Reproduce the All Convolutional Network in paper

Xin Wang
Shanghaitech University
Huaxiazhong Road 393
wangxin1@shanghaietch.edu.cn

None

Abstract

Implement All Convolutional Network in paper, reproduce the base A, B, C model and three variant networks of base C, first we run them on CIFAR-10 dataset, and then do a transfer learning on two subsets of CIFAR-100 dataset. We find that max-pooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy on several image recognition benchmarks. *In this way to break the traditional CNN architecture and create a simpler model which only be built up by the convolution layer.*

1. About the model

There are three explanations of why the pooling layers help, first the p-norm can make the representation in a CNN more invariant, second the spatial dimensionality reduction can make the deep layers cover more part of the input, third the feature-wise nature of the pooling operation could make optimization easier. We can remove each pooling layer and increase the stride of the convolutional layer, or replace the pooling layer by a normal convolution with according stride and kernel.

Then considering the fully connected layers, if the image area be covered by unit, then the simple 1 by 1 convolution can replace it.

1.1. Experiment

Experiment environment: Ubuntu 16.04
Pytorch version: 0.4.1
Details of setting: see the following part and the source code
data split: 49000 for training,1000 for validation and 10000 for test
data augmentation: random resize and random horizontal flip
Weight initialization strategy: some with super random set and some use Xavier uniform in the first few layers

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		

From model A to model C, the depth and number of parameters in the network gradually increases, and we can see in midterm task1.ipynb, if use fully connected layers instead of 1-by1 convolutions, it will get 0.5-1% worse. layers instead of 1-by1 convolutions, it will get 0.5-1% worse.

Table 2: Model description of the three networks derived from base model C used for evaluating the importance of pooling in case of classification on CIFAR-10 and CIFAR-100. The derived models for base models A and B are built analogously. The higher layers are the same as in Table 1.

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 $r = 2$

1.2. Detail proceedings

Then remove the max-pooling, Stride-CNN-C let the one layer before the max-pooling become stride 2, ALL-CNN-C add a layer of same kernel size but stride 2 to replace the max-pooling. Conv Pool-CNN-C add a dense layer before every max-pooling layer to ensure that the effect my measure is not solely due to increase model size when going from a normal CNN to its ALL-CNN counterpart.

In the task1 code, first I download and separate the CIFAR-10 data, and do some augmentations and normalizations, then let my code run with GPU.

For model A, I obey the paper and set a loop to find the best learning rate in [0.25,0.1,0.05,0.01], and decrease the learning rate by multiply 0.1 after 200,250 and 300 epochs, the loss function is the cross entropy loss , optimizer is stochastic gradient descent with fixed momentum of 0.9, weight decay of 0.001, finally I add dropout layer to the input with 20% dropping and 50% dropping after for each pooling layer(or after the layer replacing the pooling layer respectively).

The result shows that the learning rate of 0.01 is the best, and I fixed it for the following models to see the roughly result.

1.3. Result

On CIFAR-10

Model	Accuracy on validation set %	Accuracy on test set%	#Parameters
Base A	85%	83.67%	≈0.9M
Base B	82.6%	80.89%	≈1M
Base C	85.10%	80.83%	≈1.3M
Stride C	81%	76.59%	≈1.3M
ConvP C	84.8%	82.64%	≈1.4M
ALL C	81%	78.8%	≈1.4M

These models will converge within 20 epochs or so, if run the whole 350 epochs, the improvement space is not so big (1% or so), in the origin paper base A will approach 87% accuracy, mine is quite close, and the others will get 90%, but mines don't, perhaps due to the parameters and the weight initialization strategy, above is my result, base A and Base B run the entire 350 epochs and the rest only run for 20 epochs, you can compare by adding 1% to these less epoch models (I have tested it for sure)

Dropout layer can reduce the accuracy by my observation, and batch norm can improve the performance (but according to the request, I didn't use it in the above experiment), in terms of initialization, Xavier uniform performs better than the Kaiming method.

1.4. CIFAR-10 with additional data augmentation

After data augmentation with `flip, random crop, color jittering, shift, scale.` can improve the accuracy by about 2%

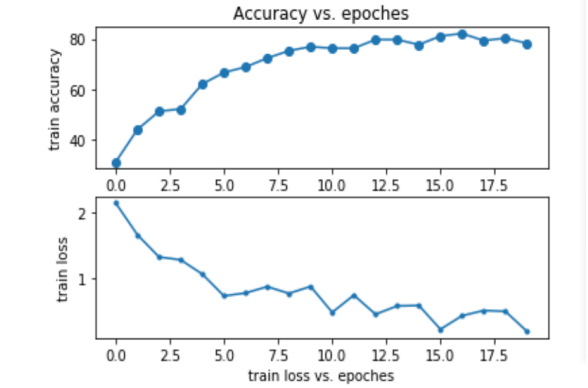
2. Transfer learning on the two subsets of CIFAR-100

Save or run the ALL-C training part above in the task2.ipynb, use pickle to transfer it into image format, then do some pretreatment and load it.

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset, and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.

Details shows in task2.ipynb, here shows the acc and loss graph of pretrained model ALL-C

```
plt.ylabel('train loss')
plt.show()
```



Then substitute the last layer and change it to a FC classifier, we can use unpickle to see what inside the CIFAR-100's subsets (class1 and class 2), retrain it(stuck)

In this way we can test in some small datasets which are similar to the pretrained data. And through the entire flow we can see the advantages of the max-pooling substitute.

Reference

[1] Deng, Jia, Dong, Wei, Socher, Richard, jia Li, Li, Li, Kai, and Fei-fei, Li. Imagenet: A large-scale hierarchical image database. In CVPR, 2009.

[2] Estrach, Joan B., Szlam, Arthur, and Lecun, Yann. Signal recovery from pooling representations. In ICML, 2014.

[3] Goodfellow, Ian J., Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In ICML, 2013.

[4] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.

[5] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998.

Appendix (training log)
#see the structure in the outside folder

```
train(model_A, optimizer, epochs=50)

check_accuracy(loader_test, model_A.cuda())
```

```
Iteration 400, loss = 0.0159
Checking accuracy on validation set
Got 842 / 1000 correct (84.20%)

Iteration 500, loss = 0.0332
Checking accuracy on validation set
Got 840 / 1000 correct (84.00%)

Iteration 600, loss = 0.0226
Checking accuracy on validation set
Got 838 / 1000 correct (83.80%)

Iteration 700, loss = 0.0239
Checking accuracy on validation set
Got 850 / 1000 correct (85.00%)

Checking accuracy on test set
Got 8367 / 10000 correct (83.67%)
```

```
train(model_B, optimizer, epochs=350)

check_accuracy(loader_test, model_B.cuda())
```

```
Iteration 400, loss = 0.1020
Checking accuracy on validation set
Got 823 / 1000 correct (82.30%)

Iteration 500, loss = 0.0485
Checking accuracy on validation set
Got 826 / 1000 correct (82.60%)

Iteration 600, loss = 0.0664
Checking accuracy on validation set
Got 813 / 1000 correct (81.30%)

Iteration 700, loss = 0.1167
Checking accuracy on validation set
Got 790 / 1000 correct (79.00%)

Checking accuracy on test set
Got 8089 / 10000 correct (80.89%)
```

```
train(model_C, optimizer, epochs=20)

check_accuracy(loader_test, model_C.cuda())
```

```
Iteration 400, loss = 0.1730
Checking accuracy on validation set
Got 851 / 1000 correct (85.10%)

Iteration 500, loss = 0.1233
Checking accuracy on validation set
Got 831 / 1000 correct (83.10%)

Iteration 600, loss = 0.1403
Checking accuracy on validation set
Got 837 / 1000 correct (83.70%)

Iteration 700, loss = 0.2473
Checking accuracy on validation set
Got 843 / 1000 correct (84.30%)

Checking accuracy on test set
Got 8083 / 10000 correct (80.83%)
```

```
train(model_STRIDED, optimizer, epochs=20)

check_accuracy(loader_test, model_STRIDED.cuda())
```

```
Iteration 400, loss = 0.4676
Checking accuracy on validation set
Got 803 / 1000 correct (80.30%)

Iteration 500, loss = 0.5750
Checking accuracy on validation set
Got 792 / 1000 correct (79.20%)

Iteration 600, loss = 0.5108
Checking accuracy on validation set
Got 810 / 1000 correct (81.00%)

Iteration 700, loss = 0.4382
Checking accuracy on validation set
Got 809 / 1000 correct (80.90%)

Checking accuracy on test set
Got 7659 / 10000 correct (76.59%)
```

```
train(model_ConvP, optimizer, epochs=20)

check_accuracy(loader_test, model_ConvP.cuda())
```

```
Iteration 400, loss = 0.1786
Checking accuracy on validation set
Got 835 / 1000 correct (83.50%)

Iteration 500, loss = 0.2903
Checking accuracy on validation set
Got 843 / 1000 correct (84.30%)

Iteration 600, loss = 0.3164
Checking accuracy on validation set
Got 840 / 1000 correct (84.00%)

Iteration 700, loss = 0.2940
Checking accuracy on validation set
Got 848 / 1000 correct (84.80%)

Checking accuracy on test set
Got 8264 / 10000 correct (82.64%)
```

```
train(model_ALLC, optimizer, epochs=20)
check_accuracy(loader_test, model_ALLC.cuda())
```

```
Iteration 400, loss = 0.3188
Checking accuracy on validation set
Got 802 / 1000 correct (80.20%)

Iteration 500, loss = 0.3622
Checking accuracy on validation set
Got 800 / 1000 correct (80.00%)

Iteration 600, loss = 0.4176
Checking accuracy on validation set
Got 810 / 1000 correct (81.00%)

Iteration 700, loss = 0.4618
Checking accuracy on validation set
Got 797 / 1000 correct (79.70%)

Checking accuracy on test set
Got 7880 / 10000 correct (78.80%)
```

Acc and loss for Base A