
Deep Image: Scaling up Image Recognition

Ren Wu
Shengen Yan
Yi Shan
Qingqing Dang
Gang Sun

WUREN@BAIDU.COM
YANSHENG@BAIDU.COM
SHANYI@BAIDU.COM
DANGQINGQING@BAIDU.COM
SUNGANG01@BAIDU.COM

Abstract

We present a state-of-the-art image recognition system, Deep Image, developed using end-to-end deep learning. The key components are a custom-built supercomputer dedicated to deep learning, a highly optimized parallel algorithm using new strategies for data partitioning and communication, larger deep neural network models, novel data augmentation approaches, and usage of multi-scale high-resolution images. On one of the most challenging computer vision benchmarks, the ImageNet classification challenge, our system has achieved the best result to date, with a top-5 error rate of 5.33%, a relative 20.0% improvement over the previous best result.

1. Introduction

On May 11th, 1997, IBM's Deep Blue achieved a historic victory by defeating world chess champion Gary Kasparov in a six-game match (Campbell et al., 2002). It came as a surprise to some, but with the correct algorithms, chess performance is a function of computational power (Condon & Thompson, 1982), (Hyatt et al., 1990), (Kuszmaul, 1995). Today, history is repeating itself: simple, scalable algorithms, given enough data and computational resources, dominate many fields, including visual object recognition (Ciresan et al., 2010), (Krizhevsky et al., 2012), (Szegedy et al., 2014), speech recognition (Dahl et al., 2012), (Hannun et al., 2014) and natural language processing (Collobert & Weston, 2008), (Mnih & Hinton, 2007), (Mnih & Hinton, 2008).

Although neural networks have been studied for many decades, only recently have they come into their own, thanks to the availability of larger training data sets along

with increased computation power through heterogeneous computing (Coates et al., 2013).

Because computational power is so important to progress in deep learning, we built a supercomputer designed for deep learning, along with a software stack that takes full advantage of such hardware. Using application specific hardware-software co-design, we have a highly optimized system. This system enables larger models to be trained on more data, while also reducing turnaround time, allowing us to explore ideas more rapidly.

Still, there are two shortcomings in current deep learning practices. First, while we know that bigger models offer more potential capacity, in practice, the size of model is often limited by either too little training data or too little time for running experiments, which can lead both to overfitting or underfitting. Second, the data collected is often limited to a specific region of the potential example space. This is especially challenging very large neural networks, which are subject to overfitting.

In this work, we show how aggressive data augmentation can prevent overfitting. We use data augmentation in novel ways, much more aggressively than previous work (Howard, 2013), (Krizhevsky et al., 2012). The augmented datasets are tens of thousands times larger, allowing the network to become more robust to various transformations.

Additionally, we train on multi-scale images, including high-resolution images. Most previous work (Krizhevsky et al., 2012), (Zeiler & Fergus, 2014) operates on downsized images with a resolution of approximately 256x256. While using downsized images reduces computational costs without losing too much accuracy, we find that using larger images improves recognition accuracy. As we demonstrate, there are many cases where the object size is small, and downsizing simply loses too much information.

Training on higher-resolution images improves the classification accuracy. More importantly, models trained on high-resolution images complement models trained on low-resolution images. Composing models trained on different scales produces results better than any model individually.

In this paper, we detail our custom designed supercomputer for deep learning, as well as our optimized algorithms and software stack built to capitalize on this hardware. This system has enabled us to train bigger neural models, work on higher-resolution images, and use more aggressive data augmentation methods. On the widely studied 1k ImageNet classification challenge, we have obtained the current state-of-the-art result, with a top-5 error rate of 5.33%.

2. Hardware/Software Co-design

It is clear that different classes of algorithms would perform differently on different computing architectures. Graphic processors, or GPUs, often perform extremely well for compute-intensive algorithms. Early work shows that for clustering algorithms, a single GPU offers 10x more performance than top-of-the-line 8 cores workstations, even on very large datasets with more than a billion data points (Wu et al., 2009). A more recent example shows that three GPU servers with 4 GPUs each, rival the same performance of a 1000 nodes (16000 cores) CPU cluster, used by the Google Brain project (Coates et al., 2013).

Modern deep neural networks are mostly trained by variants of stochastic gradient decent algorithms (SGD). As SGDs contain high arithmetic density, GPUs are excellent for this type of algorithms.

Furthermore, we would like to train very large deep neural networks without worrying about the capacity limitation of a single GPU or even a single machine, and so scaling up is a required condition. Given the properties of stochastic gradient decent algorithms, it is desired to have very high bandwidth and ultra low latency interconnects to minimize the communication costs, which is needed for the distributed version of the algorithm.

The result is the custom-built supercomputer, which we call Minwa. It is comprised of 36 server nodes, each with 2 six-core Intel Xeon E5-2620 processors. Each sever contains 4 Nvidia Tesla K40m GPUs and one FDR InfiniBand (56Gb/s) which is a high-performance low-latency interconnection and supports RDMA. The peak single precision floating point performance of each GPU is 4.29TFlops and each GPU has 12GB of memory. Thanks to the GPUDirect RDMA, the InfiniBand network interface can access the remote GPU memory without involvement from the CPU. All the server nodes are connected to the InfiniBand switch. Figure 1 shows the system architecture. The system runs Linux with CUDA 6.0 and MPI MVAPICH2, which also enables GPUDirect RDMA.

In total, Minwa has 6.9TB host memory, 1.7TB device memory, and about 0.6PFlops theoretical single precision peak performance.

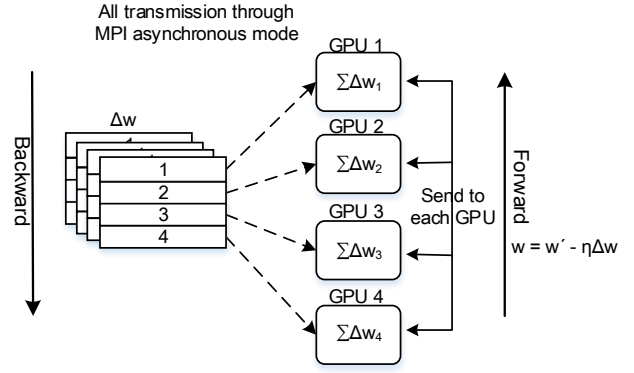


Figure 1. Example of communication strategies among 4 GPUs.

3. Optimization

Our goal is to push for extreme performance from both hardware and software for given tasks. In modern deep convolutional neural networks, convolutional layers account for most of the computation and fully-connected layers account for most of the parameters. We implement two parallelism strategies for our parallel deep neural network framework, namely model-data parallelism and data parallelism. Similar strategies have been proposed in the previous work (Krizhevsky, 2014), (Yadan et al., 2013). However, the previous work mainly focuses on a single server with multiple GPUs or small GPU clusters, so it's hard to extend directly to a large GPU cluster because of the communication bottlenecks. In our work, we focus on optimizing parallel strategies, minimizing data transfers and overlapping the computation and communications. This is needed for approaching the peak performance for the large supercomputers like Minwa.

3.1. Data Parallelism

When all the parameters can be stored in the memory of one GPU, we exploit data parallelism at all layers. In this case, each GPU is responsible for $1/N$ mini-batch of input images and all GPUs work together on the same mini-batch. In one forward-backward pass, the parameters and gradients of all layers are copied on every GPU. All GPUs compute gradients based on local training data and a local copy of weights. They then exchange gradients and update the local copy of weights.

Two strategies have helped us to achieve better parallelism. The first one is the **'butterfly synchronization'** strategy, in which all gradients are partitioned into K parts and each GPU is responsible for its own part. At the end of gradient computation, GPU k receives the k -th part from all other GPUs, accumulates there and then broadcasts the results

back to all GPUs.

Another is the ‘**lazy update**’ strategy. Once the gradients are generated in the backward pass, each GPU sends its generated gradient to the corresponding GPUs in an asynchronous way. This transfer does not need to be synchronized until its corresponding weight parameters need to be used, which only occurs later at the forward pass. This maximizes the overlapping between computation and communication.

Figure 1 shows an example of the transmission on four GPUs. Theoretically, the communication overhead of our data parallel strategy only depends on the size of the model and is independent of the number of GPUs. We also use device memory to cache the training data if there is free memory space on the device after the model is loaded.

3.2. Model-Data Parallelism

While data parallelism works well for the smaller models. It does not work if the model size cannot be fit into the memory of a single GPU. Model-data parallelism is used to address this problem. Here, data parallelism is still used at convolutional layers but fully-connected layers are instead partitioned and distributed to multiple GPUs. This works because convolutional layers have fewer parameters but more computation.

The parameters at convolutional layers are copied on every GPU and all images within one mini-batch are partitioned and assigned to all GPUs, just as we described in Section 3.1. The parameters at fully-connected layers, however, are evenly divided amongst all GPUs. All GPUs then work together to calculate the fully-connected layers and synchronize when necessary. This is similar to the approach presented in (Krizhevsky, 2014), even though we are doing it in a scaled up fashion.

3.3. Scaling Efficiency

We tested the scaling efficiency by training a model for an image classification task. The network has 8 convolutional layers and 3 fully-connected layers followed by a 1000-way softmax. Measured by the epoch time, the scalability efficiency and the speedup of going through images are shown in Figure 2 and Figure 3. For the convenience of observing the scalability of different parallel strategies, we fixed the number of images processed by each GPU to 64 (slices = 64). The time taken for hybrid parallelism and data parallelism with different numbers of GPUs is shown in Figure 2. The data parallelism performs better when the involved GPU number is larger than 16. This is because communication overhead of the data parallel strategy is constant when the size of model is fixed. The speedup is larger with larger batch size as shown in Figure 3. Com-

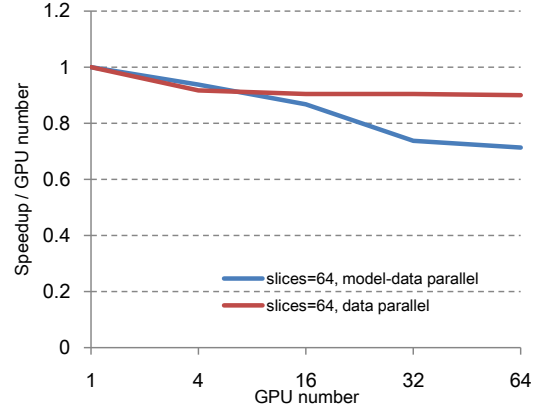


Figure 2. The scalability of different parallel approaches.

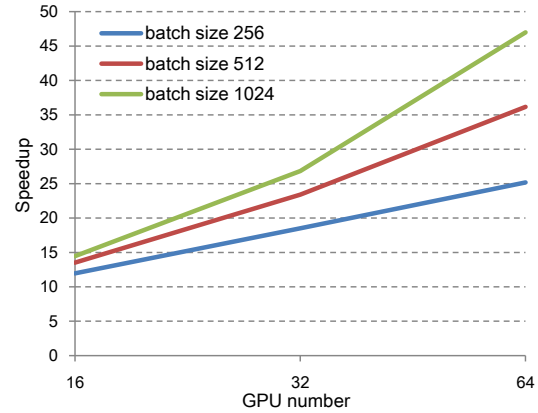


Figure 3. The speedup of going through images.

pared with a single GPU, a 47x speedup of going through images is achieved by using 64 GPUs with a mini-batch size of 1024. As the number of GPUs increases, the total device memory is also increasing, and more data can be cached on the device memory. This is helpful for improving parallel efficiency.

The ultimate test for a parallel algorithm is the convergence time, that is, the wall clock time needed for the network to reach a certain accuracy level. Figure 4 compares the time needed for the network to reach 80% accuracy using various numbers of GPUs. Note that a 24.7x speedup is obtained by using 32 GPUs.

4. Training Data

4.1. Data Augmentation

The phrase “the more you see, the more you know” is true for humans as well as neural networks, especially for mod-

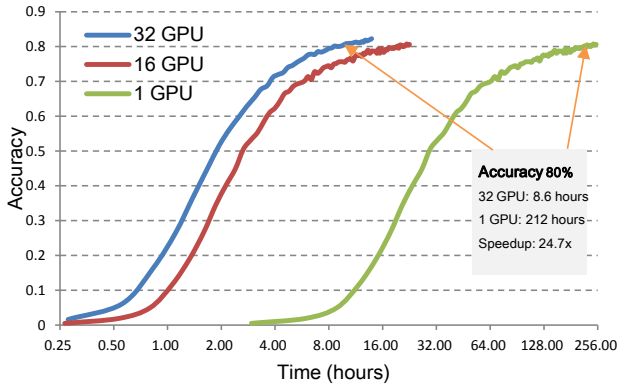


Figure 4. Validation set accuracy for different numbers of GPUs.

ern deep neural networks.

We are now capable of building very large deep neural networks up to hundreds of billions parameters thanks to dedicated supercomputers such as Minwa. The available training data is simply not sufficient to train a network of this size. Additionally, the examples collected are often just in a form of ‘good’ data - or a rather small and heavily biased subset of the possible space. It is desired to show the network with more data with broad coverage.

The authors of this paper believe that data augmentation is fundamentally important for improving the performance of the networks. We would like the network to learn the important features that are invariant for the object classes, rather than the artifact of the training images. We have explored many different ways of doing augmentation, some of which are discussed in this section.

It is clear that an object would not change its class if ambient lighting changes, or if the observer is replaced by another. More specifically, this is to say that the neural network model should be less sensitive to colors that are driven by the illuminants of the scene, or the optical systems from various observers. This observation has led us to focus on some of the key augmentations, such as color casting, vignetting, and lens distortion, as shown in the Figure 5.

Different from the color shifting in (Krizhevsky et al., 2012), we perform color casting to alter the intensities of the RGB channels in training images. Specifically, for each image, we generate three Boolean values to determine if the R, G and B channels should be altered, respectively. If one channel should be altered, we add a random integer ranging from -20 to +20 to that channel.

Vignetting means making the periphery of an image dark compared to the image center. In our implementation, there are two randomly changeable variables for a vignette effect.

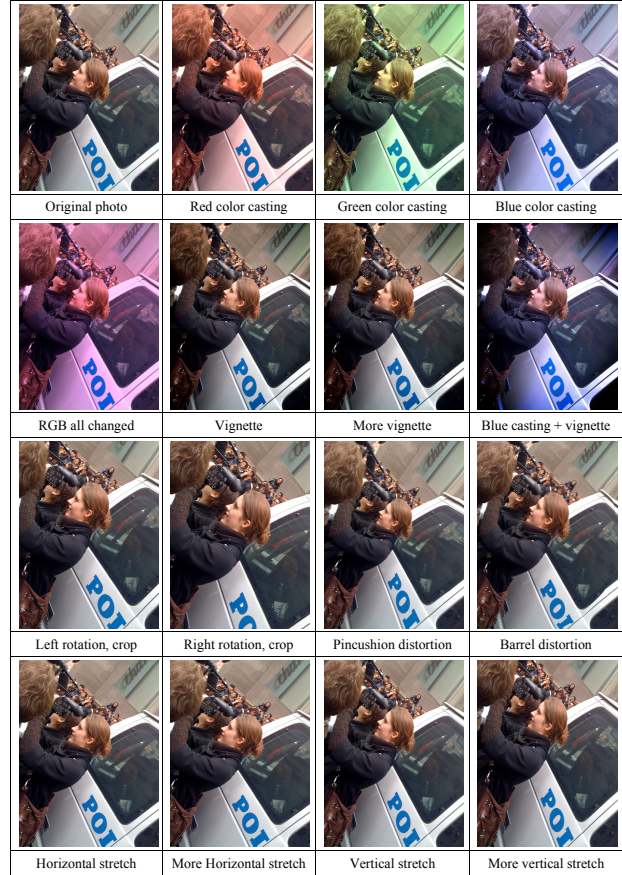


Figure 5. Effects of data augmentation.

The first is the area to add the effect. The second is how much brightness is reduced.

Lens distortion is a deviation from rectilinear projection caused by the lens of camera. The type of distortion and the degree of distortion are randomly changeable. The horizontal and vertical stretching of images can also be viewed as special kind of lens distortion.

We also adopt some augmentations that were proposed by the previous work, such as flipping and cropping (Howard, 2013), (Krizhevsky et al., 2012). To ensure that the whole augmented image feeds into the training net, all the augmentations are performed on the cropped image (except cropping).

An interesting fact worth pointing out is that both professional photographers and amateurs often use color casting and vignetting to add an artistic touch to their work. In fact, most filters offered by popular apps like Instagram are no more than various combinations of the two.

As shown in Table 1, by using different augmentation approaches, the number of training examples explodes and

Table 1. The number of possible changes for different augmentation ways.

Augmentation	The number of possible changes
Color casting	68920
Vignetting	1960
Lens distortion	260
Rotation	20
Flipping	2
Cropping	82944(crop size is 224x224, input image size is 512x512)

poses a greater challenge in terms of computational resources. However, the resulting model has better accuracy which recoups the cost as evidenced in Figure 6.

4.2. Multi-scale Training

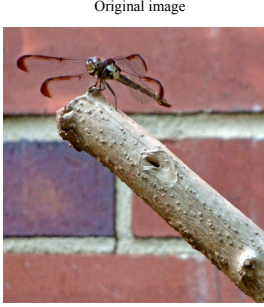
We also notice that multi-scale training with high-resolution images works better than single-scale training, especially for recognizing small objects.

Previous work (Krizhevsky et al., 2012), (Zeiler &ergus, 2014) usually downsizes the images to a fixed resolution, such as 256x256, and then randomly crops out slightly smaller areas, such as 224x224, and uses those crops for training. While this method reduces computational costs and the multiple convolutional layers can still capture the multi-scale statistics, the down sampling may disturb the details of small objects, which are important features to distinguish between different classes. If higher-resolution images such as 512x512 are used, there will be more details captured by each 224x224 crop. The model may learn more from such crops. Different from the work (Simonyan & Zisserman, 2014), (Szegedy et al., 2014) that uses scale-jittering method, we have trained separate models at different scales, including high-resolution ones (such as 512x512), and combined them by averaging softmax class posteriors.

As shown in Figure 7, the dragonfly is small and only occupies a small portion of the image. If a model trained by low-resolution images is used, the true class is out of the top-5 prediction. A high-resolution training will capture more



Figure 6. Some hard cases addressed by adding our data augmentation.



Original image			Low-resolution model			High-resolution model		
Rank	Score	Class	Rank	Score	Class	Rank	Score	Class
1	0.2287	ant	1	0.103	lacewing	1	0.103	lacewing
2	0.0997	damselfly	2	0.074	dragonfly	2	0.074	dragonfly
3	0.057	nematode	3	0.074	damselfly	3	0.074	damselfly
4	0.0546	chainlink fence	4	0.063	walking stic	4	0.063	walking stic
5	0.0522	long-horned	5	0.039	long-horned	5	0.039	long-horned
6	0.0307	walking stick	6	0.027	leathopper	6	0.027	leathopper
7	0.0287	dragonfly	7	0.025	nail	7	0.025	nail
8	0.0267	tiger beetle	8	0.023	grasshopper	8	0.023	grasshopper
9	0.0225	doormat	9	0.019	ant	9	0.019	ant
10	0.0198	flute	10	0.015	mantis	10	0.015	mantis
11	0.0198	grey whale	11	0.015	fly	11	0.015	fly
12	0.0178	mantis	12	0.013	hammer	12	0.013	hammer
13	0.0171	lacewing	13	0.012	American	13	0.012	American
14	0.0161	radiator	14	0.012	gar	14	0.012	gar
15	0.0161	scabbard	15	0.011	chainlink	15	0.011	chainlink
16	0.0157	slide rule	16	0.011	padlock	16	0.011	padlock
17	0.0148	fly	17	0.011	tree frog	17	0.011	tree frog
18	0.0129	leathopper	18	0.011	cicada	18	0.011	cicada
19	0.0101	cucumber	19	0.01	screwdriver	19	0.01	screwdriver
20	0.0094	velvet	20	0.01	harvestman	20	0.01	harvestman

Figure 7. Top-20 classification results comparison between the models trained by low-resolution and high-resolution images.



Figure 8. Some hard cases addressed by using higher-resolution images for training.

features of dragonfly and recognize the image in a second place. The high-resolution model gives similar scores for lacewing, dragonfly, and damselfly due to the similarity in species. However, the low-resolution model distinguishes them clearly by giving scores with large gap. It could be also seen that combining the two models in Figure 7 by simple averaging will give a good prediction. Figure 8 has more examples where a high-resolution model gives the correct answer while the lower-resolution one fails.

On the other hand, the small crop from a high-resolution image may not contain any object that will mislead the model by giving a label of the whole image. So, a model trained by crops from low-resolution images is necessary.

We have trained several models with the images of different scales and they are complementary. The models trained with 256x256 images and 512x512 images could give a top-5 error rate of 7.96% and 7.42% separately for the ImageNet validation dataset, but the fused model by simple averaging gives an error rate of 6.97% - an even better result than each individual model. It is worth noting that all the validation results are also achieved by testing and combining the results of multi-scale images resized from a single one.

Table 2. Basic configuration for one of our models.

Layers	Conv 1-2	Maxpool	Conv 3-4		Maxpool
# filters	64		128		
Conv 5-6-7		Maxpool	Conv 8-9-10		Maxpool
256			512		
Conv 11-12-13		Maxpool	FC 1-2	FC 2	Softmax
512			6144	1000	

5. Experiments

One of the most challenging computer vision benchmarks is the classification task of ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The task is set to evaluate the algorithms for large scale image classification. There are more than 15 million images belonging to about 22,000 categories in the ImageNet dataset, and ILSVRC uses a subset of about 1.2 million images which contains 1,000 categories. After the great success of convolutional networks (ConvNets) for this challenge, there is increasing attention both from industry and academic communities to build a more accurate ConvNet system with the help of powerful computing platforms, such as GPU and CPU clusters.

As shown in Table 3, the accuracy has been optimized a lot during the last three years. The best result of ILSVRC 2014, top-5 error rate of 6.66%, is not far from human recognition performance of 5.1% (Russakovsky et al., 2014). Our work marks yet another exciting milestone with the top-5 error rate of 5.33%, not only setting the new record but also closing the gap between computers and humans by 87.3%, almost achieved the human performance.

We only use the provided data from the dataset of ILSVRC 2014. Taking advantage of data augmentation and random crops from high-resolution training images, we are able to reduce the risk of overfitting, despite our larger models. As listed in Table 2, one basic configuration has 16 layers and is similar with VGG’s work (Simonyan & Zisserman, 2014). The number of weights in our configuration is 212.7M. For other models we have varied the number of the filters at convolutional layers and the neurons at fully-connected layers up to 1024 and 8192, respectively. Six trained models with different configurations, each focusing on various image scales and data augmentation methods respectively, are combined by simple averaging of the softmax class posteriors. The independent training procedure of these models makes them complementary and gives better results than each individual.

³This result is obtained from paper (Russakovsky et al., 2014). VGG team achieves top-5 test set error of 6.8% using multiple models after the competition (Simonyan & Zisserman, 2014).

Table 3. ILSVRC classification task top-5 performance (with provided training data only).

Team	Year	Place	Top-5 error
SuperVision	2012	1	16.42%
ISI	2012	2	26.17%
VGG	2012	3	26.98%
Clarifai	2013	1	11.74%
NUS	2013	2	12.95%
ZF	2013	3	13.51%
GoogLeNet	2014	1	6.66%
VGG ³	2014	2	7.32%
MSRA	2014	3	8.06%
Andrew Howard	2014	4	8.11%
DeeperVision	2014	5	9.51%
Deep Image	-	-	5.33%

Table 4. Single model comparison of validation error.

Team	Top-1	Top-5
GoogLeNet (Szegedy et al., 2014)	-	7.89%
VGG (Simonyan & Zisserman, 2014) ⁴	25.9%	8.0%
Deep Image	22.71%	6.18%

Inspired by (Sermanet et al., 2013), a transformed network with only convolutional layers is used to test images and the class scores from the feature map of the last layer are averaged to attain the final score. We also tested single images at multiple scales and combined the results, which will cover portions of the images with different sizes, and allow the network to recognize the object in a proper scale. The horizontal flipping of the images are also tested and combined.

As shown in Table 3, the top-5 accuracy has been improved a lot in last three years. Deep Image has set the new record of 5.33% top-5 error rate for test dataset, a 20.0% relative improvement than the previous best result.

The single model accuracy comparison is listed in Table 4 for validation dataset. Our best single model configured as Table 2 also outperforms any of the previously published ones.

In addition to having achieved the best classification result to date, our system is also more robust in real world scenarios, thanks to the aggressive data augmentation and the high-resolution images used in our training. Figure 9 shows some of the examples. The first row is the original image from ImageNet, others are captured by cellphone under different conditions. Our system recognizes these images correctly despite extreme transformation.

⁴VGG team’s single model achieves top-1 error of 24.4% and top-5 error of 7.1% on validation set after the competition (Simonyan & Zisserman, 2014).

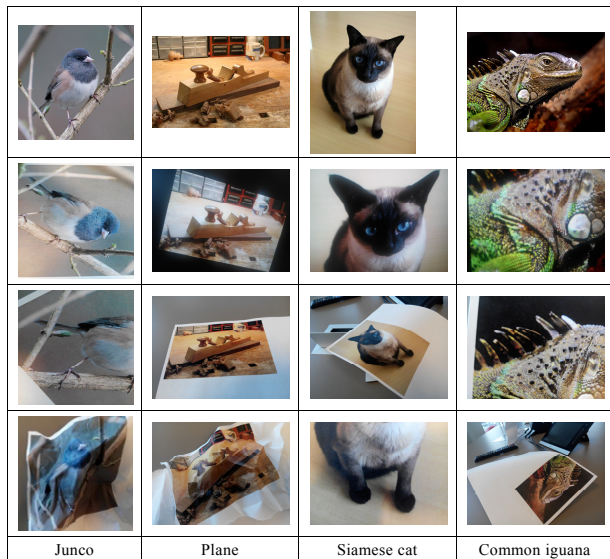


Figure 9. Experiments to test robustness.

6. Related Work

With the development of deep learning, it becomes possible to recognize and classify visual objects end-to-end without needing to create multi-stage pipelines of extracted features and discriminative classifiers. Due to the huge amount of computation, a lot of effort has been made to build a distributed system to scale the network to very large models. Dean et al. developed DistBelief (2012) to train a deep network with billions of parameters using tens of thousands of CPU cores. Within this framework, they developed asynchronous stochastic gradient descent procedure. Coates et al. built a GPU cluster with high speed interconnects, and used it to train a locally-connected neural network (2013). Similar to this work but at a much smaller scale is the work of Paine et al. (2013), in which an asynchronous SGD algorithm based on a GPU cluster was implemented. Project Adam (Chilimbi et al., 2014) is another distributed system built by Microsoft. They trained a large network for ImageNet 22K category object classification task through asynchrony system with 62 machines in ten days and achieved top-1 accuracy of 29.8%. They did not report their result for the 1k classification challenge and so it is hard to compare this work to others. In (Krizhevsky, 2014) and (Yadan et al., 2013), the authors also employed data and model parallelism or hybrid parallelization. Their systems are not scaled up and are limited to a single server with multiple GPUs.

We have trained a large convolutional neural network for the ImageNet classification challenge. This work is related to much previous work around this most popular challenge (Russakovsky et al., 2014), which has become the

standard benchmark for large-scale object classification. The SuperVision team made a significant breakthrough in ILSVRC 2012, where they trained a deep convolutional neural network with 60 million parameters using an efficient GPU implementation (Krizhevsky et al., 2012). Following the success of SuperVision, the winner of the classification task in ILSVRC 2013 was Clarifai, which was designed by using the visualization technique (Zeiler & Fergus, 2014) to guide the adjustment of the network architectures. In ILSVRC 2014, the winner GoogLeNet (Szegedy et al., 2014) and the runner-up VGG team (Simonyan & Zisserman, 2014) both increased the depth of the network significantly, and achieved top-5 classification error 6.66% and 7.32%, respectively. Besides the depth, GoogLeNet (Szegedy et al., 2014) and VGG (Simonyan & Zisserman, 2014) used multi-scale data to improve the accuracy. None of these works have used data augmentation and multi-scale training as aggressively as we have, and they have mainly used lower-resolution training images with a smaller network than ours.

7. Conclusion

We have built a large supercomputer dedicated to training deep neural networks. We have chosen the ImageNet classification challenge as the first test case. Compared to previous work, our model is larger, using higher-resolution images and seeing more examples. We have obtained the best results to date.

The success of this work is driven by tremendous computational power, and can also be described as a “brute force” approach. Earlier, Baidu’s Deep Speech used a similar approach for speech recognition and also achieved state-of-the-art results.

It is possible that other approaches will yield the same results with less demand on the computational side. The authors of this paper argue that with more human effort being applied, it is indeed possible to see such results. However, human effort is precisely what we want to avoid.

Acknowledgments

The project started in October 2013, after an interesting discussion on a high-speed train from Beijing to Suzhou. Many thanks to the Baidu SYS group for their help with hosting the Minwa supercomputer and to Zhiqian Wang for helping with benchmarking the hardware components. Thanks to Adam Coates and Andrew Ng for many insightful conversations. Thank you also to Bryan Catanzaro, Calisa Cole, and Tony Han for reviewing early drafts of this paper.

References

- Campbell, M., Hoane, A.J., and Hsu, F. Deep blue. *Artificial Intelligence*, 134:57–59, 2002.
- Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 571–582, Broomfield, CO, 2014. USENIX Association. ISBN 978-1-931971-16-4.
- Ciresan, D.C., Meier, U., Gambardella, L.M., and Schmidhuber, J. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010.
- Coates, A., Huval, B., Wang, T., Wu, D.J., Catanzaro, B.C., and Ng, A.Y. Deep learning with cots hpc systems. In *ICML (3)’13*, pp. 1337–1345, 2013.
- Collobert, R. and Weston, J. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pp. 160–167, 2008.
- Condon, J. and Thompson, K. Belle chess hardware. *Advances in Computer Chess*, 3, 1982.
- Dahl, G.E., Yu, D., Deng, L., and Acero, A. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):30–42, January 2012.
- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M.A., Senior, A.W., Tucker, P.A., Yang, K., and Ng, A.Y. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, pp. 1232–1240, 2012.
- Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A.Y. Deepspeech: Scaling up end-to-end speech recognition. *arXiv:1412.5567*, 2014.
- Howard, A.G. Some improvements on deep convolutional neural network based image classification. *CoRR*, abs/1312.5402, 2013.
- Hyatt, R.M., Nelson, H.L., and Gower, A.E. Cray blitz. *Computers, Chess, and Cognition*, abs/1312.5402, 1990.
- Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*, 2014.
- Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, pp. 1106–1114, Lake Tahoe, Nevada, United States, 2012.
- Kuszmaul, B.C. The startech massively parallel chess program. *Journal of the International Computer Chess Association*, 18(1), 1995.
- Mnih, A. and Hinton, G.E. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 641–648, Corvallis, Oregon, USA, 2007.
- Mnih, A. and Hinton, G.E. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, pp. 1081–1088, Vancouver, British Columbia, Canada, 2008.
- Paine, T., Jin, H., Yang, J., Lin, Z., and Huang, T.S. GPU asynchronous stochastic gradient descent to speed up neural network training. *CoRR*, abs/1312.6186, 2013.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z.H., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.
- Sermanet, P., Eigen, D., Zhang, X., M.M., R.F., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.
- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014, 2014.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *arXiv:1409.4842*, 2014.
- Wu, R., Zhang, B., and Hsu, M. Clustering cillions of data points using gpus. *ACM UCHPC-MAW*, 2009.
- Yadan, O., Adams, K., Taigman, Y., and Ranzato, M.A. Multi-gpu training of convnets. *CoRR*, abs/1312.5853, 2013.
- Zeiler, M.D. and Fergus, R. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference*, pp. 818–833, Zurich, Switzerland, 2014.