arXiv:1502.01852v1 [cs.CV] 6 Feb 2015

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. First, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Second, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Based on our PReLU networks (PReLU-nets), we achieve ***4.94%*** *top-5 test error on the ImageNet 2012 classification dataset. This is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [29]). To our knowledge, our result is the first to surpass human-level performance (5.1%, [22]) on this visual recognition challenge.*

1. Introduction

Convolutional neural networks (CNNs) [17, 16] have demonstrated recognition accuracy better than or comparable to humans in several visual recognition tasks, including recognizing traffic signs [3], faces [30, 28], and handwritten digits [3, 31]. In this work, we present a result that surpasses human-level performance on a more generic and challenging recognition task - the classification task in the 1000-class ImageNet dataset [22].

In the last few years, we have witnessed tremendous improvements in recognition performance, mainly due to advances in two technical directions: building more powerful models, and designing effective strategies against overfitting. On one hand, neural networks are becoming more capable of fitting training data, because of increased complexity (*e.g.*, increased depth [25, 29], enlarged width [33, 24], and the use of smaller strides [33, 24, 2, 25]), new nonlinear activations [21, 20, 34, 19, 27, 9], and sophisticated layer designs [29, 11]. On the other hand, better generalization is achieved by effective regularization techniques [12, 26, 9, 31], aggressive data augmentation [16, 13, 25, 29], and large-scale data [4, 22].

Among these advances, the rectifier neuron [21, 8, 20, 34], *e.g.*, Rectified Linear Unit (ReLU), is one of several keys to the recent success of deep networks [16]. It expedites convergence of the training procedure [16] and leads to better solutions [21, 8, 20, 34] than conventional sigmoid-like units. Despite the prevalence of rectifier networks, recent improvements of models [33, 24, 11, 25, 29] and theoretical guidelines for training them [7, 23] have rarely focused on the properties of the rectifiers.

In this paper, we investigate neural networks from two aspects particularly driven by the rectifiers. First, we propose a new generalization of ReLU, which we call *Parametric Rectified Linear Unit* (PReLU). This activation function adaptively learns the parameters of the rectifiers, and improves accuracy at negligible extra computational cost. Second, we study the difficulty of training rectified models that are very deep. By explicitly modeling the nonlinearity of rectifiers (ReLU/PReLU), we derive a theoretically sound initialization method, which helps with convergence of very deep models (*e.g.*, with 30 weight layers) trained directly from scratch. This gives us more flexibility to explore more powerful network architectures.

On the 1000-class ImageNet 2012 dataset, our PReLU network (PReLU-net) leads to a single-model result of 5.71% top-5 error, which surpasses all existing multi-model results. Further, our multi-model result achieves **4.94%** top-5 error on the test set, which is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [29]). To the best of our knowledge, our result surpasses for the first time the reported human-level performance (5.1% in [22]) on this visual recognition challenge.

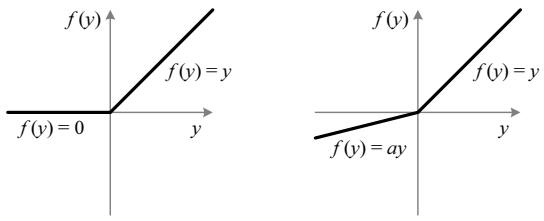

Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

2. Approach

In this section, we first present the PReLU activation function (Sec. 2.1). Then we derive our initialization method for deep rectifier networks (Sec. 2.2). Lastly we discuss our architecture designs (Sec. 2.3).

2.1. Parametric Rectifiers

We show that replacing the parameter-free ReLU activation by a learned parametric activation unit improves classification accuracy[1].

Definition

Formally, we consider an activation function defined as:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}. \tag{1}$$

Here y_i is the input of the nonlinear activation f on the ith channel, and a_i is a coefficient controlling the slope of the negative part. The subscript i in a_i indicates that we allow the nonlinear activation to vary on different channels. When $a_i = 0$, it becomes ReLU; when a_i is a learnable parameter, we refer to Eqn.(1) as *Parametric ReLU* (PReLU). Figure 1 shows the shapes of ReLU and PReLU. Eqn.(1) is equivalent to $f(y_i) = \max(0, y_i) + a_i \min(0, y_i)$.

If a_i is a small and fixed value, PReLU becomes the Leaky ReLU (LReLU) in [20] ($a_i = 0.01$). The motivation of LReLU is to avoid zero gradients. Experiments in [20] show that LReLU has negligible impact on accuracy compared with ReLU. On the contrary, our method adaptively learns the PReLU parameters jointly with the whole model. We hope for end-to-end training that will lead to more specialized activations.

PReLU introduces a very small number of extra parameters. The number of extra parameters is equal to the total number of channels, which is negligible when considering the total number of weights. So we expect no extra risk of overfitting. We also consider a channel-shared variant: $f(y_i) = \max(0, y_i) + a \min(0, y_i)$ where the coefficient is shared by all channels of one layer. This variant only introduces a single extra parameter into each layer.

[1]Concurrent with our work, Agostinelli *et al*. [1] also investigated learning activation functions and showed improvement on other tasks.

Optimization

PReLU can be trained using backpropagation [17] and optimized simultaneously with other layers. The update formulations of $\{a_i\}$ are simply derived from the chain rule. The gradient of a_i for one layer is:

$$\frac{\partial \mathcal{E}}{\partial a_i} = \sum_{y_i} \frac{\partial \mathcal{E}}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial a_i}, \tag{2}$$

where $\mathcal{E}$ represents the objective function. The term $\frac{\partial \mathcal{E}}{\partial f(y_i)}$ is the gradient propagated from the deeper layer. The gradient of the activation is given by:

$$\frac{\partial f(y_i)}{\partial a_i} = \begin{cases} 0, & \text{if } y_i > 0 \\ y_i, & \text{if } y_i \leq 0 \end{cases}. \tag{3}$$

The summation $\sum_{y_i}$ runs over all positions of the feature map. For the channel-shared variant, the gradient of a is $\frac{\partial \mathcal{E}}{\partial a} = \sum_i \sum_{y_i} \frac{\partial \mathcal{E}}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial a}$, where $\sum_i$ sums over all channels of the layer. The time complexity due to PReLU is negligible for both forward and backward propagation.

We adopt the momentum method when updating a_i:

$$\Delta a_i := \mu \Delta a_i + \epsilon \frac{\partial \mathcal{E}}{\partial a_i}. \tag{4}$$

Here μ is the momentum and ϵ is the learning rate. It is worth noticing that we do not use weight decay (l_2 regularization) when updating a_i. A weight decay tends to push a_i to zero, and thus biases PReLU toward ReLU. Even without regularization, the learned coefficients rarely have a magnitude larger than 1 in our experiments. Further, we do not constrain the range of a_i so that the activation function may be non-monotonic. We use $a_i = 0.25$ as the initialization throughout this paper.

Comparison Experiments

We conducted comparisons on a deep but efficient model with 14 weight layers. The model was studied in [10] (model E of [10]) and its architecture is described in Table 1. We choose this model because it is sufficient for representing a category of very deep models, as well as to make the experiments feasible.

As a baseline, we train this model with ReLU applied in the convolutional (conv) layers and the first two fully-connected (fc) layers. The training implementation follows [10]. The top-1 and top-5 errors are 33.82% and 13.34% on ImageNet 2012, using 10-view testing (Table 2).

layer		learned coefficients	
		channel-shared	channel-wise
conv1	7×7, 64, $_{/2}$	0.681	0.596
pool1	3×3, $_{/3}$		
conv2_1	2×2, 128	0.103	0.321
conv2_2	2×2, 128	0.099	0.204
conv2_3	2×2, 128	0.228	0.294
conv2_4	2×2, 128	0.561	0.464
pool2	2×2, $_{/2}$		
conv3_1	2×2, 256	0.126	0.196
conv3_2	2×2, 256	0.089	0.152
conv3_3	2×2, 256	0.124	0.145
conv3_4	2×2, 256	0.062	0.124
conv3_5	2×2, 256	0.008	0.134
conv3_6	2×2, 256	0.210	0.198
spp	$\{6, 3, 2, 1\}$		
fc_1	4096	0.063	0.074
fc_2	4096	0.031	0.075
fc_3	1000		

Table 1. A small but deep 14-layer model [10]. The filter size and filter number of each layer is listed. The number $/s$ indicates the stride s that is used. The learned coefficients of PReLU are also shown. For the channel-wise case, the average of $\{a_i\}$ over the channels is shown for each layer.

	top-1	top-5
ReLU	33.82	13.34
PReLU, channel-shared	32.71	12.87
PReLU, channel-wise	**32.64**	**12.75**

Table 2. Comparisons between ReLU and PReLU on the small model. The error rates are for ImageNet 2012 using 10-view testing. The images are resized so that the shorter side is 256, during both training and testing. Each view is 224×224. All models are trained using 75 epochs.

Then we train the same architecture from scratch, with all ReLUs replaced by PReLUs (Table 2). The top-1 error is reduced to 32.64%. This is a **1.2%** gain over the ReLU baseline. Table 2 also shows that channel-wise/channel-shared PReLUs perform comparably. For the channel-shared version, PReLU only introduces 13 extra free parameters compared with the ReLU counterpart. But this small number of free parameters play critical roles as evidenced by the 1.1% gain over the baseline. This implies the importance of adaptively learning the shapes of activation functions.

Table 1 also shows the learned coefficients of PReLUs for each layer. There are two interesting phenomena in Table 1. First, the first conv layer (conv1) has coefficients (0.681 and 0.596) significantly greater than 0. As the filters of conv1 are mostly Gabor-like filters such as edge or texture detectors, the learned results show that both positive and negative responses of the filters are respected. We believe that this is a more economical way of exploiting low-level information, given the limited number of filters (*e.g.*, 64). Second, for the channel-wise version, the deeper conv layers in general have smaller coefficients. This implies that the activations gradually become "more nonlinear" at increasing depths. In other words, the learned model tends to keep more information in earlier stages and becomes more discriminative in deeper stages.

2.2. Initialization of Filter Weights for Rectifiers

Rectifier networks are easier to train [8, 16, 34] compared with traditional sigmoid-like activation networks. But a bad initialization can still hamper the learning of a highly non-linear system. In this subsection, we propose a robust initialization method that removes an obstacle of training extremely deep rectifier networks.

Recent deep CNNs are mostly initialized by random weights drawn from Gaussian distributions [16]. With fixed standard deviations (*e.g.*, 0.01 in [16]), very deep models (*e.g.*, >8 conv layers) have difficulties to converge, as reported by the VGG team [25] and also observed in our experiments. To address this issue, in [25] they pre-train a model with 8 conv layers to initialize deeper models. But this strategy requires more training time, and may also lead to a poorer local optimum. In [29, 18], auxiliary classifiers are added to intermediate layers to help with convergence.

Glorot and Bengio [7] proposed to adopt a properly scaled uniform distribution for initialization. This is called "*Xavier*" initialization in [14]. Its derivation is based on the assumption that the activations are linear. This assumption is invalid for ReLU and PReLU.

In the following, we derive a theoretically more sound initialization by taking ReLU/PReLU into account. In our experiments, our initialization method allows for extremely deep models (*e.g.*, 30 conv/fc layers) to converge, while the "*Xavier*" method [7] cannot.

Forward Propagation Case

Our derivation mainly follows [7]. The central idea is to investigate the variance of the responses in each layer.

For a conv layer, a response is:

$$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l. \tag{5}$$

Here, $\mathbf{x}$ is a k^2c-by-1 vector that represents co-located $k \times k$ pixels in c input channels. k is the spatial filter size of the layer. With $n = k^2c$ denoting the number of connections of a response, W is a d-by-n matrix, where d is the number of filters and each row of W represents the weights of a filter. $\mathbf{b}$ is a vector of biases, and $\mathbf{y}$ is the response at a pixel of the output map. We use l to index a layer. We have $\mathbf{x}_l = f(\mathbf{y}_{l-1})$ where f is the activation. We also have $c_l = d_{l-1}$.

We let the initialized elements in W_l be mutually independent and share the same distribution. As in [7], we assume that the elements in $\mathbf{x}_l$ are also mutually independent and share the same distribution, and $\mathbf{x}_l$ and W_l are independent of each other. Then we have:

$$Var[y_l] = n_l Var[w_l x_l], \tag{6}$$

where now y_l, x_l, and w_l represent the random variables of each element in $\mathbf{y}_l$, W_l, and $\mathbf{x}_l$ respectively. We let w_l have zero mean. Then the variance of the product of independent variables gives us:

$$Var[y_l] = n_l Var[w_l] E[x_l^2]. \tag{7}$$

Here $E[x_l^2]$ is the expectation of the square of x_l. It is worth noticing that $E[x_l^2] \neq Var[x_l]$ unless x_l has zero mean. For the ReLU activation, $x_l = max(0, y_{l-1})$ and thus it does not have zero mean. This will lead to a conclusion different from [7].

If we let w_{l-1} have a symmetric distribution around zero and $b_{l-1} = 0$, then y_{l-1} has zero mean and has a symmetric distribution around zero. This leads to $E[x_l^2] = \frac{1}{2} Var[y_{l-1}]$ when f is ReLU. Putting this into Eqn.(7), we obtain:

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]. \tag{8}$$

With L layers put together, we have:

$$Var[y_L] = Var[y_1] \left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l] \right). \tag{9}$$

This product is the key to the initialization design. A proper initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. So we expect the above product to take a proper scalar (*e.g.*, 1). A sufficient condition is:

$$\frac{1}{2} n_l Var[w_l] = 1, \quad \forall l. \tag{10}$$

This leads to a zero-mean Gaussian distribution whose standard deviation (std) is $\sqrt{2/n_l}$. This is our way of initialization. We also initialize $\mathbf{b} = 0$.

For the first layer ($l = 1$), we should have $n_1 Var[w_1] = 1$ because there is no ReLU applied on the input signal. But the factor $1/2$ does not matter if it just exists on one layer. So we also adopt Eqn.(10) in the first layer for simplicity.

Backward Propagation Case

For back-propagation, the gradient of a conv layer is computed by:

$$\Delta \mathbf{x}_l = \hat{\mathrm{W}}_l \Delta \mathbf{y}_l. \tag{11}$$

Here we use $\Delta \mathbf{x}$ and $\Delta \mathbf{y}$ to denote gradients ($\frac{\partial \mathcal{E}}{\partial \mathbf{x}}$ and $\frac{\partial \mathcal{E}}{\partial \mathbf{y}}$) for simplicity. $\Delta \mathbf{y}$ represents k-by-k pixels in d channels, and is reshaped into a $k^2 d$-by-1 vector. We denote $\hat{n} = k^2 d$. Note that $\hat{n} \neq n = k^2 c$. $\hat{\mathrm{W}}$ is a c-by-$\hat{n}$ matrix where the filters are rearranged in the way of back-propagation. Note that W and $\hat{\mathrm{W}}$ can be reshaped from each other. $\Delta \mathbf{x}$ is a c-by-1 vector representing the gradient at a pixel of this layer. As above, we assume that w_l and Δy_l are independent of each other, then Δx_l has zero mean for all l, when w_l is initialized by a symmetric distribution around zero.

In back-propagation we also have $\Delta y_l = f'(y_l) \Delta x_{l+1}$ where f' is the derivative of f. For the ReLU case, $f'(y_l)$ is zero or one, and their probabilities are equal. We assume that $f'(y_l)$ and Δx_{l+1} are independent of each other. Thus we have $E[\Delta y_l] = E[\Delta x_{l+1}]/2 = 0$, and also $E[(\Delta y_l)^2] = Var[\Delta y_l] = \frac{1}{2} Var[\Delta x_{l+1}]$. Then we compute the variance of the gradient in Eqn.(11):

$$\begin{aligned} Var[\Delta x_l] &= \hat{n}_l Var[w_l] Var[\Delta y_l] \\ &= \frac{1}{2} \hat{n}_l Var[w_l] Var[\Delta x_{l+1}]. \end{aligned} \tag{12}$$

The scalar $1/2$ in both Eqn.(12) and Eqn.(8) is the result of ReLU, though the derivations are different. With L layers put together, we have:

$$Var[\Delta x_2] = Var[\Delta x_{L+1}] \left(\prod_{l=2}^{L} \frac{1}{2} \hat{n}_l Var[w_l] \right). \tag{13}$$

We consider a sufficient condition that the gradient is not exponentially large/small:

$$\frac{1}{2} \hat{n}_l Var[w_l] = 1, \quad \forall l. \tag{14}$$

The only difference between this equation and Eqn.(10) is that $\hat{n}_l = k_l^2 d_l$ while $n_l = k_l^2 c_l = k_l^2 d_{l-1}$. Eqn.(14) results in a zero-mean Gaussian distribution whose std is $\sqrt{2/\hat{n}_l}$.

For the first layer ($l = 1$), we need not compute Δx_1 because it represents the image domain. But we can still adopt Eqn.(14) in the first layer, for the same reason as in the forward propagation case - the factor of a single layer does not make the overall product exponentially large/small.

We note that it is sufficient to use either Eqn.(14) or Eqn.(10) alone. For example, if we use Eqn.(14), then in Eqn.(13) the product $\prod_{l=2}^{L} \frac{1}{2} \hat{n}_l Var[w_l] = 1$, and in Eqn.(9) the product $\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l] = \prod_{l=2}^{L} n_l / \hat{n}_l = c_2 / d_L$, which is not a diminishing number in common network designs. This means that if the initialization properly scales the backward signal, then this is also the case for the forward signal; and vice versa. For all models in this paper, both forms can make them converge.

Discussions

If the forward/backward signal is inappropriately scaled by a factor β in each layer, then the final propagated signal

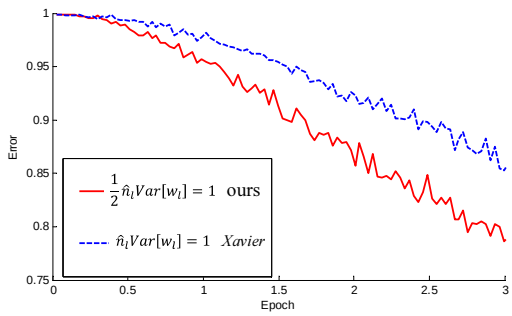

Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

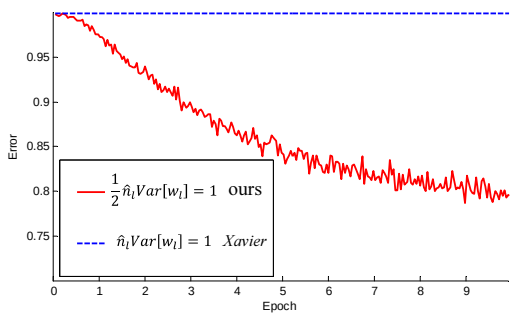

Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

will be rescaled by a factor of β^L after L layers, where L can represent some or all layers. When L is large, if $\beta > 1$, this leads to extremely amplified signals and an algorithm output of infinity; if $\beta < 1$, this leads to diminishing signals[2]. In either case, the algorithm does not converge - it diverges in the former case, and stalls in the latter.

Our derivation also explains why the constant standard deviation of 0.01 makes some deeper networks stall [25]. We take "model B" in the VGG team's paper [25] as an example. This model has 10 conv layers all with 3×3 filters. The filter numbers (d_l) are 64 for the 1st and 2nd layers, 128 for the 3rd and 4th layers, 256 for the 5th and 6th layers, and 512 for the rest. The std computed by Eqn.(14) ($\sqrt{2/\hat{n}_l}$) is 0.059, 0.042, 0.029, and 0.021 when the filter numbers are 64, 128, 256, and 512 respectively. If the std is initialized as 0.01, the std of the gradient propagated from conv10 to conv2 is $1/(5.9 \times 4.2^2 \times 2.9^2 \times 2.1^4) = 1/(1.7 \times 10^4)$ of what we derive. This number may explain why diminishing gradients were observed in experiments.

It is also worth noticing that the variance of the input signal can be roughly preserved from the first layer to the last. In cases when the input signal is not normalized (*e.g.*, it is in the range of $[-128, 128]$), its magnitude can be so large that the softmax operator will overflow. A solution is to normalize the input signal, but this may impact other hyper-parameters. Another solution is to include a small factor on the weights among all or some layers, *e.g.*, $\sqrt[L]{1/128}$ on L layers. In practice, we use a std of 0.01 for the first two fc layers and 0.001 for the last. These numbers are smaller than they should be (*e.g.*, $\sqrt{2/4096}$) and will address the normalization issue of images whose range is about $[-128, 128]$.

For the initialization in the PReLU case, it is easy to show that Eqn.(10) becomes:

$$\frac{1}{2}(1 + a^2) n_l Var[w_l] = 1, \quad \forall l, \tag{15}$$

where a is the initialized value of the coefficients. If $a = 0$, it becomes the ReLU case; if $a = 1$, it becomes the linear case (the same as [7]). Similarly, Eqn.(14) becomes $\frac{1}{2}(1 + a^2)\hat{n}_l Var[w_l] = 1$.

Comparisons with "*Xavier*" Initialization [7]

The main difference between our derivation and the "*Xavier*" initialization [7] is that we address the rectifier nonlinearities[3]. The derivation in [7] only considers the linear case, and its result is given by $n_l Var[w_l] = 1$ (the forward case), which can be implemented as a zero-mean Gaussian distribution whose std is $\sqrt{1/n_l}$. When there are L layers, the std will be $1/\sqrt{2}^L$ of our derived std. This number, however, is not small enough to completely stall the convergence of the models actually used in our paper (Table 3, up to 22 layers) as shown by experiments. Figure 2 compares the convergence of a 22-layer model. Both methods are able to make them converge. But ours starts reducing error earlier. We also investigate the possible impact on accuracy. For the model in Table 2 (using ReLU), the "*Xavier*" initialization method leads to 33.90/13.44 top-1/top-5 error, and ours leads to 33.82/13.34. We have not observed clear superiority of one to the other on accuracy.

Next, we compare the two methods on extremely deep models with up to 30 layers (27 conv and 3 fc). We add up to sixteen conv layers with 256 2×2 filters in the model in

[2]In the presence of weight decay (l_2 regularization of weights), when the gradient contributed by the logistic loss function is diminishing, the total gradient is not diminishing because of the weight decay. A way of diagnosing diminishing gradients is to check whether the gradient is modulated only by weight decay.

[3]There are other minor differences. In [7], the derived variance is adopted for uniform distributions, and the forward and backward cases are averaged. But it is straightforward to adopt their conclusion for Gaussian distributions and for the forward or backward case only.

Table 1. Figure 3 shows the convergence of the 30-layer model. Our initialization is able to make the extremely deep model converge. On the contrary, the "*Xavier*" method completely stalls the learning, and the gradients are diminishing as monitored in the experiments.

These studies demonstrate that we are ready to investigate extremely deep, rectified models by using a more principled initialization method. But in our current experiments on ImageNet, we have not observed the benefit from training extremely deep models. For example, the aforementioned 30-layer model has 38.56/16.59 top-1/top-5 error, which is clearly worse than the error of the 14-layer model in Table 2 (33.82/13.34). Accuracy saturation or degradation was also observed in the study of small models [10], VGG's large models [25], and in speech recognition [34]. This is perhaps because the method of increasing depth is not appropriate, or the recognition task is not enough complex.

Though our attempts of extremely deep models have not shown benefits, our initialization method paves a foundation for further study on increasing depth. We hope this will be helpful in other more complex tasks.

2.3. Architectures

The above investigations provide guidelines of designing our architectures, introduced as follows.

Our baseline is the 19-layer model (A) in Table 3. For a better comparison, we also list the VGG-19 model [25]. Our model A has the following modifications on VGG-19: (i) in the first layer, we use a filter size of 7×7 and a stride of 2; (ii) we move the other three conv layers on the two largest feature maps (224, 112) to the smaller feature maps (56, 28, 14). The time complexity (Table 3, last row) is roughly unchanged because the deeper layers have more filters; (iii) we use spatial pyramid pooling (SPP) [11] before the first fc layer. The pyramid has 4 levels - the numbers of bins are 7×7, 3×3, 2×2, and 1×1, for a total of 63 bins.

It is worth noticing that we have no evidence that our model A is a better *architecture* than VGG-19, though our model A has better results than VGG-19's result reported by [25]. In our earlier experiments with less scale augmentation, we observed that our model A and our reproduced VGG-19 (with SPP and our initialization) are comparable. The main purpose of using model A is for faster running speed. The actual running time of the conv layers on larger feature maps is slower than those on smaller feature maps, when their time complexity is the same. In our four-GPU implementation, our model A takes 2.6s per mini-batch (128), and our reproduced VGG-19 takes 3.0s, evaluated on four Nvidia K20 GPUs.

In Table 3, our model B is a deeper version of A. It has three extra conv layers. Our model C is a wider (with more filters) version of B. The width substantially increases the complexity, and its time complexity is about 2.3× of B (Table 3, last row). Training A/B on four K20 GPUs, or training C on eight K40 GPUs, takes about 3-4 weeks.

We choose to increase the model width instead of depth, because deeper models have only diminishing improvement or even degradation on accuracy. In recent experiments on small models [10], it has been found that aggressively increasing the depth leads to saturated or degraded accuracy. In the VGG paper [25], the 16-layer and 19-layer models perform comparably. In the speech recognition research of [34], the deep models degrade when using more than 8 hidden layers (all being fc). We conjecture that similar degradation may also happen on larger models for ImageNet. We have monitored the training procedures of some extremely deep models (with 3 to 9 layers added on B in Table 3), and found both training and testing error rates degraded in the first 20 epochs (but we did not run to the end due to limited time budget, so there is not yet solid evidence that these large and overly deep models will ultimately degrade). Because of the possible degradation, we choose not to further increase the depth of these large models.

On the other hand, the recent research [5] on small datasets suggests that the accuracy should improve from the increased number of parameters in conv layers. This number depends on the depth and width. So we choose to increase the width of the conv layers to obtain a higher-capacity model.

While all models in Table 3 are very large, we have not observed severe overfitting. We attribute this to the aggressive data augmentation used throughout the whole training procedure, as introduced below.

3. Implementation Details

Training

Our training algorithm mostly follows [16, 13, 2, 11, 25]. From a resized image whose shorter side is s, a 224×224 crop is randomly sampled, with the per-pixel mean subtracted. The scale s is randomly jittered in the range of $[256, 512]$, following [25]. One half of the random samples are flipped horizontally [16]. Random color altering [16] is also used.

Unlike [25] that applies scale jittering only during fine-tuning, we apply it from the beginning of training. Further, unlike [25] that initializes a deeper model using a shallower one, we directly train the very deep model using our initialization described in Sec. 2.2 (we use Eqn.(14)). Our end-to-end training may help improve accuracy, because it may avoid poorer local optima.

Other hyper-parameters that might be important are as follows. The weight decay is 0.0005, and momentum is 0.9. Dropout (50%) is used in the first two fc layers. The mini-batch size is fixed as 128. The learning rate is 1e-2, 1e-3,

input size	VGG-19 [25]	model A	model B	model C
224	3×3, 64 3×3, 64 2×2 maxpool, /2	7×7, 96, /2	7×7, 96, /2	7×7, 96, /2
112	3×3, 128 3×3, 128 2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2
56	3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 2×2 maxpool, /2
28	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 2×2 maxpool, /2
14	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, $\{7, 3, 2, 1\}$	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, $\{7, 3, 2, 1\}$	3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 spp, $\{7, 3, 2, 1\}$
fc_1 fc_2 fc_3	4096 4096 1000			
depth (conv+fc)	19	19	22	22
complexity (ops., $\times 10^{10}$)	1.96	1.90	2.32	5.30

Table 3. Architectures of large models. Here "/2" denotes a stride of 2.

and 1e-4, and is switched when the error plateaus. The total number of epochs is about 80 for each model.

Testing

We adopt the strategy of "multi-view testing on feature maps" used in the SPP-net paper [11]. We further improve this strategy using the dense sliding window method in [24, 25].

We first apply the convolutional layers on the resized full image and obtain the last convolutional feature map. In the feature map, each 14×14 window is pooled using the SPP layer [11]. The fc layers are then applied on the pooled features to compute the scores. This is also done on the horizontally flipped images. The scores of all dense sliding windows are averaged [24, 25]. We further combine the results at multiple scales as in [11].

Multi-GPU Implementation

We adopt a simple variant of Krizhevsky's method [15] for parallel training on multiple GPUs. We adopt "data parallelism" [15] on the conv layers. The GPUs are synchronized before the first fc layer. Then the forward/backward propagations of the fc layers are performed on a single GPU - this means that we do not parallelize the computation of the fc layers. The time cost of the fc layers is low, so it is not necessary to parallelize them. This leads to a simpler implementation than the "model parallelism" in [15]. Besides, model parallelism introduces some overhead due to the communication of filter responses, and is not faster than computing the fc layers on just a single GPU.

We implement the above algorithm on our modification of the Caffe library [14]. We do not increase the mini-batch size (128) because the accuracy may be decreased [15]. For the large models in this paper, we have observed a 3.8x speedup using 4 GPUs, and a 6.0x speedup using 8 GPUs.

model A	ReLU		PReLU	
scale s	top-1	top-5	top-1	top-5
256	26.25	8.25	**25.81**	**8.08**
384	24.77	7.26	**24.20**	**7.03**
480	25.46	7.63	**24.83**	**7.39**
multi-scale	24.02	6.51	**22.97**	**6.28**

Table 4. Comparisons between ReLU/PReLU on model A in ImageNet 2012 using dense testing.

4. Experiments on ImageNet

We perform the experiments on the 1000-class ImageNet 2012 dataset [22] which contains about 1.2 million training images, 50,000 validation images, and 100,000 test images (with no published labels). The results are measured by top-1/top-5 error rates [22]. We only use the provided data for training. All results are evaluated on the validation set, except for the final results in Table 7, which are evaluated on the test set. The top-5 error rate is the metric officially used to rank the methods in the classification challenge [22].

Comparisons between ReLU and PReLU

In Table 4, we compare ReLU and PReLU on the large model A. We use the channel-wise version of PReLU. For fair comparisons, both ReLU/PReLU models are trained using the same total number of epochs, and the learning rates are also switched after running the same number of epochs.

Table 4 shows the results at three scales and the multi-scale combination. The best single scale is 384, possibly because it is in the middle of the jittering range $[256, 512]$. For the multi-scale combination, PReLU reduces the top-1 error by 1.05% and the top-5 error by 0.23% compared with ReLU. The results in Table 2 and Table 4 consistently show that PReLU improves both small and large models. This improvement is obtained with almost no computational cost.

Comparisons of Single-model Results

Next we compare single-model results. We first show 10-view testing results [16] in Table 5. Here, each view is a 224-crop. The 10-view results of VGG-16 are based on our testing using the publicly released model [25] as it is not reported in [25]. Our best 10-view result is 7.38% (Table 5). Our other models also outperform the existing results.

Table 6 shows the comparisons of single-model results, which are all obtained using multi-scale and multi-view (or dense) test. Our results are denoted as MSRA. Our baseline model (A+ReLU, 6.51%) is already substantially better than the best existing single-model result of 7.1% reported for VGG-19 in the latest update of [25] (arXiv v5). We believe that this gain is mainly due to our end-to-end training, without the need of pre-training shallow models.

Moreover, our best single model (C, PReLU) has **5.71%** top-5 error. This result is even better than all previous multi-model results (Table 7). Comparing A+PReLU with B+PReLU, we see that the 19-layer model and the 22-layer model perform comparably. On the other hand, increasing the width (C *vs*. B, Table 6) can still improve accuracy. This indicates that when the models are deep enough, the width becomes an essential factor for accuracy.

Comparisons of Multi-model Results

We combine six models including those in Table 6. For the time being we have trained only one model with architecture C. The other models have accuracy inferior to C by considerable margins. We conjecture that we can obtain better results by using fewer stronger models.

The multi-model results are in Table 7. Our result is **4.94%** top-5 error on the test set. This number is evaluated by the ILSVRC server, because the labels of the test set are not published. Our result is 1.7% better than the ILSVRC 2014 winner (GoogLeNet, 6.66% [29]), which represents a $\sim$26% relative improvement. This is also a $\sim$17% relative improvement over the latest result (Baidu, 5.98% [32]).

Analysis of Results

Figure 4 shows some example validation images successfully classified by our method. Besides the correctly predicted labels, we also pay attention to the other four predictions in the top-5 results. Some of these four labels are other objects in the multi-object images, *e.g*., the "horse-cart" image (Figure 4, row 1, col 1) contains a "mini-bus" and it is also recognized by the algorithm. Some of these four labels are due to the uncertainty among similar classes, *e.g*., the "coucal" image (Figure 4, row 2, col 1) has predicted labels of other bird species.

Figure 6 shows the per-class top-5 error of our result (average of 4.94%) on the test set, displayed in ascending order. Our result has zero top-5 error in 113 classes - the images in these classes are all correctly classified. The three classes with the highest top-5 error are "letter opener" (49%), "spotlight" (38%), and "restaurant" (36%). The error is due to the existence of multiple objects, small objects, or large intra-class variance. Figure 5 shows some example images misclassified by our method in these three classes. Some of the predicted labels still make some sense.

In Figure 7, we show the per-class difference of top-5 error rates between our result (average of 4.94%) and our team's in-competition result in ILSVRC 2014 (average of 8.06%). The error rates are reduced in 824 classes, unchanged in 127 classes, and increased in 49 classes.

model	top-1	top-5
MSRA [11]	29.68	10.95
VGG-16 [25]	28.07†	9.33†
GoogLeNet [29]	-	9.15
A, ReLU	26.48	8.59
A, PReLU	25.59	8.23
B, PReLU	25.53	8.13
C, PReLU	**24.27**	**7.38**

Table 5. The single-model **10-view** results for ImageNet 2012 val set. †: Based on our tests.

	team	top-1	top-5
in competition ILSVRC 14	MSRA [11]	27.86	9.08†
	VGG [25]	-	8.43†
	GoogLeNet [29]	-	7.89
post-competition	VGG [25] (arXiv v2)	24.8	7.5
	VGG [25] (arXiv v5)	24.4	7.1
	Baidu [32]	24.88	7.42
	MSRA (A, ReLU)	24.02	6.51
	MSRA (A, PReLU)	22.97	6.28
	MSRA (B, PReLU)	22.85	6.27
	MSRA (C, PReLU)	**21.59**	**5.71**

Table 6. The **single-model** results for ImageNet 2012 val set. †: Evaluated from the test set.

	team	top-5 (**test**)
in competition ILSVRC 14	MSRA, SPP-nets [11]	8.06
	VGG [25]	7.32
	GoogLeNet [29]	6.66
post-competition	VGG [25] (arXiv v5)	6.8
	Baidu [32]	5.98
	MSRA, PReLU-nets	**4.94**

Table 7. The **multi-model** results for the ImageNet 2012 test set.

Comparisons with Human Performance from [22]

Russakovsky *et al*. [22] recently reported that human performance yields a 5.1% top-5 error on the ImageNet dataset. This number is achieved by a human annotator who is well trained on the validation images to be better aware of the existence of relevant classes. When annotating the test images, the human annotator is given a special interface, where each class title is accompanied by a row of 13 example training images. The reported human performance is estimated on a random subset of 1500 test images.

Our result (4.94%) exceeds the reported human-level performance. To our knowledge, our result is the first published instance of surpassing humans on this visual recognition challenge. The analysis in [22] reveals that the two major types of human errors come from fine-grained recognition and class unawareness. The investigation in [22] suggests that algorithms can do a better job on fine-grained recognition (*e.g*., 120 species of dogs in the dataset). The second row of Figure 4 shows some example fine-grained objects successfully recognized by our method - "coucal", "komondor", and "yellow lady's slipper". While humans can easily recognize these objects as a bird, a dog, and a flower, it is nontrivial for most humans to tell their species. On the negative side, our algorithm still makes mistakes in cases that are not difficult for humans, especially for those requiring context understanding or high-level knowledge (*e.g*., the "spotlight" images in Figure 5).

While our algorithm produces a superior result on this particular dataset, this does not indicate that machine vision outperforms human vision on object recognition in general. On recognizing elementary object categories (*i.e*., common objects or concepts in daily lives) such as the Pascal VOC

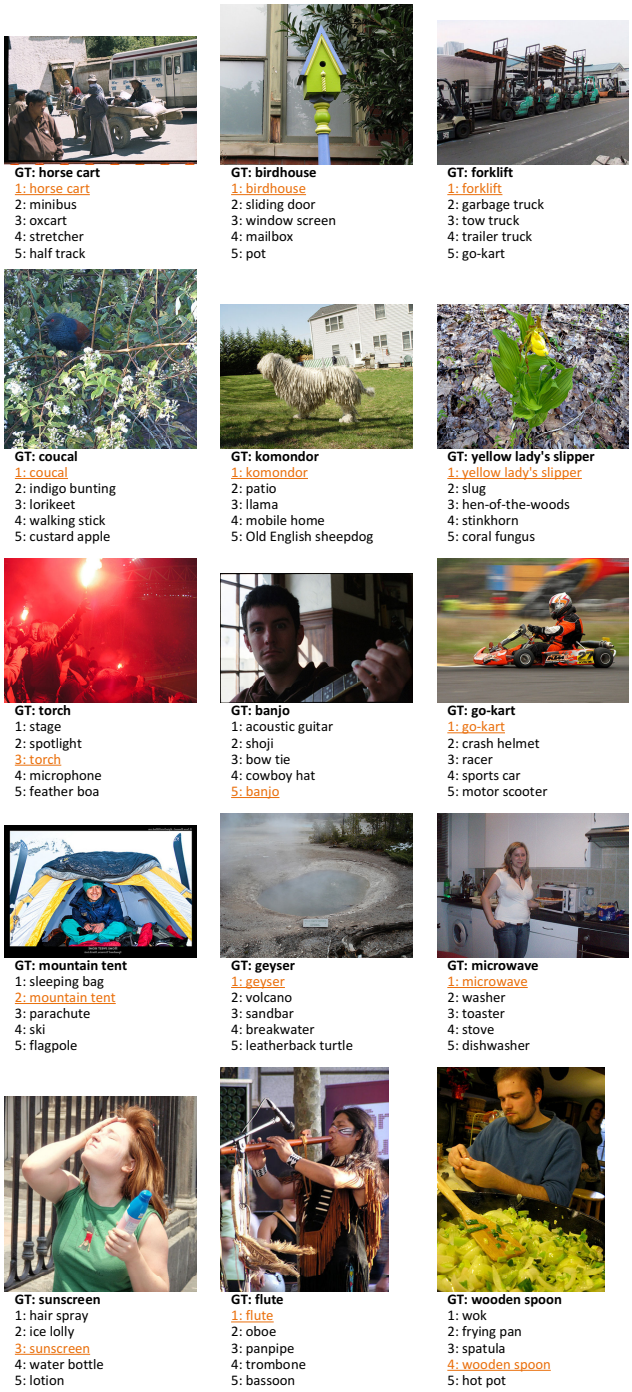

Figure 4. Example validation images successfully classified by our method. For each image, the ground-truth label and the top-5 labels predicted by our method are listed.

task [6], machines still have obvious errors in cases that are trivial for humans. Nevertheless, we believe that our results show the tremendous potential of machine algorithms to match human-level performance on visual recognition.

Figure 5. Example validation images incorrectly classified by our method, in the three classes with the highest top-5 test error. Top: "letter opener" (49% top-5 test error). Middle: "spotlight" (38%). Bottom: "restaurant" (36%). For each image, the ground-truth label and the top-5 labels predicted by our method are listed.

References

[1] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *arXiv:1412.6830*, 2014.

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[3] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[5] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv:1312.1847*, 2013.

[6] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, pages 303–338, 2010.

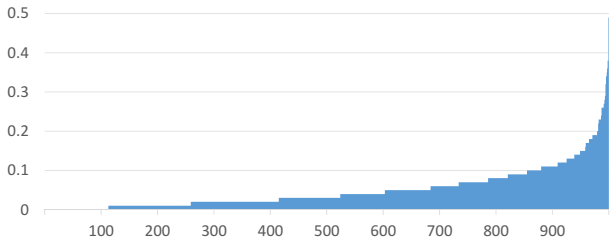

Figure 6. The per-class top-5 errors of our result (average of 4.94%) on the test set. Errors are displayed in ascending order.

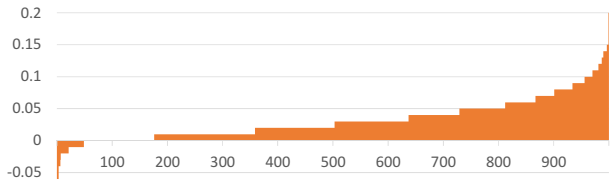

Figure 7. The difference of top-5 error rates between our result (average of 4.94%) and our team's in-competition result for ILSVRC 2014 (average of 8.06%) on the test set, displayed in ascending order. A positive number indicates a reduced error rate.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[8] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:1302.4389*, 2013.

[10] K. He and J. Sun. Convolutional neural networks at constrained time cost. *arXiv:1412.1710*, 2014.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *arXiv:1406.4729v2*, 2014.

[12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[13] A. G. Howard. Some improvements on deep convolutional neural network based image classification. *arXiv:1312.5402*, 2013.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[15] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*, 2014.

[16] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[18] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv:1409.5185*, 2014.

[19] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.

[20] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

[21] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.

[23] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2014.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, pages 1929–1958, 2014.

[27] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber. Compete to compute. In *NIPS*, pages 2310–2318, 2013.

[28] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *NIPS*, 2014.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

[30] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[31] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, pages 1058–1066, 2013.

[32] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep image: Scaling up image recognition. *arXiv:1501.02876*, 2015.

[33] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In *ECCV*, 2014.

[34] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. On rectified linear units for speech processing. In *ICASSP*, 2013.