

# Let's keep it simple, Using simple architectures to outperform deeper and more complex architectures

**Seyyed Hossein Hasanpour<sup>1</sup>**

Islamic Azad University, Science and  
Research branch, Iran  
St.h.hasanpour@iauamol.ac.ir

**Mohammad Rouhani**

Computer Vision Researcher, Technicolor R&I,  
Rennes, France  
Mohammad.Rouhani@technicolor.com

**Mohsen Fayyaz**

Deep Learning Researcher, Sensifai,  
Belgium  
Fayyaz@sensifai.com

**Mohammad Sabokrou**

Institute for Research in Fundamental Sciences  
(IPM), Iran  
Sabokro@ipm.ir

## **Abstract:**

Major winning Convolutional Neural Networks (CNNs), such as AlexNet, VGGNet, ResNet, GoogleNet, include tens to hundreds of millions of parameters, which impose considerable computation and memory overhead. This limits their practical use for training, optimization and memory efficiency. On the contrary, light-weight architectures, being proposed to address this issue, mainly suffer from low accuracy. These inefficiencies mostly stem from following an ad hoc procedure. We propose a simple architecture, called SimpleNet, based on a set of designing principles and we empirically show that SimpleNet provides a good tradeoff between the computation/memory efficiency and the accuracy. Our simple 13-layer architecture outperforms most of the deeper and complex architectures to date such as VGGNet, ResNet, and GoogleNet on several well-known benchmarks while having 2 to 25 times fewer number of parameters and operations. This makes it very handy for embedded system or system with computational and memory limitations. We achieved state-of-the-art result on CIFAR10 outperforming several heavier architectures, near state of the art on MNIST and competitive results on CIFAR100 and SVHN. Models are made available at: <https://github.com/Coderx7/SimpleNet>

**Keywords:** Deep learning, Convolutional neural network, classification

## **1 Introduction**

Since the resurgence of neural networks, deep learning methods have been gaining huge success in diverse fields of applications, amongst which, semantic segmentation, classification, object detection, image annotation and natural language processing are few to mention [1]. Convolutional Neural Network (CNN), as a powerful tool for representation learning, is able to discover complex structure in the given data and represent them in a hierarchical manner [2-4]. Then, there are fewer

---

<sup>1</sup> Corresponding author

parameters to be manually engineered, among which is the network architecture. What all of the recent architectures have in common is the increasing depth and complexity of the network that provides better accuracy for the aforementioned tasks. The winner of the ImageNet Large Scale Visual Recognition Competition 2015 (ILSVRC) [5] has achieved its success using a very deep architecture of 152 layers [4]. The runner up also deploys a deep architecture of 22 layers [3]. This trend has proved useful in the natural language processing benchmarks as well [6].

As networks get deeper, aiming to improve their discriminative power, the computations and memory usage cost and overhead get critically expensive, which has a negative effect on their applications and expansions. Despite the existence of various techniques for improving the learning algorithm, such as different initializations algorithms [7-11], normalization and regularization method and techniques [12-16], non-linearities [9, 17-19] and data-augmentation tricks [2, 13, 20-22], they are most beneficial when used on an already well performing architecture. In addition, some of these techniques may even impose more computational and memory usage overhead [12, 14]. Therefore, it would be highly desirable to propose efficient architectures with smaller number of layers and parameters that are as good as their deeper versions. Such architectures can then be further tweaked using novel advancements in the literature.

In this paper we propose a simple efficient architecture that outperforms almost all deeper architectures while using 2 to 25 times fewer parameters. Our architecture, SimpleNet, can be a very good candidate for many scenarios, especially for deploying in the embedded devices. It can be further compressed using methods such as DeepCompression [23] and thus its memory consumption can be decreased drastically. We impose a set of constraints for designing SimpleNet in order to achieve a well-crafted yet simple convolutional architecture. It is clear when the model performs well in spite of all limitations, relaxing those limitations can further boost the performance with little to no effort which is very desirable. This performance boost however has direct correlation with how well an architecture is designed. A fundamentally badly designed architecture would not be able to harness the advantages because of its inherent [flawed] design. The rest of the paper is organized as follows: Section 2 presents the most relevant works. In Section 3 we present our architecture and the set of designing principles used in the architecture. In Section 4 the experimental results are presented conducted on 4 major datasets (CIFAR10, CIFAR100, SVHN and MNIST) and more details about the architecture and different changes pertaining to each dataset are explained. Finally, conclusions and future work are summarized in Section 5.

## **2 Related Work**

In this section, we review the latest trends in related works in the literature. We categorize them into 4 sections and explain them briefly.

### **2.1 Complex networks**

Designing more effective networks were desirable and attempted from the advent of neural networks [24-26]. With the advent of deep learning methods, this desire manifested itself in the

form of creating deeper and more complex architectures [2-4, 18, 27-31]. This was first attempted and popularized by Ciresan et al [28] training a 9 layer MLP on GPU which was then practiced by other searchers [2-4, 18, 27-32].

In 2012 Krizhevsky et al [18] created a deeper version of LeNet5 [33] with 8 layers called AlexNet, unlike LeNet5, It had local contrast normalization, ReLU [18] nonlinearity instead of Tanh, and a new regularization layer called Dropout [34], this architecture achieved state of the art on ILSVRC 2012 [5]. The same year, Le et al [35] trained a gigantic network with 1 billion parameters, their work was later proceeded by Coats et al [36] which an 11 billion parameter network was trained. Both of them were ousted by much smaller network AlexNet [18].

In 2013 Lin et al [37] released their 12 layer, NIN architecture, they built micro neural networks into convolutional neural network using  $1 \times 1$  kernels. They also used global pooling instead of fully connected layers at the end acting as a structural regularizer that explicitly enforces feature maps to be confidence maps of concepts. In 2014 VGGNet by Simonyan et al [2] introduced several architectures, with increasing depth, 11 being the shallowest and 19 the deepest, they used  $3 \times 3$  conv kernels, and showed that stacking smaller kernels results in better non-linearity and achieves better accuracy. They showed the deeper, the better. The same year, GoogleNet [3] was released by Szegedy et al, with 56 convolutional layers making up a 22 modular layered network, their architecture was made up of convolutional layers with  $1 \times 1$ ,  $3 \times 3$  and  $5 \times 5$  kernels which they call, an *inception module*. Using this architecture they could decrease the number of parameters drastically compared to former architectures. They ranked first in ImageNet challenge that year. They later revised their architecture and used two consecutive  $3 \times 3$  conv layers with 128 kernels instead of the previous  $5 \times 5$  kernel layers, they also used a technique called Batch-normalization [14] for reducing internal covariate shift. This technique provided improvements in several sections which is explained thoroughly in [14]. They achieved state of the art results in ImageNet challenge.

In 2015 prior to GoogleNet achieving the state of the art on ImageNet, He et al [9], released their paper in which they used a ReLU variant called, Parametric RELU ( PReLU) to improve model fitting, they also created a initialization method specifically aimed at rectified nonlinearities, by which they could train deeper architectures better. Using these techniques, they could train a slightly modified version of VGGNet19 [2] architecture and achieved state of the art result on ImageNet. At the end of 2015, they proposed a deep architecture of 152 layers, called Residual Network (ResNet) [4] which was built on top their previous findings and achieved state of the art on ImageNet previously held by themselves. In ResNet they used what they call *residual blocks* in which layers are let to fit a residual mapping. They also used shortcut connections to perform identity mapping. This made them capable of training deeper networks easily and gain more accuracy by going deeper without becoming more complex. In fact their model is less complex than the much shallower VGGNet [2] which they previously used. They investigated architectures with 1000 layers as well. Later Huang et al [38] further enhanced ResNet with stochastic depth, where they used a training procedure, in which they would train a shorter network and then at test

time, use a deeper architecture. Using this method they could train even deeper architectures and also achieve state of the art on CIFAR dataset.

Prior to the residual network, Srivastava et al [30] released their Long Short Term Memory (LSTM) recurrent network inspired *highway networks* in which they used the initialization method proposed by He et al [9] and created a special architecture that uses adaptive gating units to regulate the flow of information through the network. They created a 100 layer and also experimented with a 1k layer network and reported the easy training of such network compared to the plain ones. Their contribution was to show that deeper architectures can be trained with Simple stochastic gradient descent.

In 2016 Szegedy et al [39] investigated the effectiveness of combining residual connections with their inceptionv3 architecture. They gave empirical evidence that training with residual connections accelerates the training of Inception networks significantly, and reported that residual Inception networks outperform similarly expensive Inception networks by a thin margin. With these variations the single-frame recognition performance on the ILSVRC 2012 classification task [5] improves significantly. With an ensemble of three residual and one Inception-v4, they achieved 3.08 percent top-5 error on the test set of the ImageNet classification challenge. The same year, Zagoria et al [31] ran a detailed experiment on residual nets [4] and came up with a novel architecture called *Wide Residual Net* (WRN) where instead of a thin deep network, they increased the width of the network in favor of its depth(decreased the depth). They showed that the new architecture does not suffer from the *diminishing feature reuse problem* [30] and slow training time. They report that a 16 layer wide residual network, outperforms any previous residual network architectures. They experimented with varying depth of their architecture from 10 to 40 layers and achieved state of the art result on CIFAR10, 100 and SVHN.

## 2.2 Model compression

The computational and memory usage overhead caused by such practices, limits the expansion and applications of deep learning methods. There have been several attempts in the literature to get around such problems. One of them is *model compression* in which it is tried to reduce the computational overhead at inference time. It was first researched by Bucila et al [40], where they tried to create a network that performs like a complex and large ensemble. In their method they used the ensemble to label unlabeled data with which they train the new neural network, thus learning the mappings learned by the ensemble and achieving similar accuracy. This idea is further worked on by Ba & Caruana [41]. They proposed a similar concept but this time they tried to compress deep and wide networks into shallower but even wider ones. Hinton et al [10] introduced their model compression model, called Knowledge Distillation (KD), which introduces a teacher/student paradigm for transferring the knowledge from a deep complex teacher model or an ensemble of such, to less complex yet still similarly deep but fine-grained student models, where each student model can provide similar performance overall and perform better on fine-grained classes where the teacher model confuses and thus eases the training of deep networks. Inspired by Hinton et al [10], Romero et al [42] proposed a novel architecture to address what they referred

to as not taking advantage of depth in the previous works related to Convolutional Neural Networks model compression. Previously, all works tried to compress a teacher network or an ensemble of networks into either networks of similar width and depth or into shallower and wider ones. However, they proposed a novel approach to train thin and deep networks, called *FitNets*, to compress wide and shallower (but still deep) networks. Their method is based on Knowledge Distillation (KD)[10] and extends the idea to allow for thinner and deeper student models. They introduce *intermediate-level hints* from the teacher hidden layers to guide the training process of the student, they showed that their model achieves the same or better accuracy than the teacher models.

## 2.3 Network Pruning

In late 2015 Han et al[23] released their work on model compression. They introduced “deep compression”, a three stage pipeline: pruning, trained quantization and Huffman coding, that work together to reduce the storage requirement of neural networks by 35 to 49 times without affecting their accuracy. In their method, the network is first pruned by learning only the important connections. Next, the weights are quantized to enforce weight sharing, finally, the Huffman coding is applied. After the first two steps they retrain the network to fine tune the remaining connections and the quantized centroids. Pruning, reduces the number of connections by 9 to 13 times; Quantization then reduces the number of bits that represent each connection from 32 to 5. On the ImageNet dataset, their method reduced the storage required by AlexNet by 35 times, from 240MB to 6.9MB, without loss of accuracy.

## 2.4 Light weight architectures

In 2015 Springenberg et al [43] released their paper where the effectiveness of simple architectures was investigated. The authors intended to come up with a simplified architecture, not necessarily shallower, that would perform better than at the time, more complex networks. They proposed different versions of their architecture and studied their characteristics, and later using a 17 layer version of their architecture they achieved a result very close to state of the art on CIFAR10 with intense data-augmentation.

In 2016 Iandola et al[44] released their paper in which they proposed a novel architecture called, *squeezenet*, a small CNN architecture that achieves AlexNet-level[18] accuracy on ImageNet With 50 times fewer parameters. To our knowledge this is the first architecture that tried to be small and yet be able to achieve a good accuracy.

In this paper, we tried to come up with a simple architecture which exhibits the best characteristics of these works and propose a 13 layer convolutional network that achieves state of the art result on CIFAR10<sup>2</sup>. Our network has fewer parameters (2 to 25 times less) compared to all previous deep architectures, and performs either superior to them or on par despite the huge difference in number of parameters and depth. For those architectures such as *squeezenet/fitnet* where the

---

<sup>2</sup> While preparing our paper we found out, our record was beaten by Wide Residual Net, which we then addressed in related works. We still have the state of the art record without data-augmentation as of zero padding and normalization.

number of parameters is less than ours but also are deeper, our network accuracy is far superior to what can be achieved with such networks. Our architecture is also the smallest (depth wise) architecture that both has a small number of parameters compared to all leading deep architectures, and also unlike previous architectures such as squeezenet or fitnet, gives higher or very competitive performance against all deep architectures.

Our model then can be compressed using deep compression techniques and be further enhanced, resulting in a very good candidate for many scenarios.

### 3 Proposed Architecture

We propose a simple convolutional network with 13 layers. The network employs a homogenous design utilizing  $3 \times 3$  kernels for convolutional layer and  $2 \times 2$  kernels for pooling operations. Figure 1 illustrates the proposed architecture.

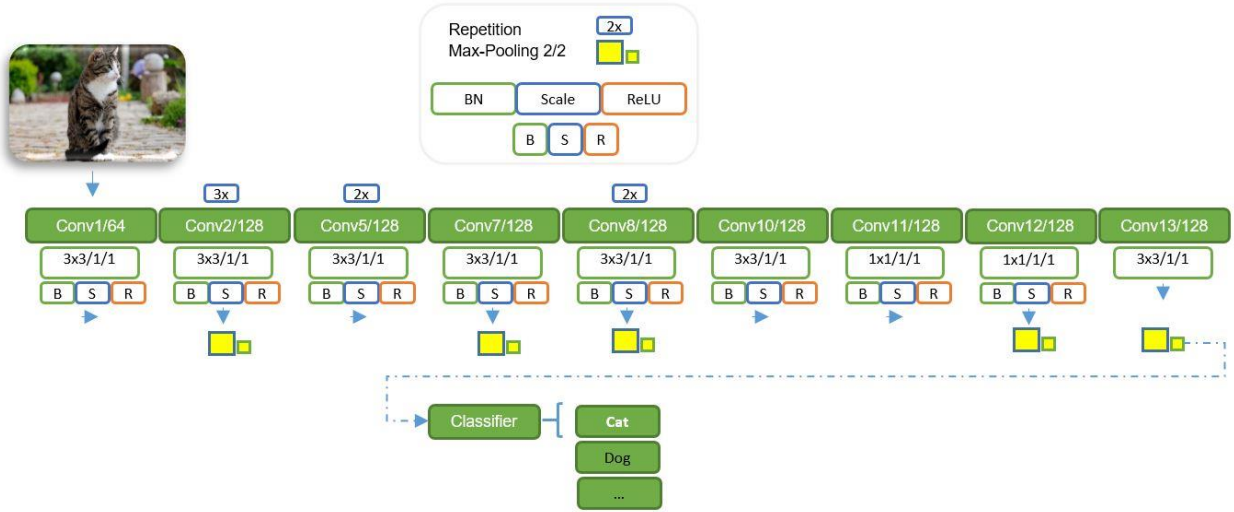


Figure 1 showing the base architecture with no drop-out

The only layers which do not use  $3 \times 3$  kernels are 11<sup>th</sup> and 12<sup>th</sup> layers, these layers, utilize  $1 \times 1$  convolutional kernels. Feature-map down-sampling is carried out using nonoverlapping  $2 \times 2$  max-pooling. In order to cope with the problem of vanishing gradient and also over-fitting, we used batch-normalization with moving average fraction of 0.95 before any ReLU non-linearity. We also used weight decay as regularizer. A second version of the architecture uses dropout to cope with over-fitting. Table 1 shows different architectures and their statistics, among which our architecture has the lowest number of parameters and operations. The extended list is provided in the appendix.

Table 1 showing different architectures statistics

Model	AlexNet	GoogleNet	ResNet152	VGGNet16	NIN	Ours
#Param	60M	7M	60M	138M	7.6M	5.4M
#OP	1140M	1600M	11300M	15740M	1100M	652M
Storage (MB)	217	51	230	512.24	29	20

### 3.2 Design intuition:

We used several principles in our work that helped us manage different issues much better and achieve desirable results. Here we present these principles with a brief explanation concerning the intuitions behind them:

- i. **Gradual Expansion and Minimum allocation:** In order to better manage the computational overhead, parameter utilization efficiency, and also network generalization power, start with a small and thin network, and then gradually expand it. Neither the depth nor the number of parameters are good indicators of how a network should perform. They are neutral factors that are only beneficial when utilized mindfully, otherwise, the design would be an inefficient network imposing un-wanted overhead. Furthermore, fewer learnable parameters also decrease the chance of over fitting and together with an enough depth it increases the networks generalization power. In order to utilize both depth and parameters more efficiently, design the architecture in a symmetric and gradual fashion, i.e. instead of creating a network with a random yet great depth, and large number of neurons per layer, start with a small and thin network then gradually add more symmetric layers. Expand the network to reach a cone shaped form. A Large degree of invariance to geometric transformations of the input can be achieved with this progressive reduction of spatial resolution compensated by a progressive increase of the richness of the representation (the number of feature maps), hence getting a coned shape, that's one of the reasons why deeper is better) [33]. Therefore a deeper network with thinner layers, tends to perform better than the same network being much shallower with wider layers. It should however be noted that, very deep and very thin architectures like their shallow and very wide counter parts are not recommended. The network needs to have proper processing and representational capacity and what this principle suggests is a method of finding the right value for depth and width of a network for this very reason.
- ii. **Homogeneous Groups of Layers:** Instead of thinking in layers, think and design in group of homogenous layers. The idea is to have several homogeneous groups of layers, each with gradually more width. The symmetric and homogenous design, allows to easily manage the number of parameters a network will withhold and also provide better information pools for each semantic level.
- iii. **Local Correlation Preservation:** Preserve locality information throughout the network as much as possible by avoiding  $1 \times 1$  kernels in early layers. The corner stone of CNN success lies in local correlation preservation. Avoid using  $1 \times 1$  kernels or fully connected layers where locality of information matters. This includes exclusively the early layers in the network.  $1 \times 1$  kernels have several desirable characteristics such as increasing networks non-linearity and feature fusion [37] which increases abstraction level, but they also ignore any local correlation in the input. Since they do not consider any neighborhood in the input and only take channels into account, they distort valuable local information. Preferably use  $1 \times 1$  kernels at the end of the network or if one intends on using tricks such as bottleneck employed by GoogleNet [3] and ResNet [4], use more layers with skip connections to

compensate the loss in information. It is suggested to replace  $1 \times 1$  kernels with  $2 \times 2$  if one plans on using them other than the end of the network. Using  $2 \times 2$  kernels both help to reduce the number of parameters and also to retain neighborhood information.

- iv. **Maximum Information Utilization:** Utilize as much information as it is made available to a network by avoiding rapid down sampling especially in early layers. To increase a network's discriminative power, more information needs to be made available. This can be achieved either by a larger dataset or larger feature-maps. If larger dataset is not feasible, the existing training samples must be efficiently harnessed. Larger feature-maps especially in early layers, provide more valuable information to the network than the smaller ones. With the same depth and number of parameters, a network which utilizes bigger feature-maps achieves a higher accuracy. Therefore instead of increasing the complexity of a network by increasing its depth and number of parameters, one can leverage more performance/accuracy by simply using larger input dimensions or avoiding rapid early down-sampling. This is a good technique to keep the complexity of the network in check and improve the network performance.
- v. **Maximum Performance Utilization:** Use  $3 \times 3$ , and follow established industrial trends. For an architecture to be easily useable and widely practical, it needs to perform fast and decently. By taking into account the current improvements in underlying libraries, designing better performing and more efficient architectures are possible. Using  $3 \times 3$  kernels, apart from already known benefits [2], allows to achieve a substantial boost in performance when using NVIDIA's cuDNNv5.x library. A speed up of about 2.7x compared to the former v4 version.<sup>3</sup> This is illustrated in Figure 2. This ability to harness every amount of performance is a decisive criterion when it comes to production and industry. A fast and robust performance translates into, less time, decreased cost and ultimately a higher profit for business owners. Apart from the performance point of view, on one hand larger kernels do not provide the same efficiency per parameter as a  $3 \times 3$  kernel does. It may be theorized that since larger kernels capture a larger area of neighborhood in the input, using them may help in ignoring noises and thus capturing better features, or more interesting correlations in the input because of larger receptive field and ultimately improving performance. But in fact the overhead they impose in addition to the loss in information they cause make them not an ideal choice. This makes the efficiency per parameter to decrease and causes unnecessary computational burden. More over larger kernels can be replaced with a cascade of smaller ones (e.g.  $3 \times 3$ ) which will still result in the same effective receptive field and also more nonlinearity, making them a better choice over larger kernels.

---

<sup>3</sup> <https://developer.nvidia.com/cudnn-whatsnew>



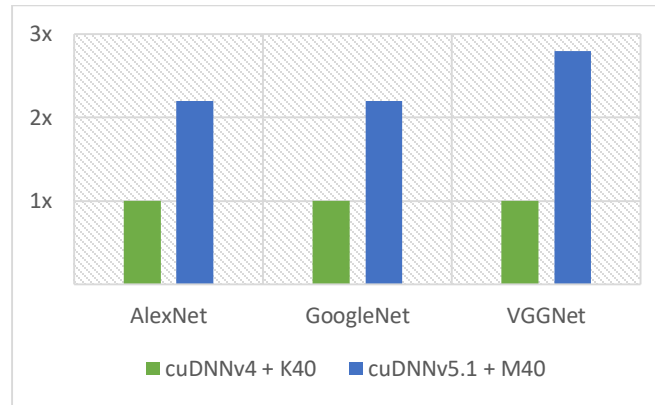


Figure 2 showing 2.7x faster training when using 3×3 kernels using cuDNNv5.x

- vi. **Rapid Prototyping:** Test the architecture with different learning policies before altering it. Most of the time, it's not the architecture that needs to be change, rather it's the optimization policy. A badly chosen optimization policy leads to bad convergence, wasting network resources. Simple things such as learning rates and regularization methods, usually have an adverse effect if not tuned correctly. Therefore it is first suggested to use an automated optimization policy to run quick tests and when the architecture is finalized, the optimization policy is carefully tuned to maximize network performance.
- vii. **8. Experiment Isolation:** Conduct experiments under equal conditions. When testing a new feature, make sure only the new feature is being evaluated. For instance, when evaluating a 5×5 kernel against a 3×3 kernel, the overall network entropy must remain equal. It is usually neglected in different experiments and changes are not evaluated in isolation or better said, under an equal condition. This can lead to a wrong deduction and thus result in an inefficient design. In order to effectively assess a specific feature and its effectiveness in the architecture design, it is important to keep track of the changes, either caused by previous experiments or by the addition of the new feature itself, and take necessary action to eliminate the sources of discrepancies.
- viii. **Minimum Entropy:** Like previous principle, here we explain about the generalization power and why lower entropy matters. It is true that the more parameter a network withholds, the faster it can converge, and the more accuracy it can achieve, but it will over-fit more as well. A model with fewer number of parameters which provides better results or performs comparable to heavier models indicates the fact that, the network has learnt much better features based on which it is making its decision. In other words, by imposing more constrains on the amount of entropy a network has, we force the network to find and use much better and more robust features. This specifically manifests itself in the generalization power, since the network decisions are based on more important and more discriminative features. It can thus perform much better compared to a network with higher number of parameters which would easily over fit as well.

- ix. **Final Regulation Stage:** While we try to formulate the best ways to achieve better accuracy in the form of rules or guidelines, they are not necessarily meant to be aggressively followed in all cases. These guidelines are meant to help achieve a good compromise between performance and the imposed overhead. Therefore start by designing according to the guidelines and then try to alter the architecture in order to get the best compromise according to your needs. In order to better tune your architecture, try not to alter or deviate a lot from multiple guidelines at once. Following a systematic procedure helps to avoid repetitive actions, and also obtain better understanding of what/which series of actions lead to specific outcomes that would normally be a hard task. Work on one aspect at a time until the desired outcome is achieved. After all it's all about the well balanced compromise between performance / imposed overhead according to one's specific needs.

As we have already briefly discussed in previous sections, the current trend in the community, has been to start with a deep and big architecture and then use different regularization methods to cope with over-fitting. The intuition behind such trend is that, it is naturally difficult to come up with an architecture with the right number of parameters/depth that suites exactly ones data requirements. While such intuition is plausible and correct, it is not without flaws.

One of the issues is the fact that, there are many use cases and applications for which there is not a huge dataset (such as ImageNet e.g.) available. Apart from the fact that less computation and memory overhead is always desirable for any circumstances and results in decreased costs, the majority of applications have access to medium/small sized datasets and yet they are already exploiting the benefits of deep learning and achieving either state of the art or very outstanding results. Individuals coming from this background, have two paths before them when they want to initiate a deep learning related project: 1) they either are going to design their own architecture which is difficult and time-consuming and has its own share of issues and 2) Use one of the existing heavy but very powerful architectures that have won competitions such as ImageNet or performed well on a related field of interest.

Using these kinds of architectures impose a lot of overhead and users should also bear the cost of coping with the resulting over-fitting. It adversely affects training time, making it more time and resource consuming. When such architectures are used for fine-tuning, the issues caused by such deep and heavy architectures such as computational, memory and time overhead, are also imposed. Therefore it makes more sense to have a less computationally expensive architecture which provides higher or comparable accuracy compared to the heavier counter parts. The lowered computational overhead results in a decreased time and power consumption which is a decisive factor for mobile applications. Apart from such benefits, reliance on better and more robust features is another important reason to opt for such networks.

## 4 Experiments

We experimented on CIFAR-10/100 [45], SVHN [46] and MNIST [33] datasets in order to evaluate and compare our architecture against the top ranking methods and deeper models that

also experimented on such datasets. We only used simple data augmentation of zero padding, and mirroring on CIFAR10/100. Other experiments on MNIST [33], SVHN [46] datasets are conducted without data-augmentation. In our experiments we used one configuration for all datasets and, we did not fine-tune anything except CIFAR10. We did this to see how this configuration can perform with no or slightest change in different scenarios. We used Caffe framework [47] for training our architecture and ran our experiments on a system with Intel Pentium G3220 CPU ,14 Gigabyte of RAM and NVIDIA GTX980.

## 4.1 CIFAR10/100

The CIFAR10/100 datasets includes 60,000 color images of which 50,000 belong to training set and 10,000 are reserved for testing (validation). These images are divided into 10 and 100 classes respectively and classification performance is evaluated using top-1 error. Table 2 shows the results achieved by different architectures.

We tried two different configurations for CIFAR10 experiment, one with no data-augmentation i.e. zero-padding and normalization and another one using data-augmentation. We name them Arch1 and Arch2 respectively. The Arc1 achieves a new state of the art in CIFAR10 when no data-augmentation is used and the Arc2 achieves 95.32. In addition to the normal architecture, we used a modified version on cifar100 and achieved 74.86 with data-augmentation. Since it had more parameters we did not include it in the following table. More results are provided in the appendix.

Table 2 showing Top CIFAR10/100 results

Method	#Params	CIFAR10	CIFAR100
VGGNet(original 16L)[48]/Enhanced	138m	91.4 / 92.45	-
ResNet-110L[4] / 1202L[4] *	1.7m/10.2m	93.57 / 92.07	74.84 / 72.18
Stochastic depth-110L[38] / 1202L[38]	1.7m/10.2m	94.77 / 95.09	75.42 / -
Wide Residual Net-(16/8)[31] / (28/10)[31]	11m/36m	95.19 / 95.83	77.11 / 79.5
Highway Network[30]	-	92.40	67.76
FitNet[42]	1M	91.61	64.96
Fractional Max-pooling* (1 tests)[13]	12M	95.50	73.61
Max-out(k=2)[12]	6M	90.62	65.46
Network in Network[37]	1M	91.19	64.32
Deeply Supervised Network[49]	1M	92.03	65.43
Max-out Network In Network[50]	-	93.25	71.14
All you need is a good init (LSUV)[51]	-	94.16	-
<b>Our Architecture*</b>	<b>5.48m</b>	<b>94.75</b>	-
<b>Our Architecture †<sup>4</sup></b>	<b>5.48m</b>	<b>95.32</b>	<b>73.42</b>

\*Note that the Fractional max pooling[13] uses a deeper architecture and also uses extreme data augmentation. \* means No zero-padding or normalization with dropout and † means Standard data-augmentation- with dropout. To our knowledge, our architecture has the state of the art result, without aforementioned data-augmentations.

<sup>4</sup> Data-augmentation method used by stochastic depth paper: <https://github.com/Pastromhaug/caffe-stochastic-depth>.

### 4.3 MNIST

The MNIST dataset [33] consists of 70,000 28x28 grayscale images of handwritten digits 0 to 9, of which 60,000 are used for training and 10,000 are used for testing. We didn't use any data augmentation on this dataset, and yet scored second to the state-of-the-art without data-augmentation and fine-tuning. We also slimmed our architecture to have only 300K parameters and achieved 99.72% accuracy beating all previous larger and heavier architectures. Table 3 shows the current state of the art results for MNIST.

Table 3 showing MNIST results

Method	Error rate
Regularization of Neural Networks using DropConnect[16]**	0.21%
Multi-column Deep Neural Networks for Image Classification[52]**	0.23%
APAC: Augmented Pattern Classification with Neural Networks[53]**	0.23%
Batch-normalized Max-out Network in Network[50]	0.24%
Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree[54]**	0.29%
Fractional Max-Pooling[13]**	0.32%
Max-out network (k=2) [12]	0.45%
Network In Network [37]	0.45%
Deeply Supervised Network [49]	0.39%
RCNN-96 [55]	0.31%
<b>Our architecture *</b>	<b>0.25%</b>

\*Note that we didn't intend on achieving the state of the art performance here (since we are using a single optimization policy without fine-tuning hyper parameters or data-augmentation for a specific task), and still we nearly achieved state-of-the-art on MNIST. \*\*Results achieved using an ensemble or extreme data-augmentation

### 4.4 SVHN

The SVHN dataset [46] is a real-world image dataset, obtained from house numbers in Google Street View images. It consists of 630,420 32x32 color images of which 73,257 images are used for training, 26,032 images are used for testing and the other 531,131 images are used for extra training. Like [12, 37, 38] we only used the training and testing sets for our experiments and didn't use any data-augmentation. We also used the slimmed version with 300K parameters and obtained a very good test error of 2.37%. Table 4 shows the current state of the art results for SVHN.

Table 4 showing SVHN results

Method	Error
Network in Network	2.35%
Deeply Supervised Net	1.92%
ResNet (reported by Huang et al. (2016))	2.01%
ResNet with Stochastic Depth	1.75%
Wide ResNet	1.64%
<b>Our architecture</b>	<b>1.79%</b>

#### 4.5 Extended test: Testing the architecture with fewer number of parameters:

Some architectures can't scale well when their processing capacity decreases. This shows the design is not robust enough to efficiently use its processing capacity. We tried a slimmed version of our architecture which has only 300K parameters to see how it performs and whether it's still efficient. The network also does not use any dropout. Table 5 shows the results for our architecture with only 300K parameters in comparison to other deeper and heavier architectures with 2 to 20 times more parameters.

Table 5-Slimmed version Results on Different Datasets

Model	Ours		Maxout	DSN	ALLCNN	dasNet	ResNet(32)	WRN	NIN
#Param	<b>310K</b>	<b>460K</b>	6M	1M	1.3M	6M	475K	600K	1M
CIFAR10	<b>91.98</b>	<b>92.33</b>	90.62	92.03	92.75	90.78	91.6	93.15	91.19
CIFAR100	<b>64.68</b>	<b>66.82</b>	65.46	65.43	66.29	66.22	67.37	69.11	-

## 5 Conclusion

In this paper, we proposed a simple convolution architecture that takes advantage of the simplicity in its design and outperforms deeper and more complex architectures in spite of having considerably fewer number of parameters and operations. We showed that a good design should be able to efficiently use its processing capacity and showed that our slimmed version of the architecture with much fewer number of parameters (300K) also outperforms deeper and or heavier architectures. Intentionally limiting ourselves to a few layers and basic elements for designing an architecture allowed us to overlook the unnecessary details and concentrate on the critical aspects of the architecture, keeping the computation in check and achieve high efficiency.

We tried to show the importance of simplicity and optimization using our experiments and also encourage more researchers to study the vast design space of convolutional neural network in an effort to find more and better guidelines to make or propose better performing architectures with much less overhead. This will hopefully greatly help to expand deep learning related methods and applications, making them more viable in more situations.

Due to lack of good hardware, we had to contend ourselves to a few configurations. We are still continuing our tests and would like to extend our work by experimenting on new applications and design choices especially using the latest achievements about deep architectures in the literature.

## 6 Acknowledgement

We would like to thank Dr. Ali Diba, Prof. Dr. Hamed Pirsiavash, Dr. Reza Saadati, and Dr. Javad Vahidi for their helpful feedback, comments and cooperation.

## References:

1. Guo, Y., et al., *Deep learning for visual understanding: A review*. Neurocomputing.
2. Simonyan, K. and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. CoRR, 2014. **abs/1409.1556**.
3. Szegedy, C., et al. *Going deeper with convolutions*. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.
4. He, K., et al., *Deep Residual Learning for Image Recognition*. CoRR, 2015. **abs/1512.03385**.
5. Russakovsky, O., et al., *ImageNet Large Scale Visual Recognition Challenge*. International Journal of Computer Vision, 2015. **115**(3): p. 211-252.
6. Sercu, T., Puhersch, C., Kingsbury, B., and LeCun, Y., *Very Deep Multilingual Convolutional Neural Networks for LVCSR*. 2015. **ArXiv e-prints, September 2015**.
7. Mishkin, D. and J. Matas, *All you need is a good init*. arXiv preprint arXiv:1511.06422, 2015.
8. Glorot, X. and Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks*. in *Aistats*. 2010.
9. He, K., et al. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. in *Proceedings of the IEEE International Conference on Computer Vision*. 2015.
10. Hinton, G., O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*. arXiv preprint arXiv:1503.02531, 2015.
11. Saxe, A.M., J.L. McClelland, and S. Ganguli, *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*. arXiv preprint arXiv:1312.6120, 2013.
12. Ian J. Goodfellow, D.W.-F., Mehdi Mirza, Aaron Courville, Yoshua Bengio, *Maxout networks*. 2013. **28**: p. 1319-1327.
13. Graham, B., *Fractional Max-Pooling*. 2014. **ArXiv e-prints, December 2014a**.
14. Ioffe, S. and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. CoRR, 2015. **abs/1502.03167**.
15. Wager, S., S. Wang, and P.S. Liang. *Dropout training as adaptive regularization*. in *Advances in neural information processing systems*. 2013.
16. Wan, L., et al. *Regularization of neural networks using dropconnect*. in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013.
17. Maas, A.L., A.Y. Hannun, and A.Y. Ng. *Rectifier nonlinearities improve neural network acoustic models*. in *Proc. ICML*. 2013.
18. Nair, V. and G.E. Hinton. *Rectified linear units improve restricted boltzmann machines*. in *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.
19. Clevert, D.-A., T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*. arXiv preprint arXiv:1511.07289, 2015.
20. Alex, K., I. Sutskever, and E.H. Geoffrey, *ImageNet Classification with Deep Convolutional Neural Networks*. 2012: p. 1097--1105.
21. Wu, R., et al., *Deep Image: Scaling up Image Recognition*. CoRR, 2015. **abs/1501.02876**.
22. Xu, B., et al., *Empirical evaluation of rectified activations in convolutional network*. arXiv preprint arXiv:1505.00853, 2015.

23. Han, S., H. Mao, and W.J. Dally, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*. CoRR, abs/1510.00149, 2015. **2**.
24. Ivakhnenko, A., *Polynomial theory of complex systems*. IEEE Transactions on Systems, Man, and Cybernetics, 1971(4): p. 364-378.
25. Fukushima, K., *Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position- Neocognitron*. ELECTRON. & COMMUN. JAPAN, 1979. **62**(10): p. 11-18.
26. Fukushima, K., *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological cybernetics, 1980. **36**(4): p. 193-202.
27. Cireşan, D., et al. *A committee of neural networks for traffic sign classification*. in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. 2011. IEEE.
28. Ciresan, D.C., et al., *Deep, big, simple neural nets for handwritten digit recognition*. Neural computation, 2010. **22**(12): p. 3207-3220.
29. CireşAn, D., et al., *Multi-column deep neural network for traffic sign classification*. Neural Networks, 2012. **32**: p. 333-338.
30. Srivastava, R.K., K. Greff, and J. Schmidhuber, *Highway networks*. arXiv preprint arXiv:1505.00387, 2015.
31. Zagoruyko, S. and N. Komodakis, *Wide Residual Networks*. arXiv preprint arXiv:1605.07146, 2016.
32. Ciresan, D., U. Meier, and J. Schmidhuber. *Multi-column deep neural networks for image classification*. in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. 2012. IEEE.
33. Lecun, Y.a.B., L. and Bengio, Y. and Haffner, P., *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 1998. **86**: p. 2278-2324.
34. Hinton, G.E., et al., *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580, 2012.
35. Le, Q.V. *Building high-level features using large scale unsupervised learning*. in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. 2013. IEEE.
36. Coates, A.H., Brody ;Wang,Tao;Wu,David J.;Ng,Andrew Y.;Catanzaro, Bryan, *Deep learning with COTS HPC systems*. 2013.
37. Lin, M., Q. Chen, and S. Yan, *Network In Network*. CoRR, 2013. **abs/1312.4400**.
38. Huang, G., et al., *Deep networks with stochastic depth*. arXiv preprint arXiv:1603.09382, 2016.
39. Szegedy, C., et al., *Inception-v4, inception-resnet and the impact of residual connections on learning*. arXiv preprint arXiv:1602.07261, 2016.
40. Buciluă, C., R. Caruana, and A. Niculescu-Mizil. *Model compression*. in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006. ACM.
41. Ba, J. and R. Caruana. *Do deep nets really need to be deep?* in *Advances in neural information processing systems*. 2014.
42. Romero, A., et al., *Fitnets: Hints for thin deep nets*. arXiv preprint arXiv:1412.6550, 2014.
43. Springenberg, J.T., et al., *Striving for Simplicity: The All Convolutional Net*. CoRR, 2014. **abs/1412.6806**.

44. Iandola, F.N., et al., *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 1MB model size*. arXiv preprint arXiv:1602.07360, 2016.
45. Krizhevsky, A.a.H., G, *Learning multiple layers of features from tiny images*. 2009.
46. Netzer, Y., et al., *Reading digits in natural images with unsupervised feature learning*. 2011.
47. Jia, Y., Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, and S. Guadarrama, and Darrell, Trevor, *Caffe: Convolutional architecture for fast feature embedding*. 2014. **arXiv preprint arXiv:1408.5093**.
48. Zagoruyko, S., *92.45% on CIFAR-10 in Torch*. 2015.
49. Lee, C.-Y., Xie, Saining, Gallagher, Patrick W., Zhang, Zhengyou, and Tu, Zhuowen., *Deeply supervised nets*. In Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015, 2015., 2015.
50. Jia-Ren Chang, Y.-S.C., *Batch-normalized Maxout Network in Network*. 2015. **arXiv:1511.02583v1**.
51. Dmytro Mishkin, J.M., *ALL YOU NEED IS A GOOD INIT*. ICLR, 2016.
52. Ciregan, D., U. Meier, and J. Schmidhuber. *Multi-column deep neural networks for image classification*. in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. 2012. IEEE.
53. Sato, I., H. Nishimura, and K. Yokoi, *APAC: Augmented Pattern Classification with Neural Networks*. arXiv preprint arXiv:1505.03229, 2015.
54. Lee, C.-Y., P.W. Gallagher, and Z. Tu. *Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree*. in *International Conference on Artificial Intelligence and Statistics*. 2016.
55. Hu, M.L.a.X., *Recurrent convolutional neural network for object recognition*. 2015.
56. Liang, M. and X. Hu. *Recurrent convolutional neural network for object recognition*. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.
57. Yosinski, J., et al., *Understanding neural networks through deep visualization*. arXiv preprint arXiv:1506.06579, 2015.
58. Graham, B., *Spatially-sparse convolutional neural networks*. arXiv preprint arXiv:1409.6070, 2014.
59. Snoek, J., et al. *Scalable bayesian optimization using deep neural networks*. in *International Conference on Machine Learning*. 2015.



## Appendix:

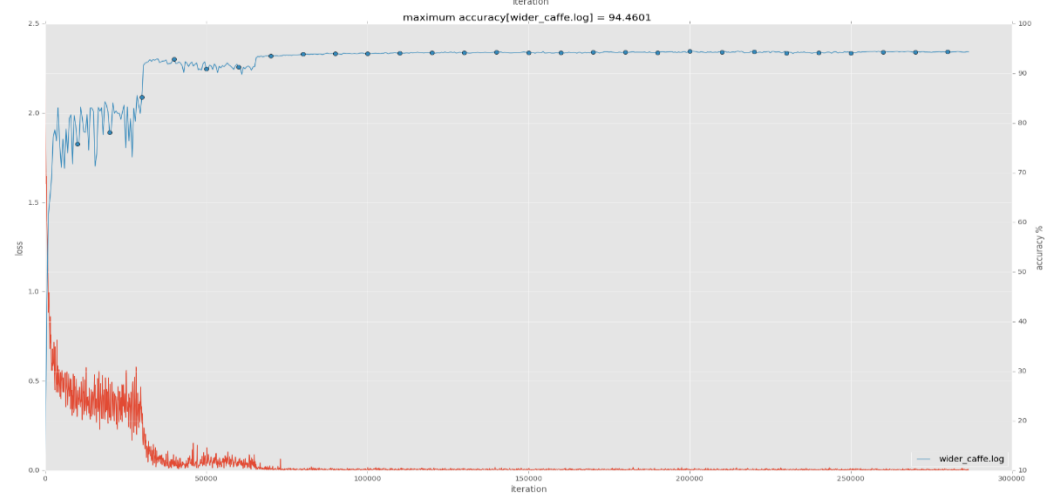
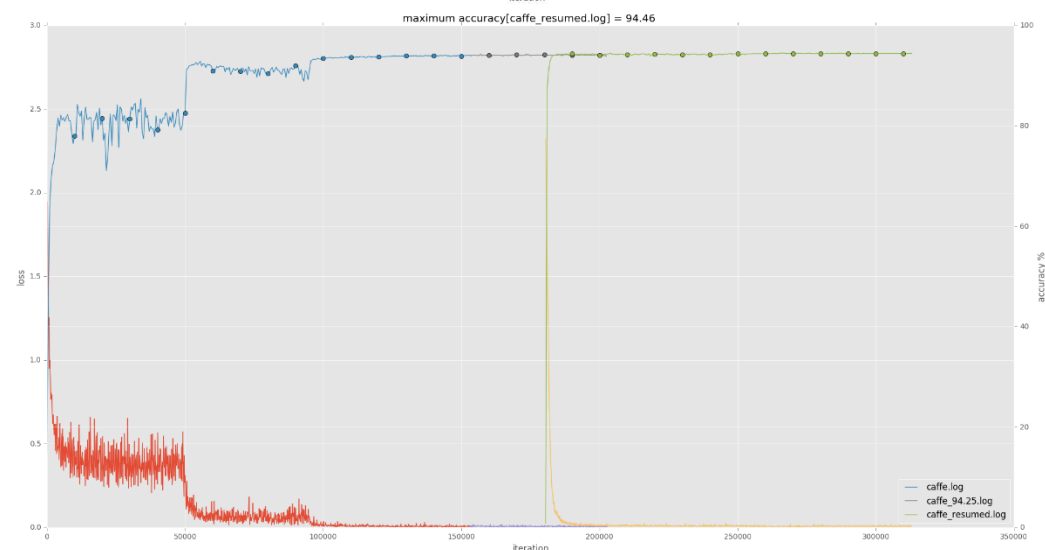
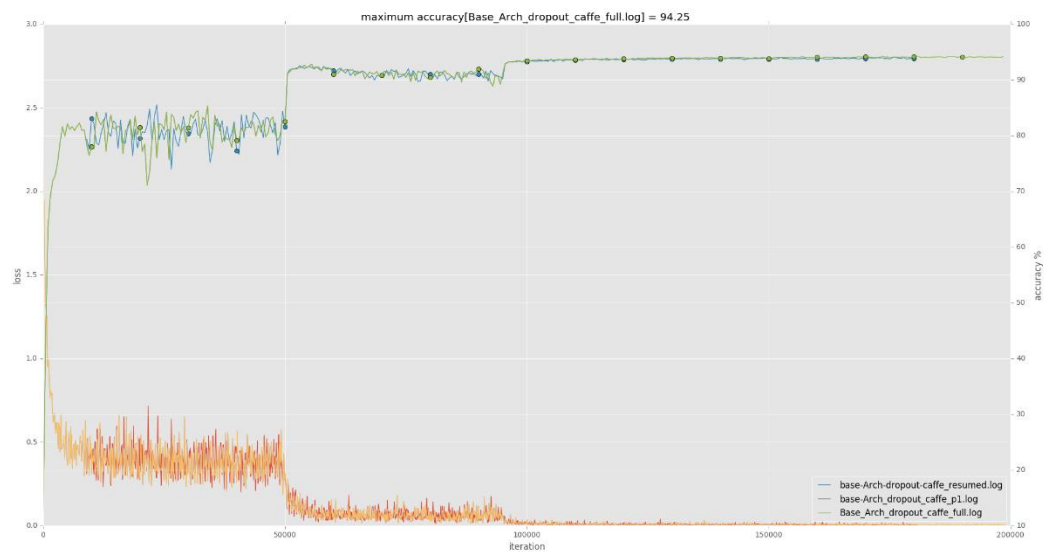
### The importance of Optimization plans:

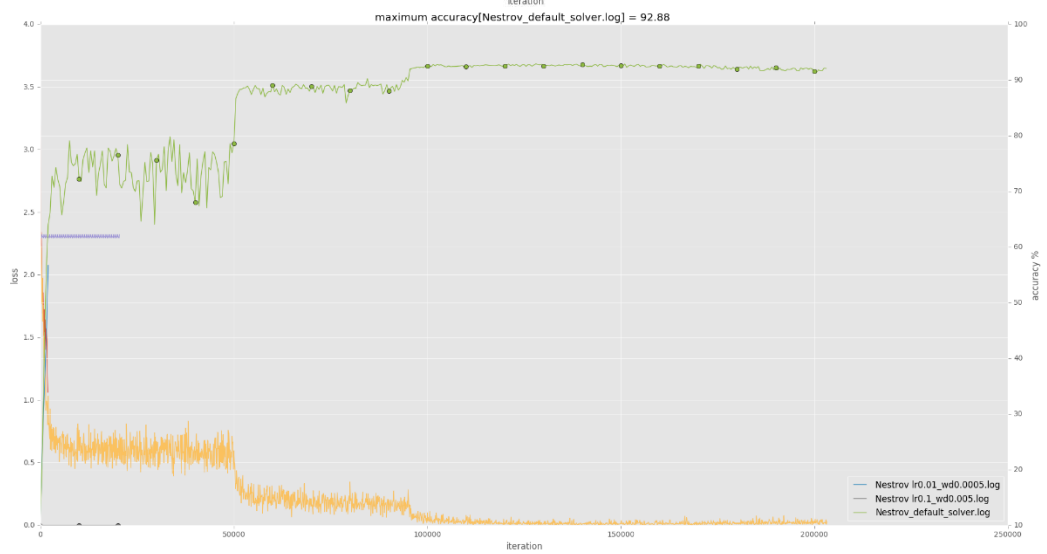
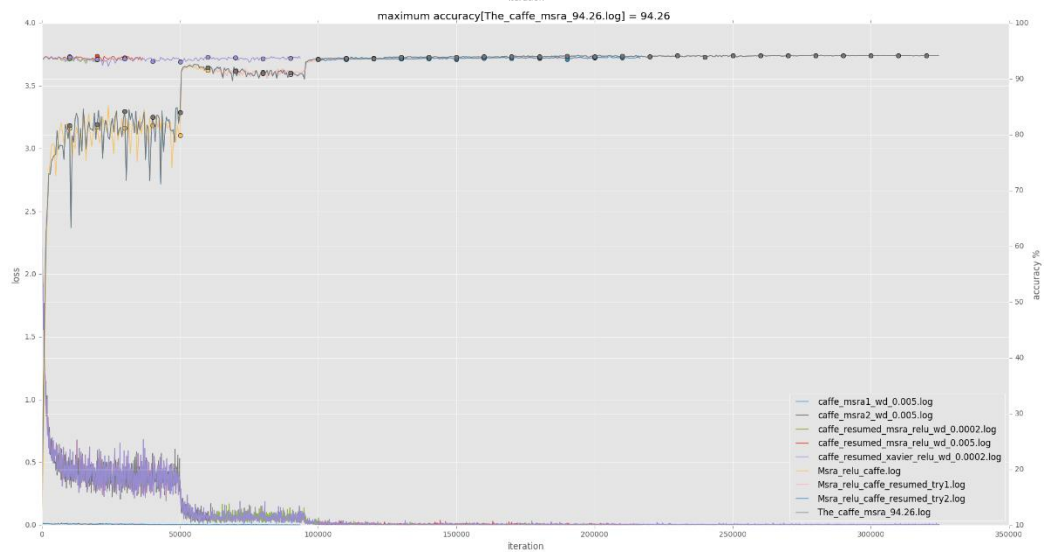
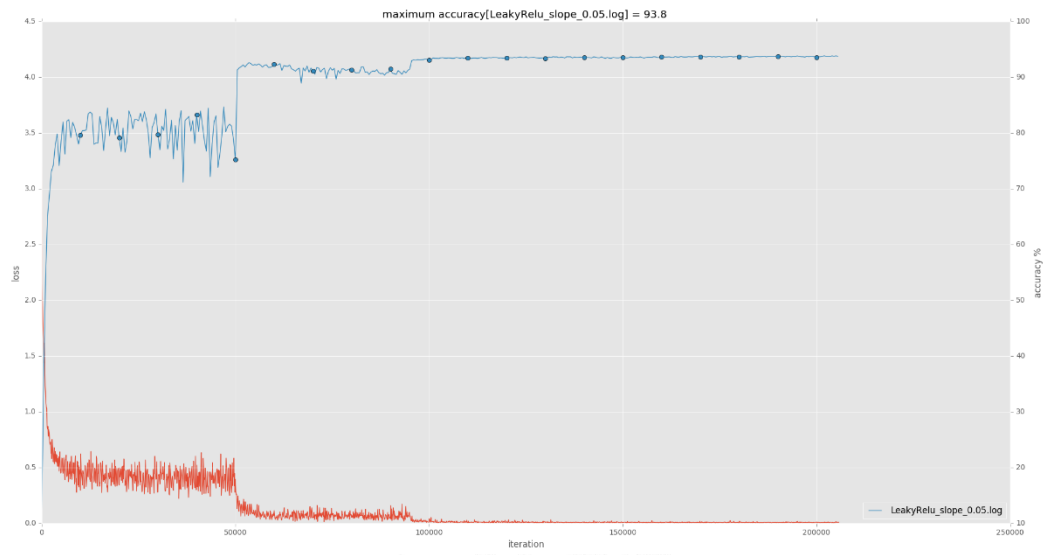
Different configurations are sampled and depicted below, which show an obvious but yet neglected point. That is all of the results below, show the very same architecture, performing differently when differently optimized. The optimization process needless to say is not a complex procedure. Looking at the charts below and previous results, it can be confidently said that a properly crafted architecture can let a significant capacity of itself to go to waste, by poor optimization decisions. Yet again a well-chosen optimization policy can make a simple trivial architecture utilize the most out of the capacity it withholds and outperform all deeper architectures with several times fewer parameters!

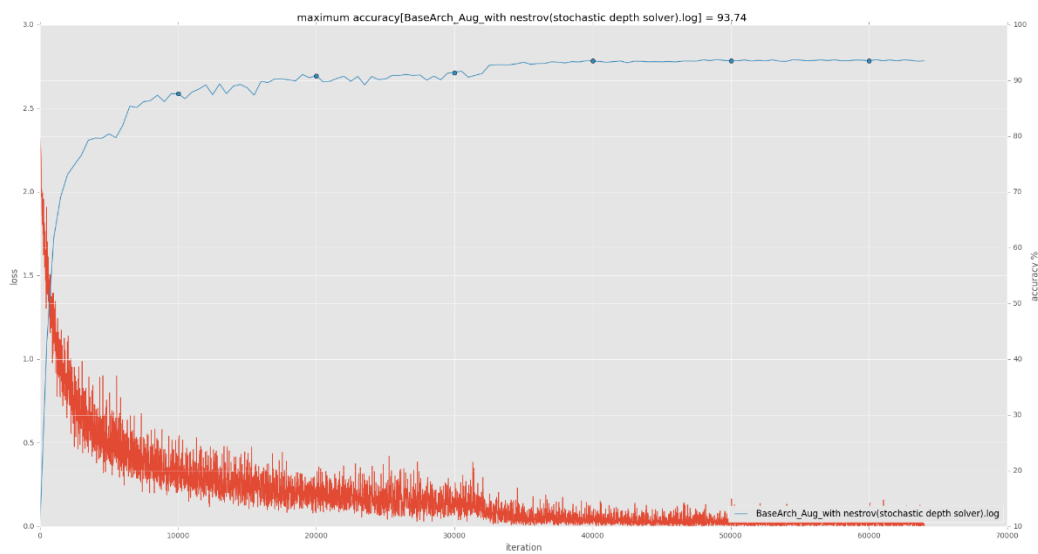
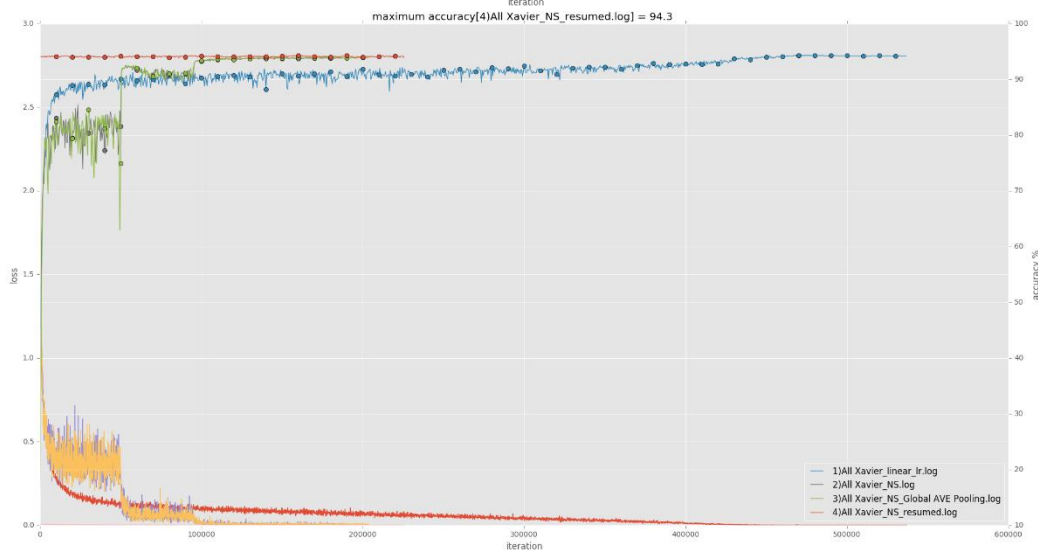
The fact is that there should be a sensible level of hierarchy of features and enough number of parameters so that in accordance to the input data, the network has the potential /capacity to provide a desirable answer. So far, the recent trend in CNN design has been redirected toward training very deep and almost always heavier architectures in order to attain good performance, and in doing so, the optimization part in our humble opinion is neglected, and the resulting outcome makes the spectrum of their practical uses yet more limited.

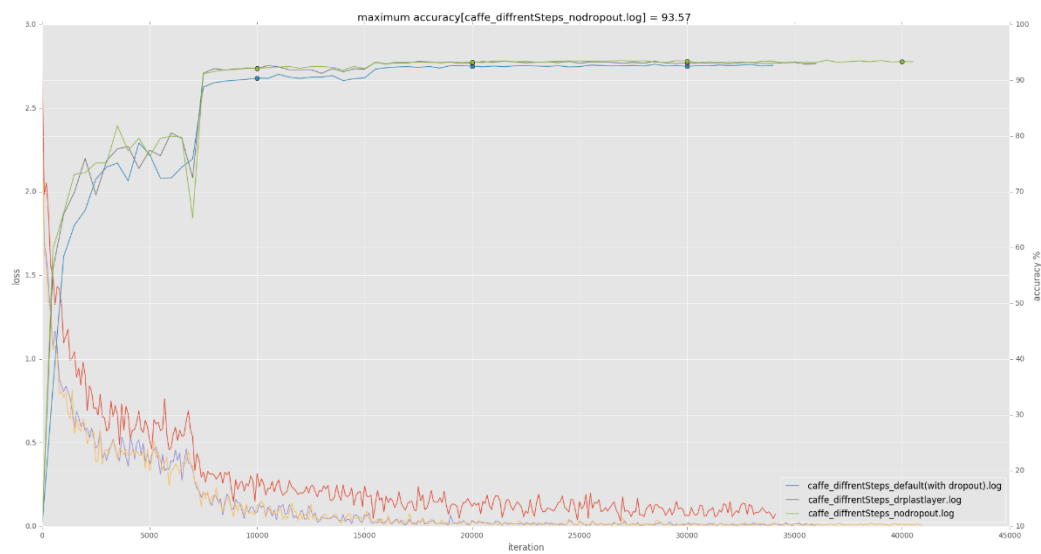
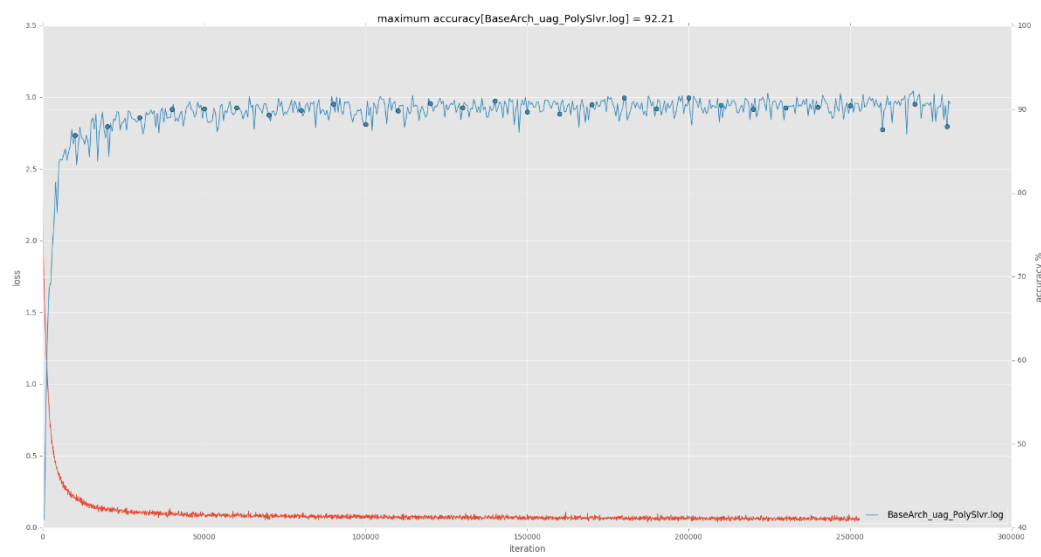
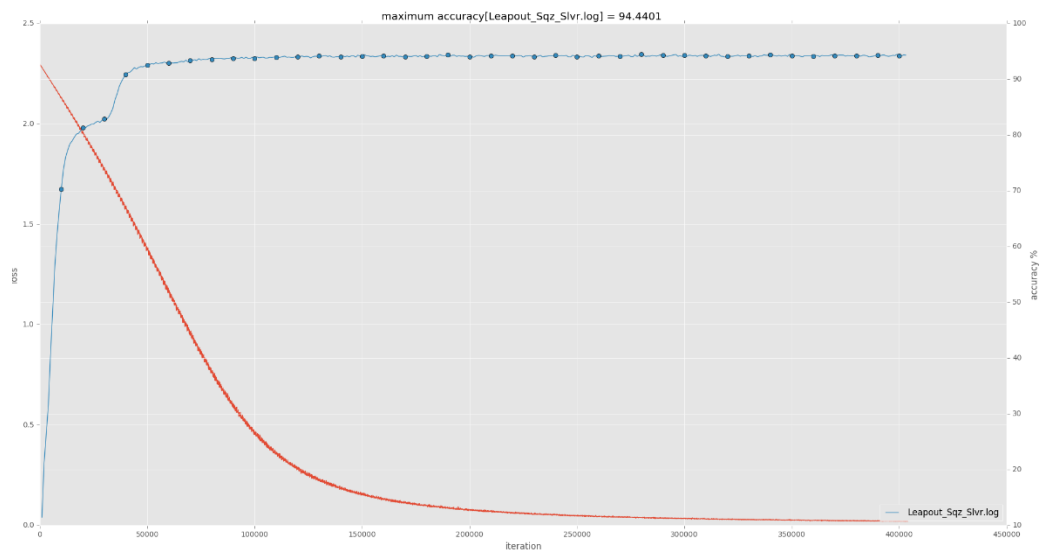
Looking at the results in our experiments and also looking at examples such as NIN[37], MIN[50], RCNN-96[56] which at the time outperformed heavier architectures, or stochastic depth[38] which enhanced ResNet performance, we can probably say with confidence that going *very* deep is not an essential criterion to gain better performance.

Our attempt practically shows that a much shallower and simpler architecture can still provide very good results. Ultimately this means that the improvements in this regard lies in the optimization plans rather than a blindly chosen depth or width of the network.





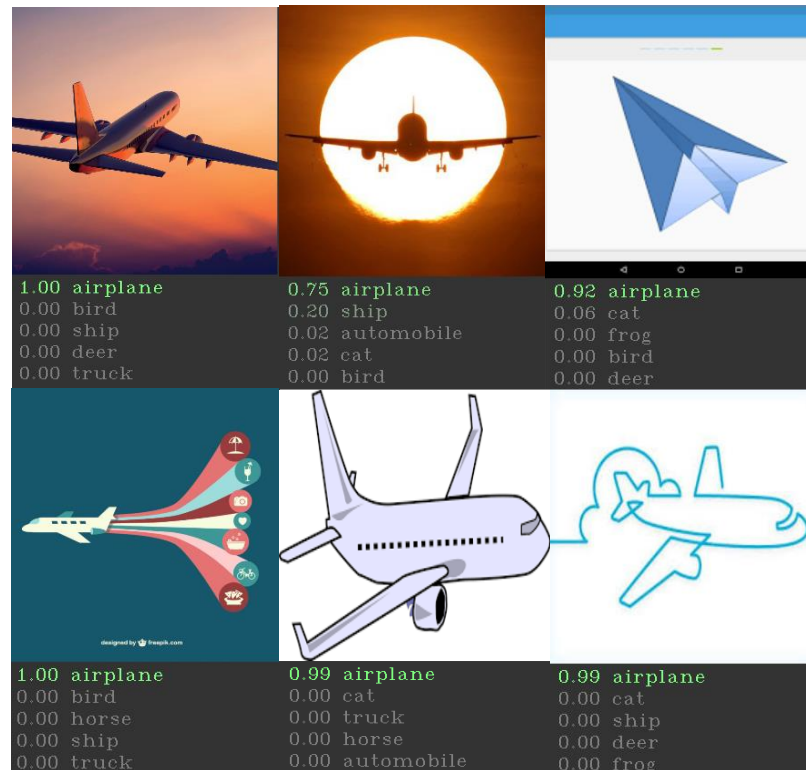




## Generalization Samples:

In this section we show some examples from CIFAR10 dataset that the network classifies correctly despite the huge difference in their appearance with other samples and also those present in training set. These pictures are taken using Deep Visualization Toolbox<sup>5</sup> by Yosinski et al[57] and early un-augmented version model.

Figure 6 some airplanes pictures with completely different appearance that the network classifies very well



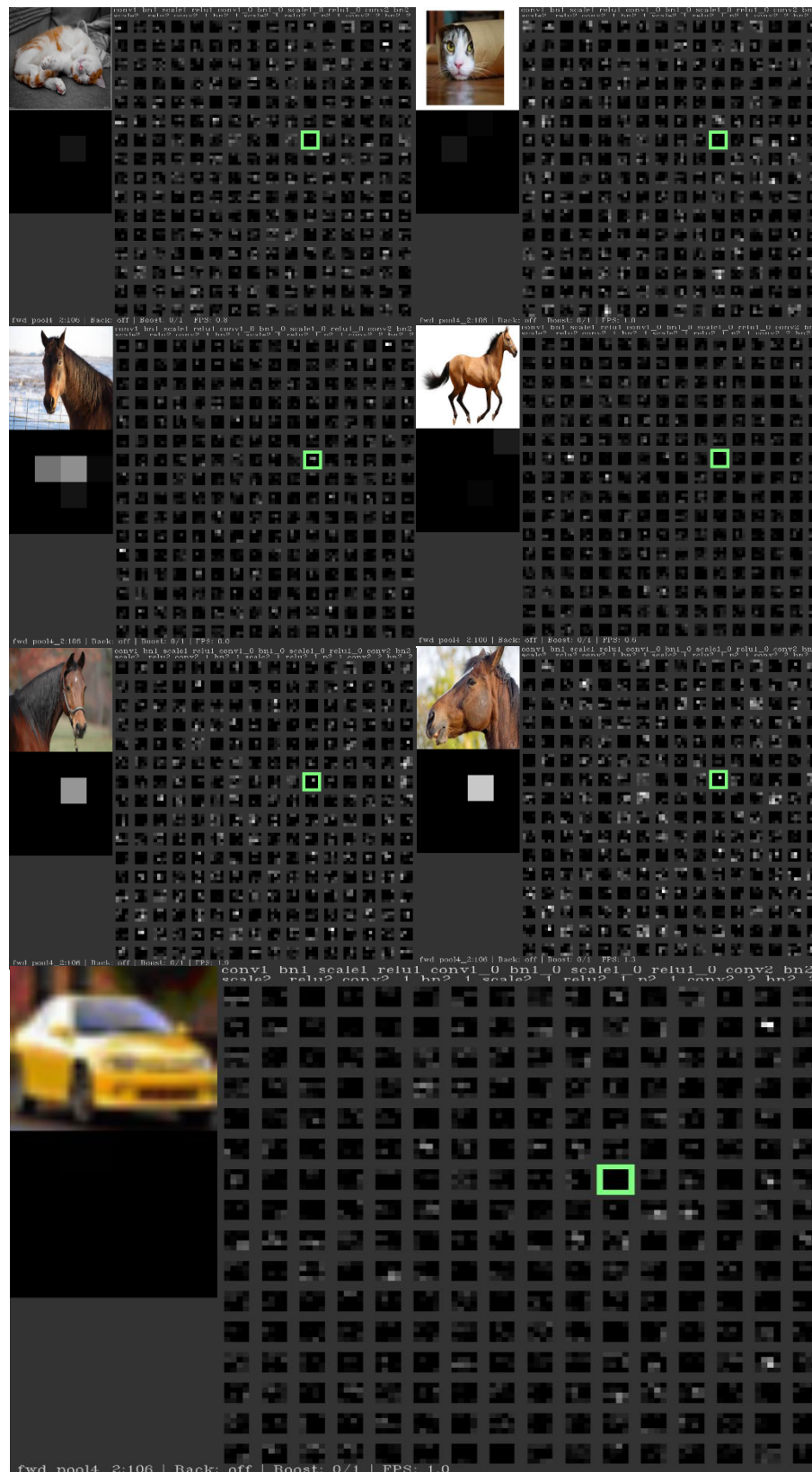
<sup>5</sup> <https://github.com/yosinski/deep-visualization-toolbox>

Figure 7 showing some cat images with a lot of deformations and also a drawing of animal



An interesting point in the above picture lies in the black dog like drawing and the interesting job the network does on the weird drawing we drew! We intentionally drew a figure that does look like to several categories and thus wanted to test how it looks like to the network and whether network uses sensible features to distinguish each class. Interestingly the network tries its best and classifies the image according to the prominent features it finds in the picture. Similarity to the animals in the first for predictions and then a truck at the end! The circled shape of the weird animal's legs is used as an indication of the truck!

Figure 8 showing a head detector the network learned which responds when seeing a head in images of animals it has never seen. The same detector does not activate when confronted with an image of a car!





## Extended Results:

### ImageNet:

ImageNet[5] dataset includes images of 1000 classes, and is split into three sets: 1.2M training images, 50K validation images, and 100K testing images. The classification performance is evaluated using two measures: the top-1 and top-5 error. We used the same architecture without any dropout and didn't tune any parameters. We just used plain SGD to see how it performs with a simple learning policy. Table 6 shows the latest result until 300K iteration from the ongoing test. Unlike others that used techniques such as scale jittering and multi crop and dense evaluation in training and testing phases, no data-augmentation is used.

Table 6 showing ImageNet2012 results

Method	T1/T5
AlexNet(60M)	57.2/80.3
VGGNet16(138M)	70.5
GoogleNet(8M)	68.7
Wide ResNet(11.7M)	69.6/89.07
<b>Our architecture(5.4M)*</b>	<b>60.97/83.54</b>
<b>ImageNet(310K) **</b>	<b>37.34/63.4</b>

\*Trained only for 300K (still training)

### CIFAR10 extended results:

Method	Accuracy	#Params
VGGNet(original 16L)[48]	91.4	138m
VGGNET(Enhanced)[48]*	92.45	138m
ResNet-110[4]*	93.57	1.7m
ResNet-1202[4]	92.07	10.2
Stochastic depth-110L[38]	94.77	1.7m
Stochastic depth-1202L[38]	95.09	10.2m
Wide Residual Net[31]	95.19	11m
Wide Residual Net[31]	95.83	36m
Highway Network[30]	92.40	-
FitNet[42]	91.61	1M
SqueezeNet[44]-(tested by us)	79.58	1.3M
ALLCNN[43]	92.75	-
Fractional Max-pooling* (1 tests)[13]	95.50	12M
Max-out(k=2)[12]	90.62	6M
Network in Network[37]	91.19	1M
Deeply Supervised Network[49]	92.03	1M
Batch normalized Max-out Network In Network[50]	93.25	-
All you need is a good init (LSUV)[51]	94.16	-
Generalizing Pooling Functions in Convolutional Neural Networks: [54]	93.95	-
Spatially-sparse convolutional neural networks[58]	93.72	-

Scalable Bayesian Optimization Using Deep Neural Networks[59]	93.63	-
Recurrent Convolutional Neural Network for Object Recognition[56]	92.91	-
RCNN-160[55]	92.91	-
<b>Our Architecture1</b>	<b>94.75</b>	<b>5.4</b>
<b>Our Architecture1 using data augmentation</b>	<b>95.32</b>	<b>5.4</b>

CIFAR100 extended results:

Method	Accuracy
GoogleNet with ELU[19]*	75.72
Spatially-sparse convolutional neural networks[58]	75.7
Fractional Max-Pooling(12M) [13]	73.61
Scalable Bayesian Optimization Using Deep Neural Networks[59]	72.60
All you need is a good init[51]	72.34
Batch-normalized Max-out Network In Network(k=5)[50]	71.14
Network in Network[37]	64.32
Deeply Supervised Network[49]	65.43
ResNet-110L[4]	74.84
ResNet-1202L[4]	72.18
WRN[31]	77.11/79.5
Highway[30]	67.76
FitNet[42]	64.96
<b>Our Architecture 1</b>	<b>73.45</b>
<b>Our Architecture 2</b>	<b>74.86</b>

- Achieved using several data-augmentation tricks

## Flops and Parameter Comparison:

	MACC	COMP	ADD	DIV	EXP	Activations	Params	SIZE(MB)
<b>SimpleNet</b>	652 M	0.838M	10	10	10 M	1 M	5 M	20.9
<b>SqueezeNet</b>	861 M	10 M	226K	1.51M	1K	13 M	1 M	4.7
<b>Inception v4*</b>	12270 M	21.9 M	5.34M	897K	1K	73 M	43 M	163
<b>Inception v3*</b>	5710 M	16.5 M	2.59M	1.71M	11K	33 M	24 M	91
<b>Inception-ResNetv2*</b>	9210 M	17.6 M	2.36M	1K	1K	74 M	32 M	210
<b>ResNet-152</b>	11300 M	22.33M	35.27M	22.03M	1K	100.26M	60.19M	230
<b>ResNet-50</b>	3870 M	10.9 M	1.62M	1.06M	1K	47 M	26 M	97.70
<b>AlexNet</b>	1140 M	1.77 M	478K	955K	478K	2 M	62 M	217.00
<b>GoogleNet</b>	1600 M	16.1 M	883K	166K	833K	10 M	7 M	22.82
<b>Network in Network</b>	1100 M	2.86 M	370K	1K	1K	3.8 M	8 M	29
<b>VGG16</b>	15740 M	19.7 M	1K	1K	1K	29 M	138 M	512.2

\*Inception v3, v4 and Inception-ResNetV2 did not have any Caffe model, so we reported their size related information from mxnet<sup>6</sup> and tensorflow<sup>7</sup> respectively. Inception-ResNet-V2 would take 60 days of training with 2 Titan X to achieve the reported accuracy<sup>8</sup>

---

<sup>6</sup> <https://github.com/dmlc/mxnet-model-gallery/blob/master/imagenet-1k-inception-v3.md>

<sup>7</sup> <https://github.com/tensorflow/models/tree/master/slim>

<sup>8</sup> [https://github.com/revilokeb/inception\\_resnetv2\\_caffe](https://github.com/revilokeb/inception_resnetv2_caffe)