Machine Learning
Final Project

Lecturer:	Prof. Lior Rokach		
Course:	Machine Learning		
Students:	Ben Baruch Noam	,	Cohen Hodaya
IDs:	302789961	,	205705205
Submission Date:	08/08/2021		

Table of Content

1. Introduction:

At the start of our project, we had to choose and evaluate a Deep Learning ensemble method. We chose the paper:
"FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence" by the authors: Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, Colin Raffel. This paper presents a new method, called: $FixMatch$. This algorithm is a simplification of existing Semi Supervised Learning (SSL) methods. In the first stage, the algorithm generates pseudo-labels using the model's predictions on weakly augmented unlabeled images. For a given image, the pseudo-label is only retained if the model produces a high-confidence prediction. In the second stage the model is then trained to predict the pseudo-label when fed a strongly-augmented version of the same image.

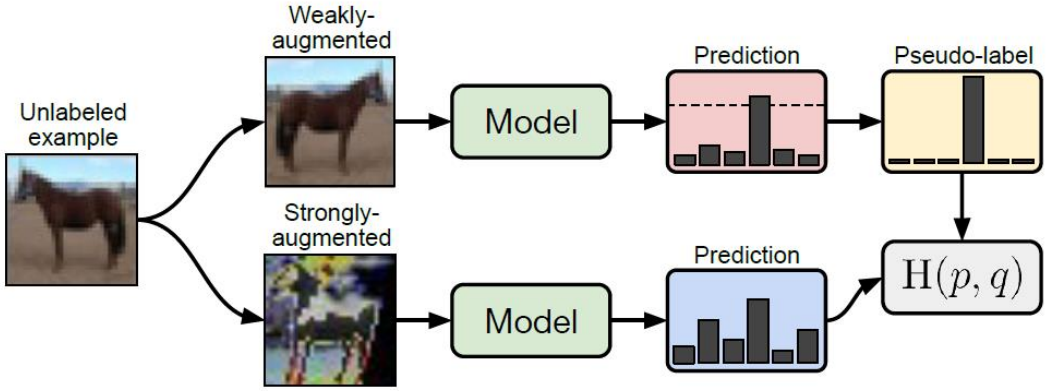

Figure 1- Diagram of $FixMatch$

1.1 Algorithm Description:

For an L-class classification problem, let $\mathcal{X} = \{(x_b, p_b) : b \in (1, \dots, B)\}$ be a batch B labeled examples, where x_b are the training examples and p_b are one-hot labels. Let $\mathcal{U} = \{u_b : b \in (1, \dots, \mu B)\}$ be a batch of μB unlabeled examples where μ is a hyperparameter that determines the relative sizes of $\mathcal{X}$ and $\mathcal{U}$. Let $p_m(y|x)$ be the predicted class distribution produced by the model for input x. We denote the cross-entropy between two probability distributions p and q as $H(p, q)$. We perform two types of augmentations as part of $FixMatch$: strong and weak, denoted by $\mathcal{A}(\cdot)$ and $\alpha(\cdot)$ respectively.

Algorithm 1 FixMatch algorithm.

Input: Labeled batch $\mathcal{X} = \{(x_b, p_b) : b \in (1, \dots, B)\}$, unlabeled batch $\mathcal{U} = \{u_b : b \in (1, \dots, \mu B)\}$, confidence threshold τ, unlabeled data ratio μ, unlabeled loss weight λ_u.
$\ell_s = \frac{1}{B} \sum_{b=1}^{B} \mathrm{H}(p_b, \alpha(x_b))$ {*Cross-entropy loss for labeled data*}
for $b = 1$ **to** μB **do**
  $q_b = p_{\mathrm{m}}(y \mid \alpha(u_b); \theta)$ {*Compute prediction after applying weak data augmentation of* u_b}
end for
$\ell_u = \frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}\{\max(q_b) > \tau\} \mathrm{H}(\arg\max(q_b), p_{\mathrm{m}}(y \mid \mathcal{A}(u_b))$ {*Cross-entropy loss with pseudo-label and confidence for unlabeled data*}
return $\ell_s + \lambda_u \ell_u$

Figure 2 - Algorithm Description in Pseudo Code

$FixMatch$ is a combination of two approaches to SSL: Consistency regularization and pseudo-labeling. Its main novelty comes from the combination of these two ingredients as well as the use of a separate weak and strong augmentation when performing consistency regularization. The loss function for $FixMatch$ consists of two cross-entropy loss terms: a supervised loss ℓ_s applied to labeled data and an unsupervised loss ℓ_u. Specifically, ℓ_s is just the standard cross-entropy loss on weakly augmented labeled examples:

$$Loss\ for\ labeled\ exmaples = \ \ell_s = \frac{1}{B}\sum_{b=1}^{\mathrm{B}} H\left(p_b, p_m\left(y|\alpha(x_b)\right)\right)$$

$FixMatch$ computes an artificial label for each unlabeled example, which is used in a standard cross-entropy loss. To obtain an artificial label, we first compute the model's predicted class distribution given a weakly-augmented version of a given unlabeled image: $q_b = p_m\left(\alpha(u_b)\right)$. Then, we use $\hat{q}_b = \ arg\ max(q_b)$ as a pseudo-label, except we enforce the cross-entropy loss against the model's output for a strongly-augmented version of u_b:

$$Loss\ for\ unlabeled\ exmaples = \ \ell_u = \frac{1}{\mu B}\sum_{b=1}^{\mu\mathrm{B}} \mathbb{I}(\max(q_b) > \tau) \cdot H\left(\hat{q}_b, p_m\left(y|\ \mathcal{A}(u_b)\right)\right)$$

where τ is a scalar hyper parameter denoting the threshold above which we retain a pseudo-label. The loss minimized by $FixMatch$ is simply $\ell_{Total=}\ell_s + \ \lambda_{\mathrm{u}} \cdot \ell_u$ where λ_{u} is a fixed scalar hyper–parameter, denoting the relative weight of the unlabeled loss.
While the formula for unlabeled loss is similar to the pseudo-labeling loss in previous formula, it is crucially different in that the artificial label is computed based on a weakly augmented image and the loss is enforced against the model's output for a strongly augmented image. This introduces a form of consistency regularization which, this is crucial to $FixMatch$ success.

1.2 Algorithm Advantages:

- Simplicity – the simple nature of this new algorithm makes it easy to understand. In addition, following the process of the algorithm makes it clear when figuring out the impact of hyper-parameters fine-tuning.
- Familiar – Since it is a combination of two well-known methods: Consistency regularization and pseudo-labeling, each of the methods is well established in extensive researches. The trial and error of fine – tuning the required hyper parameters are already figured out.
- Performances – the performance of this model are better than the current state of the art for SSL algorithms.

1.3 Algorithm Disadvantages:

- Does not work well with Adam optimizer in accuracy performances wise.
- Critical usage of correct consistency regularization, if this stage does not perform well it influences on the overall performances of the model.
- Sensitive with initial learning rate, in the article they used cosine learning rate decay, which can be described in the formula, $\eta \cdot \cos\left(\frac{7\pi k}{16K}\right)$ where η is the initial learning rate. Incorrect values caused the model to do not converge and to perform poorly.

2. Improvement Suggestion:

The improvement we suggest the $FixMatch$ algorithm, is to train the model on the training dataset with several augmentation methods, and test evaluate their performances as a hyper-parameter for each dataset trained upon. For example, images with strong spatial correlation will benefit label classification formed from a certain augmentation method while other images will be more suited with another one.

Since both images, created from weak and strong augmentation, are generated using a shared original image, some of the attributes of the image are bound to differ. By altering the methods of the augmentation, and forcing the model to figure out which are the best ones for the current dataset, we think it will improve the models performances regarding correct classification of the pseudo labeling. Since this hyper-parameter will be evaluated only on the labeled data that is more scarce, it will not influence significantly on the time it takes the model to converge and achieve state of the art results.

3. Algorithm Comparison:

In the article they used the following algorithm for comparison: "ReMixMatch: Semi-Supervised Learning with Distribution Alignment and Augmentation Anchoring".

This algorithm managed to achieve 93.73% accuracy on CIFAR10 with 250 labeled examples, and a median accuracy of 84.92% with just four labels per class. While our algorithm on equal terms managed to achieve 94.93% accuracy on CIFAR-10 with 250 labels and 88.61% accuracy with 40 – just 4 labels per class.

We used a Convolutional Neural Network classifier for the different datasets. Since it is easiest to implement, and it has good results with relative short amount of training. However, it is important to mention that the $FixMatch$ algorithm is best suited when the labeled data is scarce, which is in most cases of usage with SSL algorithms. Since we don't have an available model with the same features of SSL and unlabeled data, Convolutional Neural Network is our best option at the moment.

Here is the model structure, layers wise:

Model: "CNN_Model"

Layer (type)	Output Shape	Param #
zero_padding2d_4 (ZeroPaddin	(None, 40, 40, 3)	0
conv2d_20 (Conv2D)	(None, 40, 40, 384)	10752
activation_9 (Activation)	(None, 40, 40, 384)	0
max_pooling2d_9 (MaxPooling2	(None, 20, 20, 384)	0
dropout_8 (Dropout)	(None, 20, 20, 384)	0
conv2d_21 (Conv2D)	(None, 20, 20, 384)	147840
conv2d_22 (Conv2D)	(None, 20, 20, 384)	590208
conv2d_23 (Conv2D)	(None, 20, 20, 640)	983680

conv2d_24 (Conv2D)	(None, 20, 20, 640)	1639040
activation_10 (Activation)	(None, 20, 20, 640)	0
max_pooling2d_10 (MaxPooling	(None, 10, 10, 640)	0
dropout_9 (Dropout)	(None, 10, 10, 640)	0
conv2d_25 (Conv2D)	(None, 10, 10, 640)	3687040
conv2d_26 (Conv2D)	(None, 10, 10, 768)	1966848
conv2d_27 (Conv2D)	(None, 10, 10, 768)	2360064
conv2d_28 (Conv2D)	(None, 10, 10, 768)	2360064
activation_11 (Activation)	(None, 10, 10, 768)	0
max_pooling2d_11 (MaxPooling	(None, 5, 5, 768)	0
dropout_10 (Dropout)	(None, 5, 5, 768)	0
conv2d_29 (Conv2D)	(None, 5, 5, 768)	590592
conv2d_30 (Conv2D)	(None, 5, 5, 896)	2753408
conv2d_31 (Conv2D)	(None, 5, 5, 896)	3212160
activation_12 (Activation)	(None, 5, 5, 896)	0
max_pooling2d_12 (MaxPooling	(None, 3, 3, 896)	0
dropout_11 (Dropout)	(None, 3, 3, 896)	0
conv2d_32 (Conv2D)	(None, 3, 3, 896)	7226240
conv2d_33 (Conv2D)	(None, 3, 3, 1024)	3671040
conv2d_34 (Conv2D)	(None, 3, 3, 1024)	4195328
activation_13 (Activation)	(None, 3, 3, 1024)	0
max_pooling2d_13 (MaxPooling	(None, 2, 2, 1024)	0
dropout_12 (Dropout)	(None, 2, 2, 1024)	0
conv2d_35 (Conv2D)	(None, 2, 2, 1024)	1049600
conv2d_36 (Conv2D)	(None, 2, 2, 1152)	4719744
activation_14 (Activation)	(None, 2, 2, 1152)	0
max_pooling2d_14 (MaxPooling	(None, 1, 1, 1152)	0
dropout_13 (Dropout)	(None, 1, 1, 1152)	0

conv2d_37 (Conv2D)	(None, 1, 1, 1152)	1328256
activation_15 (Activation)	(None, 1, 1, 1152)	0
max_pooling2d_15 (MaxPooling	(None, 1, 1, 1152)	0
dropout_14 (Dropout)	(None, 1, 1, 1152)	0
flatten_1 (Flatten)	(None, 1152)	0
dense (Dense)	(None, 10)	11530
activation_16 (Activation)	(None, 10)	0

Total params: 42,503,434
Trainable params: 42,503,434
Non-trainable params: 0

4. Evaluating Algorithm Performances:

4.1 Data Split:

One of the requirements in the assignment was to evaluate the model using 20 datasets. Due to time and difficulties with adjusting each dataset preprocessing we consulted Prof. Rokach, which instructed us to divide 3-4 datasets into subsets. So, we implemented this by separating the following datasets: CIFAR10, CIFAR100, MNIST, into 20 different subsets. Each, subset contains an even amount of specific labels. For instance, FICAR10 was divided into two subsets, each subset contains 5 unique class labels. Then, each subset data was divided into train and test sets in a ratio of: 80%:20%, respectively. These subsets were used throughout the first three stages of the assignment.

- The results are attached in a separate Excel file.

4.2 Hyper – Parameter Optimization:

According to the project requirements, ad performed hyper – parameters optimizations on the first two algorithms: $Learning\ Rate - \alpha, Unlabeled\ Ratio - \ \mu, Batch\ Size - B$. On the third algorithm we used only: $Learning\ Rate - \alpha, Batch\ Size - B$, since it requires the data to be labeled in the training phase.
For all three algorithms, the value of the ideal hyper – parameters were:
$Learning\ Rate - \alpha = 0.03\ , Unlabeled\ Ratio - \ \mu = 7, Batch\ Size - B = 64$

4.3 Performance Metrics:

4.3.1 Evaluating the Algorithm in 1st stage:

In the first stage of the assignment, we were asked to implement the $FixMatch$ algorithm presented in the article and try to achieve the same results in the results of their experiments.

4.3.2 Evaluating the Algorithm in 2nd stage:

In the second stage of the assignment, we were asked to suggest an improvement to the algorithm. We decided to add a hyper-parameter responsible to optimize the data augmentation method in the training set. After gathering results for each augmentation method, the best one was chosen for the testing phase. That way, each training phase was optimized using a more suitable augmentation method. Here are the results we managed to achieve using the suggested improvement.

4.3.3 Evaluating the Algorithm in 3rd stage:

In the third stage of the assignment, we were asked to compare our results to another algorithm. For that purpose, we chose a CNN algorithm which is more suited for labeled data while the $FixMatch$ algorithm is more suited for SSL, where the labeled data is more scarce. Here are the results of the CNN algorithm.

5. Statistical Significant Testing of the Results:

As required in the project description, the Friedman test examines if the several treatments have the same distributions.

In order to support that hypothesis, we performed the Friedman test on all three algorithms, presented above. Based on the results we recorded, over all datasets (no differentiation was made), we chose to carry out the Friedman test on two metrics: Accuracy and Area Under Curve (AUC).

For the two fixed metrics, we received a statistical value lower than $p - value = 0.05,$ and the following results: $StatValue_{Acc} = 0.023$, $StatValue_{AUC} = 0.017$

This indicates that we reject the Null Hypothesis, which means that the three distributions are different and does not have a similar statistical behavior. Furthermore, according to the project requirements, we performed an additional statistical test: Post – Hoc Test.

This test calculates the differences between the distributions in terms of $p - value$ and mean difference. The results of the algorithms improvement suggested vs the original algorithm, showed a small value (close to zero), this means that their distributions behave similarly. This can also be shown from the results presented in chapter 4 and corresponds to the fact that algorithms are relatively similar, since they differ in a small change.

The other Post – Hoc tests, $FixMatch\ Vs. CNN$ and $FixMatch\ Improvement\ Vs. CNN$, has received higher values: $Post - Hoc\ Tests \cong 5$. This indicates that their distributions behave entirely different, which makes sense when looking at the results and taking in to consideration that they are based on entirely different methods.

6. Conclusion Report:

Present the three algorithms you are evaluating (stages 1-3). Include the pseudo code, results and conclusions.

In the first three stages we implemented the following algorithms:

1. FixMatch Algorithm.
2. FixMatch Algorithm with improvement, augmentation method as a hyper-parameter.
3. CNN Algorithm

We reviewed them and explained their algorithm in their respective chapters. Now, in this part we will review their results and present our conclusions.

6.1 Results and Conclusions – 1st stage – $FixMatch$ Algorithm:

In our attempt to recreate the results presented in the article, we tried different ways in making the model to converge, our best results did not match those in the article, accuracy wise. While in the article they managed to achieve an accuracy score of 94.9%, our best results managed to achieve an accuracy score of 84.73%. We believe that we may have made some alterations in the algorithm in order to converge that influenced the models quality in the learning process. Another explanation is that the time we invested in training in the model was different from the one in the paper. That is because we had to run the model on different datasets. In the article they examined several options of number of labels for each class and then, using pseudo labels they managed to train the model on the unlabeled data, we did not fully accomplished this requirement and we isolated the dimensions of our learning algorithm to the ratios mentions in the article as hyper parameter, meaning: $Unlabeled\ Ratio = \mu = \{60\%, 70\%, 80\%\}$.

We are attaching our results as a max of all the different datasets on all hyper-parameters.

Accuracy	TPR	FPR	Precision
0.847318801	0.878574296	0.009879175	0.85360025
AUC	**PR_Curve**	**Training Time**	**Inference Time**
0.86006554	0.870855258	136.486802	2.729736041

6.2 Results and Conclusions – 2nd stage – $FixMatch$ Algorithm Improvement:

We are relying on the basic algorithm, so any underlying problems that we faced in the first stage will be also present in the second stage. As far as improving the algorithm, we can say that our attempt to hyper-parametrize the augmentation method has slightly increased the accuracy results of the model. However, that improvement was not so noticeable but the numbers show that the accuracy has increased by 1.84%. However, there was a slight increase in the training and inference time that is probably due to the added augmentation hyper-parametric search.

We are attaching our results as a max of all the different datasets on all hyper-parameters.

Accuracy	TPR	FPR	Precision
0.862970997	0.87064772	0.009985037	0.835512228
AUC	**PR_Curve**	**Training Time**	**Inference Time**
0.851555468	0.86780796	143.9993706	2.879987413

6.3 Results and Conclusions – 3rd stage – CNN Algorithm:

For comparing our algorithm, we chose the CNN algorithm. The comparison algorithm presented much better results than the ones we presented previously with relatively short amount of training in comparing (training time was slightly lower). That is probably because of the usage of 100% labeled data. The CNN algorithm receives the y vector, representing all the labels of all the input data, unlike the $FixMatch$ algorithm, which receives only 20%-40% of the data as labeled. Another option for the superior performances is because that in our algorithm of comparison, we are not obligated to any restrictions and can increase the depth and width of the network as much as we want.

We are attaching our results as a max of all the different datasets on all hyper-parameters.

Accuracy	**TPR**	**FPR**	**Precision**
0.929194473	0.967602522	0.009972044	0.934119369
AUC	**PR_Curve**	**Training Time**	**Inference Time**
0.947366316	0.935142426	113.9913955	2.279827909