Delft University of Technology

Deep Learning
CS4240

Reproducibility Project

Semi-Supervised Medical Image Segmentation via Learning Consistency under Transformations

Authors:

Jari Malfliet (4668626)
J.M.L.Malfliet@student.tudelft.nl

Roemer Bakker (4367138)
R.M.B.Bakker@student.tudelft.nl

Supervisor:
Gerda Bortsova

Teaching assistant:
Olivia Huang

April 15, 2021

https://reproducedpapers.org/papers/2PQoEv5T1qDBuoBK5OCY#r4JYEkuMnyAKZi4Y6o5eg

Github: https://github.com/AQUA7XICE/Deep-learning


TU Delft
Delft
University of
Technology

1 Introduction

In this reproducibility project, a simplified version of the method in the paper on *Semi-Supervised Medical Image Segmentation via Learning Consistency under Transformations* [1] is implemented and tested on the CIFAR-10 benchmark dataset.

Goal of the reproduced paper

In medical imaging, deep learning can be used for segmentation of X-ray images (segment organs, structures etc.) to improve medical analysis. However, scarcity of labeled data limits supervised techniques in doing so. The paper concerned in this blog post aims to provide a solution by proposing a semi-supervised technique, by using both labeled and unlabeled images. Supervised learning on the labeled images is extended by learning to consistently predict segmentation under transformations on both labeled and unlabeled data. That is, an image is still segmented correctly after transformation as compared to no transformation. The paper used a U-Net-like architecture [4]. The network used in the original paper is summarised in Figure 1.

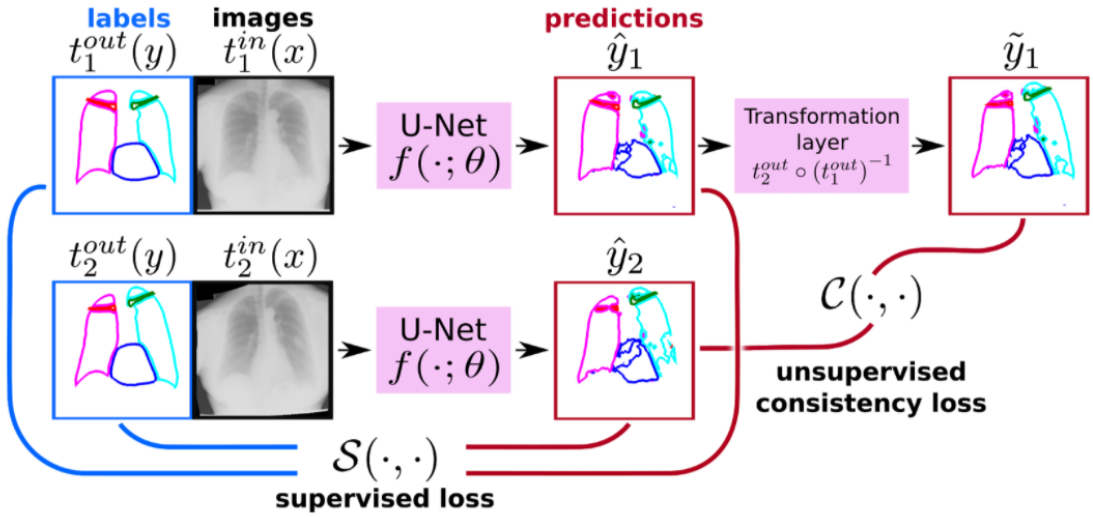

Figure 1: The network used in the original paper. The inputs are mixed batches of labeled and unlabeled images. Every image x is transformed by two random mappings t_1^{in} and t_2^{in}. The label y, if available, is transformed by t_1^{out} and t_2^{out} . $t_1^{in}(x)$ and $t_2^{in}(x)$ are fed to the two identical branches of the network. The output of one of the branches is transformed by a differentiable layer for comparison with the output of the second branch in the consistency loss $\mathcal{C}$. The network is trained end-to-end using a combination of $\mathcal{C}$ and a supervised loss $\mathcal{S}$ [1].

Reproducibility goals

The simplified method used during this reproducibility project allows for exploring the same main goal as that of the original paper, i.e. increasing network accuracy with an extension of the labeled dataset by only unlabeled images. One of the benefits of this simplified method is the reduced training time of the network. Therefore, it is easier to experiment with different types of transformations and hyperparameters.

For this project, parts of the code written by the author of the original paper are reused, namely the network architecture, the general framework and pre-processing. The code is made more user-friendly and readable by creating a single IPython notebook (Google Colab) that is able to run the code. Training, implementing transformations, hyperparameter tuning (adjusting the number of epochs) and an ablation (leaving out the supervised step) were done by us. After implementing the simplified method and during the training experiments of the simplified method, the first steps towards implementing the full method are taken, this includes coding the original architecture, the U-net segmentation network. However, no results of the full method are obtained and are thus not included in this blog post.

2 Simplified method

In the following, semi-supervised learning via consistency under transformations, as used in the reproducibility, is explained. It closely resembles the method in [1]. The goal is to find the parameters of the network that minimize the objective shown in Equation 1.

$$\min \mathcal{L}_{\mathcal{S}}(\theta) + \lambda \mathcal{L}_{\mathcal{C}}(\theta) \tag{1}$$

With $\mathcal{L}_{\mathcal{S}}$ being the supervised loss and $\mathcal{L}_{\mathcal{C}}$ being an unsupervised consistency loss. The method consists of two training steps and is quite similar to the method described in [3]. The first step is training the model, optimizing only the supervised loss, i.e. $\lambda = 0$. The resulting weights are transferred to the model used in the second step, which optimizes both the supervised and unsupervised consistency loss.

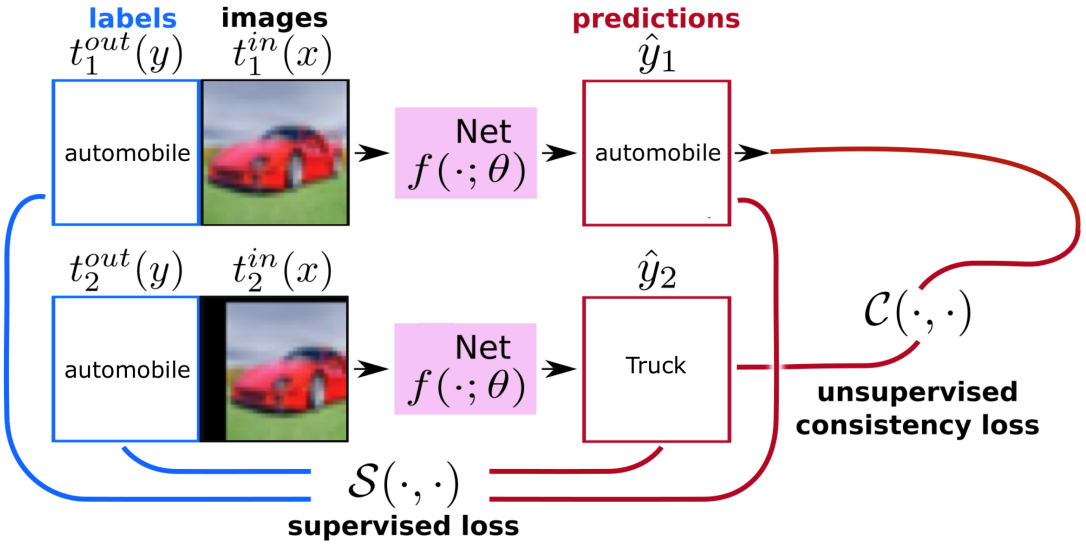

Figure 2: The simplified network lay-out

Loss function

The loss function used to compare predictions to their labels, and predictions to each other for consistency, is the Kullback-Leibler (KL) divergence loss. This particular loss function calculates the deviation of a probability function q from the true probability function q.

$$D_{KL}(p\|q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \tag{2}$$

p can be the distribution of labels, with q the distribution of predictions. Ideally the probability for each label should be the same in each distribution. The KL divergence loss is made symmetric by also computing the loss with the predictions as p, and the labels as q; then adding it to the alternative and divide by two. The same procedure can be used to compute the KL divergence loss between two predictions.

Architecture

The architecture used in the original paper is a segmentation architecture based on U-net [4] and will therefore not be used in the simplified method. In the simplified method, a smaller convolutional network, suitable for classification instead of segmentation, is used. The architecture was provided by the author, and can be seen in Appendix A.

Step one: supervised learning

In the first step, the supervised learning step, the CIFAR-10 dataset is split in three subsets. The first subset are the labeled training images, $|\mathcal{X}^{train}| = 4000$. The second subset are the validation images used in training, $|\mathcal{X}^{val}| = 1000$. The third subset are the test images used for network evaluation after training, $|\mathcal{X}^{test}| = 10000$. This datasplit is called `cifar10_default` in the used code.

The batch size is set to 128 and a transformation of the input images x is added to the batch, say x^t; the resulting batch is $[x, x^t]$. The label set is $[y, y]$ for classification, and $[y, y^t]$ for segmentation. The network gives a prediction for both parts of the batch, say $\hat{y}_1$ and $\hat{y}_2$ respectively. This set of labeled predictions is denoted by $\hat{y}_l$. Then, $\hat{y}_l$ is compared to set of labels y_1 with the loss function, resulting in the supervised loss. The network weights obtained in this step will be retained and are reinstalled at the start of the second training step.

Step two: semi-supervised learning

The weights obtained in the previous step are loaded in as starting weights. Now, both labeled and unlabeled images shall be used to train the network. The CIFAR-10 dataset is now split in four subsets: the labeled training images $|\mathcal{X}_l^{train}| = 4000$, the unlabeled training images $|\mathcal{X}_{ul}^{train}| = 45000$, the training validation images $|\mathcal{X}^{val}| = 1000$ and the network evaluation test images $|\mathcal{X}^{test}| = 10000$. This datasplit called `cifar10_ssl_default` in the code.

The core idea of semi-supervised learning here, is that the consistency loss is added to a supervised loss, resulting in a total loss. The supervised loss is calculated analogously as in step one, except that $\hat{y}_1$ and $\hat{y}_2$ now also contain unlabeled images. In this case, $\hat{y}_l$ is constructed by including the first $n_{labeled}$ images of both $\hat{y}_1$ and $\hat{y}_2$. y_1 is constructed likewise (as you might opt to use a labeled dataset as partially unlabeled).

After the supervised loss is computed, the consistency loss (unsupervised loss) is determined. For segmentation, $\hat{y}_1$ is transformed with the same transformation previously applied on x^t, and compared with $\hat{y}_2$ in the loss function: this is the consistency loss. For classification, $\hat{y}_1$ obviously cannot be not transformed. The (transformed) $\hat{y}_1$ should ideally be identical to $\hat{y}_2$. Remember that $\hat{y}_2$ is the prediction on x^t, while transformed $\hat{y}_1$ is $\hat{y}_1^t$ for segmentation.

Transformations

Three types of transformation are considered in this reproduction: random noise, random shift, and random rotation. The network is trained for only one transformation at a time, i.e. the batches only undergo one type of transformation. These three particular transformations were chosen as they could very well occur in real-life conditions. Both the original and transformed image will receive a prediction from the neural network. The consistency between those predictions is expressed by the unsupervised consistency loss. The shift and rotation transformations are implemented with the help of the elasticdeform library[1]. This library also includes a function that is able to backpropagate the gradient through the transformation. This allows for a convenient implementation of the deformation as a layer in a convolutional neural network. As can be seen in Figure 1, this is required for the prediction transformations in the full method of the reproduced paper.

Gaussian noise

A random value is sampled from a Gaussian distribution with a given standard deviation ($\sigma = 0.1$). This value is added as noise to the image; the final value is clipped between the extremes of the transformed batch. A visual example is shown in Figure 3.

[1] https://github.com/gvtulder/elasticdeform

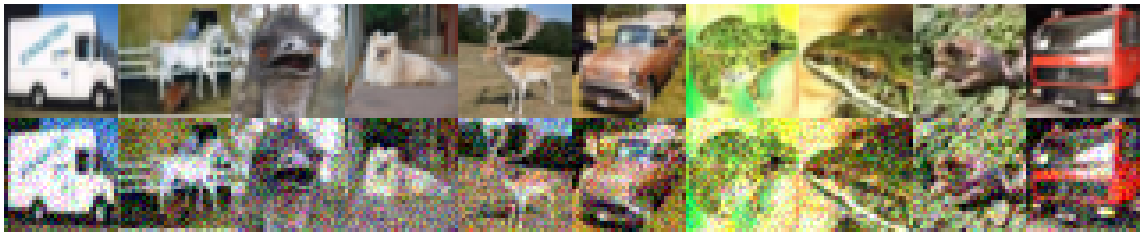

Figure 3: Random cifar images and its transform by applying random noise with mean equal to zero and a standard deviation of 0.1.

Random shift

The goal of this transformation is to shift the image by a random amount of pixels (with a maximum of 10), both horizontally and vertically. A 2D vector with two random shift values between -10 and 10 is sampled. One is used for the horizontal shift, the other for the vertical shift. A demonstration is given in Figure 4.

Figure 4: Random cifar images and its transform by randomly shifting the image with a maximum of 10 pixels.

Random rotation

The transformation consists of a rotation, randomly selected between -45 to 45 degrees with a uniform distribution. An example of 10 rotated images is shown in Figure 5.

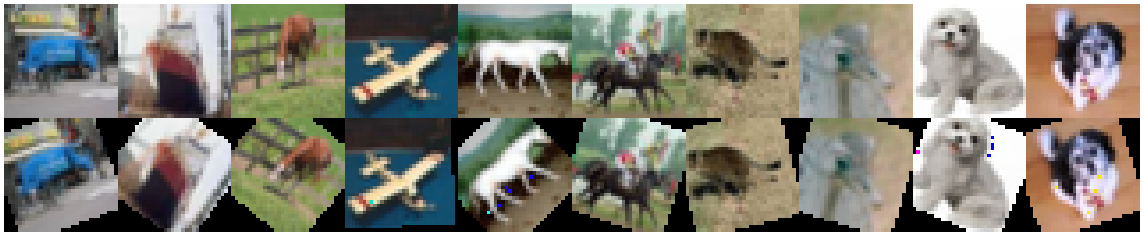

Figure 5: Random cifar images and its transform by uniformly randomly rotating the image with a value between -45 and 45 degrees.

3 Experiments & results

This section describes the performed experiments and presents the corresponding results. Four experiments per transformation were executed, making a total of 12 experiments.

Supervised learning

For supervised training of each transformation, the Adam optimizer is chosen and a learning rate of 0.001. The model is trained to optimize only supervised ($\mathcal{S}$) loss for 120 epochs. With the training images, $\mathcal{X}^{train}$, and a batch size of 128, training takes about 10 to 15 minutes.

The loss and accuracy curves can be seen in Figure 6. Looking at the KL divergence loss, it can be seen that loss on the training set approximates zero from the 30th epoch onwards for all

transformations. The training losses on the validation set drop to a minimum of 1.2 around the 20th epoch, then rise again above 2. Especially the rotation transformation validation set performs badly, with a loss of 2.8. The rise in in validation loss might be the result of overfitting of the model on the training data. In the lower plot of Figure 6, the accuracy over training is presented. The training sets converge to 1, i.e. full accuracy. For the validation sets, the accuracy stagnates at 0.6 from epoch 25 onwards.

The model is evaluated with the test images, $\mathcal{X}^{test}$. The results of this evaluation can be seen in Table 1, note that only the supervised results of this evaluation are included. The obtained accuracy is about 64% to 68%, what is mainly due to the limited size of the training set.

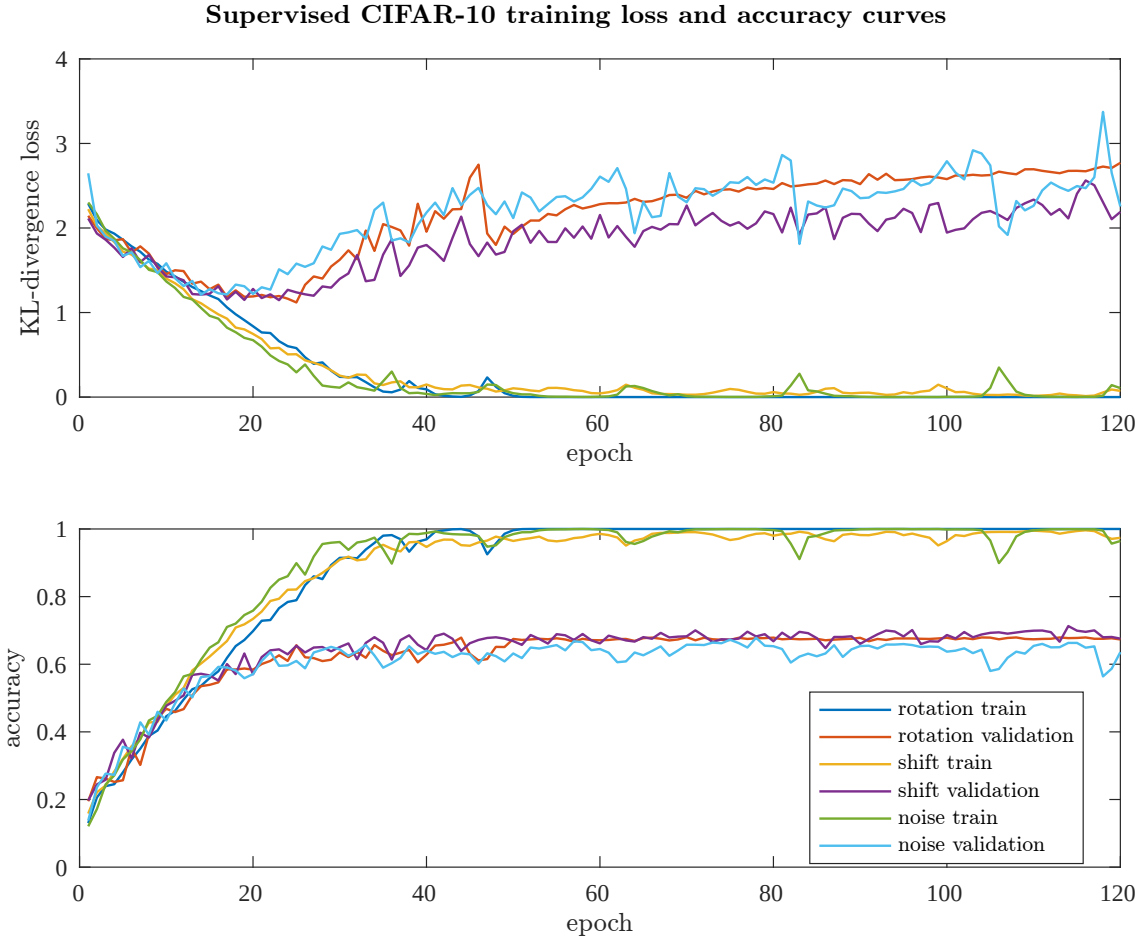

Figure 6: Loss and accuracy training curves for the supervised learning step for three different transformations: noise, shift and rotation

Semi-supervised consistency learning (*SemiC*)

The second experiment is training the network for both the supervised ($\mathcal{S}$) and the unsupervised consistency ($\mathcal{C}$) loss. This is done with a new and untrained network with an Adam optimizer and a learning rate of 0.001 and 0.00001. With the labeled and unlabeled training images, $\mathcal{X}_l^{train}$ and $\mathcal{X}_{ul}^{train}$, and a batch size of 128, training for 120 epochs takes about 4 to 5 hours.

The experiments which used a learning rate of 0.00001 did not converge during training for 120 epochs and these results are therefore not presented. For a learning rate of 0.001, all three transformations converge within the 120 epochs. The results can be seen in Table 1 (*SemiC*). For both the shift and rotation transformations the accuracy is improved with about 10% compared to the pure supervised training. The noise transformation does not show increased accuracy. This training method is seen as an ablation compared to the paper: step 1 is omitted, training is directly done without implementing supervised weights first. This is done in the next section.

Two-step semi-supervised consistency learning (*SemiCT*)

The third experiment concerns semi-supervised learning, but with pre-trained weights from the pure supervised training (also called transfer learning). This was one of the methods proposed in the original paper. The pre-trained weights used are obtained by training the pure supervised loss for 120 epochs. The learning rate is set to 0.00001 and again the Adam optimizer is chosen. The network is trained with both the labeled and unlabeled training images, $\mathcal{X}_l^{train}$ and $\mathcal{X}_{ul}^{train}$, and a batch size of 128. Training for 120 epochs takes about 4 to 5 hours.

All three transformations converged within the 120 training epochs, the results can be seen in Table 1 (*SemiCT*). The shift transformation performed better than without pre-trained weights. However, the noise and rotation transformations converge to a point where the unsupervised consistency loss is equal to zero and the supervised loss is very large. Even when the learning rate is increased from 0.00001 to 0.1, the model still converges to these local "optima". This is a bad result. Even though the predictions of the network are consistent, the labels themselves are erroneous. e.g., an image of a car and its transformed may be both predicted as a truck; it is consistent, but unsatisfactory. The reason for this behaviour lies in the pre-trained weights, but the exact reasoning is yet unknown and further research is recommended.

Two-step semi-supervised consistency learning early stopping (*SemiTCE*)

As part of extra criteria, a hyperparameter sensitivity study was done, focusing on adjusting the number of epochs in step 1 (supervised learning). This technique is known as 'early stopping'. Early stopping is defined as halting the training process once a certain convergence condition is met [2]. In our case, we chose the point where validation set accuracy stagnated in step 1. For all transformations, this was around the 25^{th} epoch, as can be seen in Figure 6. Therefore, the number of epochs in step 1 is reduced to 25. The resulting weights are installed in step 2. The learning rate was 0.00001 (the same as pre-trained for 120 epochs). Early stopping is a method to prevent overfitting, thus, one would expect better results on the test set after this particular hyperparameter tuning step, as opposed to the recommended hyperparameters.

For the noise transformation, the early stop loss is lower than for training with *SemiC* (Table 1). The supervised accuracy is 3.8% higher, while unsupervised accuracy is 0.3% lower. It is obviously better than *SemiCT*, this indicates that the problem encountered in *SemiCT* is related to weights. Overfitting is speculated to occur, and is halted by early stopping.

For the shift transformation, the loss is lower than for *SemiC*, but slightly higher still than for *SemiCT*. Its accuracy is higher than *SemiC* for both supervised and unsupervised data. Compared to *SemiCT*, the supervised accuracy is 1.1% lower, but unsupervised accuracy is higher.

Looking at early stopping for the rotation transformation, a significant performance increase can be seen compared to semiCT. However, compared to SemiC, its accuracy drops on both supervised and unsupervised sets, by 4.3% and 11.3% respectively.

For all transformations, early stopping does improve supervised accuracy compared to just running the supervised step alone. 1% for noise, 7.9% for shift, and 4.6% for rotation.

Table 1: Model evaluation test results for four types of training: pure supervised, semi-supervised consistency (*SemiC*), two-step semi-supervised consistency with 120 epoch supervised pre-trained weights (*SemiCT*) and two-step semi-supervised consistency with early stopped supervised pre-trained weights (*SemiCTE*).

Transform	**Method**	**Loss**			**Accuracy**	
		Total	$\mathcal{S}up$	$\mathcal{C}ons$	$\mathcal{S}up$	$\mathcal{C}ons$
Noise	*Supervised*	-	2.26	-	63.9%	-
	SemiC	1.72	1.69	0.03	61.1%	93.4%
	SemiCT	14.5	14.5	0.0	0.1%	100%
	SemiTCE	1.491	1.455	0.036	64.9 %	93.1%
Shift	*Supervised*	-	2.11	-	68.3%	-
	SemiC	0.84	0.76	0.08	74.8%	88.7%
	SemiCT	0.76	0.70	0.07	77.3%	89.0%
	SemiTCE	0.793	0.723	0.07	76.2%	90.0%
Rotation	*Supervised*	-	2.84	-	66.7%	-
	SemiC	1.53	1.44	0.09	75.6%	93.2%
	SemiCT	14.5	14.5	0.0	0.1%	100%
	SemiTCE	1.7	1.283	0.417	71.3%	81.9%

Discussion on the results

Looking at all the results summarised in Table 1, there can be noticed that the proposed method, semi-supervised consistency learning, can indeed contribute to obtaining a higher network accuracy. This is in line with the results in the original paper [1].

However, the type of transformation is determinative for the usefulness of the proposed method. The semi-supervised experiments with the noise transformation do not show substantial improvement compared to the pure supervised experiment (a 1% increase with early stopping). The other two transformations, shift and rotation, which are both spatial transformations, do show improvement. *SemiCT* is the best method for shift, with a supervised accuracy increase of 9%. *SemiC* is the best for rotation, with a 8.9% increase in supervised accuracy.

The weight transfer learning applied in *SemiCT* does not always show improved accuracy compared to the normal semi-supervised experiments, *SemiC*. This may be due to the overfitting happening in the first step. If the overfitting is prevented in step 1 by early stopping, the results are, namely, much better for the noise and rotation transformations. Overfitting does not happen in shift transformation training.

4 Conclusion

In this blog post, a reproducibility of a simplified method of the paper by Bortsova et al. [1] was proposed. The goal of the paper is to develop a semi-supervised method for consistent segmentation of medical images under transformation. This was simplified to a semi-supervised method to correctly classify labeled and unlabeled images from the CIFAR-10 dataset. The total loss exists of a supervised loss, based on the labeled images, and a consistency loss between predictions stemming from an image and its transformed counterpart. Three types of transformations were investigated: adding gaussian noise, shifting the image, and rotating the image. The network should be able to predict the correct label for a transformed image, and predict the same labels for an image and its transformation.

If all outcomes of the experiments are compared to each other, there can be concluded that the proposed method, semi-supervised consistency learning, can indeed contribute to obtaining a higher network accuracy. However, after running experiments with different types of transformations, there can be concluded that the degree of usefulness highly depends on the input given to the network. It seems that the proposed method performs better when a spatial transformation is applied (e.g. shift or rotation). Furthermore, one should be careful with using pre-trained weights in a network. Implementing weights that are obtained by overfitting on a training set can lead

to reaching local "optima" that might does not come close to the real optimum. Therefore, one could argue that the proposed method can be used more safely without transfer learning, without sacrificing a lot of achieved accuracy while doing this.

Although the proposed method might not be very relevant to pure image classification, because classification labels are much easier to obtain then segmentation pixel labels of entire images, the simplified method is still proven to be useful and properly working. The results of the experiments done with the simplified method show much resemblance with the original method from [1]. One should, therefore, recognize that exploring new methods in the Deep Learning field using simplified versions can actually contribute to the understanding of and trust in these new methods.

Appendix A

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 3, 32, 32)]       0
_________________________________________________________________
lambda_1 (Lambda)            (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 96)        2688
_________________________________________________________________
batch_normalization_9 (Batch (None, 32, 32, 96)        384
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 32, 32, 96)        83040
_________________________________________________________________
batch_normalization_10 (Batc (None, 32, 32, 96)        384
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 96)        83040
_________________________________________________________________
batch_normalization_11 (Batc (None, 32, 32, 96)        384
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 96)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 96)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 192)       166080
_________________________________________________________________
batch_normalization_12 (Batc (None, 16, 16, 192)       768
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 16, 16, 192)       331968
_________________________________________________________________
batch_normalization_13 (Batc (None, 16, 16, 192)       768
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 192)       331968
_________________________________________________________________
batch_normalization_14 (Batc (None, 16, 16, 192)       768
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 192)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 192)         0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 192)         331968
_________________________________________________________________
batch_normalization_15 (Batc (None, 8, 8, 192)         768
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 192)         37056
_________________________________________________________________
batch_normalization_16 (Batc (None, 8, 8, 192)         768
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 192)         37056
_________________________________________________________________
batch_normalization_17 (Batc (None, 8, 8, 192)         768
_________________________________________________________________
global_average_pooling2d_1 ( (None, 192)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1930
=================================================================
Total params: 1,412,554
Trainable params: 1,409,674
Non-trainable params: 2,880
```

References

Bortsova, G., Dubost, F., Hogeweg, L., Katramados, I., & de Bruijne, M. (2019). Semi-supervised medical image segmentation via learning consistency under transformations. *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, 810–818. Retrieved from http://dx.doi.org/10.1007/978-3-030-32226-7_90 doi: 10.1007/978-3-030-32226-7_90

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., & Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. *CoRR*, *abs/1804.09170*. Retrieved from http://arxiv.org/abs/1804.09170

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, *abs/1505.04597*. Retrieved from http://arxiv.org/abs/1505.04597