Striving for Simplicity: The All Convolutional Net

Wei-Tse Yang, Sunwei Wang, and Qingyuan Cao

18/4/2020

In this notebook, we try to reproduce the TABLE 3 in the original paper. The source code of one of the models on `Pytorch` and the training procedure on Google Colab can be found in Github. **We adopt the original training procedure and change it to the Python class. Also, we build the models from scratch on Pytorch.**

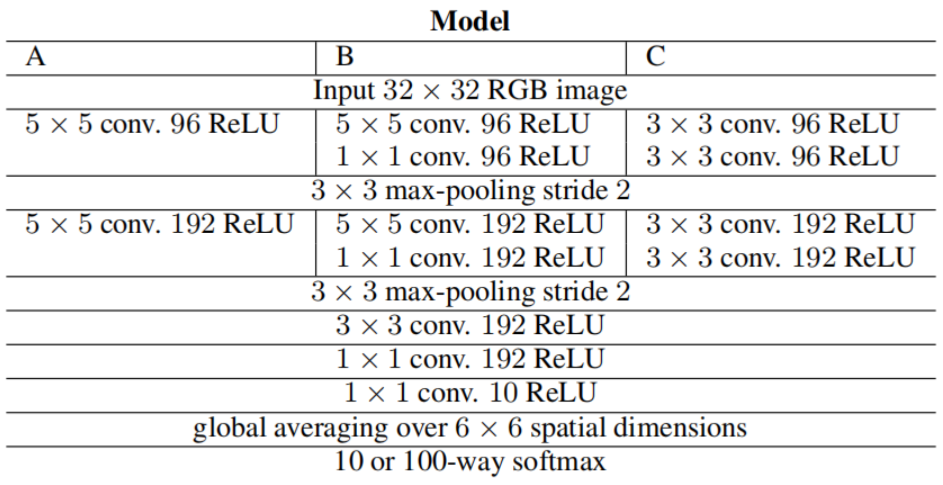
Brief Introduction

The paper shows that replacing the max-pooling with the convolutional with increased strides can improve the performance. The authors prove it by training models with max-pooling, models removing max-pooling, and models replacing max-pooling with convolutional layers. The results show the models replacing max-pooling with convolutional layers with strides generally have better performance. We provide a detaied explanation as follows. The authors tested 12 networks by designing 3 model bases and 3 branches.

Base: Model A, Model B, and Model C

Since the design of convolutional layers would influence the performance, the authors test three model bases. **Model A** uses the 5x5 strides. **Model B** uses the 5x5 strides but also adds one convolutional layer with 1x1 strides after that. **Model C** uses two convolutional layers with 3x3 strides.

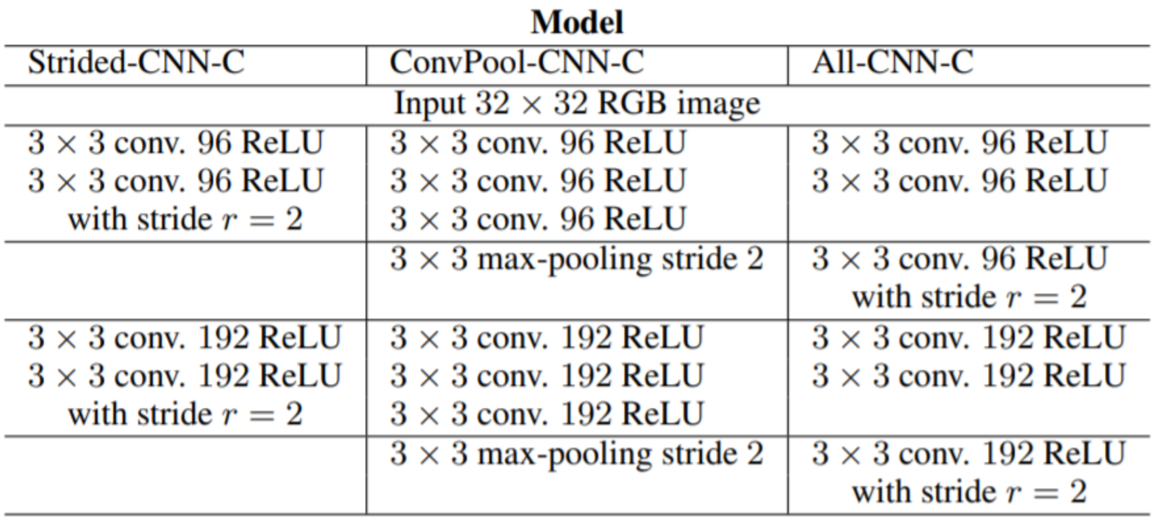
Table 1: The three base networks used for classification on CIFAR-10 and CIFAR-100.



Model		
A	B	C
Input 32×32 RGB image		
5×5 conv. 96 ReLU	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 max-pooling stride 2		
5×5 conv. 192 ReLU	5×5 conv. 192 ReLU 1×1 conv. 192 ReLU	3×3 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		

Branch: Model, Strided-CNN, ConvPool-CNN, and ALL-CNN

Each model base has one original model and thee branches. **"Model"** is the model with max-pooling. **"Strided-CNN"** is the model removing max-pooling. **"All-CNN"** is the model replacing max-pooling with convolutional strides. The better performance of "All-CNN" might result from more parameters than "Model" and "Strided-CNN". To solve it, "ConvPool-CNN" is proposed. **"ConvPool-CNN"** is the model with max-pooling and one more convolutional layer before the pooling. "ConvPool-CNN" should have the same number of parameters as “All-CNN”. Therefore, if “All-CNN” has a better performance than “ConvPool-CNN”, we can prove the better performance on “All-CNN” does not result from more parameters. We show architecture with the base of model C in the following image.



Model

Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
	Input 32×32 RGB image	
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

Experiment Setup

All 12 networks are trained on the CIFAR-10 with the stochastic gradient descent with a fixed momentum of 0.9 and 350 epochs. The learning rate γ is chosen from the set ∈ [0.25, 0.1, 0.05, 0.01]. It is also scheduled by multiplying with a fixed factor of 0.1 in the epoch S= [200, 250, 300]. The paper only presents the best performance among all learning rates. Based on the source code, the performance is directly evaluated on the CIFAR-10 test set. In other words, **the source code did not use the cross-validation for hyper-parameter tuning!**

Reproduction Results

Model	Error Rate of Paper	Error Rate of Ours
Model A	12.47%	19.27%
Strided-CNN-A	13.46%	20.27%
ConvPool-CNN-A	**10.21%**	**15.46%**
ALL-CNN-A	10.30%	15.60%
Model B	10.20%	**17.01%**
Strided-CNN-B	10.98%	23.20%
ConvPool-CNN-B	9.33%	18.22%
ALL-CNN-B	***9.10%***	*29.48%
Model C	9.74%	**13.07%**
Strided-CNN-C	10.19%	15.49%
ConvPool-CNN-C	9.31%	14.39%
ALL-CNN-C	***9.08%***	17.89%

```
1
```

```
1
```

```
# The example of training procedure
# Choose the model A
training=Training(baseModel=[True,False,False])
# Create the dataset
training.createDataset()
# Choose the branch: All-CNN
training.modifiedModel=[False,False,False,True]
# Start Training
training.Procedure()
```

Cross-Valudation

Since the source code did not use the validation set for hyper-parameters tuning, we conduct the cross-validation in this section. We split 5% of the training data to create the cross-validation set. We show the results in the following table. The performance on the test set does not have too much difference from the counterpart on the validation set. However, if we compare the test error with the original test error, the test error with cross-validation is generally higher. This is because right now we save the best model based on the validation error, and then evaluate the test set with it.

Model	Validation Set Error	Test Error	Original Test Error
Model A	21.20%	20.45%	19.27%
Strided-CNN-A	21.72%	21.38%	20.27%
ConvPool-CNN-A	?	?	**15.46%**
ALL-CNN-A	?	?	15.60%
Model B	16.65%	17.81%	**17.01%**
Strided-CNN-B	17.53%	18.68%	23.20%
ConvPool-CNN-B	17.53%	17.51%	18.22%
ALL-CNN-B	*24.53%	*25.78%	*29.48%
Model C	**14.13%**	**14.87%**	**13.07%**
Strided-CNN-C	20.89%	21.67%	15.49%
ConvPool-CNN-C	17.81%	17.60%	14.39%
ALL-CNN-C	?	?	17.89%

```
# Cross Validation can be conducted by settting validation equal to True
training=Training(validation=True,bestModel_allLR=True,baseModel=[True,False,False])
```

DropOut and Batch Normalization

Dropout is a simple method to prevent neural networks from overfitting. In our all convolutional net paper, the author stated that dropout was used to regularize all networks. Dropout was almost essential in all the state-of-the-art networks before the introduction of batch normalization(BN). With the introduction of BN, it has shown its effectiveness and practicability in the recent neural networks. However, there is evidence that when these two techniques are used combinedly in a network, the performance of the network actually becomes worse. (Ioffe & Szegedy, 2015). In our study, we will investigate the results using BN and Dropout independently and also the effect of equipping both BN and Dropout simultaneously.

BatchNorm *only*

```
model = Model(dropOut=False, BN=True)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 30, 30]           7,296
       BatchNorm2d-2           [-1, 96, 30, 30]             192
              ReLU-3           [-1, 96, 30, 30]               0
         MaxPool2d-4           [-1, 96, 14, 14]               0
            Conv2d-5          [-1, 192, 12, 12]         460,992
       BatchNorm2d-6          [-1, 192, 12, 12]             384
              ReLU-7          [-1, 192, 12, 12]               0
         MaxPool2d-8            [-1, 192, 5, 5]               0
            Conv2d-9            [-1, 192, 5, 5]         331,968
      BatchNorm2d-10            [-1, 192, 5, 5]             384
             ReLU-11            [-1, 192, 5, 5]               0
           Conv2d-12            [-1, 192, 5, 5]          37,056
      BatchNorm2d-13            [-1, 192, 5, 5]             384
             ReLU-14            [-1, 192, 5, 5]               0
           Conv2d-15             [-1, 10, 5, 5]           1,930
      BatchNorm2d-16             [-1, 10, 5, 5]              20
             ReLU-17             [-1, 10, 5, 5]               0
AdaptiveAvgPool2d-18             [-1, 10, 1, 1]               0
          Flatten-19                   [-1, 10]               0
================================================================
Total params: 840,606
```

BatchNorm with Dropout

```
model = Model(dropOut=True, BN=True)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
           Dropout-1            [-1, 3, 32, 32]               0
            Conv2d-2           [-1, 96, 30, 30]           7,296
       BatchNorm2d-3           [-1, 96, 30, 30]             192
              ReLU-4           [-1, 96, 30, 30]               0
         MaxPool2d-5           [-1, 96, 14, 14]               0
           Dropout-6           [-1, 96, 14, 14]               0
            Conv2d-7          [-1, 192, 12, 12]         460,992
       BatchNorm2d-8          [-1, 192, 12, 12]             384
              ReLU-9          [-1, 192, 12, 12]               0
        MaxPool2d-10            [-1, 192, 5, 5]               0
          Dropout-11            [-1, 192, 5, 5]               0
           Conv2d-12            [-1, 192, 5, 5]         331,968
      BatchNorm2d-13            [-1, 192, 5, 5]             384
             ReLU-14            [-1, 192, 5, 5]               0
           Conv2d-15            [-1, 192, 5, 5]          37,056
      BatchNorm2d-16            [-1, 192, 5, 5]             384
             ReLU-17            [-1, 192, 5, 5]               0
           Conv2d-18             [-1, 10, 5, 5]           1,930
      BatchNorm2d-19             [-1, 10, 5, 5]              20
             ReLU-20             [-1, 10, 5, 5]               0
AdaptiveAvgPool2d-21             [-1, 10, 1, 1]               0
          Flatten-22                   [-1, 10]               0
================================================================
Total params: 840,606
```

Dropout *only*

```
model = Model(dropOut=True, BN=False)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
           Dropout-1            [-1, 3, 32, 32]               0
            Conv2d-2           [-1, 96, 30, 30]           7,296
              ReLU-3           [-1, 96, 30, 30]               0
         MaxPool2d-4           [-1, 96, 14, 14]               0
           Dropout-5           [-1, 96, 14, 14]               0
            Conv2d-6          [-1, 192, 12, 12]         460,992
              ReLU-7          [-1, 192, 12, 12]               0
         MaxPool2d-8            [-1, 192, 5, 5]               0
           Dropout-9            [-1, 192, 5, 5]               0
           Conv2d-10            [-1, 192, 5, 5]         331,968
             ReLU-11            [-1, 192, 5, 5]               0
           Conv2d-12            [-1, 192, 5, 5]          37,056
             ReLU-13            [-1, 192, 5, 5]               0
           Conv2d-14             [-1, 10, 5, 5]           1,930
             ReLU-15             [-1, 10, 5, 5]               0
AdaptiveAvgPool2d-16             [-1, 10, 1, 1]               0
          Flatten-17                   [-1, 10]               0
================================================================
Total params: 839,242
```

Without using BatchNorm or Dropout

```
model = Model(dropOut=False, BN=False)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 30, 30]           7,296
              ReLU-2           [-1, 96, 30, 30]               0
         MaxPool2d-3           [-1, 96, 14, 14]               0
            Conv2d-4          [-1, 192, 12, 12]         460,992
              ReLU-5          [-1, 192, 12, 12]               0
         MaxPool2d-6            [-1, 192, 5, 5]               0
            Conv2d-7            [-1, 192, 5, 5]         331,968
              ReLU-8            [-1, 192, 5, 5]               0
            Conv2d-9            [-1, 192, 5, 5]          37,056
             ReLU-10            [-1, 192, 5, 5]               0
           Conv2d-11             [-1, 10, 5, 5]           1,930
             ReLU-12             [-1, 10, 5, 5]               0
AdaptiveAvgPool2d-13             [-1, 10, 1, 1]               0
          Flatten-14                   [-1, 10]               0
================================================================
Total params: 839,242
```

We compare the results of different combination of these two techniques and generated the table below:

Model	BN only	BN + Dropout	Dropout only	No BN no Dropout
Model A	no converge	no converge	19.27%	13.82%

As shown in the table above, we used general Model A to study the two techniques Batch Normalization (BN) and Dropout, and whether it increases or decreases our model performance in this case. We implemented BN layer between two convolution layers, right before feeding into ReLu activations. Dropout is also applied in between convolution layers, and it is used after the pooling layer. The original paper stated that they used Dropout only, and we found out that using BN without Dropout or combining both BN with Dropout will not let our model converge. Li, Xiang, et al. (2019) stated in their papers that the worse performance may be the result of variance shifts. However, using Dropout only does lead the model to converge, giving the result of 19.27%. But the performance is still not as good as 13.82% without using either BN or dropout. It might be due to that we did not have the time to tune hyperparameter, dropout rate, we only used the parameter from the original paper: 20% for dropping out inputs and 50% otherwise.

```
1
```

Optimizer

The default optimizer used in the paper and in our reproduction is Stochastic Gradient Descent (SGD) with momentum. We experimented with different optimizers since Adaptive Moment Estimation (Adam) is one of the most popular optimization algorithms, we decided to run Adam instead of SGD optimizer in our model. However, the model did not converge with Adam under the specific setting that we tried to reproduce. So even though in theory Adam combines the advantage of two SGD variants, RMSProp and AdaGrad, it is not very consistent in our specific setting to converge to an optimal solution.

SGD with momentum optimizer:

```python
self.optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
```

Adam optimizer:

```python
self.optimizer = optim.Adam(self.model.parameters(), lr=self.lr)
```

The optimizer is the main approach nowadays for training neural networks with minimizing its error rate. Although Adam is a newer and more popular optimizer in a lot of the project, the original paper still chose to use SGD with momentum. During our model investigation, we found out that Adam fails to converge to an optimal solution in our specific setting, SGD with momentum does in this case perform better. This might be due reasons that the original paper has already tuned the hyperparameters extensively for SGD optimizer, so they chose Learning rate γ from the set ∈ [0.25, 0.1, 0.05, 0.01], and used the momentum 0.9. We, on the other hand, used the same learning rates for the Adam optimizer without extensively tuning them.

Discussion

From our results we obtained, All-CNN models performed much worse than the ConvPool and base models except for variant model A. And when we tried to use cross-validation, All-CNN models cannot converge to an optimal solution for model A and model C. We also realized that the 0.25 learning rate in the original paper will not let most models converge, so we decided to drop it and later even tried learning rates such as small as 0.001, we found out that smaller learning rates might not guarantee the fast converge, but it will generally ensure the converge of the models.

Reference

Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).

Li, Xiang, et al. "Understanding the disharmony between dropout and batch normalization by variance shift." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019).

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

Appendix

- Github Repository
- A Python Class to build all Models
- A Python Class for Training Procedure
- The Notebooks for Model A
- The Notebooks for Model B
- The Notebooks for Model C

- The Notebooks for Cross Validation
- The Notebooks for DropOut and Batch Normalization
- The Notebooks for Optimizers