Biologically Plausible Semi-Supervised Learning by Local Training of Deep Neural Network Layers

Raffael Theiler (10-928-893), Kaan Oktay (18-796-508), Jonas Frey (19-945-336)

I. Introduction

Deep neural networks (DNNs) are commonly trained by the backpropagation algorithm with a global loss function. However, layerwise loss functions for training DNNs locally are biologically more plausible in comparison [1]. While most DNNs strongly diverge from the original biological inspiration, it is shown in [2, 3] that more biological plausible local loss functions can reduce memory requirements and computational costs while achieving comparable results to global backpropagation. In addition, semi-supervised learning approaches are becoming very popular because they require less expensive labeled training data [4, 5]. In this project, we combine these two approaches and suggest biologically inspired modifications to a semi-supervised learning method which trains DNN layers locally using supervised autoencoders [6].

II. Models and Methods

Many researchers have focused on finding biological plausible neural network training procedures and therefore multiple different methods have been proposed recently. Nøkland *et al.* trained a VGG-like architecture locally for each layer using similarity matching and prediction loss [2]. In [3], authors proposed a training mechanism by generating errors locally in each layer using fixed, random auxiliary classifiers. In [7], authors used k-hidden layer problems to sequentially build a deep network layer by layer and showed that layerwise training can scale to the ImageNet dataset.

At the same time, semi-supervised methods are gaining significant popularity in the field of deep learning. In [4], the authors merged three dominant approaches: entropy minimization, consistency and traditional regularization. In [5], authors showed that applying high quality noise to unlabelled data improved the results obtained by simple consistency regularization methods and outperformed the previous works.

Lastly, in [6], the authors proposed an autoencoder architecture which makes use of labeled information and therefore is able to learn a better latent representation of the high dimensional input data, compared to an unsupervised approach.

Based on these ideas, we implemented and analyzed mainly two different networks which were designed to solve semi-supervised tasks using local training. In both networks, the main building blocks are supervised autoencoder (SAE) [6]. Driven by the idea of only using a local loss functions to train the network, we treated one instance of a SAE as an individual local-loss block.

The loss function of the SAE consists of the reconstruction loss and the classification error. For the supervised training samples, a label is predicted by a fully connected layer attached to the bottleneck of the SAE. The cross-entropy prediction loss and the mean-squared-error (MSE) reconstruction loss function of the decoder part are added (with specified weights) to find the overall loss.

A. Model 1: Identical SAEs

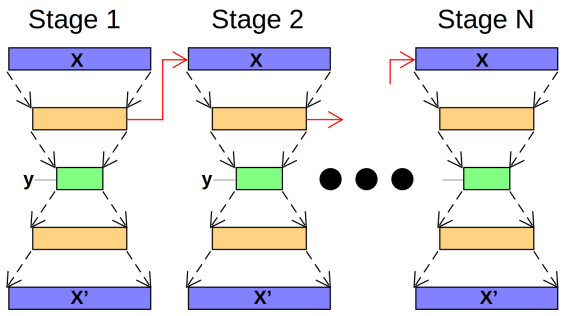

Fig. 1: Stacked identical SAEs where red arrow illustrates the layer transfer function.

The first implemented and evaluated model consists of stacked SAEs as illustrated in Fig. 1, which we proposed in our project proposal. We trained every SAE locally and fed the learned representation after the first convolutional layer to next stage's SAE input. Because, i.e. for CIFAR-10 dataset, the learned representations (24x32x32) and the input of the identical SAEs (3x32x32) have different dimensions, we evaluated two different ideas for passing the learned representations to the next stages (maps). The first idea was to use a non-trainable interpolation map to propagate the learned information. The next idea was to apply a thin, trainable convolutional map to learned representations.

Because of reasons we discuss in the later sections, we devised a different model which is introduced in the next subsection.

B. Model 2: VGG-like Structure

It became clear that the additional search for a working network structure is out of scope on top of our main objective. Even though there might be a more fitting architecture, we decided to fall back to an elaborated network and train it by embedding it layer-wise in our locally trainable SAE, as this seemed sufficient to show our conceptual idea. From [2] we knew that this procedure works, given a clever training function. Choosing the well-proven VGG structure for image recognition tasks allowed us to focus more on the local training model, our main object of investigation. The main structure of the network is shown in Fig. 2.

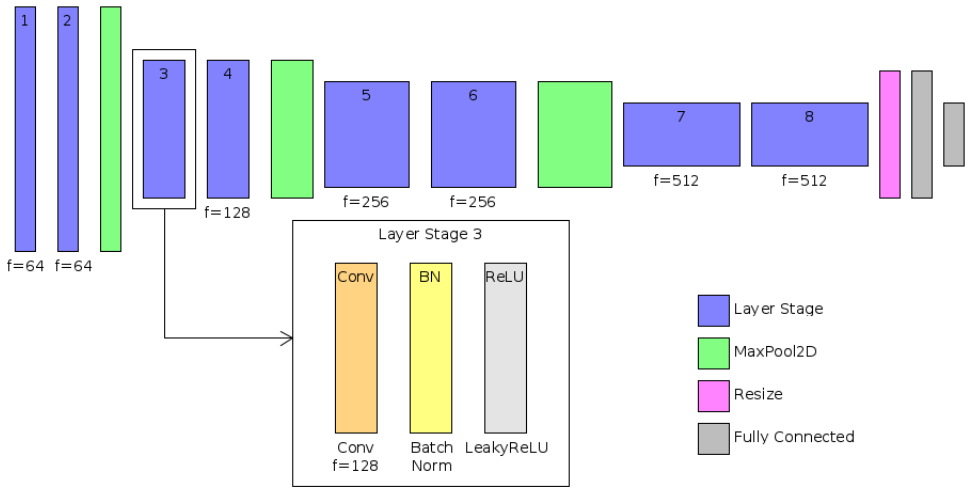

Fig. 2: VGG structure with 8 convolutional layers and final fully-connected layers.

To incorporate local training, we decompose the network into sequential "layer-stages", marked with a box in Fig. 2, each consisting of a $Conv$ layer, a $BatchNorm$ normalization and a $LeakyReLU$ activation function. An individual "layer-stage" is trained by an unique across all layers SAE which is demonstrated in Fig. 3. To train the first "layer-stage" of the VGG network locally, the "layer-stage" parameters are shared with the first encoder level of the SAE structure. After training this SAE structure, we successfully learned the parameters of the first "layer-stage" of VGG. Then this stage is cut from the optimization graph and is used to map the input images of the whole VGG structure to the input of next "layer-stage". If necessary, we apply max-pooling to the input of next-stage. Otherwise, we directly pass it. Then the second "layer stage" is trained by sharing its parameters with another SAE structure (properly designed for the size of each VGG layer) and so on. By repeating this procedure for all "layer-stages" of the VGG network, we can avoid global backpropagation and train each stage locally.

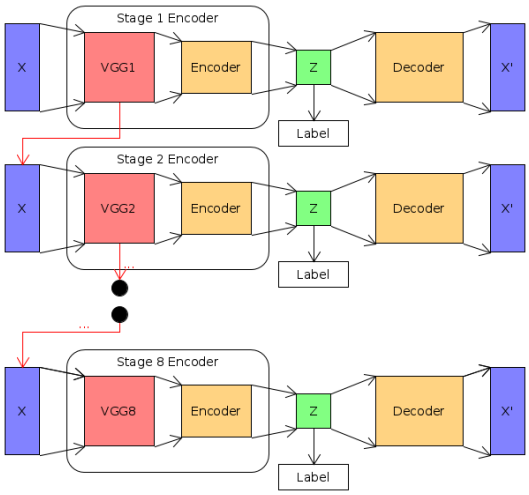

Fig. 3: Local training of VGG layers with SAE structure.

To give a fair comparison to other models, we have to distinguish between the parameters necessary to optimize at each epoch and total parameters. In a standard global backpropagation optimization procedure, all the parameters are optimized at each pass. However in our case, we just need to optimize the the parameters of SAE structure which is being trained at that point of optimization. This significantly reduces the number of parameters to optimize at each run.

Although we train the final linear layers as part of the VGG structure, our aim was to predict the labels correctly in each prediction part of local SAEs as well. We came up by an additional idea and used a mean probability prediction in which we averaged class probabilities of individual SAEs and compared it to the one hot label vector. This prediction scheme was used as the final prediction of class probabilities.

III. Results

Before we give the results, we would like to point out two different issues. First, we made experiments with the Model 1 but both proposed feature transportation methods resulted in very poor performance with the CIFAR-10 dataset, where the best configuration resulted in a prediction accuracy of 59% using a semi-supervised approach (10% of the labeled data for training). More importantly prediction accuracy dropped consistently as we go towards the later layers. A brief explanation why this idea was not able to increase the prediction performance with additional depth is provided in the discussion section. We exclude this model from the remaining parts of results section, since no reasonably good results could be achieved but the lessons learned and discovered problems are still worth further discussion.

The second issue was that we were not able to use STL-10 dataset even though we proposed so in the project proposal. STL-10 dataset has 3 times bigger images compared to CIFAR datasets and this resulted memory issues with our current network training methodology even on the GTX1080/2080 graphics cards available on the Leonhard culuster. In the

future, this issue might be resolved by more efficient implementation of the network training procedure.

For all the experiments of which we present the results after this point, we applied standard data transformations to the training images including random horizontal flip, color jittering, random crop and random rotation. We trained each stage for 25 epochs with Adam optimizer [8] with default settings and batch size 64. We used MSE loss for reconstruction task and cross-entropy loss for prediction task. Training time averaged at 1 to 2 hours. In addition, because we used only some proportion of labeled data for our semi-supervised task, we augmented our labeled data using the transformations mentioned above. We decreased our learning rate while going deeper through network as in the following setting: 8e-4, 7e-4, 6e-4, 5.1e-4, 4.2e-4, 3.3e-4, 2.4e-4 and 1.6e-4, respectively from 1st to 8th VGG layer local training with SAE. For the final linear layers of VGG structure, we used learning rate 1e-4.

It is important to mention that whether a sample is labeled or unlabeled is drawn in the initialization of the training procedure and kept fixed during the whole training procedure. Therefore, the labeled is strictly separated before the training.

The layers of our VGG used for training can be summarized as follows: **Conv64 - Conv64 - Pool - Conv128 - Conv128 - Pool - Conv256 - Conv256 - Pool - Conv512 - Conv512 - FC1024 - FC** where the dimension of the last fully-connected layer was determined by the number of classes in the respective dataset.

Network	Params (Mio)
Layer 1	4.24
Layer 2	4.31
Layer 3	2.40
Layer 4	2.55
Layer 5	2.24
Layer 6	2.83
Layer 7	2.25
Layer 8	7.63
VGG Total	13.09

TABLE I: Number of Parameters per VAE layer. An exception is VGG where we show the total number of parameters for the full vertical VGG network.

For each SAE used for local training of VGG layers, we used the following base encoder/decoder scheme for the experiments. Input to the SAE is obtained by a forward pass of the input image through previous VGG layers and current VGG layer is our first layer in the local SAE. Then we encoded it by applying max-pooling and subsequently applying a convolutional layer which divides the channel size by two. The decoder of the local SAE is basically the inversion of the encoder. The max-pooling operation of the encoder is replaced with interpolation that expands the layer size by a factor of two in the decoder structure. The prediction part of the local SAEs is the same for all layers. At first the bottleneck is reshaped and fully connected with 512 neurons and a final fully connected output layer which size is determined by the number of classes of the respective dataset. We call this encoder/decoder type C. For additional experiments, we introduced type A and B architecture which are modifications of type C. Type B architecture switches the max-pool and interpolation layers of the encoder and decoder respectively. Therefore the encoder increases the channel size and the decoder reduces it. Type A deviates from type B by adding an additional max-pooling layer combined by a subsequent convolutional layer, which multiplies the channel size by 2, before the prediction stage.

For the experiments, our first idea was to fully train one layer-stage until it converges at a time and repeat this procedure for all layer-stages sequentially. We named this training procedure "sequential training". As an alternative to this approach, we introduce the "wave training" procedure which basically trains every layer subsequently for one epoch, then start over again at the first layer and repeats the training. We will compare their performances later in this section.

As a base configuration, we used the encoder/decoder type C, 4000 labeled images (8% of training data), wave-mode training with dropout rate 0.1. In the following parts, we inspected overall performance of our base configuration, compared it to state-of-the-art baselines and demonstrated the effects of different changes to the base configuration. Unless otherwise stated, all the results were obtained using base configuration with 4000 labeled images of CIFAR-10 dataset.

A. *Overall performance of base configuration*

Here, we inspect the overall loss/accuracy performance of training and test procedures. As can be seen in Fig. 4, training loss decreases and accuracy increases over epochs for each layer given in the plot. However, as we go through deeper layers, the rate of performance increase degrades. This may be due to the fact that even in the first epochs, later layers begin with useful representations transported from previous layers.

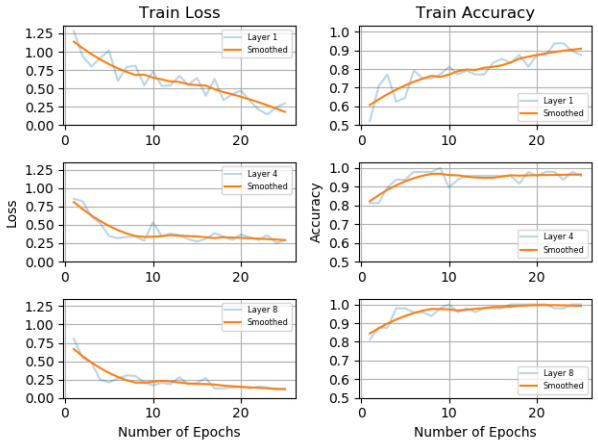

Fig. 4: Training performance for different layers.

In Fig. 5, we can see the performance on test set. Although accuracy follows a similar pattern as training set, loss of test data consistently increases over epochs for all the layers demonstrated. We might explain this discrepancy by some outlier data in the test set which introduces much higher losses compared for both reconstruction and prediction loss. And because loss and accuracy are not the same objectives,

even if we have some test examples with huge losses, we can get a descent accuracy in overall thanks to more statistically expected test examples.

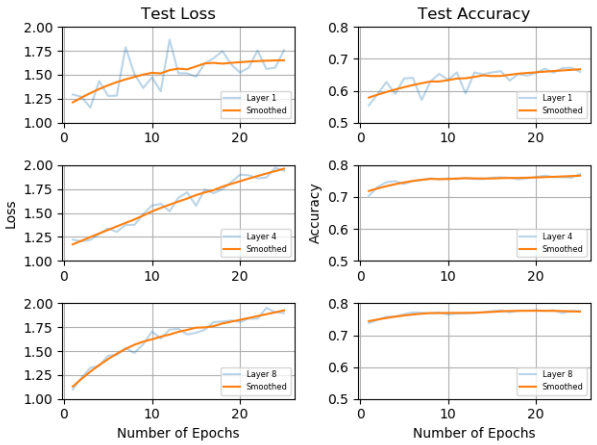

Fig. 5: Test performance for different layers.

B. Comparison to baselines

Our base configuration is suitable for semi-supervised setting. However, for comparing our local loss training performance to [2], we used all the labeled images (other parameters were left unchanged) because authors followed a fully-supervised approach. For fully-supervised setting, we used batch size 126. For both settings, we reported accuracies calculated by mean prediction probabilities method explained before.

The achieved performance in comparison to state of the art results for fully-supervised and semi-supervised training for the the CIFAR-10 and CIFAR-100 datasets are illustrated in Table II and Table III. We abbreviated our network as SAE-VGG and reported values are percentage of test accuracy error.

As can be clearly seen, even though our method is not as successful as the baselines, it is quite close to the performance of [2] in fully-supervised setting and by using the extensive tricks in [4] in a future work, there is pretty much room for further improvement in semi-supervised setting.

Method	# par	CIFAR-10	CIFAR-100
VGG8B[2]	8.9M	5.99	24.1
SimCNN[7]	-	9.60	-
SAE-VGG	13.09M	9.10	30.6

TABLE II: Comparison with fully-supervised layerwise training methods. We reported the result of [7] with 2-hidden layer setting and [2] with predsim loss setting. Number of parameters are for CIFAR-10 dataset.

Method	CIFAR-10	CIFAR-100
SWA [9]	5.00	28.8
MeanTeacher[10]	6.28	-
MixMatch[4]	4.95	25.88
SAE-VGG	20.9	48

TABLE III: Comparison with semi-supervised methods. Reported values are obtained by using 4000 and 10,000 labeled examples respectively for CIFAR-10 and CIFAR-100.

C. Effect of training procedure

After we encountered the slow convergence rate of the sequential training, we tried wave-training. This boosted the convergence rate while achieving a similar validation accuracy. Results are shown in Fig. 6 where one can easily see that we could achieve higher accuracies with wave-training than sequential one in a small number of total epochs. Additionally, the wave-trained accuracy was expected to be bumpy for the total number of epochs because it was measured over the different layers, so lower layers are less precise.

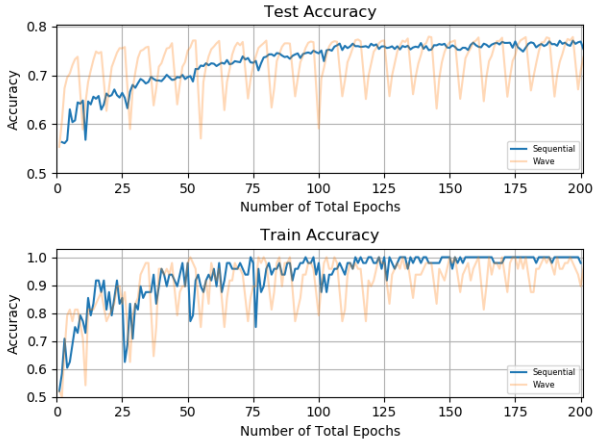

Fig. 6: Typical accuracies for sequential vs wave-trained networks.

D. Effect of mean prediction

One additional benefit of our network is that for each layer-stage, a label prediction is available. By averaging over all predicted label probabilities, which was explained before, the prediction accuracy marginally increased compared to the VGG output prediction and last layer-stage SAE prediction. Test accuracy error percentages for each method was given in Table IV.

Method	VGG	SAE	Mean
Accuracy	21.9	22.4	20.9

TABLE IV: Test error percentages for different prediction methods. VGG represents the prediction by final fully connected layer, SAE represents prediction by the last layer-stage SAE and Mean represents the prediction by average prediction probabilities.

E. Effect of the number of labeled data

We experimented with different number of labeled examples on base configuration to see the effect of it. As can be seen from Table V, scaling up the number of examples did not increase the accuracy proportionally, they just contributed marginally to the performance considering their scaling factor.

# Labels	4000	5000	10,000	20,000
Accuracy	20.9	18.9	15.1	13.6

TABLE V: Test accuracy error percentages for different number of labeled training images.

F. Effect of local SAE complexity

Lastly, we experimented the effect of the local SAE complexity because we are learning through this local structures. In Table VI, we demonstrated that increasing SAE complexity (up to 50M parameters) does not help significantly to the test accuracy performance.

SAE Type	A	B	C
Accuracy	20	20.6	20.9

TABLE VI: Test error percentages for different encoder/decoder types for local SAE structure.

G. Feature transportation through layers

To visualize the features the network was attending to, we optimized a random input image to maximally excite a filter of a convolutional layer. The implementation is based on the code provided by [11]. Visualization of excitations are demonstrated in Fig. 7.

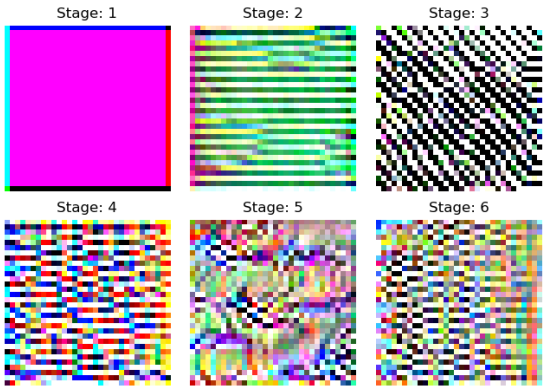

Fig. 7: Visualization of convolutional layer activations.

One sample image is provided for each depth-level 1-6 of the VGG network when fully trained. The first layer is excited the most by a complete monochrome input image except for the border pixels. This is intuitively the correct result for the first layer, since it is directly applied to the input image and has a kernel size of 3x3 with a stride zero padding of one. Therefore the maximal excitement is induced by a constant input image apart from the border region where the input to the filter is padded with zeros. In layer-stages 2 and 3, regular patterns consisting of horizontal lines for layer 2 and 45% degree lines in layer 3 maximal exists in the convolutional filter. The deeper layers are excited by an input image with no clear regular patterns. It can be stated that the complexity increases with the depth if the network.

IV. Discussion

A. Model 1

In Model 1, we directly propagated the learned representation after the first convolutional layer of the SAE encoder to the input of the next SAE stage by two different mapping. While training the network, the prediction accuracy decreased with depth as mentioned. We figured out two main flaws in this method. Firstly, compressing the high dimensional output of the convolutional layer back to the initial 3x32x32 image size, requires significant compression. Therefore even a trainable transfer mapping might not be able to transport important features. Secondly a single convolutional layer has a limited receptive field. This issue is overcome by classical CNN architectures by stacking convolutional layers on top of each other and therefore increasing the receptive field of the deeper convolutional layers. This approach is not working when training a local loss block at a time, despite the fact that our architecture stacks convolutional layers with depth. The receptive field is therefore not increasing and the network is intrinsically limited in its capability to learn large distance spatial relationships in the original input image.

B. Model 2

Using Model 2, despite the fact we were not able to outperform the given baselines, interesting results and insights were achieved.

The convergence rate of the model is directly related to the training procedure. By visual inspection of the convolutional layer activation maps and a similar resulting training loss, we conclude that both training procedure convergence to a similar or at least similar performing solution. Since the objective function is highly non-linear we can not conclude that the different training procedure results in the same solution.

The individual performance of each layer stage directly supports our intuitive belief that we are able to learn a better representation with increasing depth of the network. The ensemble of labels which can be generated by incorporating the encoder network into the prediction process is a nice side effect of the local loss training and increases the overall robustness of the network.

As we can not exploit the unlabeled data in our current model, we can use ideas like consistency-regularization and entropy minimization [4, 9] and entropy with unlabeled data in a future work. These ideas better take advantage of the vast amount of unlabeled data. We can also use ideas from slow-feature analysis [12] as it allows for learning better feature representations for unsupervised data.

V. Summary

In summary we were able to show that it is possible to combine a more biological plausible layer-wise training with established CNNs. The proposed method of using SAEs to train local network-stages is flexible enough to be a applied to a variety of other state-of-the art deep neural networks. Nonetheless, the disadvantage of extensive hyper-parameter tuning, which is already one of the biggest challenges in traditional neural networks, becomes even worse for the proposed SAE training.

Here comes the ability to train the network in an semi-supervised manner particular handy. Specifically in the age of big data and the internet, it is often orders of magnitude cheaper to access unlabeled data.

REFERENCES

[1] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.

[2] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," *arXiv preprint arXiv:1901.06656*, 2019.

[3] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in neuroscience*, vol. 12, p. 608, 2018.

[4] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," *arXiv preprint arXiv:1905.02249*, 2019.

[5] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, "Unsupervised data augmentation," *arXiv preprint arXiv:1904.12848*, 2019.

[6] L. Le, A. Patterson, and M. White, "Supervised autoencoders: Improving generalization performance with unsupervised regularizers," in *Advances in Neural Information Processing Systems*, 2018, pp. 107–117.

[7] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Greedy layerwise learning can scale to imagenet, 2018," *URL http://arxiv. org/abs*, 1812.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] B. Athiwaratkun, M. Finzi, P. Izmailov, and A. G. Wilson, "Improving consistency-based semi-supervised learning with weight averaging," *arXiv preprint arXiv:1806.05594*, vol. 2, 2018.

[10] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *Advances in neural information processing systems*, 2017, pp. 1195–1204.

[11] U. Ozbulak, *Pytorch cnn visualizations*, https://github.com/utkuozbulak/pytorch-cnn-visualizations, 2019.

[12] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural computation*, vol. 14, no. 4, pp. 715–770, 2002.