FITNETS: HINTS FOR THIN DEEP NETS

Adriana Romero[1], **Nicolas Ballas**[2], **Samira Ebrahimi Kahou**[3], **Antoine Chassang**[2], **Carlo Gatta**[4] & **Yoshua Bengio**[2]†
[1]Universitat de Barcelona, Barcelona, Spain.
[2]Université de Montréal, Montréal, Québec, Canada. †CIFAR Senior Fellow.
[3]École Polytechnique de Montréal, Montréal, Québec, Canada.
[4]Centre de Visió per Computador, Bellaterra, Spain.

ABSTRACT

While depth tends to improve network performances, it also makes gradient-based training more difficult since deeper networks tend to be more non-linear. The recently proposed knowledge distillation approach is aimed at obtaining small and fast-to-execute models, and it has shown that a student network could imitate the soft output of a larger teacher network or ensemble of networks. In this paper, we extend this idea to allow the training of a student that is deeper and thinner than the teacher, using not only the outputs but also the intermediate representations learned by the teacher as hints to improve the training process and final performance of the student. Because the student intermediate hidden layer will generally be smaller than the teacher's intermediate hidden layer, additional parameters are introduced to map the student hidden layer to the prediction of the teacher hidden layer. This allows one to train deeper students that can generalize better or run faster, a trade-off that is controlled by the chosen student capacity. For example, on CIFAR-10, a deep student network with almost 10.4 times less parameters outperforms a larger, state-of-the-art teacher network.

1 INTRODUCTION

Deep networks have recently exhibited state-of-the-art performance in computer vision tasks such as image classification and object detection (Simonyan & Zisserman, 2014; Szegedy et al., 2014). However, top-performing systems usually involve very *wide* and *deep* networks, with numerous parameters. Once learned, a major drawback of such wide and deep models is that they result in very time consuming systems at inference time, since they need to perform a huge number of multiplications. Moreover, having large amounts of parameters makes the models high memory demanding. For these reasons, wide and deep top-performing networks are not well suited for applications with memory or time limitations.

There have been several attempts in the literature to tackle the problem of model compression to reduce the computational burden at inference time. In Bucila et al. (2006), authors propose to train a neural network to mimic the output of a complex and large ensemble. The method uses the ensemble to label unlabeled data and trains the neural network with the data labeled by the ensemble, thus mimicking the function learned by the ensemble and achieving similar accuracy. The idea has been recently adopted in Ba & Caruana (2014) to compress deep and wide networks into shallower but even wider ones, where the compressed model mimics the function learned by the complex model, in this case, by using data labeled by a deep (or an ensemble of deep) networks. More recently, Knowledge Distillation (KD) (Hinton & Dean, 2014) was introduced as a model compression framework, which eases the training of deep networks by following a student-teacher paradigm, in which the student is penalized according to a softened version of the teacher's output. The framework compresses an ensemble of deep networks (*teacher*) into a *student* network of *similar depth*. To do so, the student is trained to predict the output of the teacher, as well as the true classification labels. All previous works related to Convolutional Neural Networks focus on compressing a teacher network or an ensemble of networks into either networks of similar width and depth or into shallower and wider ones; not taking advantage of depth.

Depth is a fundamental aspect of representation learning, since it encourages the re-use of features, and leads to more abstract and invariant representations at higher layers (Bengio et al., 2013). The importance of depth has been verified (1) theoretically: deep representations are exponentially more expressive than shallow ones for some families of functions (Montufar et al., 2014); and (2) empirically: the two top-performers of ImageNet use deep convolutional networks with 19 and 22 layers, respectively (Simonyan & Zisserman, 2014) and (Szegedy et al., 2014).

Nevertheless, training deep architectures has proven to be challenging (Larochelle et al., 2007; Erhan et al., 2009), since they are composed of successive non-linearities and, thus result in highly non-convex and non-linear functions. Significant effort has been devoted to alleviate this optimization problem. On the one hand, pre-training strategies, whether unsupervised (Hinton et al., 2006; Bengio et al., 2007) or supervised (Bengio et al., 2007) train the network parameters in a greedy layerwise fashion in order to initialize the network parameters in a potentially good basin of attraction. The layers are trained one after the other according to an intermediate target. Similarly, semi-supervised embedding (Weston et al., 2008) provides guidance to an intermediate layer to help learn very deep networks. Along this line of reasoning, (Cho et al., 2012) ease the optimization problem of DBM by borrowing the activations of another model every second layer in a purely unsupervised scenario. More recently, (Chen-Yu et al., 2014; Szegedy et al., 2014; Gulcehre & Bengio, 2013) showed that adding supervision to intermediate layers of deep architectures assists the training of deep networks. Supervision is introduced by stacking a supervised MLP with a softmax layer on top of intermediate hidden layers to ensure their discriminability w.r.t. labels. Alternatively, Curriculum Learning strategies (CL) (Bengio, 2009) tackle the optimization problem by modifying the training distribution, such that the learner network gradually receives examples of increasing and appropriate difficulty w.r.t. the already learned concepts. As a result, curriculum learning acts like a continuation method, speeds up the convergence of the training process and finds potentially better local minima of highly non-convex cost functions.

In this paper, we aim to address the network compression problem by taking advantage of depth. We propose a novel approach to train *thin* and *deep* networks, called *FitNets*, to compress *wide* and shallower (but still *deep*) networks. The method is rooted in the recently proposed Knowledge Distillation (KD) (Hinton & Dean, 2014) and extends the idea to allow for thinner and deeper student models. We introduce *intermediate-level hints* from the teacher hidden layers to guide the training process of the student, *i.e.*, we want the student network (FitNet) to learn an intermediate representation that is predictive of the intermediate representations of the teacher network. Hints allow the training of thinner and deeper networks. Results confirm that having deeper models allow us to generalize better, whereas making these models thin help us reduce the computational burden significantly. We validate the proposed method on MNIST, CIFAR-10, CIFAR-100, SVHN and AFLW benchmark datasets and provide evidence that our method matches or outperforms the teacher's performance, while requiring notably fewer parameters and multiplications.

2 METHOD

In this section, we detail the proposed student-teacher framework to train FitNets from shallower and wider nets. First, we review the recently proposed KD. Second, we highlight the proposed hints algorithm to guide the FitNet throughout the training process. Finally, we describe how the FitNet is trained in a stage-wise fashion.

2.1 REVIEW OF KNOWLEDGE DISTILLATION

In order to obtain a faster inference, we explore the recently proposed compression framework (Hinton & Dean, 2014), which trains a *student network*, from the softened output of an ensemble of wider networks, *teacher network*. The idea is to allow the student network to capture not only the information provided by the true labels, but also the finer structure learned by the teacher network. The framework can be summarized as follows.

Let T be a teacher network with an output softmax $\mathrm{P_T} = \mathrm{softmax}(\mathbf{a}_T)$ where $\mathbf{a}_T$ is the vector of teacher pre-softmax activations, for some example. In the case where the teacher model is a single network, $\mathbf{a}_T$ represents the weighted sums of the output layer, whereas if the teacher model is the result of an ensemble either $\mathrm{P_T}$ or $\mathbf{a}_T$ are obtained by averaging outputs from different networks (respectively for arithmetic or geometric averaging). Let S be a student network with parameters

$\mathbf{W_S}$ and output probability $\mathrm{P_S} = \mathrm{softmax}(\mathbf{a}_S)$, where $\mathbf{a}_S$ is the student's pre-softmax output. The student network will be trained such that its output $\mathrm{P_S}$ is similar to the teacher's output $\mathrm{P_T}$, as well as to the true labels $\mathbf{y_{true}}$. Since $\mathrm{P_T}$ might be very close to the one hot code representation of the sample's true label, a relaxation $\tau > 1$ is introduced to soften the signal arising from the output of the teacher network, and thus, provide more information during training[1]. The same relaxation is applied to the output of the student network ($\mathrm{P_S^\tau}$), when it is compared to the teacher's softened output ($\mathrm{P_T^\tau}$):

$$\mathrm{P_T^\tau} = \mathrm{softmax}\left(\frac{\mathbf{a}_T}{\tau}\right), \quad \mathrm{P_S^\tau} = \mathrm{softmax}\left(\frac{\mathbf{a}_S}{\tau}\right). \tag{1}$$

The student network is then trained to optimize the following loss function:

$$\mathcal{L}_{KD}(\mathbf{W_S}) = \mathcal{H}(\mathbf{y_{true}}, \mathrm{P_S}) + \lambda \mathcal{H}(\mathrm{P_T^\tau}, \mathrm{P_S^\tau}), \tag{2}$$

where $\mathcal{H}$ refers to the cross-entropy and λ is a tunable parameter to balance both cross-entropies. Note that the first term in Eq. (2) corresponds to the traditional cross-entropy between the output of a (student) network and labels, whereas the second term enforces the student network to learn from the softened output of the teacher network.

To the best of our knowledge, KD is designed such that student networks mimic teacher architectures of similar depth. Although we found the KD framework to achieve encouraging results even when student networks have slightly deeper architectures, as we increase the depth of the student network, KD training still suffers from the difficulty of optimizing deep nets (see Section 4.1).

2.2 HINT-BASED TRAINING

In order to help the training of deep FitNets (deeper than their teacher), we introduce *hints* from the teacher network. A *hint* is defined as the output of a teacher's hidden layer responsible for guiding the student's learning process. Analogously, we choose a hidden layer of the FitNet, the *guided* layer, to learn from the teacher's hint layer. We want the guided layer to be able to predict the output of the hint layer. Note that having hints is a form of regularization and thus, the pair hint/guided layer has to be chosen such that the student network is not over-regularized. The deeper we set the guided layer, the less flexibility we give to the network and, therefore, FitNets are more likely to suffer from over-regularization. In our case, we choose the hint to be the middle layer of the teacher network. Similarly, we choose the guided layer to be the middle layer of the student network.

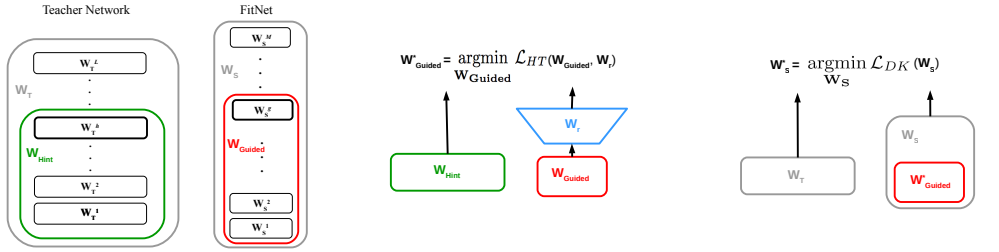
Given that the teacher network will usually be wider than the FitNet, the selected hint layer may have more outputs than the guided layer. For that reason, we add a regressor to the guided layer, whose output matches the size of the hint layer. Then, we train the FitNet parameters from the first layer up to the guided layer as well as the regressor parameters by minimizing the following loss function:

$$\mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r}) = \frac{1}{2}||u_h(\mathbf{x}; \mathbf{W_{Hint}}) - r(v_g(\mathbf{x}; \mathbf{W_{Guided}}); \mathbf{W_r})||^2, \tag{3}$$

where u_h and v_g are the teacher/student deep nested functions up to their respective hint/guided layers with parameters $\mathbf{W_{Hint}}$ and $\mathbf{W_{Guided}}$, r is the regressor function on top of the guided layer with parameters $\mathbf{W_r}$. Note that the outputs of u_h and r have to be comparable, *i.e.*, u_h and r must be the same non-linearity.

Nevertheless, using a fully-connected regressor increases the number of parameters and the memory consumption dramatically in the case where the guided and hint layers are convolutional. Let $\mathrm{N}_{h,1} \times \mathrm{N}_{h,2}$ and O_h be the teacher hint's spatial size and number of channels, respectively. Similarity, let $\mathrm{N}_{g,1} \times \mathrm{N}_{g,2}$ and O_g be the FitNet guided layer's spatial size and number of channels. The number of parameters in the weight matrix of a fully connected regressor is $\mathrm{N}_{h,1} \times \mathrm{N}_{h,2} \times \mathrm{O}_h \times \mathrm{N}_{g,1} \times \mathrm{N}_{g,2} \times \mathrm{O}_g$. To mitigate this limitation, we use a convolutional regressor instead. The convolutional regressor is designed such that it considers approximately the same spatial region of the input image as the teacher hint. Therefore, the output of the regressor has the same spatial size as the teacher hint. Given a teacher hint of spatial size $\mathrm{N}_{h,1} \times \mathrm{N}_{h,2}$, the regressor takes the output of the Fitnet's guided

[1]For example, as argued by Hinton & Dean (2014), with softened outputs, more information is provided about the relative similarity of the input to classes other than the one with the highest probability.



(a) Teacher and Student Networks (b) Hints Training (c) Knowledge Distillation

Figure 1: Training a student network using hints.

layer of size $\mathrm{N}_{g,1} \times \mathrm{N}_{g,2}$ and adapts its kernel shape $\mathrm{k}_1 \times \mathrm{k}_2$ such that $\mathrm{N}_{g,i} - \mathrm{k}_i + 1 = \mathrm{N}_{h,i}$, where $i \in \{1, 2\}$. The number of parameters in the weight matrix of a the convolutional regressor is $\mathrm{k}_1 \times \mathrm{k}_2 \times \mathrm{O}_h \times \mathrm{O}_g$, where $\mathrm{k}_1 \times \mathrm{k}_2$ is significantly lower than $\mathrm{N}_{h,1} \times \mathrm{N}_{h,2} \times \mathrm{N}_{g,1} \times \mathrm{N}_{g,2}$.

2.3 FitNet Stage-wise Training

We train the FitNet in a stage-wise fashion following the student/teacher paradigm. Figure 1 summarizes the training pipeline. Starting from a trained teacher network and a randomly initialized FitNet (Fig. 1 (a)), we add a regressor parameterized by $\mathbf{W_r}$ on top of the FitNet guided layer and train the FitNet parameters $\mathbf{W_{Guided}}$ up to the guided layer to minimize Eq. (3) (see Fig. 1 (b)). Finally, from the pre-trained parameters, we train the parameters of whole FitNet $\mathbf{W_S}$ to minimize Eq. (2) (see Fig. 1 (c)). Algorithm 1 details the FitNet training process.

Algorithm 1 FitNet Stage-Wise Training.
The algorithm receives as input the trained parameters $\mathbf{W_T}$ of a teacher, the randomly initialized parameters $\mathbf{W_S}$ of a FitNet, and two indices h and g corresponding to hint/guided layers, respectively. Let $\mathbf{W_{Hint}}$ be the teacher's parameters up to the hint layer h. Let $\mathbf{W_{Guided}}$ be the FitNet's parameters up to the guided layer g. Let $\mathbf{W_r}$ be the regressor's parameters. The first stage consists in pre-training the student network up to the guided layer, based on the prediction error of the teacher's hint layer (line 4). The second stage is a KD training of the whole network (line 6).

Input: $\mathbf{W_S}, \mathbf{W_T}, g, h$
Output: $\mathbf{W_S^*}$
1: $\mathbf{W_{Hint}} \leftarrow \{\mathbf{W_T}^1, \ldots, \mathbf{W_T}^h\}$
2: $\mathbf{W_{Guided}} \leftarrow \{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\}$
3: Intialize $\mathbf{W_r}$ to small random values
4: $\mathbf{W^*_{Guided}} \leftarrow \underset{\mathbf{W_{Guided}}}{\operatorname{argmin}} \mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r})$
5: $\{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\} \leftarrow \{\mathbf{W_{Guided}}^{*1}, \ldots, \mathbf{W_{Guided}}^{*g}\}$
6: $\mathbf{W_S^*} \leftarrow \underset{\mathbf{W_S}}{\operatorname{argmin}} \mathcal{L}_{KD}(\mathbf{W_S})$

2.4 Relation to Curriculum Learning

In this section, we argue that our hint-based training with KD can be seen as a particular form of Curriculum Learning (Bengio, 2009). Curriculum learning has proven to accelerate the training convergence as well as potentially improve the model generalization by properly choosing a sequence of training distributions seen by the learner: from simple examples to more complex ones. A curriculum learning extension (Gulcehre & Bengio, 2013) has also shown that by using guidance hints on an intermediate layer during the training, one could considerably ease training. However, Bengio (2009) uses hand-defined heuristics to measure the "simplicity" of an example in a sequence and Gulcehre & Bengio (2013)'s guidance hints require some prior knowledge of the end-task. Both of these curriculum learning strategies tend to be problem-specific.

Our approach alleviates this issue by using a teacher model. Indeed, intermediate representations learned by the teacher are used as hints to guide the FitNet optimization procedure. In addition, the teacher confidence provides a measure of example "simplicity" by means of teacher cross-entropy

term in Eq. (2). This term ensures that examples with a high teacher confidence have a stronger impact than examples with low teacher confidence: the latter correspond to probabilities closer to the uniform distribution, which exert less of a push on the student parameters. In other words, the teacher penalizes the training examples according to its confidence. Note that parameter λ in Eq. (2) controls the weight given to the teacher cross-entropy, and thus, the importance given to each example. In order to promote the learning of more complex examples (examples with lower teacher confidence), we gradually anneal λ during the training with a linear decay. The curriculum can be seen as composed of two stages: first learn intermediate concepts via the hint/guided layer transfer, then train the whole student network jointly, annealing λ, which allows easier examples (on which the teacher is very confident) to initially have a stronger effect, but progressively decreasing their importance as λ decays. Therefore, the hint-based training introduced in the paper is a generic curriculum learning approach, where prior information about the task-at-hand is deduced purely from the teacher model.

Algorithm	**# params**	**Accuracy**
Compression		
FitNet	∼2.5M	**91.61%**
Teacher	∼9M	90.18%
Mimic single	∼54M	84.6%
Mimic single	∼70M	84.9%
Mimic ensemble	∼70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Table 1: Accuracy on CIFAR-10

Algorithm	**# params**	**Accuracy**
Compression		
FitNet	∼2.5M	**64.96%**
Teacher	∼9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Table 2: Accuracy on CIFAR-100

3 Results on Benchmark Datasets

In this section, we show the results on several benchmark datasets[2]. The architectures of all networks as well as the training details are reported in the supplementary material.

3.1 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets (Krizhevsky & Hinton, 2009) are composed of 32x32 pixel RGB images belonging to 10 and 100 different classes, respectively. They both contain 50K training images and 10K test images. CIFAR-10 has 1000 samples per class, whereas CIFAR-100 has 100 samples per class. Like Goodfellow et al. (2013b), we normalized the datasets for contrast normalization and applied ZCA whitening.

CIFAR-10: To validate our approach, we trained a teacher network of maxout convolutional layers as reported in Goodfellow et al. (2013b) and designed a FitNet with 17 maxout convolutional layers, followed by a maxout fully-connected layer and a top softmax layer, with roughly $1/3$ of the parameters. The 11th layer of the student network was trained to mimic the 2nd layer of the teacher network. Like in Goodfellow et al. (2013b); Chen-Yu et al. (2014), we augmented the data with random flipping during training. Table 1 summarizes the obtained results. Our student model outperforms the teacher model, while requiring notably fewer parameters, suggesting that depth is crucial to achieve better representations. When compared to network compression methods, our algorithm achieves outstanding results; *i.e.*, the student network achieves an accuracy of 91.61%, which is significantly higher than the top-performer 85.8% of Ba & Caruana (2014), while requiring roughly 28 times fewer parameters. When compared to state-of-the-art methods, our algorithm matches the best performers.

One could argue the choice of hinting the inner layers with the hidden state of a wide teacher network. A straightforward alternative would be to hint them with the desired output. This could be addressed in a few different ways: (1) Stage-wise training, where stage 1 optimizes the 1st half of the network w.r.t. classification targets and stage 2 optimizes the whole network w.r.t. classification

[2]Code to reproduce the experiments publicly available: https://github.com/adri-romsor/FitNets

targets. In this case, stage 1 set the network parameters in a good local minima but such initialization did not seem to help stage 2 sufficiently, which failed to learn. To further assist the training of the thin and deep student network, we could add extra hints with the desired output at different hidden layers. Nevertheless, as observed in (Bengio et al., 2007), with supervised pre-training the guided layer may discard some factors from the input, which require more layers and non-linearity before they can be exploited to predict the classes. (2) Stage-wise training with KD, where stage 1 optimizes the 1st half of the net w.r.t. classification targets and stage 2 optimizes the whole network w.r.t. Eq. (2). As in the previous case, stage 1 set the network parameters in a good local minima but such initialization did not seem to help stage 2 sufficiently, which failed to learn. (3) Jointly optimizing both stages w.r.t. the sum of the supervised hint for the guided layer and classification target for the output layer. We performed this experiment, tried different initializations and learning rates with RMSprop (Tieleman & Hinton, 2012) but we could not find any combination to make the network learn. Note that we could ease the training by adding hints to each layer and optimizing jointly as in Deeply Supervised Networks (DSN). Therefore, we built the above-mentioned 19-layer architecture and trained it by means of DSN, achieving a test performance of 88.2%, which is significantly lower than the performance obtained by the FitNets hint-based training (91.61%). Such result suggests that using a very discriminative hint w.r.t. classification at intermediate layers might be too aggressive; using a smoother hint (such as the guidance from a teacher network) offers better generalization. (4) Jointly optimizing both stages w.r.t. the sum of supervised hint for the guided layer and Eq. (2) for the output layer. Adding supervised hints to the middle layer of the network did not ease the training of such a thin and deep network, which failed to learn.

CIFAR-100: To validate our approach, we trained a teacher network of maxout convolutional layers as reported in Goodfellow et al. (2013b) and used the same FitNet architecture as in CIFAR-10. As in Chen-Yu et al. (2014), we augmented the data with random flipping during training. Table 2 summarizes the obtained results. As in the previous case, our FitNet outperforms the teacher model, reducing the number of parameters by a factor of 3 and, when compared to state-of-the-art methods, the FitNet provides near state-of-the-art performance.

3.2 SVHN

The SVHN dataset (Netzer et al., 2011) is composed by 32×32 color images of house numbers collected by GoogleStreet View. There are 73,257 images in the training set, 26,032 images in the test set and 531,131 less difficult examples. We follow the evaluation procedure of Goodfellow et al. (2013b) and use their maxout network as teacher. We trained a 13-layer FitNet composed of 11 maxout convolutional layers, a fully-connected layer and a softmax layer.

Algorithm	**# params**	**Misclass**
Compression		
FitNet	∼1.5M	2.42%
Teacher	∼4.9M	$\mathbf{2.38}\%$
State-of-the-art methods		
Maxout		2.47%
Network in Network		2.35%
Deeply-Supervised Networks		$\mathbf{1.92}\%$

Table 3: SVHN error

Algorithm	**# params**	**Misclass**
Compression		
Teacher	∼361K	0.55%
Standard backprop	∼30K	1.9%
KD	∼30K	0.65%
FitNet	∼30K	$\mathbf{0.51}\%$
State-of-the-art methods		
Maxout		0.45%
Network in Network		0.47%
Deeply-Supervised Networks		$\mathbf{0.39}\%$

Table 4: MNIST error

Table 3 shows that our FitNet achieves comparable accuracy than the teacher despite using only 32% of teacher capacity. Our FitNet is comparable in terms of performance to other state-of-art methods, such as Maxout and Network in Network.

3.3 MNIST

As a sanity check for the training procedure, we evaluated the proposed method on the MNIST dataset (LeCun et al., 1998). MNIST is a dataset of handwritten digits (from 0 to 9) composed of 28x28 pixel greyscale images, with 60K training images and 10K test images. We trained a teacher network of maxout convolutional layers as reported in Goodfellow et al. (2013b) and designed a

FitNet twice as deep as the teacher network and with roughly 8% of the parameters. The 4th layer of the student network was trained to mimic the 2nd layer of the teacher network.

Table 4 reports the obtained results. To verify the influence of using hints, we trained the FitNet architecture using either (1) standard backprop (w.r.t. classification labels), (2) KD or (3) Hint-based Training (HT). When training the FitNet with standard backprop from the softmax layer, the deep and thin architecture achieves 1.9% misclassification error. Using KD, the very same network achieves 0.65%, which confirms the potential of the teacher network; and when adding hints, the error still decreases to 0.51%. Furthermore, the student network achieves slightly better results than the teacher network, while requiring 12 times less parameters.

3.4 AFLW

AFLW (Koestinger et al., 2011) is a real-world face database, containing 25K annotated images. In order to evaluate the proposed framework in a face recognition setting, we extracted positive samples by re-sizing the annotated regions of the images to fit 16x16 pixels patches. Similarly, we extracted 25K 16x16 pixels patches not containing faces from ImageNet (Russakovsky et al., 2014) dataset, as negative samples. We used 90% of the extracted patches to train the network.

In this experiment, we aimed to evaluate the method on a different kind of architecture. Therefore, we trained a teacher network of 3 ReLU convolutional layers and a sigmoid output layer. We designed a first FitNet (FitNet 1) with 15 times fewer multiplications than the teacher network, and a second FitNet (FitNet 2) with 2.5 times fewer multiplications than the teacher network. Both FitNets have 7 ReLU convolutional layers and a sigmoid output layer.

The teacher network achieved 4.21% misclassification error on the validation set. We trained both FitNets by means of KD and HT. On the one hand, we report a misclassification error of 4.58% when training FitNet 1 with KD and a misclassification error of when 2.55% when training it with HT. On the other hand, we report a missclassifation error of 1.95% when training FitNet 2 with KD and a misclassification error of 1.85% when training it with HT. These results show how the method is extensible to different kind of architectures and highlight the benefits of using hints, especially when dealing with thinner architectures.

4 Analysis of empirical results

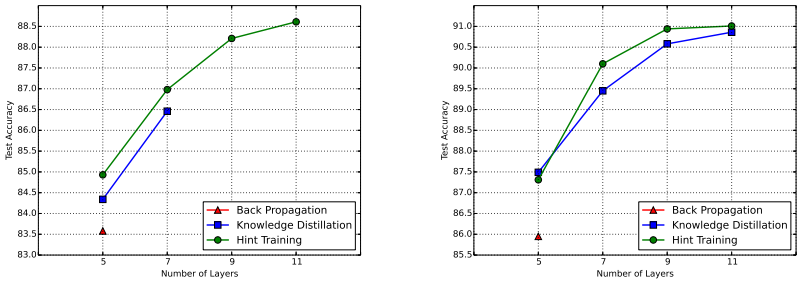
We empirically investigate the benefits of our approach by comparing various networks trained using standard backpropagation (cross-entropy w.r.t. labels), KD or Hint-based Training (HT). Experiments are performed on CIFAR-10 dataset (Krizhevsky & Hinton, 2009).

We compare networks of increasing depth given a fixed computational budget. Each network is composed of successive convolutional layers of kernel size 3×3, followed by a maxout non-linearity and a non-overlapping 2×2 max-pooling. The last max-pooling takes the maximum over all remaining spatial dimensions leading to a 1×1 vector representation. We only change the depth and the number of channels per convolution between different networks, *i.e.* the number of channels per convolutional layer decreases as a network depth increases to respect a given computational budget.

4.1 Assisting the training of deep networks

In this section, we investigate the impact of HT. We consider two computational budgets of approximately 30M and 107M operations, corresponding to the multiplications needed in an image forward propagation. For each computational budget, we train networks composed of 3, 5, 7 and 9 convolutional layers, followed by a fully-connected layer and a softmax layer. We compare their performances when they are trained with standard backpropagation, KD and HT. Figure 2 reports test on CIFAR-10 using early stopping on the validation set, *i.e.* we do not retrain our models on the training plus validation sets.

Due to their depth and small capacity, FitNets are hard to train. As shown in Figure 2(a), we could not train 30M multiplications networks with more than 5 layers with standard backprop. When using KD, we succesfully trained networks up to 7 layers. Adding KD's teacher cross-entropy to the training objective (Eq. (2)) gives more importance to easier examples, *i.e.* samples for which the teacher network is confident and, can lead to a smoother version of the training cost (Bengio, 2009). Despite some optimization benefits, it is worth noticing that KD training still suffers from



(a) 30M Multiplications (b) 107M Multiplications

Figure 2: Comparison of Standard Back-Propagation, Knowledge Distillation and Hint-based Training on CIFAR-10.

Network	**# layers**	**# params**	**# mult**	**Acc**	**Speed-up**	**Compression rate**
Teacher	5	∼9M	∼725M	90.18%	1	1
FitNet 1	11	∼250K	∼30M	89.01%	**13.36**	**36**
FitNet 2	11	∼862K	∼108M	91.06%	4.64	10.44
FitNet 3	13	∼1.6M	∼392M	91.10%	1.37	5.62
FitNet 4	19	∼2.5M	∼382M	**91.61%**	1.52	3.60

Table 5: Accuracy/Speed Trade-off on CIFAR-10.

the increasing depth and reaches its limits for 7-layer networks. HT tends to ease these optimization issues and is able to train 13-layer networks of 30M multiplications. The only difference between HT and KD is the starting point in the parameter space: either random or obtained by means of the teacher's hint. On the one hand, the proliferation of local minima and especially saddle points in highly non-linear functions such as very deep networks highlights the difficulty of finding a good starting point in the parameter space at random (Dauphin et al., 2014). On the other hand, results in Figure 2(a) indicate that HT can guide the student to a better initial position in the parameter space, from which we can minimize the cost through stochastic gradient descent. Therefore, HT provides benefits from an optimization point of view. Networks trained with HT also tend to yield *better test performances* than the other training methods when we fix the capacity and number of layers. For instance, in Figure 2(b), the 7-layers network, trained with hints, obtains a $+0.7\%$ performance gain on the test set compared to the model that does not use any hints (the accuracy increases from 89.45% to 90.1%). As pointed by Erhan et al. (2009), pre-training strategies can act as regularizers. These results suggest that HT is a stronger regularizer than KD, since it leads to better generalization performance on the test set. Finally, Figure 2 highlights that deep models have better performances than shallower ones given a fixed computational budget. Indeed, considering networks that are trained with hints, an 11-layer network outperforms a 5-layer network by an absolute improvement of 4.11% for 107M multiplications and of 3.4% for 30M multiplications. Therefore, the experiments validate our hypothesis that given a fixed number of computations, we leverage depth in a model to achieve faster computation and better generalization.

In summary, this experiment shows that (1) using HT, we are able to train deeper models than with standard back-propagation and KD; and (2) given a fixed capacity, deeper models performed better than shallower ones.

4.2 Trade-off Between Model Performance and Efficiency

To evaluate FitNets efficiency, we measure their total inference times required for processing CIFAR-10 test examples on a GPU as well as their parameter compression. Table 5 reports both the speed-up and compression rate obtained by various FitNets w.r.t. the teacher model along with their number of layers, capacity and accuracies. In this experiment, we retrain our FitNets on training plus validation sets as in Goodfellow et al. (2013b), for fair comparison with the teacher.

FitNet 1, our smallest network, with $36\times$ less capacity than the teacher, is one order of magnitude faster than the teacher and only witnesses a minor performance decrease of 1.3%. FitNet 2, slightly increasing the capacity, outperforms the teacher by 0.9%, while still being faster by a strong 4.64 factor. By further increasing network capacity and depth in FitNets 3 and 4, we improve the performance gain, up to 1.6%, and still remain faster than the teacher. Although a trade-off between speed and accuracy is introduced by the compression rate, FitNets tend to be significantly faster, matching or outperforming their teacher, even when having low capacity.

A few works such as matrix factorization (Jaderberg et al., 2014; Denton et al., 2014) focus on speeding-up deep networks' convolutional layers at the expense of slightly deteriorating their performance. Such approaches are complementary to FitNets and could be used to further speed-up the FitNet's convolutional layers.

Other works related to quantization schemes (Chen et al., 2010; Jégou et al., 2011; Gong et al., 2014) aim at reducing storage requirements. Unlike FitNets, such approaches witness a little decrease in performance when compressing the network parameters. Exploiting depth allows FitNets to obtain performance improvements w.r.t. their teachers, even when reducing the number of parameters $10\times$. However, we believe that quantization approaches are also complementary to FitNets and could be used to further reduce the storage requirements. It would be interesting to compare how much redundancy is present in the filters of the teacher networks w.r.t. the filters of the FitNet and, therefore, how much FitNets filters could be compressed without witnessing significant performance drop. This analysis is out of the scope of the paper and is left as future work.

5 Conclusion

We proposed a novel framework to compress *wide* and *deep* networks into *thin* and *deeper* ones, by introducing *intermediate-level hints* from the teacher hidden layers to guide the training process of the student. We are able to use these hints to train very deep student models with less parameters, which can generalize better and/or run faster than their teachers. We provided empirical evidence that hinting the inner layers of a thin and deep network with the hidden state of a teacher network generalizes better than hinting them with the classification targets. Our experiments on benchmark datasets emphasize that deep networks with low capacity are able to extract feature representations that are comparable or even better than networks with as much as 10 times more parameters. The hint-based training suggests that more efforts should be devoted to explore new training strategies to leverage the power of deep networks.

Acknowledgments

We thank the developers of Theano (Bastien et al., 2012) and Pylearn2 (Goodfellow et al., 2013a) and the computational resources provided by Compute Canada and Calcul Québec. This work has been partially supported by NSERC, CIFAR, and Canada Research Chairs, Project TIN2013-41751, grant 2014-SGR-221 and Spanish MINECO grant TIN2012-38187-C03.

References

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *NIPS*, pp. 2654–2662. 2014.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. Theano: new features and speed improvements. Deep Learning & Unsupervised Feature Learning Workshop, NIPS, 2012.

Bengio, Y. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *NIPS*, pp. 153–160, 2007.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *TPAMI*, 2013.

Bucila, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *KDD*, pp. 535–541, 2006.

Chen, Yongjian, Guan, Tao, and Wang, Cheng. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

Chen-Yu, L., Saining, X., Patrick, G., Zhengyou, Z., and Zhuowen, T. Deeply-supervised nets. *CoRR*, abs/1409.5185, 2014.

Cho, Kyunghyun, Raiko, Tapani, Ilin, Alexander, and Karhunen, Juha. A two-stage pretraining algorithm for deep Boltzmann machines. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2012.

Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pp. 1269–1277. 2014.

Erhan, D., Manzagol, P.A., Bengio, Y., Bengio, S., and Vincent, P. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *AISTATS*, pp. 153–160, 2009.

Gong, Yunchao, Liu, Liu, Yang, Min, and Bourdev, Lubomir. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013a.

Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. In *ICML*, 2013b.

Gulcehre, C. and Bengio, Y. Knowledge matters: Importance of prior information for optimization. In *ICLR*, 2013.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Hinton, G. Vinyals, O. and Dean, J. Distilling knowledge in a neural network. In *Deep Learning and Representation Learning Workshop, NIPS*, 2014.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Jégou, Hervé, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

Koestinger, M., Wohlhart, P., Roth, P.M., and Bischof, H. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, pp. 473–480, 2007.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Montufar, G.F., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. In *NIPS*. 2014.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Reading digits in natural images with unsupervised feature learning. In *Deep Learning & Unsupervised Feature Learning Workshop, NIPS*, 2011.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge, 2014.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., D.A., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Tieleman, T. and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Weston, J., Ratle, F., and Collobert, R. Deep learning via semi-supervised embedding. In *ICML*, 2008.

A SUPPLEMENTARY MATERIAL: NETWORK ARCHITECTURES AND TRAINING PROCEDURES

In the supplementary material, we describe all network architectures and hyper-parameters used throughout the paper.

A.1 CIFAR-10/CIFAR-100

In this section, we describe the teacher and FitNet architectures as well as hyper-parameters used in both CIFAR-10/CIFAR-100 experiments.

A.1.1 TEACHERS

We used the CIFAR-10/CIFAR-100 maxout convolutional networks reported in Goodfellow et al. (2013b) as teachers. Both teachers have the same architecture, composed of 3 convolutional hidden layers of 96-192-192 units, respectively. Each convolutional layer is followed by a maxout non-linearity (with 2 linear pieces) and a max-pooling operator with respective windows sizes of 4x4, 4x4 and 2x2 pixels. All max-pooling units have an overlap of 2x2 pixels. The third convolutional layer is followed by a fully-connected maxout layer of 500 units (with 5 linear pieces) and a top softmax layer. The CIFAR-10/CIFAR-100 teachers are trained using stochastic gradient descent and momentum. Please refer to Goodfellow et al. (2013b) for more details.

A.1.2 FITNETS

Here, we describe the FitNet architectures used in the Section 3 and Section 4. Each FitNet is composed of successive zero-padded convolutional layers of kernel size 3×3, followed by a maxout non-linearity with two linear pieces. A non-overlapping 2×2 max-pooling follows some of the convolutional layers; each network has a total of 3 max-pooling units. The last max-pooling takes the maximum over all remaining spatial dimensions, leading to a 1×1 vector representation. The last convolutional layer is followed by a fully-connected and a softmax layer, as the ones on CIFAR-10/100 teachers.

Table 6 describes the architectures used for the depth experiment in Figure 2. Table 7 describes the architectures for the efficiency-performance trade-off experiment in Table 5. The results reported in Table 1, Table 2 and Table 3 correspond to the FitNet 4 architecture.

All FitNet parameters were initialized randomly in U(-0.005,0.005). We used stochastic gradient descent with RMSProp (Tieleman & Hinton, 2012) to train the FitNets, with an initial learning rate 0.005 and a mini-batch size of 128. Parameter λ in Eq. (2) was initialized to 4 and decayed linearly during 500 epochs reaching $\lambda = 1$. The relaxation term τ was set to 3.

On CIFAR-10, we divided the training set into 40K training examples and 10K validation examples. We trained stage 1 by minimizing Eq. (3) and stopped the training after 100 epochs of no validation error improvement, performing a maximum of 500 epochs. After that, we trained stage 2 by minimizing Eq. (2) using RMSprop, the same stopping criterion and the same hyper-parameters as stage 1. We picked the optimal number of epochs according to the above-mentioned stopping criterion and retrained the FitNet on the whole 50K training examples (training + validation sets).

On CIFAR-100, we trained directly on the whole training set using stochastic gradient descent with RMSprop, the same hyper-parameters as CIFAR-10 FitNets and the number of epochs determined by CIFAR-10 stopping criterion.

A.2 MNIST

In this section, we describe the teacher and FitNet architectures as well as the hyper-parameters used in the MNIST experiments.

We trained a teacher network of maxout convolutional layers as reported in Goodfellow et al. (2013b). The teacher architecture has three convolutional maxout hidden layers (with 2 linear pieces each) of 48-48-24 units, respectively, followed by a spatial max-pooling of 4x4-4x4-2x2 pixels, with

5 Layer	7 Layer	9 Layer	11 Layer
conv 3x3x64 (3x3x128) pool 2x2	conv 3x3x16 (3x3x32) conv 3x3x32 (3x3x64) pool 2x2	conv 3x3x16 (3x3x32) conv 3x3x32 (3x3x32) pool 2x2	conv 3x3x16 (3x3x16) conv 3x3x16 (3x3x32) conv 3x3x16 (3x3x32) pool 2x2
conv 3x3x64 (3x3x128) pool 2x2	conv 3x3x32 (3x3x80) conv 3x3x64 (3x3x80) pool 2x2	conv 3x3x32 (3x3x64) conv 3x3x32 (3x3x80) conv 3x3x32 (3x3x80) pool 2x2	conv 3x3x32 (3x3x48) conv 3x3x32 (3x3x64) conv 3x3x32 (3x3x80) pool 2x2
conv 3x3x64 (3x3x128) pool 8x8	conv 3x3x64 (3x3x128) pool 8x8	conv 3x3x48 (3x3x96) conv 3x3x64 (3x3x128) pool 8x8	conv 3x3x48 (3x3x96) conv 3x3x48 (3x3x96) conv 3x3x64 (3x3x128) pool 8x8
fc softmax	fc softmax	fc softmax	fc softmax
hint: 2←2	hint: 4←2	hint: 5←2	hint: 7←2

Table 6: Fitnet architectures with a computational budget of 30M (or 107M) of multiplications: conv $s_x \times s_y \times c$ is a convolution of kernel size $s_x \times s_y$ with c outputs channels; pool $s_x \times s_y$ is a non-overlapping pooling of size $s_x \times s_y$; fc stands for fully connected. hint: FitNet $\leftarrow$ teacher specifies the hint and guided layers used for hint-based training, respectively.

FitNet 1	FitNet 2	FitNet 3	FitNet 4
conv 3x3x16 conv 3x3x16 conv 3x3x16 pool 2x2	conv 3x3x16 conv 3x3x32 conv 3x3x32 pool 2x2	conv 3x3x32 conv 3x3x48 conv 3x3x64 conv 3x3x64 pool 2x2	conv 3x3x32 conv 3x3x32 conv 3x3x32 conv 3x3x48 conv 3x3x48 pool 2x2
conv 3x3x32 conv 3x3x32 conv 3x3x32 pool 2x2	conv 3x3x48 conv 3x3x64 conv 3x3x80 pool 2x2	conv 3x3x80 conv 3x3x80 conv 3x3x80 conv 3x3x80 pool 2x2	conv 3x3x80 conv 3x3x80 conv 3x3x80 conv 3x3x80 conv 3x3x80 conv 3x3x80 pool 2x2
conv 3x3x48 conv 3x3x48 conv 3x3x64 pool 8x8	conv 3x3x96 conv 3x3x96 conv 3x3x128 pool 8x8	conv 3x3x128 conv 3x3x128 conv 3x3x128 pool 8x8	conv 3x3x128 conv 3x3x128 conv 3x3x128 conv 3x3x128 conv 3x3x128 conv 3x3x128 pool 8x8
fc softmax	fc softmax	fc softmax	fc softmax
hint: 6←2	hint: 6←2	hint: 8←2	hint: 11←2

Table 7: Performance-Efficiency FitNet architectures.

an overlap of 2x2 pixels. The 3rd hidden layer is followed by a fully-connected softmax layer. As is Goodfellow et al. (2013b), we added zero padding to the second convolutional layer.

We designed a FitNet twice as deep as the teacher network and with roughly 8% of the parameters. The student architecture has 6 maxout convolutional hidden layers (with 2 linear pieces each) of 16-16-16-16-12-12 units, respectively. Max-pooling is only applied every second layer in regions of 4x4-4x4-2x2 pixels, with an overlap of 2x2 pixels. The 6th convolutional hidden layer is followed by a fully-connected softmax layer.

The teacher network was trained as described in Goodfellow et al. (2013b). The FitNet was trained in a stage-wise fashion as described in Section 2. We divided the training set into a training set of 50K samples and a validation set of 10K samples.

All network parameters where initialized randomly in U(-0.005,0.005). In the first stage, the 4th layer of the FitNet was trained to mimic the 2nd layer of the teacher network, by minimizing Eq. (3) through stochastic gradient descent. We used a mini-batch size of 128 samples and fixed the learning rate to 0.0005. We initialized λ to 4 and decayed it for the first 150 epochs until it reached 1. The training was stopped according to the following criterion: after 100 epochs of no validation error improvement and performning a maximum of 500 epochs. We used the same mini-batch size, learning rate and stopping criterion to train the second stage. The relaxation term τ was set to 3.

A.3 SVHN

In this section, we describe the teacher and FitNet architectures as well as the hyper-parameters used in the SVHN experiments.

We used SVHN maxout convolutional network described in as Goodfellow et al. (2013b) teacher. The network is composed of 3 convolutional hidden layers of 64-128-128 units, respectively, followed by a fully-connected maxout layer of 400 units and a top softmax layer. The teacher training was carried out as in Goodfellow et al. (2013b).

We used the FitNet 4 architecture outlined in Table 7, initializing the network parameters randomly in U(-0.005,0.005) and training with the same hyper-parameters as in CIFAR-10. In this case, we used the same early stopping as in CIFAR-10, but we did not retrain the FitNet on the whole training set (training + validation). The same hyper-parameters where used for both stages.

A.4 AFLW

In this section, we describe the teacher and FitNet architectures as well as the hyper-parameters used in the AFLW experiments.

We trained a teacher network of 3 ReLU convolutional layers of 128-512-512 units, respectively, followed by a sigmoid layer. Non-overlapping max-pooling of size 2×2 was performed after the first convolutional layer. We used receptive fields of 3-2-5 for each layer, respectively.

We designed two FitNets of 7 ReLU convolutional layers. Fitnet 1's layers have 16-32-32-32-32-32-32 units, respectively, followed by a sigmoid layer. Fitnet 2's layers have 32-64-64-64-64-64-64 units, respectively, followed by a sigmoid layer. In both cases, we used receptive fields of 3×3 and, due to the really small image resolution, we did not perform any max-pooling.

All network parameters of both FitNets where initialized randomly in U(-0.05,0.05). Both FitNets were trained in the stage-wise fashion described in Section 2. We used 90% of the data for training. In the first stage, the 5th layer of the FitNets were trained to mimic the 3rd layer of the teacher network, by minimizing Eq. (3) through stochastic gradient descent. We used a mini-batch size of 128 samples and initialized the learning rate to 0.001 and decayed it for the first 100 epochs until reaching 0.01. We also used momentum. We initialized momentum to 0.1 and saturated it to 0.9 at epoch 100. We picked the best validation value after a 500 epochs. We used the same mini-batch size, learning rate and stopping criterion to train the second stage. The relaxation term τ was set to 3.