Reevaluating The All Convolutional Net

Jonas Wechsler, Lars Vagnes, Nicholas Jang

DD2424 Deep Learning

Abstract. We analyze and reproduce a network composed of all convolutional layers proposed by Springenberg et al. and create a similar model that trains much faster and achieves comparable results. First, we assess the training method proposed and remove techniques that increase training time while not providing benefit. Second, we gauge the efficacy of different initialization techniques and training techniques to accelerate training. Third, we use the decreased training time to allow us to conduct a thorough hyper-parameter search. We decrease the training time by 90% and achieve a final validation accuracy of 91.14%.

1 Introduction

The most prevalent CNN architecture to be found in literature consists of alternating convolutional and pooling layers. In the last several years much effort has been spent on further improving this architecture by way of utilizing more complex activation functions, unsupervised pre-training, and improved methods of regularization. However, Springenberg et al. demonstrate that even with these proposed improvements, the standard CNN architecture does not exhibit superior performance on image classification tasks compared to the simpler All-CNN architecture [1]. In fact, All-CNN, the architecture proposed by Springenberg et al., performs on par or better. This is significant because the the All-CNN model, implements dimensionality reduction between layers, not by pooling, but by convolutional layers with stride 2.

The All-CNN models achieve some of the lowest error rates on the CIFAR-10 dataset out of all published models. Springenberg et al. claim that the All-CNN-C model achieves a 9.08% error rate without data augmentation and 7.25% error rate with data augmentation after 350 epochs of training. First, we attempt to replicate their results based on the architecture and hyper-parameters described in the literature. Second, we decrease the training time required to reach the <10% error rate cited in the paper. The All-CNN-C model, when trained as described in the paper, takes 12 hours or longer on a modern GPU. The paper emphasizes that they did not do a thorough search for parameters such as learning rate, and a more extensive search could yield better results. We decrease the time required to train the model three ways: First, we remove training techniques that increase training time but offer negligible benefit, in particular the use of ZCA whitening and a dropout layer applied to the input. Second, we integrate other techniques not described in the paper that accelerate training, and compare different initializations techniques. Third, we decrease the training time

to 100 epochs and use the time savings to conduct a more thorough parameter search.

2 Background

2.1 Striving for Simplicity: The All Convolutional Net by Dmytro Mishkin et al.

Springenberg et al. propose three different all-convolutional "base nets", which are described in Table 1, as well as three additional variations of each network. In this paper we are attempting to verify and augment the work on the All-CNN-C model (described in Table 2).

Springenberg et al.'s training scheme for the All-CNN-C model is as follows. They train the model using stochastic gradient descent and a momentum of 0.9. The learning rate decreases by a factor of 0.1 after a set number of epochs elapsed. They train the first 200 epochs with the initial learning rate, and then decrease the rate after every 50 epochs for a total of 350 epochs. They do not specify an initial learning rate for this model, but the learning rate is chosen from the set [0.25, 0.1, 0.05, 0.01]. A weight decay of 0.001 was also applied. Data augmentation was done by randomly horizontally flipping images and applying a 5px random translation in all directions. Images were also whitened and contrast normalized. There was no specification of batch size or number of iterations. The network had dropout applied to the input and pooling layers, as described in Table 2.

For contrast normalization, we found the 2nd and 98th percentile of intensities for the three color channels, and then stretching the intensity so that those percentiles are the max and min intensities for the image. For an image of values in the range [0,1] with a 2nd percentile of ℓ and a 98th percentile of h, the following equation is applied to each channel for each pixel p:

$$p_{new} = \frac{p - \ell}{h - \ell}$$

2.2 All you need is a good INIT by Dmytro Mishkin et al.

In the process of training deep CNNs, problems can occur with exploding or (more rarely) vanishing gradients. Others have developed a wide variety of end-to-end training practices to avoid these phenomena, but they are lacking in standardization.

Mishkin et al. proposed a novel approach to network initialization, Layer Sequential Unit Variance initialization (LSUV), by which they demonstrate that state of the art performance can be achieved on very deep and thin CNN's through this simple initialization techinique [2]. This technique is an extension of the orthonormal initialization proposed by Saxe et al [3]. The LSUV initialization algorithm consists of two main steps: first initialize all weights with Gaussian noise with unit variance. Second use SVD or QR decomposition to transform the

weights into orthonormal basis. Then proceed to estimate the variance of each individual layers output, and force it to 1 by scaling the weights accordingly.

2.3 Xavier initialization, He initialization, and Batch Normalization

In our approach, we compare the LSUV initialization with other initializations, namely Xavier initialization combined with batch normalization and He normal and He uniform initialization. Xavier initialization draws from Glorot and Bengio's paper [4] to combat the vanishing and exploding gradient problem. They initialize weights based on a uniform distribution such that the variances up and down the network remain the same. He et al. propose an initialization based on the Xavier initialization but adapted for a non-linear activation function, namely ReLU [5]. In our approach, we explore both the He uniform and He normal initialization, drawn from the uniform and normal distribution respectively. Ioffe and Szegedy [6] introduce batch normalization which can accelerate deep network training and mitigate bad initializations. By scaling each mini-batch with regard to its mean and variance, batch normalization acts like a regularizer and allows more reliable training.

Table 1. The three base networks proposed by Springenberg et al.

A	**B**	**C**
Input 32 × 32 RGB Image		
5 × 5 conv. 96 ReLU	5 × 5 conv. 96 ReLU	3 × 3 conv. 96 ReLU
	1 × 1 conv. 96 ReLU	3 × 3 conv. 96 ReLU
3 × 3 max-pooling stride 2		
5 × 5 conv. 192 ReLU	5 × 5 conv. 192 ReLU	3 × 3 conv. 192 ReLU
	1 × 1 conv. 192 ReLU	3 × 3 conv. 192 ReLU
3 × 3 max-pooling stride 2		
3 × 3 conv. 192 ReLU		
1 × 1 conv. 192 ReLU		
1 × 1 conv. 10 ReLU		
Global averaging over 6 × 6 spatial dimensions		
10 or 100-way softmax		

3 Approach

In order to verify the results of the All-CNN-C model, we trained for 350 epochs with learning rates of 0.01, 0.05, and 0.1. No batch size was specified in the paper, so we used a batch size of 64. The paper stated that they used a ZCA whitening filter. We first applied the filter after the random translations during training, which dramatically increased training time. We found that applying the filter to all data points before training made no difference in accuracy, and

Table 2. The All-CNN-C model proposed by Springenberg et al. The 20% dropout after the input layer is omitted in our model.

All-CNN-C
Input 32×32 RGB Image
20% dropout*
3×3 conv. 96 ReLU
3×3 conv. 96 ReLU
3×3 conv. 96 ReLU with 2×2 stride
50% dropout
3×3 conv. 96 ReLU
3×3 conv. 96 ReLU
3×3 conv. 96 ReLU with 2×2 stride
50% dropout
3×3 max-pooling stride 2
3×3 conv. 192 ReLU
1×1 conv. 192 ReLU
1×1 conv. 10 ReLU
Global averaging over 6×6 spatial dimensions
10 or 100-way softmax

that furthermore omitting the filter altogether made little difference in accuracy, so the filter was omitted from our experiments.

We conducted a parameter search for the optimal learning rate and decay rate for the model. First, we searched for the learning rate from 0.005 to 0.1, using 20 epochs per run. For these tests we kept the decay rate fixed at 1e-5. We then searched for the decay rate from 1e0 to 1e-9, using 100 epochs per run. We used a batch size of 32 epochs for all our tests. As with the All-CNN-C model, we did not include ZCA whitening because it did not provide a significant boost in accuracy. We found that omitting the first dropout layer increased accuracy in our model. This can be attributed to the large relative size of the network compared to the training data, coupled with the data augmentation that goes into the network, which prevents the network from overfitting without the additional dropout. For both the All-CNN-C model and our parameter search we use Xavier initialization without batch normalization.

We then tested the initialization mentioned in the LSUV paper with the default All-CNN-C model and compared the results with other initializations. We first tested the LSUV initialization, then the He normal and He uniform initialization, and finally with Xavier initialization with batch normalization. To keep our results consistent, we tested over 100 epochs with the All- CNN-C model with a learning rate of 0.01 and decay of 1e-6. For the batch normalization model, we added a batch normalization layer before the ReLU activation function for each convolutional layer. After finding the best-performing parameters and initialization method, we ran the model with both.

4 Experiments, Observations, Conclusions

We found it difficult to reproduce the results of the All-CNN-C model as given by Springenberg et al., because some key hyper-parameters were not specified by the paper. In particular, the model would have performed much better with a smaller decay rate. Decay rate varies per iteration of the model, but batch size was not specified, so the number of iterations we use over training may vary widely from the original model. The size of the network makes it impractical to search for decay rate or batch size. The model may also perform better with a larger batch size.

When the model was run with large learning rates (0.1, 0.25), it made no improvements in cost or accuracy in the first 100 epochs, so the results were omitted. Model accuracy changed negligibly after 160 epochs, and stabilizes entirely after 200 epochs. For the All-CNN-C model, a larger learning rate (0.05) performs better, as shown in figure 1. However, for our model, we found that a smaller learning rate (0.015) performs better, which could also indicate that the decay rate is too large for the All-CNN-C model. Note our architecture does differ in that it leaves out the first dropout layer on the input. The best performing version of All-CNN-C got 84.7% with a learning rate of 0.05, which is far below the cited accuracy of 92.75%.

We compared learning rates over 20 epochs while keeping the decay static and small. Because we are running the model for a small number of epochs, a change in decay rate shouldn't change the results. The parameter search yielded a concave function that peaks with a learning rate of 0.01 or 0.015, with the accuracy quickly falling off with smaller learning rates. The dropout in our model prevents it from overfitting, which means that the model with the highest training accuracy should perform the best and have the least noise, so we used a learning rate of 0.015, which had the highest train accuracy.

We then compare decay rates on the 100 epoch model with learning rates 0.012, 0.015, and 0.017. We used these three learning rates for two reasons: to see if we needed to conduct a finer search for the learning rate, and to see if a changing decay rate changed the affect the learning rate had. We found that the learning rate of 0.015 still performed best, and a changing decay rate did not change the optimal learning rate. We found that decay rates of 1e-4 and 1e-5 perform best, so we used a decay rate of 3e-5. We found that the decay rate used by Springenberg et al. is much too small for our model, even though our model has fewer epochs, which again suggests that the decay rate was too large for the All-CNN-C model.

We tested the LSUV, He normal, and He uniform initializations along with the Xavier initialization with batch normalization. In figures 4 through 8 we show the accuracy and loss of our ALL-CNN-C models over 100 epochs. The final test accuracies were 87.68%, 88.23%, 87.42%, and 91.13% for the LSUV, He normal, He uniform, and batch normalization models respectively. The final validation losses were 0.466, 0.480, 0.474, and 0.338 for the LSUV, He normal, He uniform, and batch normalization models respectively. Overall, the batch normalization model had significantly higher test and training accuracies and lower validation

Fig. 1. The after-epoch accuracy for the All-CNN-C model, trained with a changing learning rate as specified in section 2.

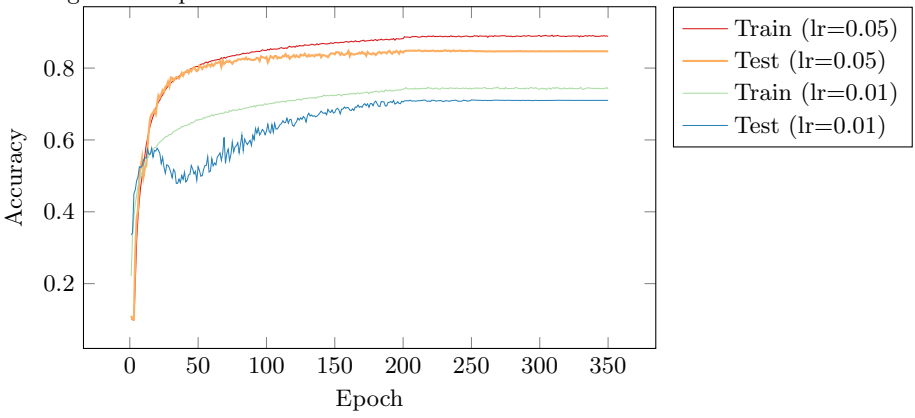


Fig. 2. Accuracy per Learning Rate over 20 Epoch runs. Training and testing accuracy are taken as the individual maximum over the run, so testing accuracy may surpass training accuracy due to noise.

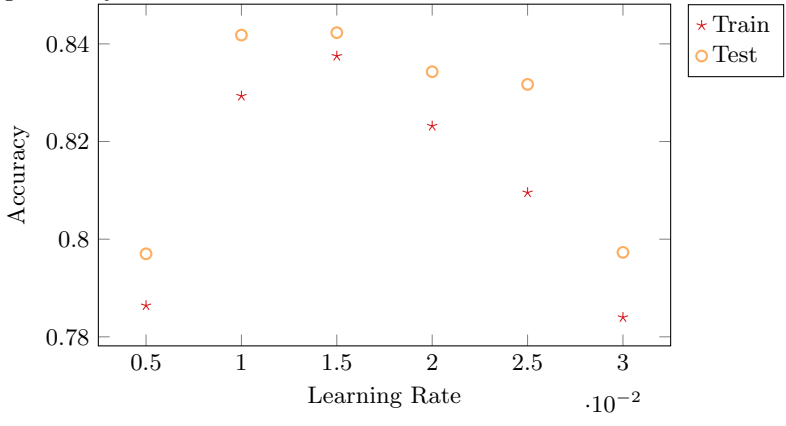
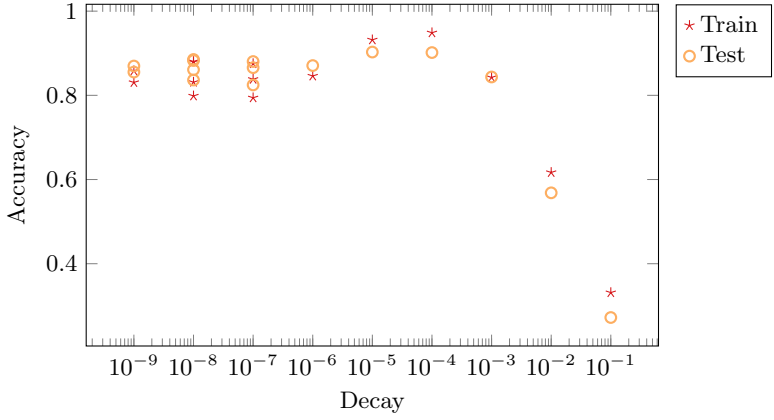


Table 3. Cifar-10 classification accuracy

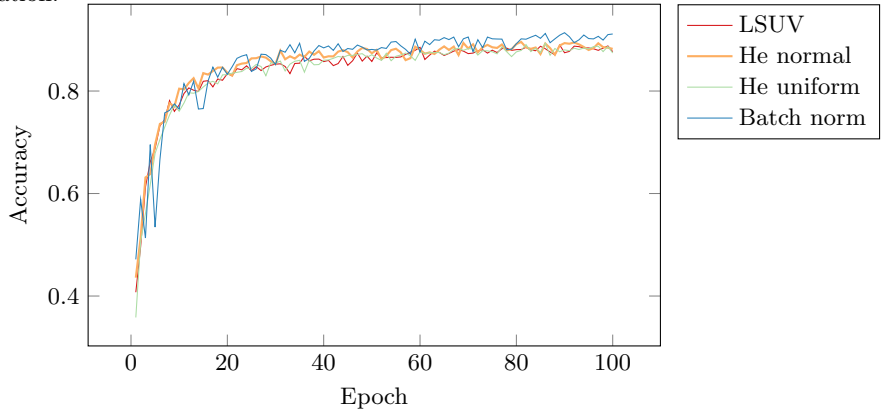
Method	Initialization type	Accuracy
100 epoch model w/ param search	**Xavier w/ batch norm**	**91.14%**
100 epoch model w/o param search	**Xavier w/ batch norm**	**91.13%**
100 epoch model w/o param search	**He normal**	**88.23%**
100 epoch model w/o param search	**LSUV**	**87.68%**
100 epoch model w/o param search	**He uniform**	**87.42%**
100 epoch model w/ param search	**Xavier w/o batch norm**	**90%**
All-CNN-C (ours)	**Xavier w/o batch norm**	**84.7%**
All-CNN-C (paper)	**unspecified**	**92.75%**

Fig. 3. Accuracy per Decay Rate over 100 Epoch runs.



and training losses over 100 epochs. In addition, the batch normalization model consistently outperformed other models throughout the 100 epochs. We find that the LSUV initialization did not enhance the performance of our 100 epoch network. In fact, the He normal, He uniform, and LSUV initialized models all performed fairly similar.

Fig. 4. The test accuracy of the 100 epoch model with LSUV initialization, He normal initialization, He uniform initialization, and Xavier initialization with batch normalization.



Running our 100 epoch model with Xavier initialization, batch normalization, and the training hyper-parameters yields a verification accuracy of 91.14% and a training accuracy of 96.41%. Training the 350 epoch model took an average of 143 seconds per epoch, while the 100 epoch model took an average of 50 seconds.

Fig. 5. The training accuracy of the 100 epoch model with LSUV initialization, He normal initialization, He uniform initialization, and Xavier initialization with batch normalization.

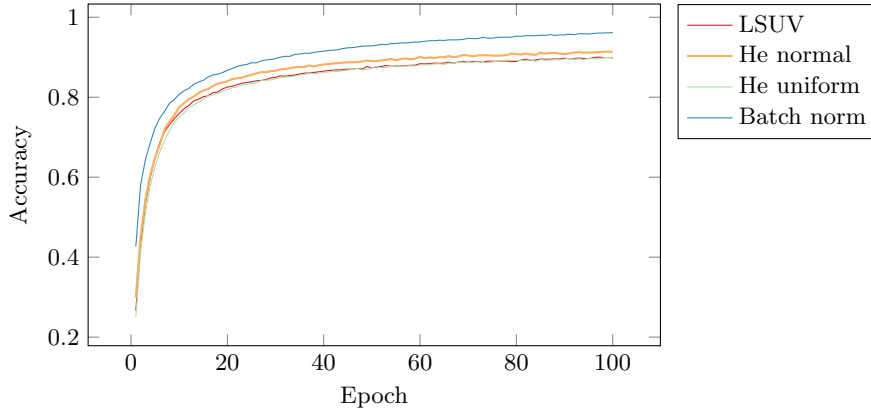
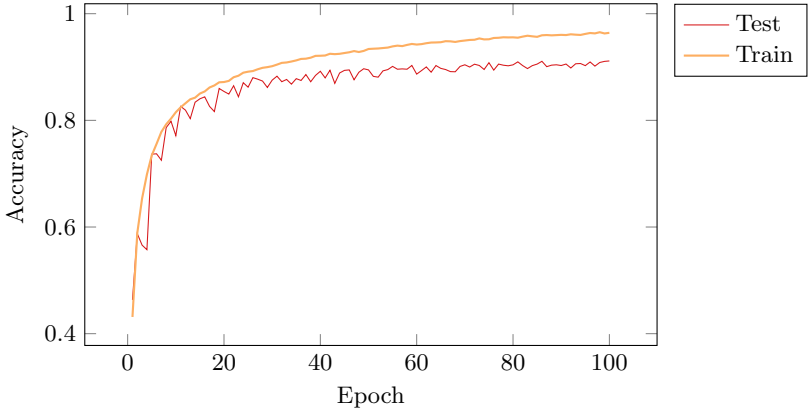


Fig. 6. The training accuracy of the 100 epoch model with learning rate 0.015, decay rate 3e-5, and Xavier initialization with batch norm.



Both were done on a modern GPU with the same hardware. The original model took about 14 hours to train, while ours took less than an hour and a half. This gave a 90% decrease in training time.

We have a few key conclusions from our study:

- Some methods that have been shown to increase performance for other architectures may not increase performance for your architecture, while they will significantly increase training time.
- With sufficiently large data sets and data augmentation, input dropout can be omitted.
- It is possible to achieve comparable results with a shorter training time and a more thorough analysis of the training method and hyper-parameters.
- Sufficient model specification is critical to reproducing a model.

References

1. Jost Tobias Springenberg, Alexey Dosovitskiy, T.B.M.R.: Striving for simplicity: The all convolutional net (2015)
2. Dmytro Mishkin, J.M.: All you need is a good init (2016)
3. Andrew M. Saxe, James L. McClelland, S.G.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks (2013)
4. Xavier Glorot, Y.B.: Understanding the difficulty of training deep feedforward neural networks (2010)
5. Kaiming He, Xiangyu Zhang, S.R.J.S.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classifications (2015)
6. Sergey Ioffe, C.S.: Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015)