Efficient Recurrent Residual Networks Improved by Feature Transfer

MSc Thesis

written by

Yue Liu

under the supervision of **Dr. Silvia-Laura Pintea**, **Dr. Jan van Gemert**, and **Dr. Ildiko Suveg** and submitted to the Board of Examiners for the degree of

Master of Science

at the *Delft University of Technology.*

Date of the public defense:	**Members of the Thesis Committee:**
August 31, 2017	Prof. Marcel Reinders
	Dr. Jan van Gemert
	Dr. Julian Urbano Merino
	Dr. Silvia-Laura Pintea
	Dr. Ildiko Suveg (Bosch)
	Dr. Gonzalez Adrlana (Bosch)

Efficient Recurrent Residual Networks Improved by Feature Transfer

Yue Liu
Delft University of Technology
y.liu-33@student.tudelft.nl

Silvia-Laura Pintea
Delft University of Technology
S.L.Pintea@tudelft.nl

Jan van Gemert
Delft University of Technology
J.C.vanGemert@tudelft.nl

Ildiko Suveg
Bosch Security Systems
Ildiko.Suveg@nl.bosch.com

Abstract

Over the past several years, deep and wide neural networks have achieved great success in many tasks. However, in real life applications, because the gains usually come at a cost in terms of the system resources (e.g., memory, computation and power consumption), it is impractical to run top-performing but heavy networks such as VGGNet and GoogleNet directly on mobile and embedded devices, like smartphones and cameras. To tackle this problem, we propose the use of recurrent layers in residual networks to reduce the redundant information and save the parameters. Furthermore, with the help of feature map knowledge transfer, the performance of Recurrent Residual Networks (ReResNet) can be improved so as to reach similar accuracy to some complex state-of-the-art architectures on CIFAR-10, even with much fewer parameters. In this paper, we demonstrate the efficiency of ReResNet possibly improved by Feature Transfer on three datasets, CIFAR-10, Scenes and MiniPlaces.

1. Introduction

Neural networks have been gaining numerous success on many disciplines such as computer vision [1, 2] and speech recognition [3, 4]. To achieve high accuracy on large quantities of data, the general trend is to construct deeper and wider networks [1, 5, 6, 7]. However, in practice, top-performing architectures that require a fairly large number of parameters like VGGNet and GoogleNet, are not efficient to use on constrained devices in terms of the system memory storage, computational capability and power consumption.

One big challenge for deep learning in real life applications is the memory concern. Most of the mobile and embedded platforms are equipped with small storage, but deep and wide networks conventionally occupy gigabytes of memory to store, so devices users may spend a lot of time installing and updating the applications, which is not desirable in practice. Moreover, since the models often run in the order of tens of MBs and the peak memory needs could reach hundreds of MBs, it is hard to fulfil the runtime memory requirements [8]. As the secondary effect of the large memory overhead in most cases, computation that is required to perform inference, is also shaped by network architectures [9]. For instance, compared with multi-layer perceptrons which are more memory-bound, convolutional neural networks that are widely used in computer vision and speech recognition are more computation-bound [10]. Although some constrained devices like smartphones have significant computational power, we still face the challenges in executing a large amount of computations required by a heavy network. Additionally, when we extend further from the computation concern, because of excessive execution time for each single inference, the high power-consumption level of large networks makes real life implementations infeasible for continuous monitoring [11, 12].

To address the above problems, several approaches to do model compression that aims to construct simple but effective neural network architectures have been proposed. One interesting method, knowledge transfer, which is to train a light student network to mimic the behavior of a well-trained heavy teacher network, generates a high level of performance [13, 14, 15, 16, 17]. Thus in this paper, we aim to come up with a compact model and then make better use of knowledge transfer to further boost the performance.

Artificial neural networks are originally inspired by the biological neural networks in human brains. Some studies have found that recurrent connections exist ubiquitously in the neocortex [18], visual signals always go through the same neuron recursively [19, 20], and the recurrent connections are very important for capturing the context information [21]. To incorporate those facts of human brains, we make use of recurrent residual layers introduced in [22] to

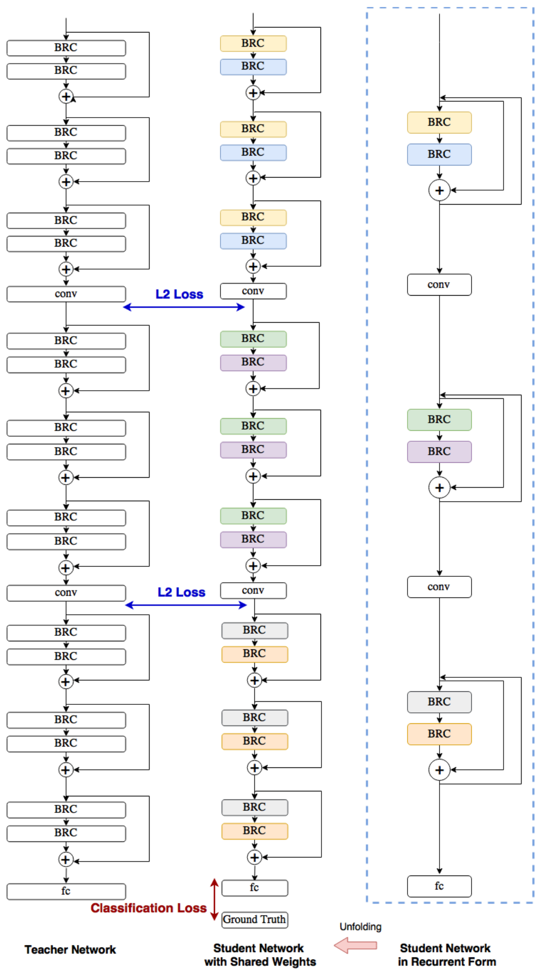

Figure 1: Overall architecture of a ReResNet improved by feature map knowledge transfer. "BRC" denotes layer with pipeline "Batch Normalization(BN)-Rectified Linear Units(ReLU)-Convolution(Conv)". Layers with the same color share one set of convolution parameters over time. In student networks (both with Shared Weights and in Recurrent Form), the convolutional layers with no color (shown in white) operate spatial dimension reduction with a stride of 2. In this example, the teacher network is a general ResNet that is wider than its student, while the student network is a recurrent ResNet having three different recurrent residual blocks in total. Both layers in each block are shared for three times (see the unfolded version, Student Network with Shared Weights), so the student only has six different residual layers (shown in six different colors) which is 3× fewer than its teacher. Together with the classification loss, two extra intermediate losses extracted after the first and second recurrent blocks are added to the final loss function.

tie convolutional weights, and propose a new use of it so that networks become more compact. We show that the accuracy gap between our recurrent residual network (ReResNet) and its original sequential ResNet [7] is very small, especially when it is combined with knowledge transfer and trained with the guidance of feature map representations from a powerful network.

One example of our proposed method's overall architecture is shown in Figure 1. We share the weights of all convolutions at the same spatial scale, thus making the student network equivalent to a recurrent network so as to save parameters. Moreover, by minimizing the total loss that consists of classification loss and two extra intermediate losses, the student network is able to learn the intermediate representations of its teacher and get good accuracy. In our experiments on CIFAR-10 [23], the results of ReResNet improved by feature map knowledge transfer are even as competitive as that of some large state-of-the-art networks.

To summarize, our main contributions are:

- We propose the use of recurrent residual layers to reduce the redundant information and save the parameters while preserving good accuracy. We show that with the help of recurrent layers, it is possible to deepen the network to get better performance by making ReResNets recur more times, with almost the same parameter budget.

- We propose a new teacher-student knowledge transfer pattern with a general ResNet as the teacher and a ReResNet as the student. We show that adding extra losses at multiple depths of a ReResNet aids its training and improves the performance.

- We show that despite the compactness of the model, our new use of ReResNet that can be improved by Feature Transfer achieves good performance. We demonstrate the efficiency of our proposed method on a widely-used dataset CIFAR-10 for object classification [23], a small dataset Scenes for indoor scene classification [24] and a large dataset MiniPlaces for indoor and outdoor scenes [25].

The paper is structured as follows: we first present the related work in Sec. 2; we explain our proposed ReResNet improved by Feature Transfer in Sec. 3; then in Sec. 4 we validate our method on CIFAR-10, Scenes and MiniPlaces, and discuss the questions such as "how can a ReResNet perform on a benchmark/small/big dataset?" and "how many times can a ReResNet recur?"; finally, we conclude our paper in Sec. 5, and attach some visualization results in Appendix.

2. Related work

2.1. Recurrent neural networks

Recurrent neural networks (RNNs) are naturally used to process sequential and temporal data such as video [26] and speech signal [27], but they have seen limited use on static visual signal processing. To explore RNNs on image data, the idea of sharing weights across convolutional layers was first studied by Eigen *et al.* [28]. They showed that even though no extra parameters are introduced, adding suitable number of recurrent layers tend to improve the performance (the network was called rCNN for short). Based on that, Liang *et al.* [29] introduced a time-folded version called RCNN, and additionally allowed feed-forward input to each shared convolutional layer. They showed that although RCNN and rCNN have exactly the same number of parameters, the former outperforms the latter to some extent. Feedback Networks proposed by Zamir *et al.* [30] applied convolutional LSTM model to enable early predictions. Liao and Poggio [22] achieved good classification performance when adopting shared weights on ResNets, which are exactly recurrent networks that form a cycle. Based on that, we propose the new use of recurrent residual layers to save the parameters while preserving good accuracy. Compared with ShaResNets recently proposed by Boulch and Alexandre [31] that only share the adopted spatial relations on ResNets, our paper introduces two variants that are $2\times$ lighter but still effective by sharing both convolutions in each residual block, and investigates how many times one ReResNet can recur at most with the same parameter budget.

2.2. Efficient Networks

To construct simple but effective neural networks, several approaches have been proposed in recent literature. The obtaining of efficient networks can be generally categorized into either pre-trained architecture compression, or direct small network construction with knowledge transfer.

Pre-trained architecture compression aims to make trained models have smaller size by directly reducing their number of parameters or computations. Speeding up with Low Rank Expansions proposed by Jaderberg *et al.* [32] exploits the existing redundancy between different feature channels and filters by approximating a learnt full rank filter as combinations of several rank-1 filters. To produce sparse weights instead of dense weights, the method Deep Compression proposed by Han *et al.* [33] is based on a three-stage pipeline: weight pruning [34, 35, 36], trained quantization [37] and Huffman coding [38], and it saves the parameter storage by $35\times$ on AlexNet and $49\times$ on VGG-16 without loss of accuracy. Molchanov *et al.* [39] further proposed iterative pruning of parameters from deep neural networks. Recently, Zhou *et al.* [40] presented a novel weights quantization method, called Incremental Network Quantization(INQ), to convert the parameters from original 32-bit floating point to 5-bit, 4-bit, 3-bit, and even 2-bit, and proves to have impressive efficiency on different architectures, such as AlexNet, VGG-16, GoogleNet and ResNet.

Another method to achieve model compression and acceleration is knowledge transfer, which is to directly train a compact neural network with the help of some complex models. Bucilu *et al.* [13] first trained a single neural network of modest size to mimic a much larger ensemble, so that with fewer parameters, the student model can reach same accuracy as its ensemble teacher model. Ba and Caruna [14] then replaced the ensemble teacher network with a deep neural network, and trained a shallow student model to mimic logits got from its teacher. For some tasks where having multiple convolutions is less important it gave similar accuracy as the state-of-art with the same parameter budget. Hinton *et al.* [15] further developed this approach by considering both soften labels and hard labels in loss function, and finally it succeeded in training smaller and/or shallower models that are nearly as accurate as its teacher model. Then, Romero *et al.* [16] proposed FitNets and introduced an intermediate hint from the teacher model, showing that distilled student model which is deeper but thinner, could achieve the same or even better accuracy than its teacher with fewer parameters and computations. Recently, Zagoruyko *et al.* [17] presented an attention transfer model to learn less deep student networks better, but unlike FitNets approach, they did not separate the training process into two. Instead, they added extra losses to force a student network to mimic the attention maps of its teacher model. Our feature map transfer is similar to the attention model, but rather than the general ResNets, we use ReResNets as our student models.

3. Method

Recurrent Residual Networks (ReResNets) are based on Residual Networks (ResNets) [7], for which identity mapping shortcuts are inserted into plain convolutional networks to solve the problem of gradient vanishing.

In addition to the case where all convolutional layers have independent weights, we consider ReResNets in which ResNets are encapsulated into recurrent models by using tied weights in certain parts of the architecture. Sec.3.1 introduces two variants of such ReResNets. Furthermore, in Sec.3.2 we describe a feature map knowledge transfer method that helps speed up the training process and make small networks perform better.

3.1. Recurrent residual networks

The key module of ReResNet is the recurrent residual block that is shared at the same spatial scale. The output of

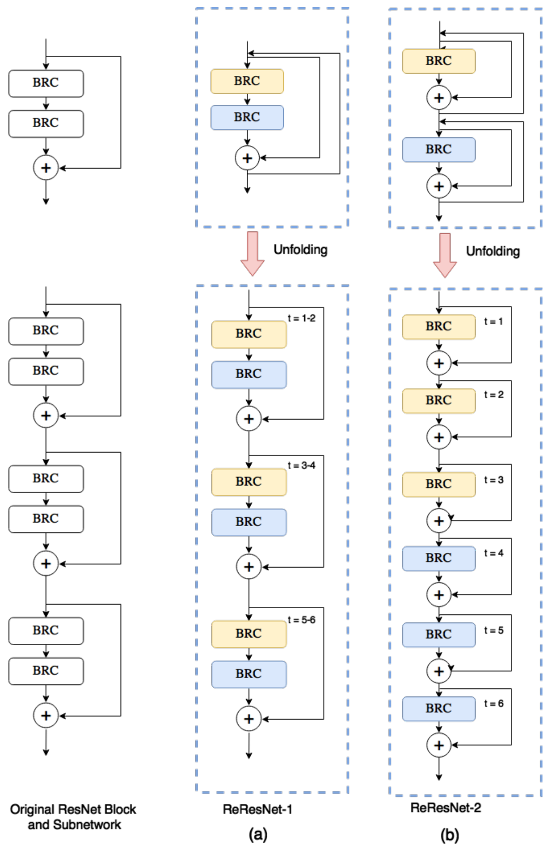

Figure 2: Subnetworks of original ResNet and two ReResNet variants with the same number of layers. "BRC" denotes pipeline with "BN-ReLU-Conv". In original ResNet, layers without color (shown in white) are not tied over time, while in two ReResNet variants, layers with the same color (yellow or blue) share one set of convolution parameters. Subnetworks in dashed boxes are our folded/unfolded ReResNet units. (a) ReResNet-1: The ReResNet unit is the same as the original residual block, each with two BRC layers inside. Both convolutions in each residual block are tied over time. The first convolution is shared at time $t = 1$, $t = 3$, and $t = 5$, while the second is shared at time $t = 2$, $t = 4$ and $t = 6$; (2) ReResNet-2: Break apart the original residual block of two BRC layers with one identity mapping, into two blocks of one BRC layer with two identity connections in total. Two different residual blocks work together as one ReResNet unit, and each block has one BRC layer inside. The first convolution is shared at time $t = 1$, $t = 2$, and $t = 3$, while the second is shared at time $t = 4$, $t = 5$ and $t = 6$.

the ith block at time step t is defined as:

$$y_i(t) = f_i(y_i(t-n); W_i, W_t). \tag{1}$$

where n simulates the cycle time it takes to go through one residual block, assuming each convolution takes constant time 1. W_i denotes the shared parameters of the ith block and W_t is the time-specific parameters. f_i is the residual function of the ith block, composed of batch normalization (BN) [41], rectified linear units (ReLU) [42] and convolutional layers (Conv).

The subnetwork structures of two proposed ReResNet variants are illustrated in Figure 2. Our experiments show that if we share all parameters in each block including BN learnable scaling and shifting parameters, there will be a considerable drop in accuracy in ReResNets. So as in [22], we only share the weights of convolutional layers W_{conv} as W_i, while the rest parameters are all set to time-specific as W_t in Eq. 1. Details of two variants are shown below:

- ReResNet-1: One residual block that consists of two BN-ReLU-Conv layers is a ReResNet unit. Each block has cycle time $n = 2$ and shared parameters $W_i = \{W_{conv1,i}, W_{conv2,i}\}$. The recurrent connection enables both convolutions in each residual block to be shared alternately over time.

- ReResNet-2: The original residual block with two BN-ReLU-Conv layers is now broken apart into two different residual blocks with only one layer for each. Two residual blocks work together as one ReResNet unit, and each block has cycle time $n = 1$ and shared parameters $W_i = W_{conv,i}$. In contrast with ReResNet-1, ReResNet-2 shares two convolutions in sequence. Not until the first block finishes its recurrence does the second start to be shared, so the first convolution is common to the first half of the subnetwork, while the second convolution is tied for the rest.

The unfolded network architectures of ReResNet-1 and ReResNet-2 show that both networks have the same number of layers but with different sharing configurations. More nonlinearity is introduced as the network is deepened, however, since in most cases convolutions introduce most of the parameters in the network and they are shared now with recurrent layers, the memory required to store all parameters of our ReResNets is not increased by much.

Since we use tied convolutional weights, as in [29, 31] for $\forall W \in W_i$ the gradient back-propagation is adjusted according to

$$\left(\frac{\partial L}{\partial W}\right)_{ReResNet} = \sum_{W' \in W_i} \left(\frac{\partial L}{\partial W'}\right)_{original}. \tag{2}$$

where L is the loss function we aim to optimize. The gradient of a shared weight is the sum of the gradients of all weights in W_i.

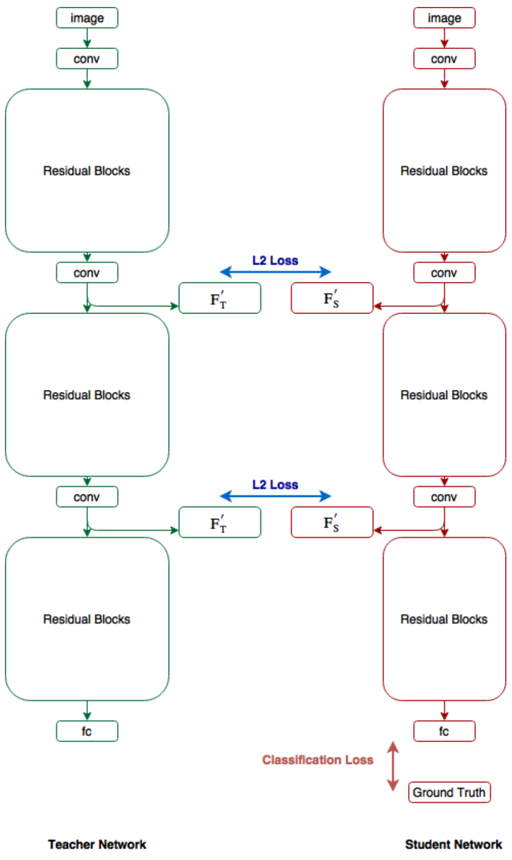

Figure 3: Architecture of feature transfer. The teacher network is wider than the student network with respect to the number of convolutional filters. F_T' and F_S' are mathematical representations of the feature maps that match each other's tensor shape. The student network is trained with the sum of classification loss and two intermediate losses as the final loss function, to get correct predictions and achieve similar intermediate feature representations as the powerful teacher network.

3.2. Feature map knowledge transfer

In this paper, Feature map knowledge transfer is for further improving the performance of a small network ReResNet.

In a convolutional neural network, each convolution can be regarded as one feature extractor. Early feature maps correspond to low-level features, while with feature maps going deeper, the network will detect more complex high-level representations. Based on the assumption that for the same task, similar networks achieve similar intermediate representations in a solution flow, as in [17] we implement feature map knowledge transfer. The main idea of it is to distill feature maps from a trained complex teacher network and then transfer the knowledge to a small student network by adding multiple intermediate losses to the final loss function. An ideal student network will not only have correct predictions but also learn the intermediate outputs. In this paper, we use "feature transfer' to refer to the above process.

The architecture of feature transfer is shown in Figure 3. Let $F \in R^{h\times w\times c}$ be the feature map that is directly extracted after each residual group, where h, w, and c are height, width and number of channels respectively. In this paper, we consider the case of having a wider teacher network with more convolutional filters than the student network. Therefore, to get the intermediate loss functions between features of two networks, mathematical representations F_T' and F_S' that match each other's tensor shape are needed. To construct a mapping function from original feature maps ($F_T \in R^{h\times w\times m}$ and $F_S \in R^{h\times w\times n}$) to the ones that can be used for intermediate loss calculation ($F_T' \in R^{h\times w\times x}$ and $F_S' \in R^{h\times w\times x}$), we discuss two options:

- 1-channel feature map generation: As proposed in [17], we add up the squared values across channels to get $F_T' \in R^{h\times w\times 1}$ and $F_S' \in R^{h\times w\times 1}$. The mathematical expressions are shown below.

$$F_T' = \sum_{i=1}^{i=m} \mid F_{T,i} \mid^2 . \tag{3}$$

$$F_S' = \sum_{i=1}^{i=n} \mid F_{S,i} \mid^2 . \tag{4}$$

  where i goes over each channel.

- Convolutional regressor: Convolve original feature maps with filters that have the size of $k_1 \times k_2 \times m \times x$ and $k_1 \times k_2 \times n \times x$, where $k_1 \times k_2$ denotes the kernel shape. m and n are original numbers of channels for $F_T \in R^{h\times w\times m}$ and $F_S \in R^{h\times w\times n}$, and x is the consistent number of channel for feature maps to be used $F_T' \in R^{h\times w\times x}$ and $F_S' \in R^{h\times w\times x}$.

Let W_T and W_S represent the weights of teacher and student networks, and $\mathcal{L}_{cls}(W)$ and $\mathcal{L}_{ft}(W)$ denote the classification and feature transfer $\mathcal{L}2$ loss function. Like in [17], the total loss $\mathcal{L}_{total}(W)$ is defined as:

$$\mathcal{L}_{total}(W_S) = \mathcal{L}_{cls}(W_S) + \lambda\mathcal{L}_{ft}(W_S). \tag{5}$$

$$\mathcal{L}_{ft}(W_S) = \sum_{i=1}^{i=g} \left\| \frac{F_T^{'(i)}(W_T)}{\left\|F_T^{'(i)}(W_T)\right\|_2} - \frac{F_S^{'(i)}(W_S)}{\left\|F_S^{'(i)}(W_S)\right\|_2} \right\|_2 . \tag{6}$$

where λ represents the weight to balance each loss function, and g calculates how many intermediate $\mathcal{L}2$ losses we have between the teacher model and the student model ($g = 2$ in the example of Figure 3). Due to the fact that teacher and student networks may have features with different scales, the normalization of generalized feature map representations is necessary.

The training process is summarized below in Algorithm 1. After randomly initializing teacher parameters W_T, we first train a powerful teacher network to get trained W_T^*. Then similarly, we randomly initialize student parameters W_S and get trained W_S^* with the sum of classification loss and multiple losses at different depths as the loss function. If convolutional regressor is chosen to match the shapes of feature representations, the regressor parameters W_r are trained together with the student parameters W_S. By minimizing the total loss function, we can then leverage feature knowledge acquired from a teacher network to a student network and achieve feature map knowledge transfer.

Algorithm 1 Feature map knowledge transfer

```
Input: $W_T, W_S, W_r, \lambda$
Output: $W_S^*$
1: $W_T^* \leftarrow \arg\min_{W_T} \mathcal{L}_{cls}(W_T)$
2: if 1-channel feature map generation then
3:     $W_S^* \leftarrow \arg\min_{W_S}(\mathcal{L}_{cls}(W_S) + \lambda\mathcal{L}_{ft}(W_S; {W_T}^*))$
4: else if convolutional regressor then
5:     $W_S^* \leftarrow \arg\min_{W_S}(\mathcal{L}_{cls}(W_S) + \lambda\mathcal{L}_{ft}(W_S, W_r; W_T^*))$
6: end if
```

4. Experimental evaluation

We experiment on three datasets CIFAR-10 [23], Scenes [24] and MiniPlaces [25]. To evaluate our methods, we compare ReResNets with their sequential ResNets counterparts, and show the gains we get from Feature Map Knowledge Transfer.

4.1. Experiments on CIFAR-10

4.1.1 Implementation

CIFAR-10 consists of a set of small 32×32 RGB images from 10 different object classes, and it is divided into 50,000 training and 10,000 test images. We prepare the dataset by subtracting the mean and normalize its scale to $[-1, 1]$ along each channel. For data augmentation, as in [7] we pad 4 pixels on each side, take random 32×32 crops and then do random horizontal flips.

The model was trained from scratch. We use a weight decay of 0.0001 and momentum of 0.9. The weights are initialized as in [43]. Experiments were run for 80,000 iterations with batch size 128. The learning rate started with 0.1 and was then divided by 10 at 40,000 iterations and 60,000 iterations. We use softmax and cross-entropy loss for classification.

name	output size	operation type
conv1	32×32	$[3 \times 3, 16]$, stride 1
group1	32×32	$\begin{bmatrix} 3 \times 3, 16 \times k \\ 3 \times 3, 16 \times k \end{bmatrix} \times 3N$
transition1	16×16	$[3 \times 3, 32 \times k]$, stride 2
group2	16×16	$\begin{bmatrix} 3 \times 3, 32 \times k \\ 3 \times 3, 32 \times k \end{bmatrix} \times 3N$
transition2	8×8	$[3 \times 3, 64 \times k]$, stride 2
group3	8×8	$\begin{bmatrix} 3 \times 3, 64 \times k \\ 3 \times 3, 64 \times k \end{bmatrix} \times 3N$
avg-pool	1×1	$[8 \times 8]$, stride 1

Table 1: Structure of our ResNet(3N-k)/ReResNet-x(3N-k) on CIFAR-10. Network depth and width are determined by factors N and k respectively. "3×3" denotes 3×3 convolutions with batch normalization and rectified linear units.

The general architecture of our ResNet/ReResNet on CIFAR-10 is shown in Table 1. It is composed of an initial convolution, 3 residual groups of width $16 \times k$, $32 \times k$, and $64 \times k$ (each with 3N blocks inside), and 2 transition convolutions of width $32 \times k$ and $64 \times k$ with strides 2 to perform downsampling. Then it goes through an average pooling layer and a final classification layer. In this paper, we refer to this network as "ResNet(3N-k)/ReResNet-x(3N-k)" by following "ResNet(d-w)/ReResNet-x(d-w)", where x indicates the type of recurrent as discussed in Section 3.1, while d and w are the depth and width factors. Each residual group has d blocks and we apply $[16 \times k, 32 \times k, 64 \times k]$ filters to convolutional layers in each residual block.

4.1.2 ReResNet performance on CIFAR-10

On CIFAR-10, we experiment with two variants of ReResNets. The teacher network ResNet(3-2) has the accuracy of 93.28% and it is 2× wider than its students for both cases of having 3 or 6 blocks per residual group.

The quantitative evaluation of our proposed ReResNet together with its feature map transfer can be found in Table 2. Sharing the weights of convolutional layers in ReResNets helps reduce the size of network significantly. Especially when we add 1-channel feature map transfer, there is no huge accuracy gap between each ReResNet and its original sequential ResNet counterpart. To our understanding, the weights of residual blocks in each group are somehow

network name	params	no FT	FT	gains
ResNet(3-1)	316K	**91.05%**	91.50%	0.45%
ReResNet-1(3-1)	122K	89.81%	**90.29%**	0.48%
ReResNet-2(3-1)	122K	89.25%	89.90%	0.64%
ResNet(6-1)	607K	**92.00%**	92.45%	0.45%
ReResNet-1(6-1)	124K	89.99%	**90.49%**	0.50%
ReResNet-2(6-1)	124K	89.00%	89.82%	0.82%

Table 2: Test accuracy on CIFAR-10. "Feature Transfer" is denoted as "FT" for short, and it is conducted with the option "1-channel feature map generation". In this set of experiments, for feature transfer, knowledge is distilled from a teacher network ResNet(3-2) with accuracy 93.28%.

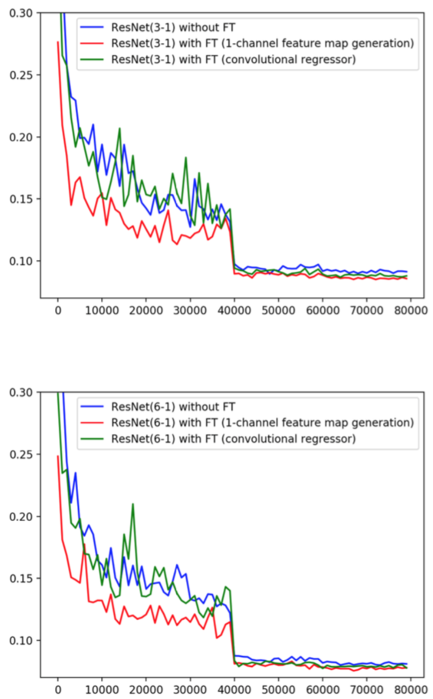

Figure 4: Test accuracy of ResNet(3-1) and ResNet(6-1) without/with two feature transfer options, 1-channel feature map generation and convolutional regressor, on CIFAR-10.

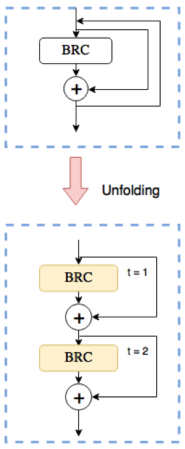

Figure 5: Illustration of ReResNet-3(r). r denotes the recurring times. In this example r is set to be 2.

similar, and the same parts of each block share similar operations, so the ReResNet makes better use of the trainable parameters when compared to its sequential ResNet counterpart.

We also plot test accuracy of ResNet(3-1) and ResNet(6-1) that are trained independently, and those adopt the distilled knowledge from ResNet(3-2), in Figure 4. It illustrates that feature map knowledge transfer speeds up the convergence greatly, and can improve the accuracy in the end. The feature transfer option "1-channel feature map generation" outperforms "convolutional regressor". We argue that this is likely because to calculate intermediate loss, the latter option introduces extra parameters W_r that will not contribute to final classification.

4.1.3 How many times can a ReResNet recur?

To investigate how many times a ReResNet can recur at most, we introduce a new type of ReResNet with only one residual block with one BRC layer inside as a ReResNet unit (see Figure 5 for its illustration). We refer to it as ReResNet-3(r), where r represents how many times we recur. We compare the test accuracy of ReResNet-1, ReResNet-2 and ReResNet-3 with different recurring times (see Table 3). It shows that having recurrent loops to grow the network depth leads to better performance without any significant increase in model size. This is because adding extra recurrent layers introduces more nonlinearity, and it increases the network capacity to learn more complex functions. However, it is worth noting that recurring too many times will make the networks suffer from overfitting.

network name	params	accuracy
ReResNet-1(1-1)	121K	89.44%
ReResNet-1(3-1)	122K	89.81%
ReResNet-1(6-1)	124K	**89.99%**
ReResNet-2(1-1)	121K	88.98%
ReResNet-2(3-1)	122K	**89.25%**
ReResNet-2(6-1)	124K	89.00%
ReResNet-3(1)	73K	86.95%
ReResNet-3(2)	73K	88.17%
ReResNet-3(3)	73K	88.33%
ReResNet-3(4)	74K	**88.39%**
ReResNet-3(5)	74K	88.24%
ReResNet-3(6)	74K	87.95%
ReResNet-3(12)	75K	88.03%

Table 3: Comparison of test accuracy with different recurring times on CIFAR-10.

4.1.4 Comparison with state-of-the-art methods.

We compare ReResNet improved by feature transfer with state-of-the-art methods on CIFAR-10 (see Table 4). ReResNet+FT can be as light as $\sim$120K parameters, but achieve much better results than Circulant cNN [44] with similar network size. Compared with big networks like NIN [45], DSN [46], Highway [47, 48], FitNet [16], Maxout [49] and Prob maxout [50], our proposed method is significantly more efficient. And we also believe that the performance can be further improved by utilizing a better teacher network for feature transfer.

4.2. Experiments on Scenes

4.2.1 Implementation

Scenes is a very challenging dataset for indoor scene classification. It consists of 67 categories, and there are 101 to 738 RGB images per category. By following the standard protocol as in [24], we use 80/20 images per category for training and testing.

As in [51], we resize images to 224$\times$224 pixels. Since the dataset is very small, we conduct transfer learning by extracting features from the last residual group of a pre-trained ResNet-50 [7] on ImageNet [52] and then train a small ResNet/ReResNet on top of that. ResNet-50 uses "bottleneck" design. For each residual block, it has a stack of three convolutional layers (1×1, 3×3, 1×1 convolutions) instead of two (3×3, 3×3 convolutions). The overall architecture of our ResNet/ReResNet on Scenes is shown in Table 5.

network name	params	accuracy
Circulant cNN [44]	120K	84.29%
ResNet [7]	270K	91.25%
ResNet [7]	460K	92.49%
NIN [45]	970K	91.19%
DSN [46]	970K	92.03%
Highway [47, 48]	1.25M	91.20%
FitNet [16]	2.5M	91.61%
Maxout [49]	>5M	90.62%
Prob maxout [50]	>5M	90.61%
ReResNet-1(3-1)+FT	122K	90.29%
ReResNet-1(6-1)+FT	124K	**90.49%**
ReResNet-2(3-1)+FT	219K	**91.10%**
ReResNet-2(6-1)+FT	366K	91.82%

Table 4: Comparison of test accuracy with state-of-the-art methods on CIFAR-10. FT is implemented with 1-channel feature map generation.

The weight decay was set to 0.00001 and momentum was 0.9. We ran experiments for 10,000 iterations with batch size 128. The learning rates were 0.01 for the first 2,000 iterations and 0.001 for the rest. Same weights initialization and loss function were used as CIFAR-10. Here, the network on Scenes is referred to as "ResNet(3-k)/ReResNet-x(3-k)".

name	output size	operation type
conv1*	112×112	$[7 \times 7, 64]$
max-pool*	56×56	$[3 \times 3]$, stride 2
group1*	56×56	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
group2*	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
group3*	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
group4*	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
transition1	7×7	$[3 \times 3, 64 \times k]$, stride 1
group1	7×7	$\begin{bmatrix} 3 \times 3, 64 \times k \\ 3 \times 3, 64 \times k \end{bmatrix} \times 3$
transition2	7×7	$[3 \times 3, 128 \times k]$, stride 1
group2	7×7	$\begin{bmatrix} 3 \times 3, 128 \times k \\ 3 \times 3, 128 \times k \end{bmatrix} \times 3$
avg-pool	1×1	$[7 \times 7]$, stride 1

Table 5: Structure of our ResNet(3-k)/ReResNet-x(3-k) on Scenes. The upper part with * signs in name corresponds to operations of ResNet-50 pre-trained on ImageNet, while the bottom shows the architecture of our transfer learning small ResNet/ReResNet. Network depth factor is always 3, which means at the same spatial scale, each residual group is made up of three residual blocks, and the width is determined by factor k.

4.2.2 ReResNet performance on a small dataset

Results on Scenes are shown in Table 6. The teacher network ResNet(3-4) with accuracy 71.8% is 2× wider than ResNet(3-2)/ReResNet-1(3-2)/ReResNet-2(3-2) and 4× wider than ReResNet-1(3-1)/ReResNet-1(6-1). For ResNet(3-2) we try 6 feature transfer losses (each after per residual block) instead of two which is shown in Figure 3. We see by adding more losses, the test accuracy goes up, from 70.51% to 71.42%.

network name	params	no FT	FT	gains
ResNet(3-2)	7.10M	**69.61%**	70.51%(2) 71.42%(6)	0.90%(2) 1.81%(6)
ReResNet-1(3-2)	4.15M	67.62%	68.73%	1.11%
ReResNet-2(3-2)	4.15M	69.25%	**70.43%**	1.18%
ReResNet-1(3-1)	2.37M	65.22%	67.53%	2.31%
ReResNet-1(6-1)	3.48M	62.83%	66.05%	3.22%

Table 6: Test accuracy on Scenes. "Feature Transfer" is denoted as "FT" for short, and it is conducted with the option "1-channel feature map generation". The values in brackets for FT indicate how many losses we add in $\mathcal{L}_{ft}$. In this experiment, for feature transfer, knowledge is distilled from teacher networks ResNet(3-4) with accuracy 71.80%.

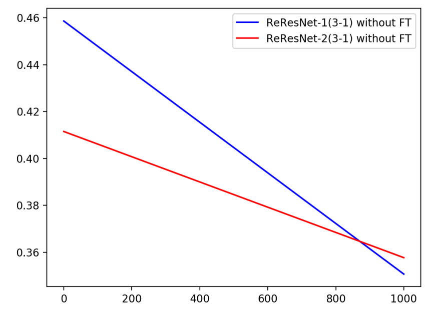

Figure 6: Test accuracy got after the first 1,000 iterations of ReResNet-1(3-1) and ReResNet-2(3-1) on CIFAR-10.

Recurrent loop for ReResNet-1(3-2) and ReResNet-2(3-2) degrades the performance of ReResNet(3-2), but feature transfer introduces some improvements. The gains are about 1.0%. Unlike our observation on CIFAR-10, ReResNet-2 works better than ReResNet-1 with the same number of parameters. To our understanding, this is because sharing both convolutions alternatively in each residual block is more likely to confuse the optimization process on a small dataset. This can be verified by another experiment on CIFAR-10, where we just ran for 1,000 iterations with a batch size 128. The test accuracy plot (Figure 6) shows that when the network doesn't see a lot of training samples, the network is not robust to any possible confusion caused by ReResNet-1. Therefore, in this case, ReResNet-2 classifies the images better.

Interestingly, wider ReResNet-1(3-2) leads to better performance than thinner ReResNet-1(3-1), but deeper

name	outpuht size	operation type
conv1	56×56	$\left[7 \times 7, 32\right]$, stride 2
group1	56×56	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$
transition1	28×28	$\left[3 \times 3, 64\right]$, stride 2
group2	28×28	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 4$
transition2	14×14	$\left[3 \times 3, 128\right]$, stride 2
group3	14×14	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 6$
transition3	7×7	$\left[3 \times 3, 256\right]$, stride 2
group4	7×7	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 3$
avg-pool	1×1	$\left[7 \times 7\right]$, stride 1
fully-connected	1×1	1000d-fc

Table 7: Structure of our ResNet/ReResNet-x on MiniPlaces.

network name	params	top-1	top-5
ResNet	6.07M	47.54%	76.82%
ReResNet-1	1.98M	**47.56%**	**77.42%**

Table 8: Test accuracy on MiniPlaces.

ReResNet-1(6-1) causes an accuracy drop on shallower ReResNet-1(3-1). We believe this also has something to do with the size of dataset. Because of the gradient vanishing problem, deep networks are more difficult to train, but transferring extra knowledge to guide the optimization helps ease the training and improve the performance. From the results of ReResNet-1(3-1) and ReResNet-1(6-1) we see the deeper the network is, the larger gains we could get from feature transfer.

4.3. Experiments on MiniPlaces

4.3.1 Implementation

MiniPlaces is also for scene classification, but unlike Scenes, it has both indoor and outdoor scene images. As a subset of MIT Places2 dataset [25], it consists of RGB images from 100 categories. The training set has 100,000 images and test set has 10,000 images. Initially each image has a size of 128×128, but when we feed images into the network we take a random crop of 112×112. The mean/std normalization is the same as that of CIFAR-10.

Inspired by ResNet-34 trained on ImageNet, we build a ResNet for MiniPlaces. See Table 7 for detailed architecture. Networks were run for 200,000 iterations with a batch size 256. We start with learning rate 0.1, and it was divided by 10 at 69,000 and 160,000 iterations. The network was trained from scratch, with weight decay 0.00001 and momentum 0.9. For this larger dataset, we did not have time to tune the hyperparameters, so we use it only to verify the efficiency of ReResNet-1.

4.3.2 ReResNet performance on a big dataset

Table 8 represents the top-1 and top-5 accuracy of our ReResNet-1 compared with its sequential ResNet counterpart on MiniPlaces. Because larger networks are more likely to have redundant parameters, the performance can benefit from enabling the weights to be better used. In this case, sharing weights so that it becomes recurrent makes more sense. So we see the accuracy of ReResNet-1 is even better than its counterpart, with 67% fewer parameters. We assume that as the dataset becomes larger and the network goes more complicated, sharing ResNet weights will become more efficient.

5. Conclusion

In this paper, we propose two sharing configurations for ReResNets that make better use of trainable parameters, to greatly reduce the network size without degrading the performance too much. When we combine it with feature map knowledge transfer by adding extra intermediate losses, we see the possibility to aid better training and compensate the slight performance decrease caused by the parameter reduction. Adding more recurrent layers may increase the network capacity to achieve better accuracy, but there is a critical value for recurring times at which the networks start to suffer from overfitting and have performance degradation. Over three datasets, CIFAR-10, Scenes and MiniPlaces, with fewer parameters ReResNet improved by Feature Transfer performs well. On CIFAR-10, when ReResNet is trained with the guidance from a powerful network, the architecture can be extremely light with $\sim$120K parameters, but the results are as competitive as that of other much heavier state-of-the-art architectures. We hope these findings will help efficient networks to be embedded in more real life scenarios.

One limitation of our method is that our sharing configurations are too restrictive. We use tied convolutional weights without considering forget gates to control the information passing. Moreover, it is worth trying multiple paths as in [2, 46, 29] by adding feed-forward inputs to the recurrent inputs, to see if shorter paths will help gradient back-propagation during training so as to facilitate the learning process, which could be another future work.

Acknowledgement

The work was supported by Bosch Security Systems. This project aims at making surveillance cameras automatically switch between scenes at a practical level.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[4] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent dbn-hmms. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4688–4691. IEEE, 2011.

[5] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[6] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Nicholas Lane, Sourav Bhattacharya, Akhil Mathur, Claudio Forlivesi, and Fahim Kawsar. Dxtk: Enabling resource-efficient deep learning on mobile and embedded devices with the deepx toolkit. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services, ser. MobiCASE*, volume 16, pages 98–107, 2016.

[9] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*, pages 7–12. ACM, 2015.

[10] Chao Li, Yi Yang, Min Feng, Srimat Chakradhar, and Huiyang Zhou. Optimizing memory efficiency for deep convolutional neural networks on gpus. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 633–644. IEEE, 2016.

[11] Andreas Plieninger, STEUERUNGS-und REGELUNGSTECHNIK, and J Conradt. Deep learning neural networks on mobile platforms. 2016.

[12] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[13] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM, 2006.

[14] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[17] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

[18] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience*, volume 806. Cambridge, MA: MIT Press, 2001.

[19] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Thomas D Albright and Gene R Stoner. Contextual influences on visual processing. *Annual review of neuroscience*, 25(1):339–379, 2002.

[22] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[24] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE, 2009.

[25] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[26] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[28] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013.

[29] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375, 2015.

[30] Amir R Zamir, Te-Lin Wu, Lin Sun, William Shen, Jitendra Malik, and Silvio Savarese. Feedback networks. *arXiv preprint arXiv:1612.09508*, 2016.

[31] Alexandre Boulch. Sharesnet: reducing residual network parameter number by sharing weights. *arXiv preprint arXiv:1702.08782*, 2017.

[32] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[33] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[34] Lorien Y Pratt. *Comparing biases for minimal network construction with back-propagation*, volume 1. Morgan Kaufmann Pub, 1989.

[35] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989.

[36] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *Neural Networks, 1993., IEEE International Conference on*, pages 293–299. IEEE, 1993.

[37] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[38] Jan Van Leeuwen. On the construction of huffman trees. In *ICALP*, pages 382–410, 1976.

[39] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

[40] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[42] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[44] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[45] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[46] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pages 562–570, 2015.

[47] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[48] R K Srivastava, K Greff, and J Schmidhuber. Training very deep networks. In *NIPS*, pages 2377–2385, 2015.

[49] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[50] Jost Tobias Springenberg and Martin Riedmiller. Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*, 2013.

[51] Ammar Mahmood, Mohammed Bennamoun, Senjian An, and Ferdous Sohel. Resfeats: Residual network based features for image classification. *arXiv preprint arXiv:1611.06656*, 2016.

[52] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[54] David Eng and Jenny Hong. Tractable neural networks for identity recognition.

[55] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

[56] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[57] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[58] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[59] Jonathan Shen, Noranart Vesdapunt, Vishnu N Boddeti, and Kris M Kitani. In teacher we trust: Learning compressed models for pedestrian detection. *arXiv preprint arXiv:1612.00478*, 2016.

[60] Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[61] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

A. Visualizations of Feature maps on CIFAR-10 and Scenes

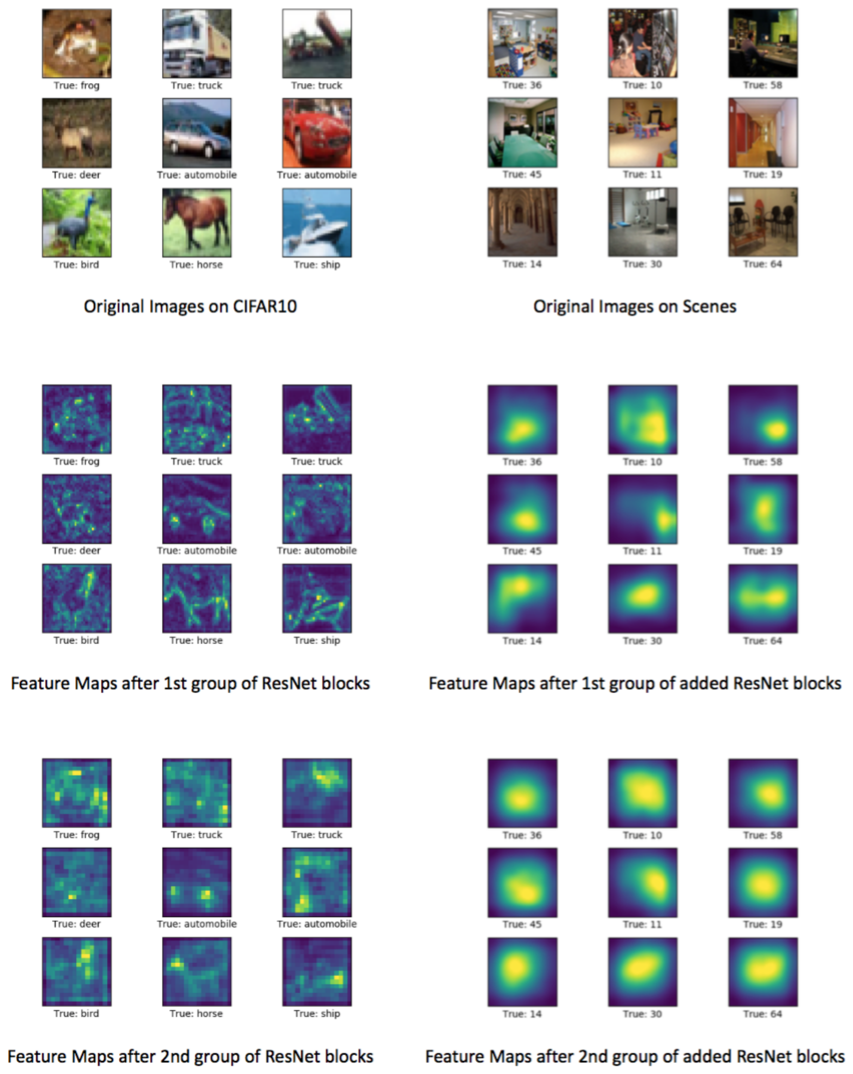
True: frog
True: truck
True: truck
True: deer
True: automobile
True: automobile
True: bird
True: horse
True: ship
Original Images on CIFAR10
True: 36
True: 10
True: 58
True: 45
True: 11
True: 19
True: 14
True: 30
True: 64
Original Images on Scenes
Feature Maps after 1st group of ResNet blocks
Feature Maps after 1st group of added ResNet blocks
Feature Maps after 2nd group of ResNet blocks
Feature Maps after 2nd group of added ResNet blocks

Figure 7: Visualizations of 1-channel feature map on CIFAR-10 and Scenes. Feature maps are got after 1st and 2nd residual groups. On CIFAR-10, we train our ResNet from scratch. The visualizations show that early feature maps have higher activations around the whole object, while latter feature maps focus on higher-level features like the automobile wheels and horse face. On Scenes, we use a pre-trained ResNet50 on ImageNet to extract features right before the last layer, and then add 2 ResNet groups to do transfer learning. Since pre-trained model is used to do object classification on ImageNet, feature maps after the second group of our added ResNet blocks are more background-focused than the feature maps after the first group after transfer learning.

B. Visualizations of Feature Transfer on Scenes

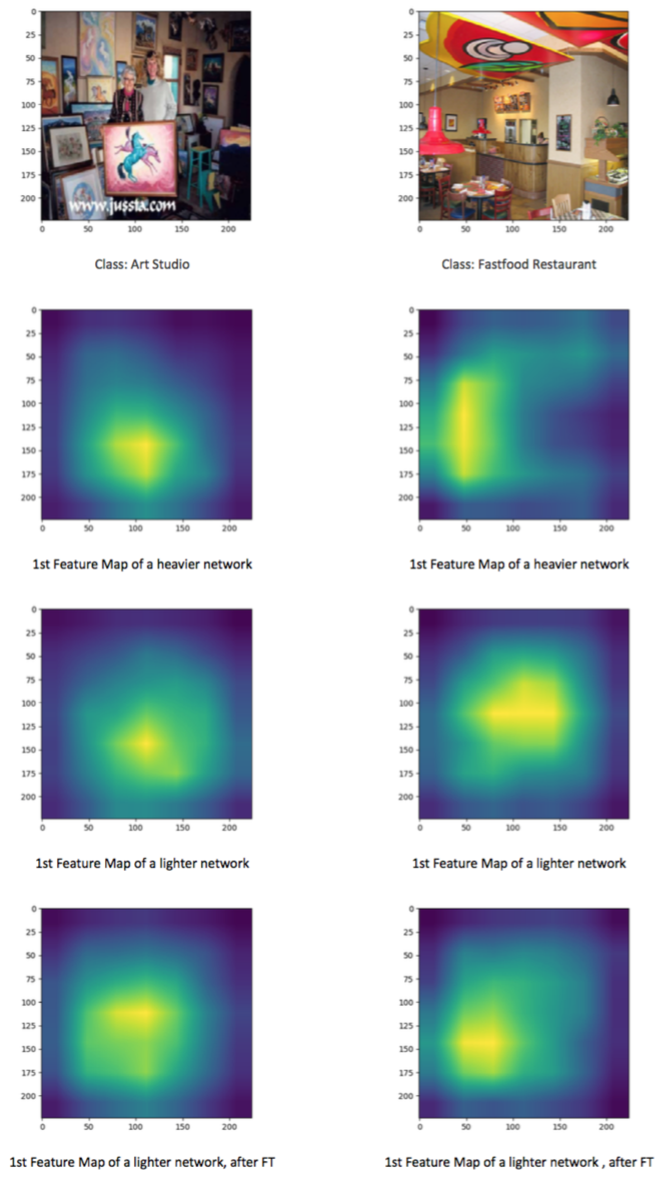

Figure 8: Visualizations of 1-channel feature map transfer on Scenes. The heavier network is the teacher model with 23.64M parameters (accuracy 71.80%), and the lighter one is the student model with 7.10M parameters (accuracy 69.61% and 70.51% after FT). From visualizations, we see feature transfer helps the student's feature maps to become more similar to those of the teacher network.

C. Supplementary Information about Knowledge Transfer

The key idea behind Knowledge Transfer is to train a compact neural network to approximate the function learned by another neural network which is more complex. Here, since the smaller model is learned from the bigger one, the former is called ”student model”, while the latter one is its ”teacher model”. In this section, we will explain some existing knowledge transfer methods that are used for model compression.

- *Ensemble Compression* is used to train a single neural network of modest size to mimic a much larger ensemble, so that with fewer parameters, the student model can reach same accuracy as its ensemble teacher model [13].
- *Train Shallow Networks using Deep Networks* shows how to train a shallow student model to mimic logits got from its deep teacher model. For some tasks where having multiple convolutions is less important, it gives similar accuracy as state-of-the-art networks with the same parameter budget [14].
- *Knowledge Distillation* incorporates a parameter to control the relative importance of teacher model’s accuracy by considering hard labels in loss function, as well as a temperature parameter to make soften labels thus passing targets towards the uniform distribution, and finally it succeeds in training smaller and/or shallower models that are nearly as accurate as its ensemble teacher model [15, 17].
- *FitNet* introduces a hint from the teacher model, and shows that distilled student model which is deeper but thinner, can be more efficient and accurate than its teacher even with low capacity [16, 53, 54].

C.1. Ensemble compression

An ensemble is a collection of networks whose predictions are combined in some way, e.g., bagging, boosting, random forests, Bayesian averaging and stacking. Many ensemble models have excellent performance, but they are too large and slow to use, especially for devices with limited memory storage, computational capability and power consumption in practice.

“Ensemble Compression” with knowledge transfer was first researched by Bucilu *et al.* [13]. Since one big challenge for deep learning is the lack of large labeled dataset and more training samples usually lead to better predictive performance, they first use an ensemble to label a large unlabeled dataset and then train a new student neural network on these newly labeled data to learn the function that is learned by the ensemble teacher model. Although the student models have $1000\times$ fewer parameters, they are surprisingly as accurate as their ensemble teacher models.

C.2. Train shallow networks using deep networks

Ba and Caruna [14] replaced the ensemble teacher model with a deep neural network, and came up with a new idea: if shallow networks are able to learn deep networks while preserving good accuracy with the same number of parameters, then we could say deep network does not really need to be deep. See Figure 9 for the flow chart.

They train deep teacher networks in a conventional way. They use softmax cross-entropy cost function, and logits, the logarithm probability values before the softmax activation, as the shallow student models’ regression targets. This is to say, instead of using softmax output

$$p_k = \frac{e_k^z}{\sum_j e_j^z}. \tag{7}$$

where z is logit value, and p_k is the softmax probability computed for each class k, they train student models directly on logit z, to avoid the information loss while passing through logits to probability space.

While training the shallow network, they minimize the loss function

$$L(W, \beta) = \frac{1}{2T} \sum_t ||g(x^{(t)}; W, \beta) - z^{(t)}||^2. \tag{8}$$

where W and β are the weights between the tth input $x^{(t)}$ to the hidden layer, and the hidden layer to the prediction output $g(x^{(t)}; W, \beta)$). Weights are updated using standard back-propagation and stochastic gradient descent.

One trick here is that, they insert one linear layer between the input and the hidden layer to speed up the mimic learning. In order to make the shallow network have the same number of parameters as the deep network, the shallow one has to have many hidden units, but the training with large weight matrix W to convergence is very slow, which leads to a problem in practice. If D and H are respectively the dimensions of input and hidden units, both the computation time and memory space it needs are proportional to $O(HD)$. However, if we introduce one linear layer with k units between the input and

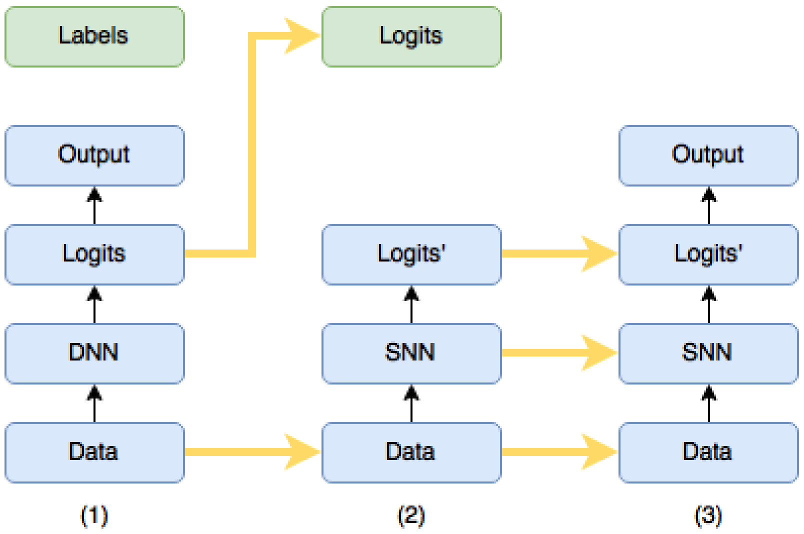

Figure 9: Train one shallow neural network (SNN) using one deep nueral network (DNN) as the teacher: (1) Build a complex model; (2) Train a simple model to mimic teacher logits; (3) Apply it. Green blocks refer to the ground truth used for training, and yellow arrows are identical mappings.

the non-linear hidden layer, we can reduce the complexity from $O(HD)$ to $O(k(H + D))$, which can speed up learning dramatically. Therefore, now the new loss function is

$$L(W, \beta) = \frac{1}{2T} \sum_t ||g(x^{(t)}; UV, \beta) - z^{(t)}||^2. \tag{9}$$

where U and V are the weight matrices separated from a single weight matrix W.

The reason why student models can be more accurate on predicted labels than they trained on original labels is further discussed by them. Apart from the fact that teacher model may eliminate some label errors, we know new soft labels, the logits, are more informative than the original hard 0/1 labels, and are a function only of the available input features. Assume we have three classes: cat, tiger and pig. Then the label of a picture of cat should be [1 0 0], which means each class is totally independent and has no relations between each other. However, the truth is that the similarity between cat and tiger should be higher than that of cat and pig, and this information is not available by using simple 0/1 labels. So the method of using soft labels as regression targets can be seen as a form of regularization which helps avoid overfitting in shallow student models.

The results show that this method works very well on TIMIT speech corpus so that its shallow student model can be as accurate as state-of-the-art deep models with similar number of parameters as state-of-the art, but much fewer parameters than its deep teacher model. See Table 9, SNN-MIMIC-8k is trained with model compression to mimic ECNN, and it performs as well as a DNN with a similar number of parameters.

However, the results on object recognition dataset CIFAR-10 [23] are less convincing since the size of the network has become $30\times$ larger than the deep model, but the accuracy is still several points less than state-of–the-art networks. Urban *et al.* [55] explains the reason, saying that convolution is more important for image classification than it is for TIMIT speech recognition, so given the same parameter budget, the shallow student model can hardly be as accurate as its deep convolutional teacher model with fewer convolutions. Furthermore, the paper [55] concludes that for some problems such as CIFAR-10, although given the same number of parameters, the student model trained with soft targets leads to better accuracy than the model with original hard targets, there is still a huge gap between the convolutional and non-convolutional student models, and multiple (at least 3-4) convolutional layers are needed for better performance.

	Architecture	# Param.	# Hidden units	PER
SNN-8k	8k + dropout trained on original data	∼12M	∼8k	23.1%
SNN-50k	50k + dropout trained on original data	∼100M	∼50k	23.0%
SNN-400k	250L-400k + dropout trained on original data	∼180M	∼400k	23.6%
DNN	2k-2k-2k + dropout trained on original data	∼12M	∼6k	21.9%
CNN	c-p-2k-2k-2k + dropout trained on original data	∼13M	∼10k	**19.5%**
ECNN	ensemble of 9 CNNs	∼125M	∼90k	**18.5%**
SNN-MIMIC-8k	250L-8k no convolution or pooling layers	∼12M	∼8k	**21.6%**
SNN-MIMIC-400k	250L-400k no convolution or pooling layers	∼180M	∼400k	**20.0%**

Table 9: Train Shallow Networks using Deep Networks: Comparison of shallow and deep models with phone error rate(PER) on TIMIT core test set [14].

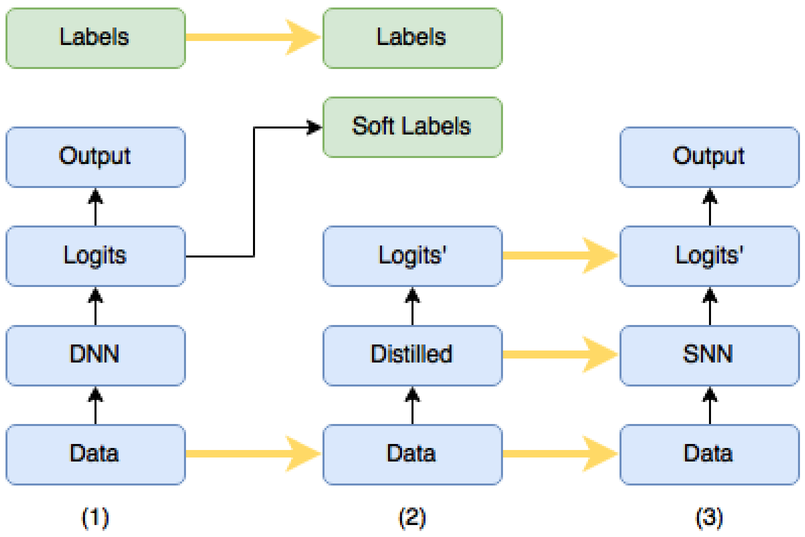

Figure 10: Knowledge Distillation: (1) Build a complex ensemble of deep networks and get soft labels with high temperature; (2) Train a simple distilled model on soft labels with the same high temperature and hard labels with temperature 1; (3) Apply it with temperature 1. Green blocks refer to the ground truth used for training, and yellow arrows are identical mappings.

C.3. Knowledge Distillation

In order to improve deep learning performance, people like to train many different models on the same data and then average the predictions [56]. However, due to its cumbersome architecture it is not suitable for deployment. Known from Ensemble Compression, it's possible to transfer a large ensemble of models to a single small model. Therefore, Hinton *et al.* [15] further developed the approach descried in Sec. C.2 using a different compression technique, called Knowledge Distillation (KD). See Figure 10 for the flow chart.

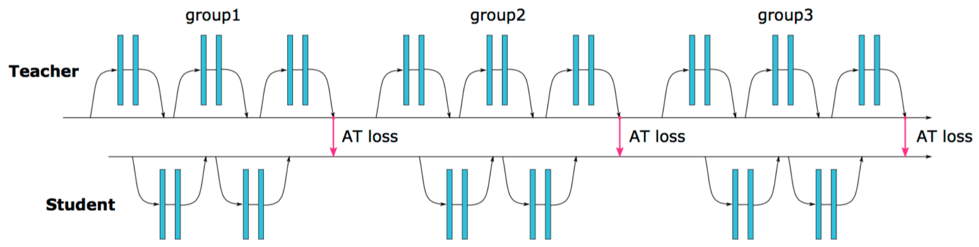

Figure 11: Knowledge Distillation: Schematics of teacher-student attention transfer for the case when both networks are residual, and the teacher is deeper [17].

After adjusting the softmax function described above by introducing "temperature" T, the new softmax output turns into

$$p_k = \frac{e_k^{z/T}}{\sum_j e_j^{z/T}}. \tag{10}$$

where z is logit value, T is called "temperature", and p_k is the softmax probability computed for each class k. In the case of ensemble compression, the soft targets are set with the arithmetic or geometric mean of all deep teacher networks predictive distributions in the ensemble. Normally for the standard softmax, the temperature is set to 1. While raising its value, we get a softer probability distribution over classes, with which we will get more information per training case than hard targets, and less variance in the gradient between training cases. In that way, we are able to use less training data and much higher learning rate. For example, if the output of the neural network is [1 2 3], with standard softmax the probability distribution is [0.09 0.24 0.67] which is closer to one-hot code, and when $T = 4$ we get [0.25 0.33 0.43] which is softer and better as a regression target. However, temperature T cannot be extremely large, because if so, three classes tend to have equal probability, which makes three labels indistinguishable and will definitely lead to bad performance.

The process of the simplest form of knowledge distillation is as follows. We first train a deep teacher ensemble model with a high temperature in its softmax until it produces suitably soft probability labels, and then use the same high temperature when training its distilled student model to match the regression targets. When we apply the student model for testing or inference in practice, we use the temperature of 1.

A more complex and better performing form of distillation is applied when we know the correct labels of the dataset. We can train the distilled model to produce correct labels by weighted averaging the functions of the cross-entropy with its soft targets, and the cross-entropy with the correct labels. Thus, it allows its student model to capture not only the information of true labels, but also the fine structure over all classes, and it makes the teacher penalize the training examples according to its confidence. The loss function can be written as

$$L_{KD} = \alpha L_{soft} + (1 - \alpha) L_{hard}. \tag{11}$$

where α is a regularization that enforces the student model to learn more from its teacher than from the true labels, L_{KD}, L_{soft} and L_{hard} are respectively the loss functions we finally want with Knowledge Distillation, only with soft labels' guidance, and only with hard labels' guidance. Empirically, we know a lower weight applied on hard targets will lead to better performance.

The result of KD shows that it compresses an ensemble of deep teacher networks into a simple student network of similar depth, but works significantly better than when it is trained directly from the same training data. One problem here is that, while increasing the depth of the student network, it still has difficulty in optimizing deep nets.

Based on Knowledge Distillation, Zagoruyko *et al.* [17] recently proposed a method to improve the performance of a student convolutional neural network by forcing it to mimic the teacher model's attention maps. The illustration for the case when both teacher and student networks are residual and the teacher is deeper is shown in Figure 11. The experiments were conducted on three teacher-student pairs: WRN-16-2/WRN-16-1 (networks have 16 residual layers and teacher model is $2\times$ wider than the student model), WRN-16-1/WRN-40-1 and WRN-16-2/WRN-40-2 (both networks are of the same width but teacher model is deeper than its student). WRN is a new introduced Wide Residual Network described in [57]. Apart from the standard cross entropy with hard targets and soft label loss controlled by parameter temperature T, this new method also

student	teacher	student	AT	F-ActT	KD	AT+KD	teacher
NIN-thin, 0.2M	NIN-wide, 1M	9.38	8.93	9.05	8.55	8.33	7.28
WRN-16-1, 0.2M	WRN-16-2, 0.7M	8.77	7.93	8.51	7.41	7.51	6.31
WRN-16-1, 0.2M	WRN-40-1, 0.6M	8.77	8.25	8.62	8.39	8.01	6.58
WRN-16-2, 0.7M	WRN-40-2, 2.2M	6.31	5.85	6.24	6.08	5.71	5.23

Table 10: Knowledge Distillation: Activation-based attention transfer(AT) with various architectures on CIFAR-10. Error is computed as median of 5 runs with different seed. F-ActT means full-activation transfer [17].

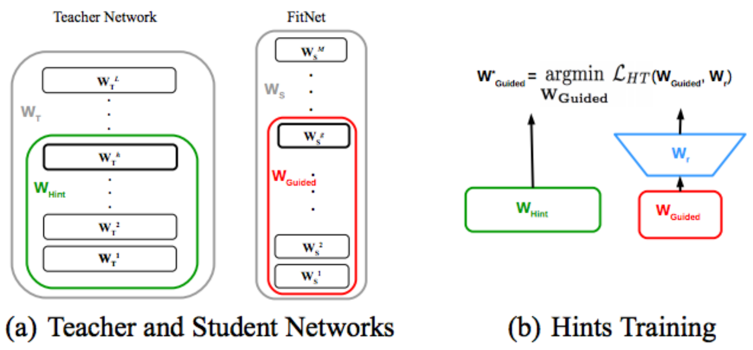

Figure 12: FitNets: Training a student network using hints [16].

takes the attention map loss into account. For the activation layer pairs that we want to transfer attention maps, we calculate the distance and sum all values up as the loss function. For networks with different depth as shown in Figure 11, we extract the attention maps on output activations of each group of residual blocks; for those with same depth we just have the attention transfer layer after every residual block. The results in Table 10 show that Knowledge Distillation combined with Attention Transfer is able to give us smaller error rates.

C.4. FitNets: thin and deep networks

Up till then, all previous work tried to train a student neural network of similar width and depth e.g., Knowledge Distillation, or into shallower but wider ones e.g., shallow networks trained by deep networks. Depth has been verified to be very important [58], because deep representations are more complex thus being more expressive than shallow networks, but neither of the methods explained above takes advantage of depth. The paper written by Romero *et al.* [16] extends the approach of Knowledge Distillation, and allows the training of thinner and deeper student models. The idea comes from the guess that making networks deeper allows the model to generalize better, while having thinner models help us reduce the computational burden a lot.

FitNet compresses teacher models which are shallower and wider into their thinner and deeper student networks. In order to make it easier to train deep networks and guide the student optimization procedure, FitNets introduces hints from the teacher model, making a hint/guided layer pair and wanting the guided layer in student model to be able to predict the corresponding hint layer in its teacher network. The pair is chosen to be the middle layers of both models.

FitNet separates its training process into two, hint-based training(HT) and normal KD training. In HT, since there are more output units in teacher's hint layer, they add a regressor to the student's guided layer to match the shape of both. Apart from the guided weights, this mapping also needs to be trained by minimizing the following loss function.

$$L_{HT}(W_{Guided}, W_{Regression}) = \frac{1}{2}||u_h(x; W_{Hint}) - r(v_g(x; W_{Guided}); W_{Regression})||^2. \tag{12}$$

where u_h and v_g are the function representations up to hint/guided layers with weight matrices W_{Hint} and W_{Guided}, and r is the regression function upon the guided layer in student model with parameters $W_{Regression}$. This step produces strong initialization for the first half of the student network. One trick applied here is, instead of fully-connected regressor they use a

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

Table 11: FitNets: Accuracy/Speed Trade-off on CIFAR-10 [16].

convolutional regressor to reduce the number of parameters and memory consumption. The work flow of hint-based training is shown in Figure 12(a) and (b).

The second training stage, normal KD training works as follows. First we get pre-trained FitNet parameters W_{Guided} and $W_{Regression}$ after minimizing the loss function $L_{HT}(W_{Guided}, W_{Regression})$, then we follow the process of Knowledge Distillation to minimize L_{KD} as discussed in Sec C.3.

The Table 11 measures FitNets efficiency, showing that FitNet is significantly faster while matching or outperforming its teacher model, even when having fewer parameters and computations. And that is a very surprising result.

Inspired by FitNets, for a specific pedestrian detection task on Caltech Pedestrian Dataset, Shen *et al.* [59] uses ResNet-200 [53] as the teacher model and unmodified ResNet-18, ResNet-18-thin which cuts half of the number of channels for every layer, and ResNet-18-Small which fixes the channels of every layer at 32 as student models [53]. The result shows that by introducing hint layer and the confidence of teacher model by adding cross-entropy with correct labels, the student model performs $8\times$ faster than its teacher with $21\times$ less memory usage, while only observing a drop in log-average error rate of 4.9%.

Additionally, for another Identity Recognition task, Eng *et al.* [54] slightly adjusts the loss function by adding a particular loss just for identity recognition, called ”Triplet Embedding Loss”. They use Labeled Faces in the Wild dataset [60], and its student models are all much smaller than the original teacher FaceNet NN4 Small2 Model [61] which contains around 5M parameters. As a result, they significantly reduce training time and memory usage with very little loss in training accuracy.