IDIAP COMMUNICATION REPORT

LOCAL AFFINE APPROXIMATIONS FOR IMPROVING KNOWLEDGE TRANSFER

Suraj Srinivas [a] Francois Fleuret

Idiap-Com-01-2018

MARCH 2018

[a] Idiap Research Institute

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

Local Affine Approximators for Improving Knowledge Transfer

Suraj Srinivas & François Fleuret
Idiap Research Institute and EPFL
{suraj.srinivas, francois.fleuret}@idiap.ch

Abstract

The Jacobian of a neural network, or the derivative of the output with respect to the input, is a versatile object with many applications. In this paper we discuss methods to use this object efficiently for knowledge transfer. We first show that matching Jacobians is a special form of distillation, where noise is added to the input. We then show experimentally that we can perform better distillation under low-data settings. We also show that Jacobian-penalty regularizers can be used to improve robustness of models to random noise.

1 Introduction

Deep neural networks are non-linear functions with high expressive power, and given a network architecture, it is straightforward to train models from data. However, what if we are interested in training multiple alternate architectures? How can we use already trained models to accelerate the training of others? Methods that can do this are called knowledge transfer methods. A perfect knowledge transfer method would enable us to seamlessly transform one neural network architecture into another, while preserving input-output mapping and generalization. Such interchangability would allow us to more easily explore the space of neural network architectures. This capability could be used for neural network architecture search, model compression, or creating diverse ensembles, among other applications.

This paper deals with the problem of knowledge transfer using a first-order approximation of the neural network. This approach has also been recently explored by Czarnecki *et al.*[1], who considered the general idea of matching Jacobians, and by Zagoruyko *et al.*[2] who proposed knowledge transfer by viewing Jacobians as attention maps. In both of these cases, it was not clear what are the right kinds of penalty to impose on Jacobians. It was also unclear how these methods relate to previous knowledge transfer approaches such as knowledge distillation [3, 4].

In this paper, we show that we can train alternate architectures using much smaller number of data points than the full training set. Specifically for the CIFAR100 dataset, we are able to reach within 2% of the final generalization error by using only $1/5^{th}$ of the entire dataset. The overall contributions of our paper are: (1) We show that matching Jacobians is a special case of distillation, where noise is added to the inputs, (2) Using these regularizers, we improve knowledge transfer performance, particularly for low-data settings, (3) We show that training neural networks with Jacobian penalty improves stability with respect to noise.

2 Related Work

There is a lot of literature which uses the Jacobian in neural networks. Recently Sobolev training [1], showed that using higher order derivatives along with the targets can help in training with low data. This work is very similar to ours. While we also make similar claims, we clarify the relationship of this method with activation matching (knowledge distillation) and show how it can help. Same

is the case with Zagoruyko *et al.*[2], who introduce the idea of matching attention maps. One possible method they consider was computing the Jacobian. This work also finds that combining both activation matching and jacobian matching is helpful.

Jacobian-based regularizers were used in early works [5], where they looked at penalizing the Jacobian norm. The intuition was to make the model more robust to small changes in the input. We find that this conforms to our analysis as well.

There is also plenty of work relating to knowledge transfer between neural networks. Knowledge Distillation [4] first showed that one can use softmax with temperature to perform knowledge transfer with neural nets. Ba and Caruana [3] found that squared error between logits worked better than the softmax method, and they used this method to train shallow nets with equivalent performance to deep nets. A recent paper [6] also found that adding noise to logits helps during teacher-student training. We show that the use of the Jacobian can be interpreted as adding such noise analytically.

3 Local Affine Approximators of Neural Networks

Let us consider the first order Taylor series expansion of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ around a small neighbourhood $\{\mathbf{x} + \Delta\mathbf{x} : \|\Delta\mathbf{x}\| \leq \epsilon\}$. It can be written as

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x}) + \nabla_x f(\mathbf{x})^T(\Delta\mathbf{x}) + \mathcal{O}(\epsilon^2) \approx f(\mathbf{x}) + \nabla_x f(\mathbf{x})^T(\Delta\mathbf{x}) \tag{1}$$

This is the local affine approximation of the function $f(\cdot)$ around the local neighbourhood. This approximation is useful to analyse non-linear objects like deep neural networks. The source of non-linearity for neural nets lie in the elementwise non-linear activations (like ReLU, sigmoid) and pooling operators. *It is easy to see that to construct an affine approximation of the entire neural network, one must construct affine approximations of all such non-linearities in the network.*

Special case: ReLU and MaxPool For the ReLU nonlinearity, the affine approximation is simple to compute, as the derivative $\frac{\mathrm{d}\sigma(z)}{\mathrm{d}z}$ is either 0 or 1 (except at $z = 0$, where it is undefined). Similarly for max-pooling as well. Going back to the definition in Equation 1, for piecewise linear nets there exist $\epsilon > 0$ such that the super-linear terms are exactly zero, *i.e.*; $f(\mathbf{x}+\Delta\mathbf{x}) = f(\mathbf{x})+\nabla_x f(\mathbf{x})^T(\Delta\mathbf{x})$, *i.e.*; $\mathbf{x}$ and $\Delta\mathbf{x}$ lie on the same plane.

Consider piecewise linear functions with the non-differentiability at zero. Also consider neural networks with no external bias units among the model parameters. For such cases we see that the affine approximation reduces to a linear approximation, *i.e.*; the overall 'bias' term is zero. This means the following: let $\mathbf{x}_0$ be a point arbitrarily close to zero such that $f(\mathbf{x}_0+\Delta\mathbf{x}) = f(\mathbf{x}_0)+\nabla_x f(\mathbf{x}_0)^T(\Delta\mathbf{x})$. Then, given $f(\mathbf{x}_0) \rightarrow 0$, $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0 \rightarrow \mathbf{x}$ and the fact that $\mathbf{x}$ and $\mathbf{x}_0$ have the same Jacobian, we have the following - $f(\mathbf{x}) = \nabla_x f(\mathbf{x})^T\mathbf{x}$. *In other words, the output can be obtained by multiplying the Jacobian with the input*. We find that this relation holds approximately even for ReLU nets with a bias unit as they are typically quite small (*e.g.* VGG19). Thus in this case the Jacobian captures almost all of the information required to make a decision. However, this breaks down for ReLU nets with batch normalization (*e.g.* ResNets) as they introduce their own mean subtraction terms.

As a corollary for other non-linearities, even with no external bias units, the overall 'bias' may be non-zero. For example, consider the hard-tanh nonlinearity, which is given by $\sigma(z) = z$ for $-1 \leq z \leq 1$, and saturates to -1 on the left and $+1$ on the right. In such a case, when the non-linearity saturates, the corresponding slope is zero and we get $\sigma(z + dz) = \sigma(z) = \pm 1$ bias depending on where it saturates. This reminds us that in general the Jacobian does not capture all information about the neural network, especially in the case of such saturating non-linearities, or when external bias units are added.

4 Knowledge Transfer

In this section, we consider the problem of knowledge transfer using Local Affine Approximations. This problem may be posed as follows: given a *teacher* network $\mathcal{T}$ which is trained on a dataset $\mathcal{D}$, we wish to enhance the training of a student network $\mathcal{S}$ on $\mathcal{D}$ using hints from $\mathcal{T}$. Recent works [1, 2] sought to match the Jacobians of $\mathcal{S}$ and $\mathcal{T}$ via squared error terms. However, two aspects are not

clear in these formalisms: (i) what penalty term must be used between Jacobians, and (ii) how this idea of matching Jacobians relates to simpler methods such as activation matching [3, 4]. Recall that previous distillation works involved matching the activations of $\mathcal{S}$ and $\mathcal{T}$. To resolve these issues, we make the following claim.

Claim. *Matching Jacobians of two networks is equivalent to matching soft targets with noise added to the inputs during training.*

More concretely, we make the following proposition.

Proposition 1. *Consider the squared error cost function for matching soft targets of two neural networks with k-length targets ($\in \mathbb{R}^k$), given by $\ell(\mathcal{T}(\mathbf{x}), \mathcal{S}(\mathbf{x})) = \sum_{i=1}^{k}(\mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x}))^2$, where $\mathbf{x} \in \mathbb{R}^D$ is an input data point. Let $\boldsymbol{\xi}$ $(\in \mathbb{R}^D) = \sigma$ $\mathbf{z}$ be a scaled version of a unit normal random variable $\mathbf{z}$ $\in \mathbb{R}^D$ with scaling factor $\sigma \in \mathbb{R}$. Then the following is true.*

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}+\boldsymbol{\xi}) - \mathcal{S}^i(\mathbf{x}+\boldsymbol{\xi})\right)^2\right] = \sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x})\right)^2 + \sigma^2 \sum_{i=1}^{k} \|\nabla_x \mathcal{T}^i(\mathbf{x}) - \nabla_x \mathcal{S}^i(\mathbf{x})\|_2^2 + \mathcal{O}(\sigma^4)$$

Notice that in this expression, we have decomposed the loss function into two components - one representing the usual distillation loss on the samples, and the second regularizer term representing the jacobian matching loss. The final error terms are small for small σ and can be ignored. The above proposition is a simple consequence of using the first-order Taylor series expansion around x. Note that the error term is zero for piecewise-linear nets. An analogous statement is true for the case of cross entropy error between soft targets, leading to:

$$\mathbb{E}_{\boldsymbol{\xi}}\left[-\sum_{i=1}^{k}\mathcal{T}_s^i(\mathbf{x}+\boldsymbol{\xi})\log\left(\mathcal{S}_s^i(\mathbf{x}+\boldsymbol{\xi})\right)\right] \approx -\sum_{i=1}^{k}\mathcal{T}_s^i(\mathbf{x})\log(\mathcal{S}_s^i(\mathbf{x})) - \sigma^2\sum_{i=1}^{k}\frac{\nabla_x \mathcal{T}_s^i(\mathbf{x})^T \nabla_x \mathcal{S}_s^i(\mathbf{x})}{\mathcal{S}_s^i(\mathbf{x})} \tag{2}$$

where $\mathcal{T}_s^i(\mathbf{x})$ denotes the same network $\mathcal{T}^i(\mathbf{x})$ but with a softmax or sigmoid (with temperature parameter T if needed) added at the end. We ignore the super-linear error terms for convenience.

In simple terms, these statements show that matching Jacobians is a natural consequence of matching not only the raw CNN outputs at the given data points, but also at the infinitely many data points nearby. We can use a similar idea to derive regularizers for the case of regular neural network training as well. These regularizers seek to make the underlying model *robust* to small amounts of noise added to the inputs.

Proposition 2. *Consider the squared error cost function for training a neural network with k targets, given by $\ell(y(\mathbf{x}), \mathcal{S}(\mathbf{x})) = \sum_{i=1}^{k}(y^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x}))^2$, where $\mathbf{x} \in \mathbb{R}^D$ is an input data point, and $y^i(\mathbf{x})$ is the i^{th} target output. Let $\boldsymbol{\xi}$ $(\in \mathbb{R}^D) = \sigma$ $\mathbf{z}$ be a scaled version of a unit normal random variable $\mathbf{z}$ $\in \mathbb{R}^D$ with scaling factor $\sigma \in \mathbb{R}$. Then the following is true.*

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}\left(y^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x}+\boldsymbol{\xi})\right)^2\right] = \sum_{i=1}^{k}\left(y^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x})\right)^2 + \sigma^2\sum_{i=1}^{k}\|\nabla_x \mathcal{S}^i(\mathbf{x})\|_2^2 + \mathcal{O}(\sigma^4)$$

A statement similar to Proposition 2 has been previously derived by [7], who observed that the regularizer term for linear models corresponds exactly to the well-known Tikhonov regularizer. This regularizer was also proposed by [5]. The ℓ_2 weight decay regularizer for neural networks was derived as an extension of this for neural networks. However, this result shows that a more

appropriate extension would be to penalize the norms of the Jacobian rather than weights directly. We can derive a similar result for the case of cross-entropy error as well, which is given by -

$$\mathbb{E}_{\boldsymbol{\xi}}\left[-\sum_{i=1}^{k} y^i(\mathbf{x})\log(\mathcal{S}_s^i(\mathbf{x}+\boldsymbol{\xi}))\right] \approx -\sum_{i=1}^{k} y^i(\mathbf{x})\log(\mathcal{S}_s^i(\mathbf{x})) + \sigma^2 \sum_{i=1}^{k} y^i(\mathbf{x})\frac{\|\nabla_x \mathcal{S}_s^i(\mathbf{x})\|_2^2}{\mathcal{S}_s^i(\mathbf{x})^2} \quad (3)$$

We notice here again that the regularizer involves $\mathcal{S}_s^i(\mathbf{x})$, which has the sigmoid / softmax nonlinearity applied on top of the final layer of $\mathcal{S}^i(\mathbf{x})$. Deriving all the above results is a simple matter of using first-order Taylor series expansions, and a second-order expansion in the case of Equation 3.

In all cases above we see that the regularizers for cross-entropy error term seem more unstable when compared to those for squared error. We find that this is true experimentally as well. As a result, we use squared error loss for distillation.

Approximating the Full Jacobian One can see that both in the case of Proposition 1 and 2, we are required to match the full Jacobian. This is computationally very expensive to perform. To reduce the computational load, we can approximate the summation of Jacobian terms with the single largest term. However, we cannot estimate this without computing the Jacobians themselves. As a result, we use a heuristic where the output variable involving the correct answer $c \in [1, k]$ is used for computing the Jacobian. Alternately, if we do not want to use the labels, we may instead use the output variable with the largest magnitude.

5 Experiments

We perform two kinds of experiments to show the effectiveness of using Jacobians. First, we perform knowledge transfer on low-data on CIFAR10 and CIFAR100 datasets. Second, we show that penalizing Jacobian norm increases stability of the networks to random noise.

5.1 Knowledge Transfer

For the knowledge transfer experiments, we use VGG-like architectures with batch normalization. The main difference is that we keep only the convolutional layers and have only one fully connected layer rather than three. Our workflow is as follows. First, a 9-layer 'teacher' network is trained on the full CIFAR10 / CIFAR100 dataset. Then, a smaller 4-layer 'student' network is trained - but this time on small subsets of CIFAR10 / 100 datasets rather than the full dataset. We perform experiments to see how performance of various methods differ on changing the size of the dataset for the student network. Experimental details are given in the supplementary section.

Regarding baselines, we compare against regular training of student network from scratch (Row 1). For the distillation experiments, we compare with and without using labels. Baselines with labels have the suffix '+CE' which stands for 'Cross-Entropy' loss using the ground truth labels. In all cases, we use the squared-error distillation loss shown in Proposition 1. We found that the squared loss worked much better than the cross-entropy loss for distillation as did [3]. The loss function we use for the CIFAR10 case is slightly different from that predicted by theory. We use a form of normalized squared error, whose exact form is specified in the supplementary material. However in our CIFAR100 experiments we use the simple squared error loss.

From both Tables 1, 2 we can conclude that (1) it is generally beneficial to do any form of distillation to improve performance, (2) matching Jacobians along with activations outperforms matching only activations in low-data settings, (3) not matching Jacobians is often beneficial for large data.

One curious phenomenon we observe is that having a cross-entropy (CE) error term is often not crucial to maintain good regularization performance. For CIFAR10, we find that not using this term often yields better results. This is perhaps because the CIFAR10 teacher network has high accuracy (90.94%) when compared to the CIFAR100 teacher (64.78%).

For Table 2, we see that when training with activations, Jacobians and regular cross-entropy training (fourth row), we reach an accuracy of 52.43% when training with 100 data points per class. We observe that the overall accuracy of raw training using the full dataset is 54.28%. This implies that

Table 1: Knowledge transfer performance for the CIFAR10 dataset. We find that matching both activations and Jacobians performs the best for low-data settings. The student network is VGG-4 while the teacher is a VGG-9 network which achieves 90.94% accuracy.

# of Data points per class	5	10	50	100	500	1000	5000 (full)
Regular Cross-Entropy (CE) training	28.92	33.8	53.6	61.05	76.01	77.63	84.56
Match Activations + CE	35.8	40.79	59.98	67.22	80.55	81.77	87.65
Match Jacobians + CE	34.47	39.43	58.18	65.05	76.82	78.21	84.76
Match {Activations + Jacobians} + CE	39.78	43.4	62.17	69.15	80.28	82.23	87.29
Match Activations only	36.73	41.61	60.71	68.32	**82.19**	**84.00**	**88.64**
Match {Activations + Jacobians}	**41.43**	**46.03**	**63.5**	**70.17**	80.98	82.75	85.56

Table 2: Knowledge transfer performance for the CIFAR100 dataset. We find that matching both activations and Jacobians along with cross-entropy error performs the best for low-data settings. The student network is VGG-4 while the teacher is a VGG-9 network which achieves 64.78% accuracy.

# of Data points per class	1	5	10	50	100	500 (full)
Regular Cross-Entropy (CE) training	5.69	13.9	20.03	37.6	44.92	54.28
Match Activations + CE	12.13	26.97	33.92	46.47	50.92	**56.65**
Match Jacobians + CE	6.78	23.94	32.03	45.71	51.47	53.44
Match {Activations + Jacobians} + CE	**13.78**	**33.39**	**39.55**	**49.49**	**52.43**	54.57
Match Activations only	10.73	28.56	33.6	45.73	50.15	56.59
Match {Activations + Jacobians}	13.09	33.31	38.16	47.79	50.06	51.33

we are able to reach close to the full training accuracy using $1/5^{th}$ of the data requirement. For the CIFAR10 dataset, we see that the closest performance is achieved by training with only the activations.

5.2 Noise robustness

We now look at the Jacobian regularizers intended to increase robustness to noise. We train 9-layer VGG networks on CIFAR100 with varying amount of the regularization strength, and measure their classification accuracies in presence of noise added to the normalized images. From Table 3 we find that using higher regularization strengths is better for robustness, even when the initial accuracy at the zero-noise case is lower. This aligns well with our theory.

6 Conclusion

In this paper we have considered the problem of matching local affine approximators of deep neural networks for knowledge transfer. We showed that adding the Jacobian loss term to soft-target loss is equivalent to using the soft-target loss with noisy inputs. We experimentally find that adding such a Jacobian loss term typically improves over using only the distillation loss. We also showed that these regularizers can help in making neural networks more robust to noise.

Table 3: Noise robustness tests on VGG-9 nets trained with the Jacobian-penalty regularizers on the CIFAR100 dataset

Noise standard deviation	0	0.1	0.2	0.3	0.4
Regularization = 0	64.78	61.9 ± 0.07	47.53 ± 0.23	29.63 ± 0.16	17.69 ± 0.17
Regularization = 0.01	64.67	61.85 ± 0.15	49.47 ± 0.07	32.24 ± 0.28	19.63 ± 0.17
Regularization = 0.1	**65.62**	$\mathbf{63.36 \pm 0.18}$	53.57 ± 0.23	37.38 ± 0.18	23.99 ± 0.19
Regularization = 0.5	64.26	62.85 ± 0.05	56.57 ± 0.21	44.99 ± 0.28	31.97 ± 0.15
Regularization = 1.0	63.59	62.66 ± 0.13	$\mathbf{57.53 \pm 0.17}$	$\mathbf{47.48 \pm 0.14}$	$\mathbf{35.43 \pm 0.11}$

References

[1] Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. Sobolev training for neural networks. *NIPS*, 2017.

[2] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *ICLR*, 2017.

[3] LJ Ba and R Caruana. Do deep networks really need to be deep. *Advances in neural information processing systems*, 27:1–9, 2014.

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning Workshop*, 2015.

[5] Harris Drucker and Yann Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 1992.

[6] Bharat Bhusan Sau and Vineeth N Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*, 2016.

[7] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 1995.

Supplementary Material

Proofs

Proposition 1. *Consider the squared error cost function for matching soft targets of two neural networks with k-length targets ($\in \mathbb{R}^k$), given by $\ell(\mathcal{T}(\mathbf{x}), \mathcal{S}(\mathbf{x})) = \sum_{i=1}^{k}(\mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x}))^2$, where $\mathbf{x} \in \mathbb{R}^D$ is an input data point. Let $\boldsymbol{\xi}$ $(\in \mathbb{R}^D) = \sigma$ $\mathbf{z}$ be a scaled version of a unit normal random variable $\mathbf{z}$ $\in \mathbb{R}^D$ with scaling factor $\sigma \in \mathbb{R}$. Then the following is locally true.*

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}+\boldsymbol{\xi}) - \mathcal{S}^i(\mathbf{x}+\boldsymbol{\xi})\right)^2\right] = \sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x})\right)^2 + \sigma^2 \sum_{i=1}^{k}\|\nabla_x \mathcal{T}^i(\mathbf{x}) - \nabla_x \mathcal{S}^i(\mathbf{x})\|_2^2 + \mathcal{O}(\sigma^4)$$

Proof. There exists σ and $\boldsymbol{\xi}$ small enough that first-order Taylor series expansion holds true

$$\begin{aligned} & \mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}+\boldsymbol{\xi}) - \mathcal{S}^i(\mathbf{x}+\boldsymbol{\xi})\right)^2\right] \\ = \quad & \mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}) + \boldsymbol{\xi}\nabla_x \mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x}) - \boldsymbol{\xi}\nabla_x \mathcal{S}^i(\mathbf{x})\right)^2\right] + \mathcal{O}(\sigma^4) \\ = \quad & \sum_{i=1}^{k}\left(\mathcal{T}^i(\mathbf{x}) - \mathcal{S}^i(\mathbf{x})\right)^2 + \mathbb{E}_{\boldsymbol{\xi}}\left[\sum_{i=1}^{k}(\boldsymbol{\xi})^2\left(\nabla_x \mathcal{T}^i(\mathbf{x}) - \nabla_x \mathcal{S}^i(\mathbf{x})\right)^2\right] + \mathcal{O}(\sigma^4) \end{aligned} \tag{4}$$

To get equation 4, we use the fact that mean of $\boldsymbol{\xi}$ is zero. To complete the proof, we use the diagonal assumption on the covariance matrix of $\boldsymbol{\xi}$.

□

Proofs of other statements are similar. For proof of equation 3, use a second order Taylor series expansion of $\log(\cdot)$ in the first step.

Experimental details

Network Architectures

The architecture for our networks follow the VGG design philosophy. Specifically, we have blocks with the following elements:

- 3×3 conv kernels with c channels of stride 1
- Batch Normalization
- ReLU

Whenever we use Max-pooling (M), we use stride 2 and window size 2.

The architecture for VGG-9 is - $[64-M-128-M-256-256-M-512-512-M-512-512-M]$. Here, the number stands for the number of convolution channels, and M represents max-pooling. At the end of all the convolutional and max-pooling layers, we have a Global Average Pooling (GAP) layer, after which we have a fully connected layer leading up to the final classes.

The architecture for VGG-4 is - $[64 - M - 128 - M - 512 - M]$.

Similar architectures are used for both CIFAR10 and CIFAR100, and the only difference is in the final fully connected layer which has more output classes for CIFAR100.

Loss function

The loss function we use takes the following form in general.

$$\ell(\mathcal{S}, \mathcal{T}) = \alpha \times (\text{CE}) + \beta \times (\text{Match Activations}) + \gamma \times (\text{Match Jacobians})$$

In our experiments, α, β, γ are either set to 1 or 0. In other words, all regularization constants are 1.

Here, 'CE' refers to cross-entropy with ground truth labels. 'Match Activations' refers to squared error term over pre-softmax activations of the form $(y_s - y_t)^2$. 'Match Jacobians' refers to the same squared error term, but for Jacobians. We use the approximation specified in Section 4. For CIFAR100, the loss takes the following form:

$$\text{Match Jacobian} = \|\nabla_x \mathcal{T}^i(\mathbf{x}) - \nabla_x \mathcal{S}^i(\mathbf{x})\|_2^2$$

For CIFAR10, we use a slightly different loss which we found works better:

$$\text{Match Jacobian} = \left|\left| \frac{\nabla_x \mathcal{T}^i(\mathbf{x})}{\|\nabla_x \mathcal{T}^i(\mathbf{x})\|} - \frac{\nabla_x \mathcal{S}^i(\mathbf{x})}{\|\nabla_x \mathcal{S}^i(\mathbf{x})\|} \right|\right|_2^2$$

Optimization

We run optimization for 500 epochs. We use the Adam optimizer, with an initial learning rate of $1e-3$, and a single learning rate annealing (to $1e-4$) at 400 epochs.