|X'|} \sum_{x,p \in X'} H(label, model(X^{'}))$
   $L_U = MeanSquareError(guess\_label, model(U^{'}))$
   Finally we add two parts loss by a factor λ_μ, that is
   $L = L_X + \lambda_\mu \cdot L_U$.

3.2. Implementation Details

Our approach is based on the pytorch implementation of MixMatch[16]. It is only implemented to suit the CIFAR-10[8] dataset, where the images are $32 \times 32 \times 3$. Therefore, we need to re-implement some codes.

1. Dataloader: we use the STL-10[2] dataset integrated in torchvision[10]. We also use samplers to split the dataset, so that we can test

2. Data augmentation: we tried many combinations of data augmentations. Finally we choose the RandomCrop and RandomHorizontalFlip. We set the crop size as the 90% of the image size with constant padding.

3. Learning rate: the original learning rate is 0.002, which is too high for STL-10. So we tried a lot of learning rate values and finally choose 2e-4.

4. Changes in model: Because the images in STL-10[2] is much larger, which is $96 \times 96 \times 3$, we need to change the model to adapt the data. At first, we just resize the images to 32×32. However, during the resize process, some information is lost and the result is pretty bad. The accuracy is only about 70%.
   So we are exploring other methods. We choose a natural and simple way: pooling layer. We add a maxpooling layer following the block1. So that the weights of previous conv1 and block1 sub-networks will learn to extract most important information during the backward propagation, instead of just pooling the images at the very beginning.

5. We also tried to split the final average pooling layer where kernel size is 8 into 4 pooling layer whose kernel size

Specification of Best Model: As for the training of our best model, our parameters is in table 2. The validation accuracy trend graph is 2 and error rate graph is 1. The graph for loss trend against epochs is at 3.

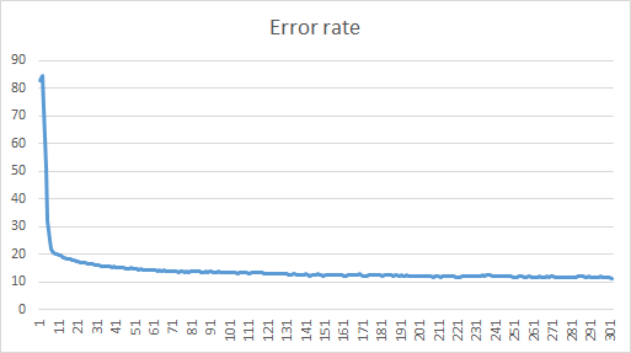

Figure 1. Error Rate of Best Model

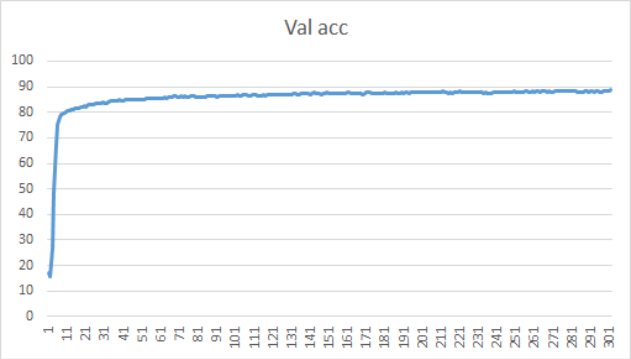

Figure 2. Validation accuracy of Best Model

Figure 3. Loss graph of best model

T	λ_μ	learning rate	α	ema-decay	batch size
0.5	1	2e-4	0.75	0.999	64

Table 1. Parameters for our best model

3.3. Other method we have ever tried

At first, we started with Variational autoencoder(VAE)[3]. We train a VAE on the whole unlabeled dataset. And we extract the encoder and combine it with a classifier network. Then we train and refine this new network on the labeled dataset.

However, our VAE wasn't trained very well. I put them in a sub-folders inside code folder.

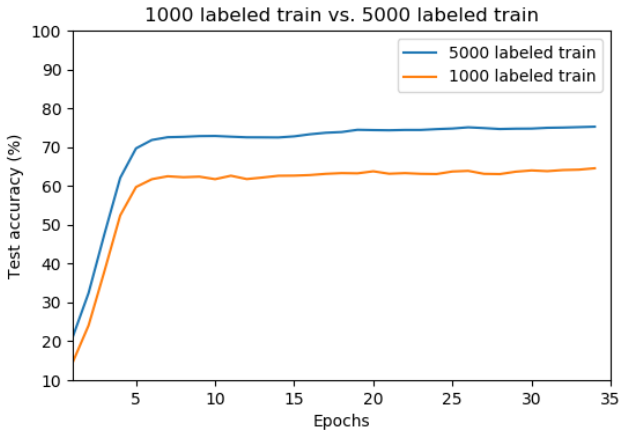

Figure 4. Different label numbers

4. Experiments

4.1. Best Model Versus Benchmark

Our benchmark model is the 18-layered Residual Network (ResNet-18) whose training set is the 5,000 labeled data in STL-10. After training this network for 1024 epochs, the best accuracy achieved on the test set is 69.09%.

Our best model, which is trained on both the labeled and unlabeled data, on the contrary, is able to achieve an accuracy of 88.6125%. This significant difference between these two accuracy scores can be contributed to the learned latent representation from the self-labeled unlabeled training set. In addition, it could also be partly contributed to the image transforms that are applied to the training data in our model. From the graph 2 we can see that the model training are converging very soon. Only about 11 epoch the model gets over 80%.

4.2. Varying Amounts of Labeled Training Data

A feature of an SSL model is that it ought to be able to make the most out of as little the amount of labeled training data as provided. Given that the STL-10 data set has 5,000 labeled training data, we want to compare the performance of our model when it is given the full set of labeled training and when it gets only 1,000 of these labeled data. Setting constant the number of iterations per epoch (1024), image transformations, temperature T (0.5) and lambda_u (75), [FIGURE ONE] exhibits the marked difference in the accuracy path due to the availability of training data. At epoch

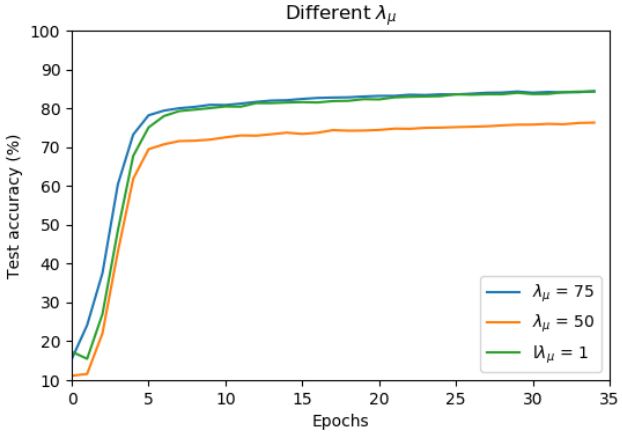

Figure 5. Different λ_μ

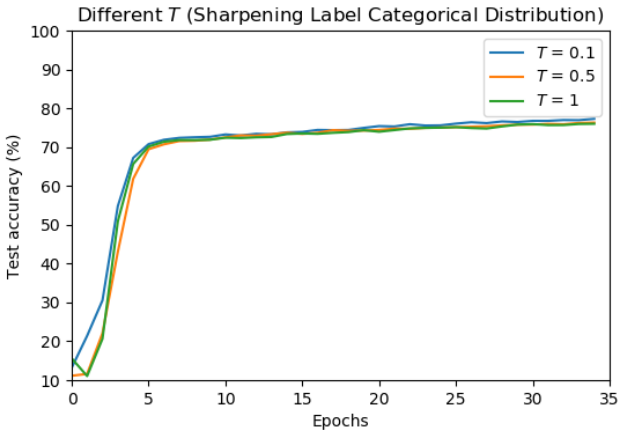

Figure 7. Different T

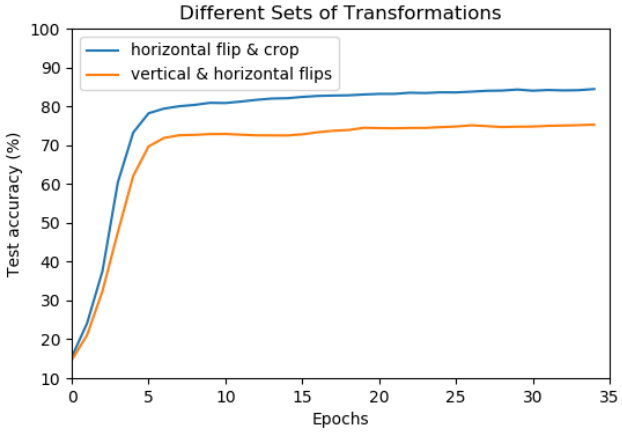

Figure 6. Different transform

35, the test accuracy with 1000 labeled data is 64.55% and with 5000 is 75.25%. While only a fifth of the labeled data set is given, the model is able to get to 85% of the accuracy had it received the full training set. This shows that our SSL model learns beyond just the given labeled data but also a great deal with latent representation from the provided pool of unlabeled data.

4.3. Hyper-parameter testing

After testing a series of adjustments to the parameters, below are the adjustments to have lead to significant model improvement.

4.3.1 *K = 2*, Different Transformations

Comparing two sets of two random image transformations, we are able to show the more effective set. The standard transformers are horizontal and vertical flips of input data at probability = 0.5. We are able to achieve 9% increase in accuracy by replacing the latter flip with random cropping. The accuracy jumps from 75.25% to 84.42%. by epoch 35.

4.3.2 λ_μ and Ablation Test

A recommendation in the original MixMatch paper [1] working with the STL-10 data set is to set $\lambda_\mu = 50$. Together with previously described image transformations and the full labeled training set, the model performance for each value of λ_μ of that this recommendation for our particular model is not to be heeded. Model with $\lambda_\mu = 50$ performs the worst of all, while the ablation test of setting $\lambda_\mu = 1$ reveals that our model performs best when we attach no weight to the unsupervised loss. Because setting $\lambda_\mu = 1$, it is part of our best model.

4.3.3 Temperature *T* and Ablation Test

We adjust the level of entropy of the label distribution through adjusting the sharpening function's the variable "temperature" T that ranges from 0 to 1. The closer T is to zero, the less the entropy, thus the sharper (more "one-hot") the categorical distribution. We tested with three different values of T: 0.1, 0.5 and 1 for ablation testing. While the model with more sharpening ($T = 0.1$) shows better performance at in the first few epochs, sharpening appears to have little contribution to model performance because test accuracy scores of these models converge to the same level after roughly 7 epochs. In other words, removing T or making it low to induce sharpness does not have impact on model performance.

Model	images	T	Trans	λ_μ	Accu
Resnet-18	5000	None	Ver & hor flips	None	69.09
MixMatch	1000	0.1	Ver & hor flips	50	64.55
MixMatch	5000	0.1	Ver & hor flips	50	$\sim$75
MixMatch	5000	1	Ver & hor flips	50	$\sim$ 7 5
MixMatch	5000	0.5	Ver & hor flips	50	75.2
MixMatch	5000	0.5	hor flip & crop	50	84.4
MixMatch	5000	0.5	hor flip & crop	50	$\sim$ 75
MixMatch	5000	0.5	hor flip & crop	75	$\sim$ 88
MixMatch	5000	0.5	hor flip & crop	1	88.612

Table 2. Comparision of different models

5. Discussion & Conclusion

In this paper we implemented MixMatch[1], a self-supervised learning method to do the image classification. We trained and tested it on STL-10 data set. We tried different data augmentations, Parameters(T, λ_μ), number of labeled data. Finally we achieved 88.6125% top-1 accuracy.

Through experimenting with the model, we arrived at different set of hyper-parameters from what appeared to work best for the model in the original paper. Our best model shows that the unsupervised loss does not require any weight for efficient performance. In addition, while our best model keeps the sharpening variable T = 0.5, our experiment shows that this value would not matter to the model that much as long as we train our model for a long time.

For future work, we would like to experiment with even more and different image transformers. In particular, we think that trying out spatial transform augmentations would give interesting results. In addition, We want to test with an enlarged Residual Network, playing with different layers, without causing GPU memory error.

References

[1] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. Mixmatch: A holistic approach to semi-supervised learning, 2019. 1, 4, 5

[2] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. 1, 2

[3] C. Doersch. Tutorial on variational autoencoders, 2016. 3

[4] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015. 1

[5] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005. 1

[6] G. Hinton and D. Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993. 1

[7] X. Ji, J. F. Henriques, and A. Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9865–9874, 2019. 1

[8] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012. 2

[9] B. Lecouat, C.-S. Foo, H. Zenati, and V. Chandrasekhar. Manifold regularization with gans for semi-supervised learning. *arXiv preprint arXiv:1807.04307*, 2018. 1

[10] S. Marcel and Y. Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1485–1488, New York, NY, USA, 2010. ACM. 2

[11] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016. 1

[12] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1163–1171, 2016. 1

[13] K. Swersky, J. Snoek, and R. P. Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013. 1

[14] M. Ulicny, V. A. Krylov, and R. Dahyot. Harmonic networks with limited training samples. *arXiv preprint arXiv:1905.00135*, 2019. 1

[15] Wikipedia contributors. Semi-supervised learning — Wikipedia, the free encyclopedia, 2019. [Online; accessed 19-December-2019]. 1

[16] YUI. Mixmatch. https://github.com/YU1ut/MixMatch-pytorch, 2019. 2

[17] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 1

[18] J. Zhao, M. Mathieu, R. Goroshin, and Y. Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015. 1