LET'S KEEP IT SIMPLE, USING SIMPLE ARCHITECTURES TO OUTPERFORM DEEPER AND MORE COMPLEX ARCHITECTURES

Seyyed Hossein Hasanpour[*]
Department of Computer Science
Islamic Azad University, Science and Research branch
Tehran, 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)
Tehran, 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, with which we empirically show, a well-crafted yet simple and reasonably deep architecture can perform on par with deeper and more complex architectures. 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

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 Guo et al. (2015). What has made this enormous success possible is the ability of deep architectures to do feature learning automatically, eliminating the need for a feature engineering stage. In this stage which is the most important one amongst others, the preprocessing pipelines and data transformation are designed using human ingenuity and prior knowledge Bengio et al. (2013) and has a profound effect on the end result. It is highly dependent on the level of engineers experience and expertise and if done poorly the result would be disappointing. It however, cannot scale or be generalized for other tasks well. Furthermore, in deep learning methods, instead of manual and troublesome feature engineering, feature learning is carried out automatically in an efficient way. Deep methods also scale very well to different tasks of different essence. This proved extremely successful which one can say by looking

[*] Corresponding author

at the diverse fields it has been being used.
CNNs, have been one of the most popular deep learning methods and also a major winner in many computer vision and natural language processing related tasks lately Simonyan & Zisserman (2014); Szegedy et al. (2015); He et al. (2015b). Since CNNs take into account the locality of the input, they can find different levels of correlation through a hierarchy of consecutive application of convolution filters. This way they are able to find and exploit different levels of abstractions in the input data and using this perform very well on both coarse and fine level details. Therefore the depth of a CNN plays an important role in the discriminability power the network offers. The deeper the better.
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) Russakovsky et al. (2015) has achieved its success using a very deep architecture of 152 layers He et al. (2015b). The runner up also deploys a deep architecture of 22 layers Szegedy et al. (2015). This trend has proved useful in the natural language processing benchmarks as well Sercu et al. (2015).
While this approach has been useful, there are some inevitable issues that arise when the network gets more complex. Computation and memory usage cost and overhead is one of the critical issues that is caused by the excessive effort put on making networks deeper and more complex in order to make them perform better. This has a negative effect on the expansion of methods and applications utilizing deep architectures. Despite the existence of various techniques for improving the learning algorithm, such as different initialization algorithms Glorot & Bengio (2010); He et al. (2015a); Hinton et al. (2015); Mishkin & Matas (2015); Saxe et al. (2013), normalization and regularization method and techniques Graham (2014a); Goodfellow et al. (2013); Ioffe & Szegedy (2015); Wager et al. (2013); Wan et al. (2013), non-linearities Clevert et al. (2015); He et al. (2015a); Maas et al. (2013); Nair & Hinton (2010) and data-augmentation tricks Alex et al. (2012); Graham (2014a); Simonyan & Zisserman (2014); Wu et al. (2015); Xu et al. (2015), 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 Goodfellow et al. (2013); Ioffe & Szegedy (2015). 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.
The main contribution of our work is the proposal of a simple architecture, with minimum reliance on new features that outperforms almost all deeper architectures with 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 Han et al. (2015) and thus its memory consumption can be decreased drastically. We intentionally imposed some limitation on ourselves when designing the architecture and tried to create a mother architecture with minimum reliance on new features proposed recently, to show the effectiveness of 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. However a fundamentally badly designed architecture would not be able to harness the advantages because of its inherent [flawed] design, therefore we also provide the intuitions behind the overall design choices.

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 design of 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 and acknowledgment is covered in section 6.

2 Related Works

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 Fukushima (1979; 1980); Ivakhnenko (1971). With the advent of deep learning methods, this desire manifested itself in the form of creating deeper and more complex architectures Ciresan et al. (2010); Cirean et al. (2011); CireAn et al. (2012); He et al. (2015b); Alex et al. (2012); Simonyan & Zisserman (2014); Srivastava et al. (2015); Szegedy et al. (2015); Zagoruyko & Komodakis (2016). This was first attempted and popularized by Ciresan et al. (2010) training a 9 layer MLP on GPU which was then practiced by other researchers Cirean et al. (2011); CireAn et al. (2012); Ciregan et al. (2012); He et al. (2015b); Alex et al. (2012); Simonyan & Zisserman (2014); Srivastava et al. (2015); Szegedy et al. (2015); Zagoruyko & Komodakis (2016).

In 2012 Alex et al. (2012) created a deeper version of LeNet5 Lecun et al. (1998) with 8 layers called AlexNet, unlike LeNet5, It had local contrast normalization, ReLU Nair & Hinton (2010) nonlinearity instead of Tanh, and a new regularization layer called Dropout Hinton et al. (2012), this architecture achieved state of the art on ILSVRC 2012. The same year, Le (2013) trained a gigantic network with 1 billion parameters, their work was later proceeded by Coates et al. (2013) which an 11 billion parameter network was trained. Both of them were ousted by much smaller network AlexNet Alex et al. (2012).

In 2013 Lin et al. (2013) released their 12 layer, NIN architecture, they built micro neural networks into convolutional neural network using 1×1 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 Simonyan & Zisserman (2014) introduced several architectures, with increasing depth, 11 being the shallowest and 19 the deepest, they used 3×3 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 Szegedy et al. (2015) was released, with 56 convolutional layers making up a 22 modular layered network, their architecture was made up of convolutional layers with 1×1, 3×3 and 5×5 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 3×3 conv layers with 128 kernels instead of the previous 5×5 layers, they also used a technique called Batch-Normalization Ioffe & Szegedy (2015) for reducing internal covariate shift. This technique provided improvements in several sections which is explained thoroughly in Ioffe & Szegedy (2015). 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. (2015a), 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 Simonyan & Zisserman (2014) architecture and achieve state of the art result on ImageNet. At the end of 2015, they proposed a deep architecture of 152 layers, called Residual Network (ResNet) He et al. (2015b) 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 Simonyan & Zisserman (2014) which they previously used. They investigated architectures with 1000 layers as well. Later Huang et al. (2016) 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 CIFAR10 dataset.

Prior to the residual network, Srivastava et al. (2015) released their Long Short Term Memory (LSTM) recurrent network inspired highway networks in which they used the initialization method proposed by He et al. (2015a) 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 networks 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. (2016) 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 Russakovsky et al. (2015) 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, Zagoruyko & Komodakis (2016) ran a detailed experiment on residual nets He et al. (2015b) 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 Srivastava et al. (2015) 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 Bucilu et al. (2006), 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 (2014). 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. (2015) 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. (2015), Romero et al. (2014) 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)Hinton et al. (2015) 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. (2015) 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 2014 Springenberg et al. (2014) 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. Later in 2015, they proposed different versions of their architecture and studied their characteristics, and 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. (2016) released their paper in which they proposed a novel architecture called, SqueezeNet, a small CNN architecture that achieves AlexNet-level 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[1]. 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 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 and Design intuition

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

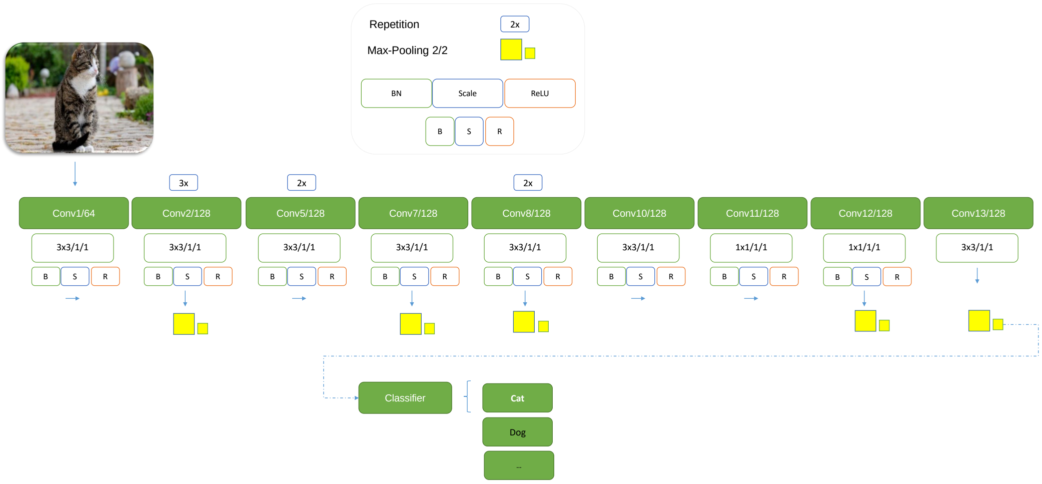

Figure 1: Showing the base architecture with no drop-out

The only layers which do not use 3 × 3 kernels are 11th and 12th layers, these layers, utilize 1 × 1 convolutional kernels. Feature-map down-sampling is carried out using nonoverlaping 2 × 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.

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:

[1] 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. We also have the state of the art in terms of accuracy/parameters ratio.

Table 1: showing different architectures statistics

Model	AlexNet	GoogleNet	ResNet152	VGG16	NIN	**SimpleNet**
Param	60M	7M	60M	138M	7.6M	**5.4M**
OP	7.27G	16.04G	11.3G	154.7G	11.06G	**652M**
Storage (MB)	217	40	230	512.24	29	**20**

3.0.1 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 result in an inefficient network imposing unwanted 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 conned shape, that's one of the reasons why deeper is better) Lecun et al. (1998). 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.

3.0.2 Homogeneous Groups of Layers

Instead of thinking in layers, think and design in group of homogeneous layers. The idea is to have several homogeneous groups of layers, each with gradually more width. The symmetric and homogeneous design, allows to easily manage the number of parameters a network will withhold and also provide better information pools for each semantic level.

3.1 Local Correlation Preservation

Preserve locality information throughout the network as much as possible by avoiding 1×1 kernels in early layers. The corner stone of CNN success lies in local correlation preservation. Avoid using 1×1 kernels or fully connected layers where locality of information matters. This includes exclusively the early layers in the network. 1×1 kernels have several desirable characteristics such as increasing networks non-linearity and feature fusion Lin et al. (2013) 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 1×1 kernels at the end of the network or if one intends on using tricks such as bottleneck employed by GoogleNet Szegedy et al. (2015) and ResNet He et al. (2015b), use more layers with skip connections to compensate the loss in information. It is suggested to replace 1×1 kernels with 2×2 if one plans on using them other than the end of the network. Using 2×2 kernels both help to reduce the number of parameters and also to retain neighborhood information.

3.2 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.

3.3 Maximum Performance Utilization

Use 3×3, and follow established industrial trends. For an architecture to be easily usable 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 3×3 kernels, apart from already known benefits Simonyan & Zisserman (2014), allows to achieve a substantial boost in performance when using NVIDIA's cuDNNv5.$\times$ library. A speed up of about $2.7\times$ compared to the former v4 version [2]. 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 3×3 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. 3×3) which will still result in the same effective receptive field and also more nonlinearity, making them a better choice over larger kernels.

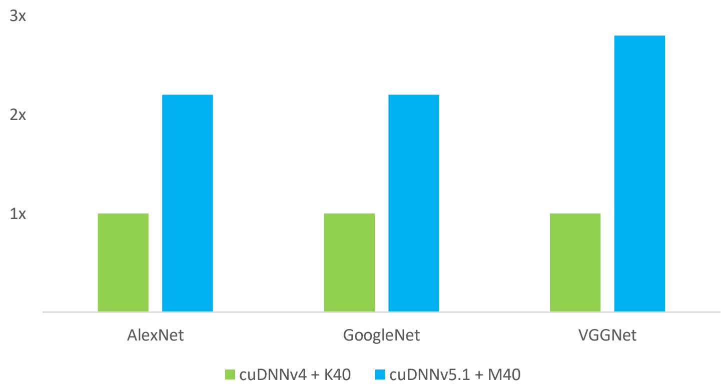

Figure 2: Using 3×3 kernels results in $2.7\times$ faster training when using cuDNNv5.$\times$

3.4 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.

[2] https://developer.nvidia.com/cudnn-whatsnew

3.5 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 3×3 kernel against a 5×5 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.

3.6 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 learned 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.

3.7 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. Ultimately, 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 Krizhevsky & Hinton (2009), SVHN Netzer et al. (2011) and MNIST Lecun et al. (1998) 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 , SVHN 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 Jia et al. (2014) 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 Krizhevsky & Hinton (2009) 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: Top CIFAR10/100 results.

Method	**#Params**	**CIFAR10**	**CIFAR100**
VGGNet(16L) Zagoruyko (2015)/Enhanced	138m	91.4 / 92.45	-
ResNet-110L / 1202L He et al. (2015b) *	1.7/10.2m	93.57 / 92.07	74.84/72.18
SD-110L / 1202L Huang et al. (2016)	1.7/10.2m	94.77 / 95.09	75.42 / -
WRN-(16/8)/(28/10) Zagoruyko & Komodakis (2016)	11/36m	95.19 / 95.83	77.11/79.5
Highway Network Srivastava et al. (2015)	N/A	92.40	67.76
FitNet Romero et al. (2014)	1M	91.61	64.96
FMP* (1 tests) Graham (2014a)	12M	95.50	73.61
Max-out(k=2) Goodfellow et al. (2013)	6M	90.62	65.46
Network in Network Lin et al. (2013)	1M	91.19	64.32
DSN Lee et al. (2015)	1M	92.03	65.43
Max-out NIN Jia-Ren Chang (2015)	-	93.25	71.14
LSUV Dmytro Mishkin (2016)	N/A	94.16	N/A
SimpleNet-Arch 1∗	5.48M	**94.75**	-
SimpleNet-Arch 2 †	5.48M	**95.32**	**73.42**

*Note that the Fractional Max Pooling Graham (2014a) 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.

4.2 MNIST

The MNIST dataset Lecun et al. (1998) 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: Top MNIST results

Method	Error rate
DropConnectWan et al. (2013)**	0.21%
Multi-column DNN for Image ClassicationCiregan et al. (2012)**	0.23%
APACSato et al. (2015)**	0.23%
Generalizing Pooling Functions in CNNLee et al. (2016)**	0.29%
Fractional Max-PoolingGraham (2014a)**	0.32%
Batch-normalized Max-out NIN Jia-Ren Chang (2015)	0.24%
Max-out network (k=2) Goodfellow et al. (2013)	0.45%
Network In Network Lin et al. (2013)	0.45%
Deeply Supervised Network Lee et al. (2015)	0.39%
RCNN-96 Liang & Hu (2015)	0.31%
SimpleNet *	**0.25%**

*Note that we didn't intend on achieving the state of the art performance here as 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

4.3 SVHN

The SVHN dataset Netzer et al. (2011) 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 Huang et al. (2016); Goodfellow et al. (2013); Lin et al. (2013) 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: Top SVHN results.

Method	Error rate
Network in NetworkLin et al. (2013)	2.35
Deeply Supervised NetLee et al. (2015)	1.92
ResNetHe et al. (2015b) (reported by Huang et al. (2016) (2016))	2.01
ResNet with Stochastic DepthHuang et al. (2016)	1.75
Wide ResNetZagoruyko & Komodakis (2016)	1.64
SimpleNet	**1.79**

4.4 Extended test

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 CIFAR10/100 datasets.

Model	Param	CIFAR10	CIFAR100
SimpleNet	**310K - 460K**	**91.98 - 92.33**	**64.68 - 66.82**
Maxout Goodfellow et al. (2013)	6M	90.62	65.46
DSN Lee et al. (2015)	1M	92.03	65.43
ALLCNN Springenberg et al. (2014)	1.3M	92.75	66.29
dasNet Stollenga et al. (2014)	6M	90.78	66.22
ResNet He et al. (2015b) (Depth32, tested by us)	475K	91.6	67.37
WRN Zagoruyko & Komodakis (2016)	600K	93.15	69.11
NIN Lin et al. (2013)	1M	91.19	—

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 express our deep gratitude to Dr. Ali Diba the CTO of Sensifai for his great help and cooperation in this work. We also would like to express our great appreciation to Dr. Hamed Pirsiavash for his insightful comments and constructive suggestions. We would also like to thank Dr. Reza Saadati, and Dr. Javad Vahidi for their valuable help in early stage of the work.

References

Alex, Krizhevsky, Sutskever, Ilya, and Geoffrey, E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

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

Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Bucilu, Cristian, Caruana, Rich, and Niculescu-Mizil, Alexandru. Model compression. In *SIGKDD*, pp. 535–541. ACM, 2006. ISBN 1595933395.

Cirean, Dan, Meier, Ueli, Masci, Jonathan, and Schmidhuber, Jrgen. A committee of neural networks for traffic sign classification. In *IJCNN*, pp. 1918–1921. IEEE, 2011. ISBN 1424496357.

CireAn, Dan, Meier, Ueli, Masci, Jonathan, and Schmidhuber, Jrgen. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012. ISSN 0893-6080.

Ciregan, Dan, Meier, Ueli, and Schmidhuber, Jrgen. Multi-column deep neural networks for image classification. In *CVPR*, pp. 3642–3649. IEEE, 2012. ISBN 1467312266.

Ciresan, Dan Claudiu, Meier, Ueli, Gambardella, Luca Maria, and Schmidhuber, Jrgen. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010. ISSN 0899-7667.

Clevert, Djork-Arn, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Coates, Adam, Huval, Brody, Wang, Tao, Wu, David, Catanzaro, Bryan, and Andrew, Ng. Deep learning with cots hpc systems. In *ICML*, pp. 1337–1345, 2013.

Dmytro Mishkin, Jiri Matas. All you need is a good init. In *ICLR*, 2016.

Fukushima, Kunihiko. Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *ELECTRON. & COMMUN. JAPAN*, 62(10):11–18, 1979.

Fukushima, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. ISSN 0340-1200.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pp. 249–256, 2010.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *ICML*, volume 28, pp. 1319–1327, 2013.

Graham, Ben. Fractional max-pooling. ArXiv e-prints, December 2014a, 2014a. URL `http://arxiv.org/abs/1412.6071`.

Graham, Benjamin. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014b.

Guo, Yanming, Liu, Yu, Oerlemans, Ard, Lao, Songyang, Wu, Song, and Lew, Michael S. Deep learning for visual understanding: A review. *Neurocomputing*, 2015. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2015.09.116. URL `http://www.sciencedirect.com/science/article/pii/S0925231215017634`.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, pp. 1026–1034, 2015a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015b. URL `http://arxiv.org/abs/1512.03385`.

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

Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and Weinberger, Kilian. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

Iandola, Forrest N, Han, Song, Moskewicz, Matthew W, Ashraf, Khalid, Dally, William J, and Keutzer, Kurt. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Ivakhnenko, A. G. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971. ISSN 0018-9472.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

Jia-Ren Chang, Yong-Sheng Chen. Batch-normalized maxout network in network. arXiv:1511.02583v1, 2015.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Le, Quoc V. Building high-level features using large scale unsupervised learning. In *ICASSP*, pp. 8595–8598. IEEE, 2013. ISBN 1520-6149.

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

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply supervised nets. *AISTATS*, 2015. URL `http://jmlr.org/proceedings/papers/v38/lee15a.html`.

Lee, Chen-Yu, Gallagher, Patrick W, and Tu, Zhuowen. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *AISTATS*, 2016.

Liang, Ming and Hu, Xiaolin. Recurrent convolutional neural network for object recognition. In *CVPR*, pp. 3367–3375, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *CoRR*, abs/1312.4400, 2013. URL `http://arxiv.org/abs/1312.4400`.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, volume 30, 2013.

Mishkin, Dmytro and Matas, Jiri. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814, 2010.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop*, 2011.

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

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. Imagenet large scale visual recognition challenge. In *ICCV*, volume 115, pp. 211–252, 2015.

Sato, Ikuro, Nishimura, Hiroki, and Yokoi, Kensuke. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*, 2015.

Saxe, Andrew M, McClelland, James L, and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

Sercu, Tom, Puhrsch, Christian, Kingsbury, Brian, and LeCun, Yann. Very deep multilingual convolutional neural networks for lvcsr. ArXiv e-prints, September 2015., 2015. URL `http://arxiv.org/abs/1509/08967`.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Ali, Mostofa, and Adams, Ryan P. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin A. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL `http://arxiv.org/abs/1412.6806`.

Srivastava, Rupesh Kumar, Greff, Klaus, and Schmidhuber, Jrgen. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Stollenga, Marijn F, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Jürgen. Deep networks with internal selective attention through feedback connections. In *NIPS*, pp. 3545–3553, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *CVPR*, pp. 1–9, 2015.

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alex. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

Wager, Stefan, Wang, Sida, and Liang, Percy S. Dropout training as adaptive regularization. In *NIPS*, pp. 351–359, 2013.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Cun, Yann L, and Fergus, Rob. Regularization of neural networks using dropconnect. In *ICML*, pp. 1058–1066, 2013.

Wu, Ren, Yan, Shengen, Shan, Yi, Dang, Qingqing, and Sun, Gang. Deep image: Scaling up image recognition. *CoRR*, abs/1501.02876, 2015. URL `http://arxiv.org/abs/1501.02876`.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Yosinski, Jason, Clune, Jeff, Nguyen, Anh, Fuchs, Thomas, and Lipson, Hod. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Zagoruyko, Sergey. 92.45% on cifar-10 in torch. 2015.

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

A Appendix

A.1 Extended Results

In this section the extended results pertaining to CIFAR10 and CIFAR100 are provided along with early results on ImageNetRussakovsky et al. (2015) dataset. ImageNet 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 use techniques such as scale jittering and multi-crop and dense evaluation in training and testing phases, no data-augmentation is used in achieving the following results.

Table 6: Showing some of major architectures results on ImageNet

Method	**T1/T5 Accuracy Rate**
AlexNet(60M)Alex et al. (2012)	57.2/80.3
VGGNet16(138M)Simonyan & Zisserman (2014)	70.5
GoogleNet(8M) Szegedy et al. (2015)	68.7
Wide ResNet(11.7M)Zagoruyko & Komodakis (2016)	69.6/89.07
SimpleNet(5.4M)	**60.97/83.54**
SimpleNet(310K)	**37.34/63.4**

Table 7: CIFAR10 extended results

Method	**Accuracy**	**#Params**
VGGNet(16L)Zagoruyko (2015)	91.4	138m
VGGNET(Enhanced-16L)Zagoruyko (2015)*	92.45	138m
ResNet-110He et al. (2015b)*	93.57	1.7m
ResNet-1202He et al. (2015b)	92.07	10.2m
Stochastic depth-110LHuang et al. (2016)	94.77	1.7m
Stochastic depth-1202LHuang et al. (2016)	95.09	10.2m
Wide Residual NetZagoruyko & Komodakis (2016)	95.19	11m
Wide Residual NetZagoruyko & Komodakis (2016)	95.83	36m
Highway NetworkSrivastava et al. (2015)	92.40	-
FitNetRomero et al. (2014)	91.61	1M
SqueezNetIandola et al. (2016)-(tested by us)	79.58	1.3M
ALLCNNSpringenberg et al. (2014)	92.75	-
Fractional Max-pooling* (1 tests)Graham (2014a)	95.50	12M
Max-out(k=2)Goodfellow et al. (2013)	90.62	6M
Network in NetworkLin et al. (2013)	91.19	1M
Deeply Supervised NetworkLee et al. (2015)	92.03	1M
Batch normalized Max-out NINJia-Ren Chang (2015)	93.25	-
All you need is a good init (LSUV)Dmytro Mishkin (2016)	94.16	-
Generalizing Pooling Functions in CNNLee et al. (2016)	93.95	-
Spatially-Sparse CNNsGraham (2014b)	93.72	-
Scalable Bayesian Optimization Using DNNSnoek et al. (2015)	93.63	-
Recurrent CNN for Object RecognitionLiang & Hu (2015)	92.91	-
RCNN-160Liang & Hu (2015)	92.91	-
SimpleNet-Arch1	94.75	5.4m
SimpleNet-Arch1 using data augmentation	95.32	5.4m

Table 8: CIFAR100 extended results

Method	Accuracy
GoogleNet with ELUClevert et al. (2015)*	75.72
Spatially-sparse CNNsGraham (2014b)	75.7
Fractional Max-Pooling(12M) Graham (2014a)	73.61
Scalable Bayesian Optimization Using DNNsSnoek et al. (2015)	72.60
All you need is a good initDmytro Mishkin (2016)	72.34
Batch-normalized Max-out NIN(k=5)Jia-Ren Chang (2015)	71.14
Network in NetworkLin et al. (2013)	64.32
Deeply Supervised NetworkLee et al. (2015)	65.43
ResNet-110LHe et al. (2015b)	74.84
ResNet-1202LHe et al. (2015b)	72.18
WRNZagoruyko & Komodakis (2016)	77.11/79.5
HighwaySrivastava et al. (2015)	67.76
FitNetRomero et al. (2014)	64.96
SimpleNet-Arch1	73.45
SimpleNet-Arch2	74.86

*Achieved using several data-augmentation tricks

Table 9: Flops and Parameter Comparison of Models trained on ImageNet

Model	MACC	COMP	ADD	DIV	Activations	Params	SIZE(MB)
SimpleNet	1.9G	1.82M	1.5M	1.5M	6.38M	6.4M	24.4
SqueezeNet	861.34M	9.67M	226K	1.51M	12.58M	1.25M	4.7
Inception v4*	12.27G	21.87M	53.42M	15.09M	72.56M	42.71M	163
Inception v3*	5.72G	16.53M	25.94M	8.97M	41.33M	23.83M	91
Incep-Resv2*	13.18G	31.57M	38.81M	25.06M	117.8M	55.97M	214
ResNet-152	11.3G	22.33M	35.27M	22.03M	100.11M	60.19M	230
ResNet-50	3.87G	10.89M	16.21M	10.59M	46.72M	25.56M	97.70
AlexNet	7.27G	17.69M	4.78M	9.55M	20.81M	60.97M	217.00
GoogleNet	16.04G	161.07M	8.83M	16.64M	102.19M	7M	40
NIN	11.06G	28.93M	380K	20K	38.79M	7.6M	29
VGG16	154.7G	196.85M	10K	10K	288.03M	138.36M	512.2

*Inception v3, v4 did not have any Caffe model, so we reported their size related information from MXNet and Tensorflow respectively. Inception-ResNet-V2 would take 60 days of training with 2 Titan X to achieve the reported accuracy. Statistics are obtained using `http://dgschwend.github.io/netscope`

A.2 Generalization Samples

In order to see how well the model generalizes, and whether it was able to develop robust features, we tried some images that the network has never faced and used them with a model trained on CIFAR10 dataset. As the results show, the network classifies them correctly despite the fact that they are very different from the images used for training. These visualizations are done using Deep Visualization Toolbox by Yosinski et al. (2015) and early un-augmented version of SimpleNet.

An interesting point in the figure 4 lies in the black dog/cat like drawing and the interesting predictions the network does on the strange drawing we drew! We intentionally drew a figure that does look like several categories inside CIFAR10 dataset, and thus wanted to test how it looks like to the network and whether the network uses sensible features to distinguish between each class. Interestingly the network tries its best and classifies the image according to the prominent features it finds in the picture. The similarity to some animals present in the dataset is manifested in the first four predictions and then a truck at the end denotes the circular shape of the animal's legs might have been used as an indication of the existence of the truck! Suggesting the network is trying to use prominent features to

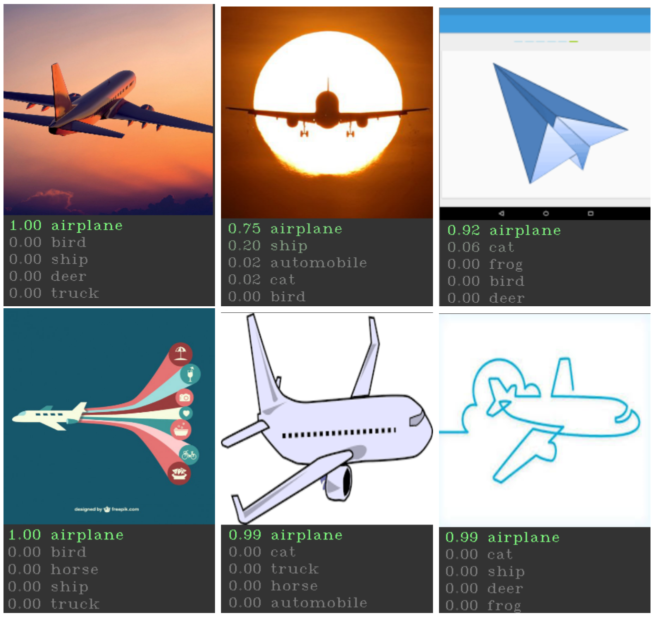

Figure 3: Some airplanes pictures with completely different appearances that the network classifies very well.

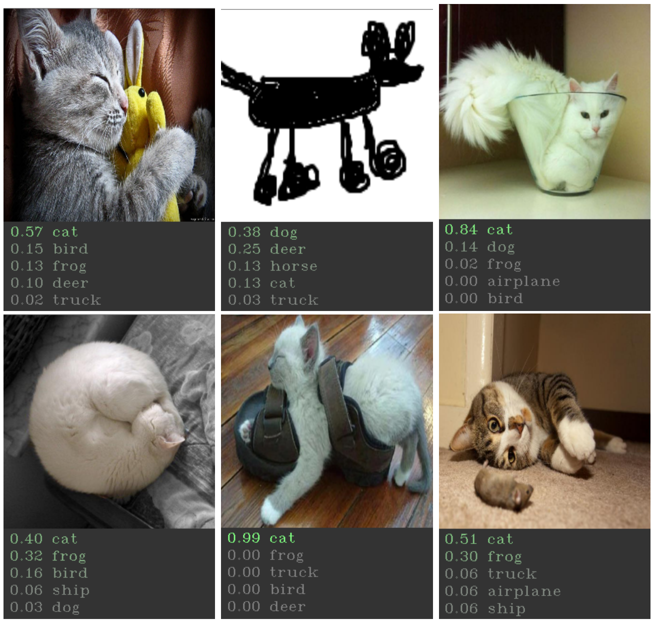

Figure 4: showing some cat images with a lot of deformations and also a drawing of animal.

identify each class rather than some random features. Investigating the internals of the network also shows, such predictions are because of a well developed feature combinations, by which the network performs its deduction. Figure 5 shows the network has developed a proper feature to distinguish the head/shoulder in the input, and a possible deciding factor by which to distinguish between animals and non animals. As it can be seen from the samples, while the results are very encouraging and in high confidence, they are still far from prefect. This observation may suggest 3 possible reasoning: 1) The network does not have the capability needed to perfectly deduce as we expect. 2) More data is needed for the network to develop better features, a small dataset such as CIFAR10 with no data augmentation is not simply enough to provide such capability we expect. 3) The current optimization process that we employ to train our deep architectures is insufficient and or incapable of providing such capability easily or at all. Apart from the current imperfections, results show that even a simple architecture, when properly devised, can perform decently.

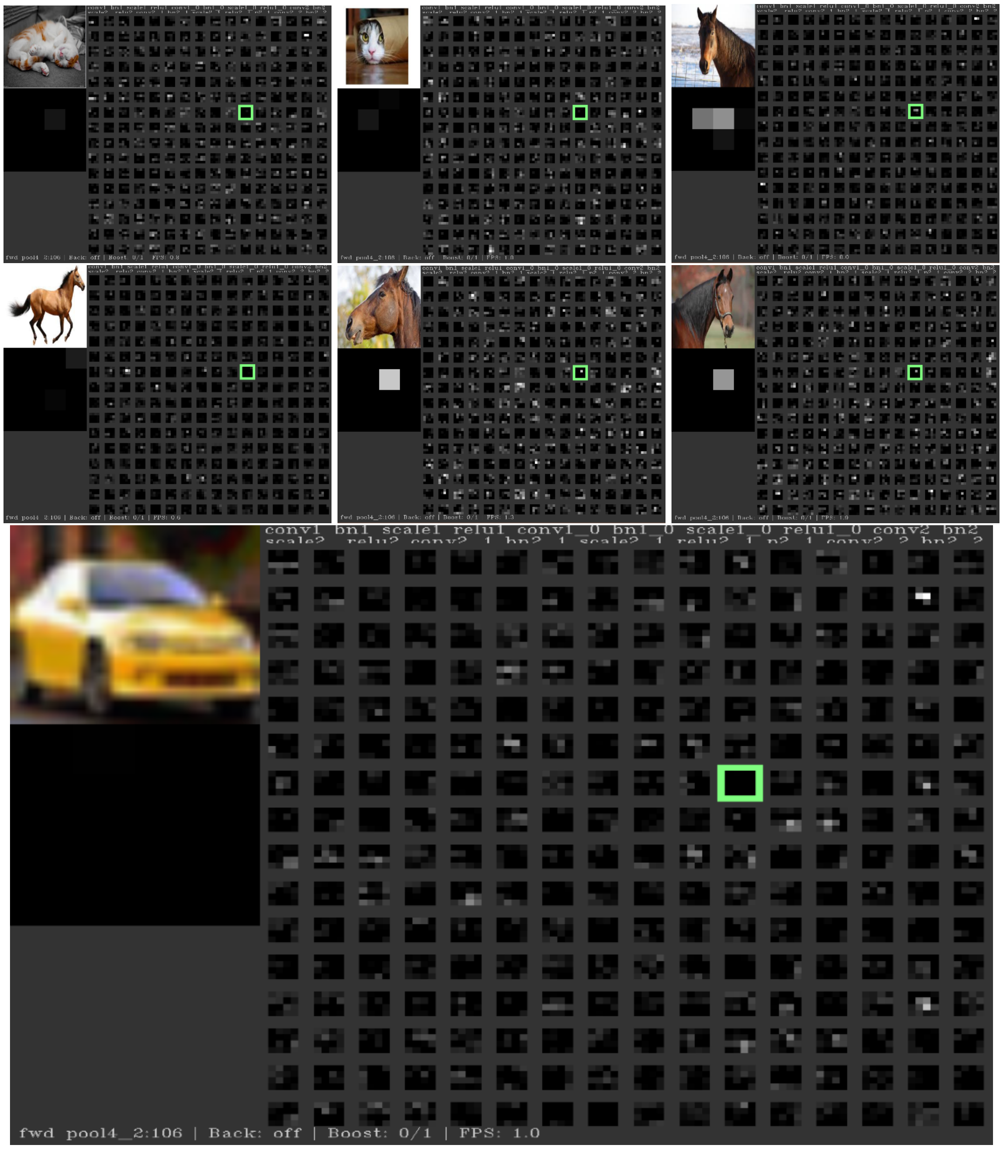
conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 1.0

Figure 5: Showing a head detector the network has learned, which responds when facing a head in images of animals it has never seen. The same detector does not activate when confronted with an image of a car!