Novel Approach for DNN Model Compression
Software Internship,
WS 2017-18, University of Heidelberg

Ashis Ravindran
Master Scientific Computing
ashis.ravindran@stud.uni-heidelberg.de

Enrique Fita Sanmartin
Master Scientific Computing
fita@stud.uni-heidelberg.de

July 31, 2018

Abstract

Its well known that training a deep neural network requires a huge amount of computational resources, whereas in the real-world case, the trained cumbersome model would drastically hinder the performance in the deployment stage in terms of computation and speed, despite its proven decent accuracy. Thus it is important to efficiently distil the knowledge from the large model (teacher) to get a smaller one (student) not trading too much on the original accuracy. Inspired by [1], this project ponders over the possibilities of inter-network training. We compare how different configurations improve the transfer of knowledge (a.k.a Dark Knowledge) from ensemble/teacher model to smaller/student model and propose a new configuration for inter-model training called the Pyramid. The result shows that the newly proposed configuration (henceforth: The Pyramid) achieves the state of art classification accuracy for a compressed, smaller, model. All experiments were carried out using VGG19 & VGG19 based sub-architectures, trained using STL10 & CIFAR100. The newly derived model using the proposed approach has 10x fewer parameters and 4x faster than VGG19.

1 Introduction

Since 2012 ImageNet challenge won by AlexNet, the focus on image intelligence/ computer vision has shifted to Deep Learning. From every year then on, newly introduced architectures got deeper and deeper, steadily increasing the performance on the datasets. As deep neural networks aka Convolutional neural networks are essentially matrix operations for feature extraction, more the deeper the architecture got, more the number of parameters it had. This meant alarmingly high levels of computational resource requirement to train and test the model. For eg. VGG19 has 30M trainable parameters, ResNet 18 has 11M trainable parameters etc. More parameters mean more floating point operations as part of forward pass and backpropagation. Constant improvements in GPU computing by Nvidia has leveraged a lot of this bottleneck however the target application of these trained models lies in a lightweight, portable environments like embedded chips in mobile phones, cameras, laptops, where deploying a bulky model have its limitation on memory footprint as well as floating point operations.

The mobility of the future lies in autonomous vehicles, aka self-driving cars. Environment understanding using a camera is a critical part of autonomous driving whether its lane recognition task, object classification task, segmentation tasks to tasks understanding the drivers mood and focus. With the inception of deep learning, lots of these challenges can be (or, are already) solved well with high accuracy,

leading to less human errors and accidents. Thus deep learning techniques have virtually risen to promising one solution for all status, however, leaving behind one critical aspect- models deployability. Excessive memory footprint and computational requirements demand costly hardware for online performance. Thus its imperative that we make models smaller in terms of the number of parameters or faster in terms of test time performance, both meaning the same.

Here with this project, we propose a set of training schemes, which rely on transfer learning techniques, in compressing a neural network. We call this training strategy The Pyramid and explore variations of it viz. Forward scheme and Backward scheme, and compare results.

2 Previous Work

One of the first well documented efforts in model compression dates back to pre deep learning era, in 2006 by Rich Caruana,in [1], where the problem of compressing networks was addressed. This was achieved by first testing the pre trained model using a test set (unlabeled data), followed by training a smaller network using the same data and predictions for the data from the pre trained model.
In the deep learning era, distilling of the knowledge in an CNN was tried out by Hinton in the seminal work in [2] which made a smaller network more efficient. There introduced the concept of dark knowledge which helped a smaller network to perform better when trained using a bigger teacher (pre-trained) network, a process which is named distillation. Such a trained small network is called distilled model. Here, instead of using hard targets to train 'student', they used softened target predictions from soft max layer of the 'teacher' network.

$$out = \frac{\exp(z_i/T)}{\sum \exp(z_j/T)}$$

where T is a temperature. The backpropagation in the student CNN was done by matching outputs from teacher's FC layers by MSE loss. Hence the overall training of the 'student' was accomplished using MSE loss of soft targets from 'teacher' NN and Cross entropy loss from hard target labels of the dataset. Extension of this approach was attempted Romero et al., 2014 [5] by also matching inner layers using MSE loss in the overall training, called 'FitNets'. Again another variation was introduced by Junho et al., 2017 [3] by matching Gram matrices created between intermediate layers of 'teacher' and 'student' network. Matching was accomplished by MSE loss between the Gram matrices. Parallely, there were efforts in compressing deep neural networks by pruning techniques as in [4] which was about rooting out the least activated neurons(filters) and then removing them from the model. Pruning techniques are largely mathematical and are less effective than the transfer learning ones like [3].
Nevertheless, pitching on these works, we propose a transfer learning training/distilling strategy called 'The Pyramid' for more effective compression, do comparative study with the above mentioned works and present the results.

3 The Pyramid Statergy

The core idea of *the Pyramid* is that when given a neural network X, in this case: VGG19, we create intermediate neural networks $X_1, X_2, X_3, \ldots, X_n$, each smaller or shallower than the previous one. Here we create smaller sub models from VGG19, termed as 'SVGG17','SVGG14','SVGG11','SVGG8' & 'SVGG5'. For eg. 'SVGG14' model has 13 convolutional layers & 1 FC layer. (More on sub models in coming sections)
We train these models contiguously, where the big model provides the extra 'dark knowledge' to the small model in the hierarchy. We propose the pyramid configuration in 2 forms viz. **Forward and Backward**.

3.1 Forward Pyramid

Given a neural network X, we will create intermediate neural networks $X_1, X_2, X_3, \ldots, X_n$. These intermediate neural networks will form the pyramid like configuration, since each neural network will be shallower than the previous one. Each neural network will learn from the preceding neural network. The layers of the neural networks will be classified in 3 blocks:

- **Training layers**: These layers will be formed by the initial layers of the previous level of the pyramid. The layers will be initialized with the weights of the preceding level. A subset of these layers, located at the end of this block of layers, will be compressed in the next stage. This subset will be called the **compressing layers**.
- **Compressed layers**: These layers will be the ones that will represent the compressing layers of the previous stage. This block must contain less layers than the compressing layers' block, and the input and output sizes of this block of layers must match the input and output sizes of the compressing layers.
- **Frozen layers**: Formed by the layers of the previous levels which has been already compressed and trained. As in the training layers, it also inherits the initialization. These layers will not be trained anymore.

We work over a fine-tuned VGG19. The levels of the pyramid (sub models derived from VGG19) are as follows:

- **SVGG17**: Inherits the architecture from VGG19, except the fully connected layers of VGG19, that are compressed into one fully connected layer (compressed layer) as in figure 1.
- **SVGG14**: Inherits the architecture from SVGG17 but compresses the block formed by the last 4 convolutional layers of SVGG17 (compressing layers) into one (compressed layer) as in figure 1.
- **SVGG11**: Inherits the architecture from SVGG14 but compresses the block formed by the last 4 convolutional layers of the training layers of SVGG14 (compressing layers) into one (compressed layers) as in figure 1.
- **SVGG8**: Inherits the architecture from SVGG11 but compresses the block formed by the last 4 convolutional layers of the training layers of SVGG11 (compressing layers) into one (compressed layers) as in figure 1.
- **SVGG5**: Inherits the architecture from SVGG8 but compresses the block formed by the last 4 convolutional layers of the training layers of SVGG8 (compressing layers) into one (compressed layers) as in figure 1.

3.2 Backward Pyramid

The backward pyramid follows the same strategy as the forward pyramid. We also reduce the size of the original architecture, X, gradually by creating intermediate neural networks $X = X_0, X_1, X_2, X_3, \ldots, X_n$ each one shallower that the preceding one. Each neural network will learn from the preceding neural network as is done in the forward pyramid. The distinction of the backward and forward pyramid style lies in the order of compression of the layers. In the forward pyramid we divide the layers of the original network in different disjoint sets. Each of these sets will be compressed in a level of the pyramid. The same process is followed in the backward pyramid, but while in the forward pyramid the order of compression is from the last sets of layers to the first set of layers, in the backward style the order is backwards, i.e. the order of compression goes from the first set of layers to the last set of layers (Figure 2).

The backward pyramid style is divided in the same number of levels of the forward pyramid, but the architecture of the intermediate levels has been modified since the order of the layers that are compressed is modified. The architectures that remain the same are the first level, (the original neural network), and the last level.

- **SVGG17**: Inherits the architecture from VGG19, except the initial convolutional layers of VGG19, that are compressed into one convolutional layer (compressed layer) as in the figure 2.
- **SVGG14**: Inherits the architecture from SVGG17 but compresses the block formed by the initial 4 convolutional layers of SVGG17 (compressing layers) into one (compressed layer) as in the figure 2.
- **SVGG11**: Inherits the architecture from SVGG14 but compresses the block formed by the initial 4 convolutional layers of the training layers of SVGG14 (compressing layers) into one (compressed layers) as in the figure 2.
- **SVGG8**: Inherits the architecture from SVGG11 but compresses the block formed by the initial 4 convolutional layers of the training layers of SVGG11 (compressing layers) into one (compressed layers) as in the figure 2.
- **SVGG5**: Inherits the architecture from SVGG8 but compresses the block formed by the initial 4 convolutional layers of the training layers of SVGG8 (compressing layers) into one (compressed layers) as in the figure 2.

4 Training

In order to train the neural network X_i, we perform a two stages process:

1. **First Stage**: We consider X_i and the preceding network X_{i-1} which serves as a teacher of X_i. In this stage takes place the transfer of knowledge from X_{i-1} to X_i. Let D denote the set of the layers of X_{i-1} that are being compressed from level $i-1$ to level i of the pyramid and C the set of the layers of X_i that will represent the set D in level X_i, i.e. the set D after performing the compression. X_i inherits the parameters of X_{i-1} for the layers that have the same structure. That is all the layers except the ones that are in C, since they represent the layers of D which have a different structure. In the first stage we transfer the knowledge of X_{i-1} to X_i by minimizing some loss, L_1, that depends on the outputs of C and D, (O_C and O_D) over the parameters Θ_i of network X_i

$$\min_{\Theta_i} L_1(O_C, O_D; \Theta_i).$$

   Notice that since only the outputs of C are being considered in L_1 the parameters of the layers that are after C are independent of this loss. Therefore this minimization only affect to the parameters Θ_i of X_i that precede C, including them too. Let's denote them $\Theta_i^{\leq C}$

$$\min_{\Theta_i} L_1(O_C, O_D; \Theta_i) = \min_{\Theta_i^{\leq C}} L_1(O_C, O_D; \Theta_i^{\leq C}).$$

2. **Second Stage**: The second stage trains X_i as is usually done. It minimizes the cross-entropy between the output of X_i, o_{X_i} and the real labels t:

$$\min_{\Theta_i} L_2(O_{X_i}, t; \Theta_i).$$

   We can interpret the first stage as a good initialization of the parameters of the parameters of X_i in order to train the network in the second stage.

4.1 Information transferred. Loss L_1

We used different two different versions of the L_1 loss:

1. **L2-loss**[1]: We base on [2] and [5] that transfer information by minimizing the L2-loss between the outputs of different layers of the teacher network and student network. In our case, we perform the L2-loss between the output of the last layer of the set D of the network X_{i-1}, O_D, and the last layer of the output of the last layer of the set C of X_i, O_C

$$L_1(O_C, O_D; \Theta_i^{\leq C}) = ||O_C - O_D||_2^2.$$

2. **Gram-loss**: In this case we base on the paper [3]. Given the outputs of two different layers $F^1 \in \mathbb{R}^{h \times w \times m}$ and $F^2 \in \mathbb{R}^{h \times w \times n}$, where h and w represent the height and the width, which must be the same, and m and n are the depths respectively. They define the Gram matrix G^{F^1,F^2} of these two outputs as:

$$G_{i,j}^{F^1,F^2} = \sum_{s=1}^{h} \sum_{t=1}^{w} \frac{F_{s,t,i}^1 \cdot F_{s,t,j}^2}{h \cdot w}.$$

   In the paper they minimize the L2-distance of the Gram matrices between the student and teacher network at different points of the networks. In our setting we will define the Gram matrix of the network X_{i-1}, G^{i-1} as the one that is created by the outputs of the first and last layer of the block D. Analogously, we define Gram matrix of the network X_i, G^i as the one that is created by the outputs of the first and last layer of the block C. Then

$$L_1(O_C, O_D; \Theta_i^{\leq C}) = ||G^{i-1} - G^i||_2^2.$$

   This loss is more complex than the simple L2-loss considered in the first stage, since we are minimizing the distance of the Gram matrices. The Gram matrices carry more information since they measure how much the features change from layer to layer.

[1] Referred as Pyramid 2 stages in the results

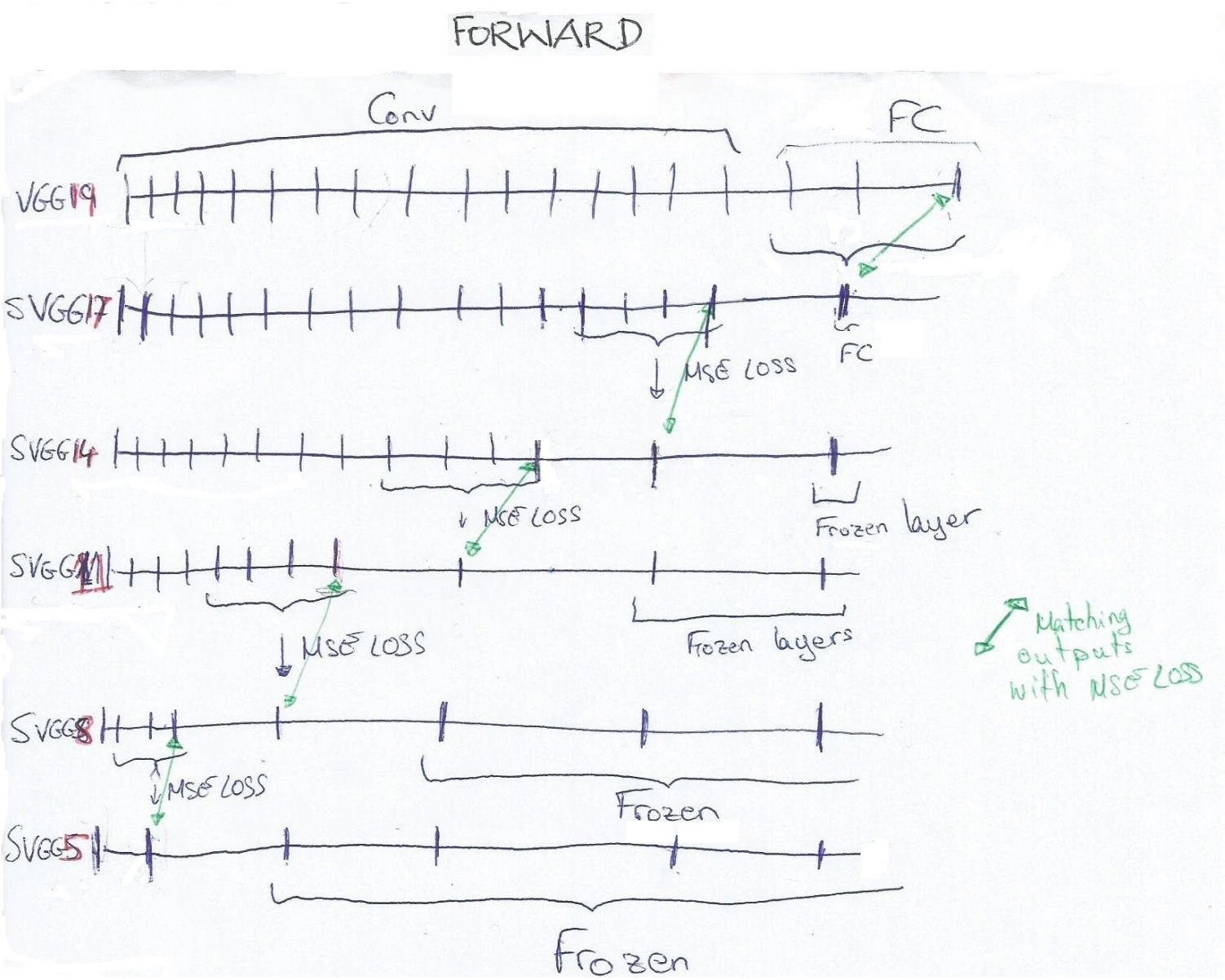


Figure 1: Forward Scheme (FC=Fully connected layers, Conv=Convolutional layers)

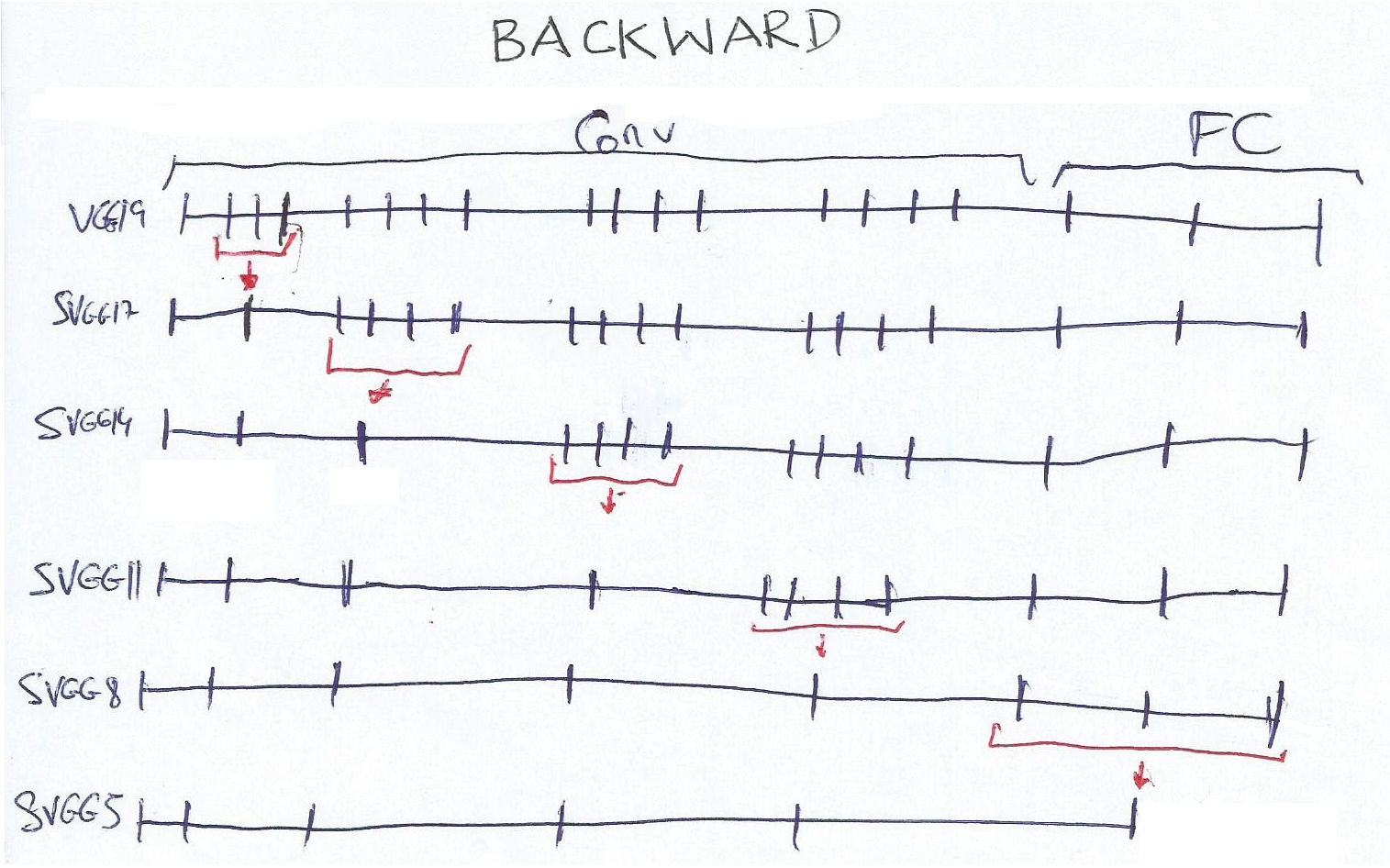


Figure 2: Backward Scheme (FC=Fully connected layers, Conv=Convolutional layers)

5 Experiements and Results

We have tested the proposed methods on the datasets STL10 and CIFAR100 by means of the VGG19 & VGG19 based sub architectures introduced previously. We compare the accuracies obtained by our methods with Fitnet [5] and Gram Direct [3]. We also compared them with the accuracy obtained by the sub architectures of VGG19 if they were just finetuned from VGG19, i.e. use the same weights of VGG19 for the layers that have the same size and train the network only using the cross entropy loss (referred as Scratch in the results tables).

STL10 and CIFAR100 are image recognition datasets with 10 and 100 classes respectively. The images of STL10 are 96x96 pixels and there are 500 training images and 800 test images per class. The images of CIFAR100 are 32x32 pixels and there are 500 training images and 100 test images per class. For CIFAR100 we increased the size of the image to 96x96 pixels. We noticed that shallower networks outperformed the deeper networks just being trained with CIFAR100 by scratch. We associate this issue to the fact that the network was too deep for an image so small. By doing this size augmentation we avoid the anomaly that we suffered by training the shallower networks with CIFAR100.

Tables 1, 2, 3 and 4 sum up the results obtained. Notice that for the methods that are related with the Gram matrix they do not have accuracy for SVGG5 in the backward case, since the transfer knowledge occurs in the fully connected layer. For the fully connected layer we have not defined any Gram matrix, therefore we could not perform any further compression. Analogously occurs for SVGG17 in the forward method. In this case we take as initial level of the pyramid scheme the architecture of SVGG17 with the weights of other method (PYRAMID 2 STAGES weights), and carry on the compression with the corresponding Gramwise method.

We observe the same patterns on both datasets. For the forward pyramid the Gram version gives the best results, while for the Backward pyramid the pyramid 2 stages gives the best results. We observe that the forward pyramid performs better for the earlier levels but from SVGG8 on the performance reduce considerably. On the other hand, the Backward pyramid has the opposite tendency. The last levels of the pyramid (SVGG8 and SVGG5) perform better.

This behaviour can be explained by the fact that in the forward pyramid for the first levels more layers are trained in the first stage, which means that it has more freedom to adapt its parameters to mimic the teacher network of that level. In the later levels, the number of layers trained in the first stage is reduced and its performance also reduces. In contrast to the forward pyramid, in the backward pyramid the first levels have less layers trained on the first stage and the last levels have more, changing the tendency of the results. Furthermore, the first layers are the ones in charge of obtaining the first features. In the backward pyramid, these layers are being forced to be retrained. Since the student network is not able to mimic perfectly the output of the first layers of the teacher network because there are less layers trained (less freedom to copy), the initial features are not so good and the performance is reduced. Surprisingly, in the last levels, the backward pyramid improves the results of the previous levels and even the forward pyramid.

In this sense the backward pyramid is more effective since it needs less computation to be trained since the number of layers trained in the backward pyramid is less than the ones trained on the forward pyramid. Furthermore, for the backward pyramid, the pyramid 2 stages gives the better result which since it is a simpler loss than the Gram loss it is faster to calculate. What we can not explain is why the Gram loss does not perform so good with the Backward pyramid, if this loss carries more information and is more complex.

Figure 3 shows the time performance of every level of the pyramid. We contrast the size of the network with respect to the Wall Clock Time taken for classify 13000 96x96 pixel images. We are able to make VGG19 up to 4 times faster. In general we see that the intermediate levels of the backward scheme are faster than the ones of the forward scheme, which implies that the convolutional layers are most costly than the fully connected layers in the case of VGG19.

We have proposed two different distilling technique, Forward pyramid and Backward pyramid, that perform by means of a gradual compression of a neural network performed in two stages. We have also proposed different loss functions for the distillation stage and we have compared them with state of the art methods. The Forward pyramid is able to compress better for the first levels of the compression (SVGG17, SVGG14,SVGG11), while the Backward pyramid achieves a better efficiency in terms of accuracy for the last levels of the pyramid (SVGG8, SVGG5). The result shows that the newly proposed configuration (henceforth: The Pyramid) achieves the state of art classification accuracy for a compressed, smaller, model. The newly derived model using the proposed approach has 10x fewer parameters and 4x faster than VGG19.

Table 1: **FORWARD ARCHITECTURE ACCURACIES STL10**

Architecture	Scratch	Pyramid 2-stages	Pyramid Gram Normal	Fitnet	Gram Direct
VGG19	**94.175**				
SVGG17	93.925	**93.925**	(same as pyramid 2-stage)	91.3375	(same as pyramid 2-stage)
SVGG14	92.2875	**93.8625**	**93.775**	92.025	93.7875
SVGG11	89.95	91.5875	**92.5125**	90.1	92.0375
SVGG8	82.9125	88.05	**90.125**	85.7125	88.9125
SVGG5	73.76	84.375	**87.0625**	84.93	76.3125

Table 2: **FORWARD ARCHITECTURE ACCURACIES CIFAR100**

Architecture	Scratch	Pyramid 2-stages	Pyramid Gram Normal	Fitnet	Gram Direct
VGG19	80.66				
SVGG17	80.61	**80.98**	(same as pyramid 2-stage)	75.61	(same as pyramid 2-stage)
SVGG14	80.12	79.86	**80.16**	77.89	79.47
SVGG11	75.11	75.15	**78.47**	75.39	76.23
SVGG8	68.92	72.68	**74.71**	70.32	74.56
SVGG5	61.24	69.72	**72.18**	70.31	67.67

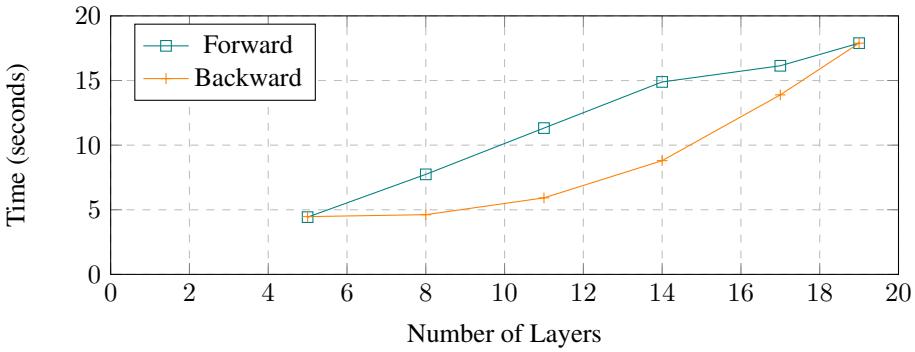
Table 3: **BACKWARD ARCHITECTURE ACCURACIES STL10**

Architecture	Scratch	Pyramid 2-stages	Pyramid Gram Normal	Fitnet	Gram Direct
VGG19	94.175				
SVGG17	81.25	**93.35**	91.9125	90.475	82.3625
SVGG14	72.912	**91.95**	90.075	89.1	68.925
SVGG11	72.15	**90.9375**	87.925	87.25	69.2375
SVGG8	72	**90.85**	90.125	89.1	69.5125
SVGG5	72.8	**91.0125**		85.8375	

Table 4: **BACKWARD ARCHITECTURE ACCURACIES CIFAR100**

Architecture	Scratch	Pyramid 2-stages	Pyramid Gram Normal	Fitnet	Gram Direct
VGG19	80.66				
SVGG17	68.96	**80.24**	78.95	76.65	76.37
SVGG14	63.89	**78.98**	77.78	76.48	65.42
SVGG11	63.16	**76.18**	74.28	75.57	65.71
SVGG8	60.18	**74.69**	74.27	73.42	71.4
SVGG5	61.56	**74.71**		69.41	

Figure 3: **TIME PERFORMANCE**:
CNN Size w.r.t. Wall Clock Time(13000 [96x96] images)



References

[1] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA, 2006. ACM.

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

[3] Yim Junho, Joo Donggyu, Bae Jihoon, and Kim Junmo. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *CVPR*, 2017.

[4] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

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