The All Convolutional Net

E4040.2016Fall.XXXD.report

Haoyang Chen hc2812, Sitao Luan sl3903, Han Cui hc2813

Columbia University

Abstract

*We build CNN models that consist solely of convolutional layers and use such architecture to yield competitive performance. However, the challenge is the performance is lower than it in original paper, and we try to overcome this by data augmentation and early stopping. Finally, our neural network model yields ** error.*

1. Introduction

CNNs are widely used for object recognition. The models are built using convolution and max-polling layers followed by a small number of fully connected layers. There are two major directions to improve the performance of this pipeline: on the one hand, using more complex activation functions and layer-wise pre-training; on the other hand, experimenting different architectures in CNNs.

We build CNN models that consist solely of convolutional layers with occasional dimensionality reduction by using a stride of 2 on several image recognition datasets. The intuition is that dimension reduction is performed by strided convolution rather than max-polling. The objective of this project is to use such architecture to yield competitive performance on classification. The technical challenge is the performance of the reproduction is not as high as it in the original paper. We overcome this by adding data augmentation (adding rotated images, randomly horizontally flipping images) before training model, and tuning the parameters in the model: changing weights initialization, and implementing early stopping to regularize the model.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

Original Paper proposed 3 base models used for classification on CIFAR-10 and CIFAR-100 (Table 2.1). These three models reflect current best practices for setting up CNNs for object recognition. And then they experimented with three additional variants for each base model. The derived models for base model C are described in Table 4.1, 4.2 & 4.3.

The model in Table 4.1 is called “Strided-CNN”. The max-pooling layer is removed and the stride of the convolution layers preceding the max-pool layers is increased by 1.

The model in Table 4.2 is called “ConvPool-CNN”. A dense convolution is placed before each max-pooling layer.

The model in Table 4.3 is “All-CNN” whose max-pooling layer is replaced by a convolution layer.

The experiments with the “ConvPool-CNN” are necessary to ensure the effect of replacing max-pooling layer with convolution layer.

Table 2.1: The three base networks

Model A	Model B	Model 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, stride 1		
1 ∗ 1 conv. 192 ReLU, stride 1		
1 ∗ 1 conv. 10 ReLU, stride 1		
averaging over 8 ∗ 8 spatial dimensions		
10-way softmax		

2.2 Key Results of the Original Paper

Original paper experimented the networks in both CIFAR-10, CIFAR-100, and ImageNet. In our project, we only experimented on CIFAR-10, thus we only showed the results on CIFAR-10 here. The result is shown in Table 2.2.

Table 2.2: CIFAR-10 classification error

Model	Error (%)	# parameters
Model A	12.47%	0.9 M
Strided-CNN-A	13.46%	0.9M
ConvPool-CNN-A	10.21%	1.28M
All-CNN-A	10.30%	1.28M
Model B	10.20%	1M
Strided-CNN-B	10.98%	1M
ConvPool-CNN-B	9.33%	1.35M
All-CNN-B	9.10%	1.35M
Model C	9.74%	1.3M
Strided-CNN-C	10.19%	1.3M
ConvPool-CNN-C	9.31%	1.4M
All-CNN-C	9.08%	1.4M

3. Methodology

We implemented the 3 networks derived from model C in the original paper with dropout and without dropout. We also tried to improve the performance of the model and used different initialization method. Also, we put forward out own architecture with data augmentation, batch normalization and padding strategy.

3.1. Objectives and Technical Challenges

Objectives: Our objectives are implementing the architectures in the original paper and show that the all-CNN could perform good enough to replace the max-pooling layer in the standard CNN. And we want to experiment some different architectures to improve the model in the paper.

Challenges: Due to limited time and computing resources, especially our budgets, we only implemented 3 models derived from model C in the original paper. For model A and B, since we wouldn't expect some better results, we just skipped them. Also by using numpy, it was difficult for us to implement the whitening. And that might be the main reason why our results were not as good as the original paper.

3.2. Problem Formulation and Design

Initialization: We tried to use a Gaussian random generator for our initialization. First it performed worse than uniform random generator. But after 5 epochs, they became similar.

Augmentation: First of all, we generated 80,000 augmentation data with rotation from -3 to 3 degrees. Then combined them with our original training data and randomly flipped them during the training process.

Batch Normalization: In our own model, between the last convolutional layer and the first fully connected layer, we used batch normalization.

Dropout: Besides the 3 models in the original paper, we tried models without dropout. In our own model, we kept the dropout layers as it is.

Learning rate: We experimented different learning rates (0.25, 0.1, 0.05, 0.01) in All-CNN-C for the training process. Only 0.01 with weight decay could give us good results. 0.25 and 0.1 would just make the training stop before 30 epochs due to early stopping. And 0.05 would not lead to a good result, thus we choose 0.01 for all models.

4. Implementation

Based on the methodologies in section 2 and section 3, we implemented the deep neural networks on CIFAR-10 dataset.

4.1. Deep Learning Network

In experiments on CIFAR-10, we implement three different network models which are described in the original paper. Architectures of these networks are described in Table 4.1, Table 4.2 and Table 4.3. Based on the three models, we also built our own neural network which is given in Table 4.4.

Table 4.1: Architecture of Strided-CNN

Layer	Layer Description
Input	Input $32 * 32$ RGB image
Conv1	$3 * 3$ conv. 96 ReLU, stride 1
Conv2	$3 * 3$ conv. 96 ReLU, stride 2, dropout 0.5
Conv3	$3 * 3$ conv. 192 ReLU, stride 1
Conv4	$3 * 3$ conv. 192 ReLU, stride 2, dropout 0.5
Conv5	$3 * 3$ conv. 192 ReLU, stride 1
Conv6	$1 * 1$ conv. 192 ReLU, stride 1
Conv7	$1 * 1$ conv. 10 ReLU, stride 1
Pool1	averaging over $8 * 8$ spatial dimensions
Softmax	10-way softmax

Table 4.2: Architecture of ConvPool-CNN

Layer	Layer Description
Input	Input $32 * 32$ RGB image
Conv1	$3 * 3$ conv. 96 ReLU, stride 1
Conv2	$3 * 3$ conv. 96 ReLU, stride 1
Conv3	$3 * 3$ conv. 96 ReLU, stride 1
Pool1	$3 * 3$ max-pooling stride 2, dropout 0.5
Conv4	$3 * 3$ conv. 192 ReLU, stride 1
Conv5	$3 * 3$ conv. 192 ReLU, stride 1
Conv6	$3 * 3$ conv. 192 ReLU, stride 1
Pool2	$3 * 3$ max-pooling stride 2, dropout 0.5
Conv7	$3 * 3$ conv. 192 ReLU, stride 1
Conv8	$1 * 1$ conv. 192 ReLU, stride 1
Conv9	$1 * 1$ conv. 10 ReLU, stride 1
Pool3	averaging over $8 * 8$ spatial dimensions
Softmax	10-way softmax

Table 4.3: Architecture of All-CNN

Layer	Layer Description
Input	Input $32 * 32$ RGB image
Conv1	$3 * 3$ conv. 96 ReLU, stride 1
Conv2	$3 * 3$ conv. 96 ReLU, stride 1
Conv3	$3 * 3$ conv. 96 ReLU, stride 2, dropout 0.5
Conv4	$3 * 3$ conv. 192 ReLU, stride 1
Conv5	$3 * 3$ conv. 192 ReLU, stride 1
Conv6	$3 * 3$ conv. 192 ReLU, stride 2, dropout 0.5
Conv7	$3 * 3$ conv. 192 ReLU, stride 1
Conv8	$1 * 1$ conv. 192 ReLU, stride 1
Conv9	$1 * 1$ conv. 10 ReLU, stride 1
Pool1	averaging over $8 * 8$ spatial dimensions
Softmax	10-way softmax

Table 4.4: Architecture of our network for CIFAR-10

Layer	Layer Description
Input	Input 32 ∗ 32 RGB image
Conv1	3 ∗ 3 conv. 96 ReLU, stride 1
Conv2	3 ∗ 3 conv. 96 ReLU, stride 1
Conv3	3 ∗ 3 conv. 96 ReLU, stride 2, dropout 0.5
Conv4	3 ∗ 3 conv. 192 ReLU, stride 1
Conv5	3 ∗ 3 conv. 192 ReLU, stride 1
Conv6	3 ∗ 3 conv. 192 ReLU, stride 2, dropout 0.5
Conv7	3 ∗ 3 conv. 192 ReLU, stride 1
Conv8	1 ∗ 1 conv. 192 ReLU, stride 1
Conv9	1 ∗ 1 conv. 16 ReLU, stride 1, batchNorm
Pool1	averaging over 8 ∗ 8 spatial dimensions
FC1	10 ReLU fully connected layer, dropout 0.5
Softmax	10-way softmax

All networks were trained using stochastic gradient descent with fixed momentum of 0.9. The learning rate γ was adapted using a schedule of S = [200, 250, 300] in which γ is multiplied by a fixed multiplier of 0.1 after 200, 250 and 300 epochs respectively. Dropout was used to regularize all networks. We applied dropout to the input image as well as after each max-pooling layer (or after the layer replacing the max-pooling layer) respectively. The dropout probabilities were 20% for dropping out inputs and 50% otherwise. Moreover, all models were regularized with weight decay $\lambda = 0.001$. In experiment of our neural network model, we perform data augmentations. These include adding rotated images and horizontally flipping 20% of all images.

In the original paper, All-CNN model was trained on both CIFAR-10 and CIFAR-100 datasets. However, the other two models, Strided-CNN and ConvPool-CNN were trained only on CIFAR-10 dataset. To keep the results comparable, we only trained the models on CIFAR-10 dataset.

4.2. Software Design

Based on the architecture, we designed the process to train the models which is shown in Fig 4.1.

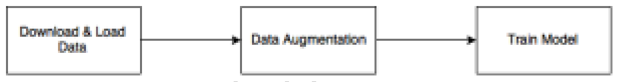

Fig 4.1: The whole train process

The first step is to load the CIFAR-10 dataset into the model. The process for this step is shown is Fig 4.2. At first, we checked whether the dataset existed in the dictionary. If no, we would download the dataset first. Then we would load the train dataset and the test dataset. And we extracted 10% of images from train dataset as validation dataset, the extracted process is shown in Table 4.5.

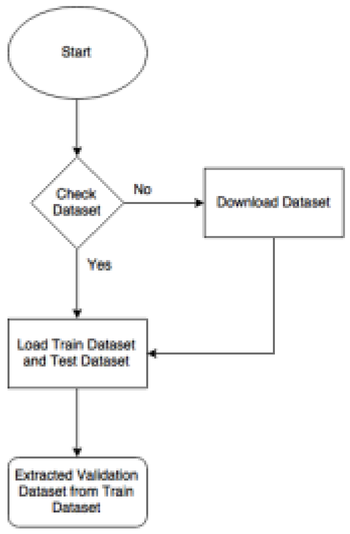
Start
Check Dataset
No
Download Dataset
Yes
Load Train Dataset and Test Dataset
Extracted Validation Dataset from Train Dataset
Fig 4.2: Download and Load Data

Table 4.5: Extract Validation Dataset

valid_set = last 10% original train_set
train_set = first 90% original train_set

The next step is to do data augmentation. The process is shown in Fig 4.3. We would add rotated images first, the rotation angle is between [-4, 4]. We added twice the number of train dataset. After that, we would horizontally flip the images randomly with the 20% probability.

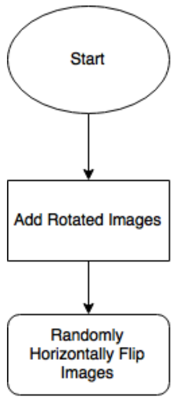

Fig 4.3: Data Augmentation

The last step is to train the model as shown in Fig 4.4.

In train model, first, we need to initialize the weights and biases with random values. We use two schemes: select values from uniform distribution or from Gaussian distribution. Also, we use early stopping to regularize the model. The process of early stopping is shown in Table 4.6.

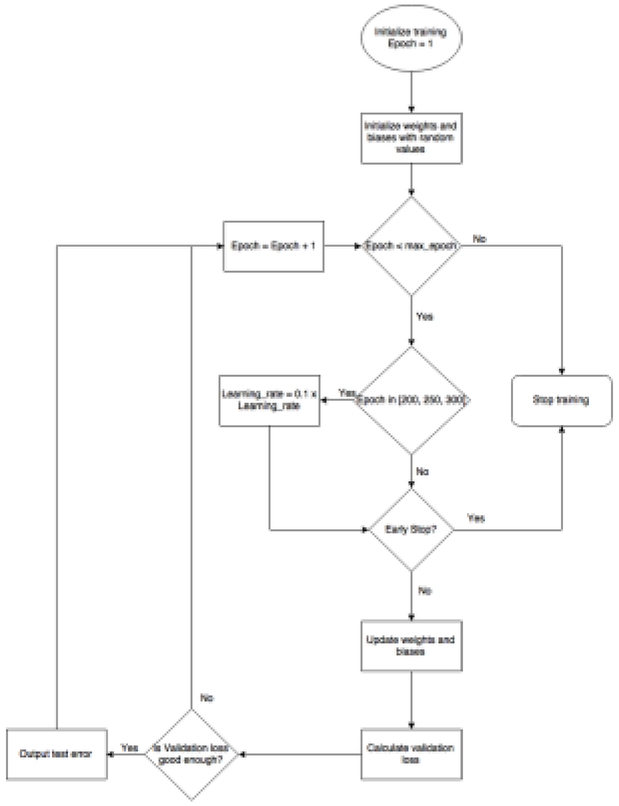
Initialize training
Epoch = 1
Initialize weights and biases with random values
Epoch < max_epoch
No
Epoch = Epoch + 1
Yes
Learning_rate = 0.1 x Learning_rate
Yes
Epoch in [200, 250, 300]
Stop training
No
Early Stop?
Yes
No
Update weights and biases
No
Output test error
Yes
Is Validation loss good enough?
Calculate validation loss

Fig 4.4: Train Model

Table 4.6: Early stopping

```
Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before give up.
Let $\theta_0$ be the initial parameters.
Initializing: $\theta = \theta_0$, i=0, j=0, v = inf, $\theta^* = \theta$, $i^* = i$
While j < p do:
  Update $\theta$ by running the training algorithm for n steps
  i = i + n
  v' = ValidationSetError($\theta$)
  if v' < v then:
    j = 0
    $\theta^* = \theta$
    $i^* = i$
    v=v'
  else: j = j + 1
Best parameters are $\theta^*$, best # of training steps is $i^*$
```

To update the parameters, we use stochastic gradient descent with momentum. The algorithm is shown in Table 4.7.

Table 4.7: Stochastic gradient descent with momentum

```
While Stopping criterion not met do:
  Sample a minibatch on m exmaples from the training Set $\{x^1, \ldots, x^m\}$.
  Set g = 0
  For i = 1 to m do:
    Compute gradient estimate:
    $g = g + \nabla_\theta L(f(x^i; \theta), y^i)/m$
  Compute velocity update: $v = \alpha v - \eta g$
  Apply update: $\theta = \theta + v$
```

5. Results

5.1. Project Results

We trained the model with learning rate = 0.01, number of epochs = 350, batch size = 64. And as the description in the original paper, training using vanilla stochastic gradient descent with momentum could reach state of art performance. So, we did as the paper said.

An interesting observation in our training process was that the test error would sometimes be lower that the validation error.

5.2. Comparison of Results

Table 5.1: CIFAR-10 classification error comparison

Model	Test Error (%)	Training Time (min)
Original Models		
Strided-CNN-C	15.825000%	3248.93m
ConvPool-CNN-C	15.025000%	3540.18m
All-CNN-C	14.125000%	3348.73m
Without Dropout		
Strided-CNN-C	18.562500%	2283.38m
ConvPool-CNN-C	17.250000%	2338.65m
All-CNN-C	16.625000%	2363.73m
Our Model		
All-CNN-C	17.725000%	4027.50m

From the table 5.1 we can see that, ALL-CNN could always outperform other models with a similar training time.

Dropout will cost us more time for training, but is able to provide us with 2-3% improvement of the accuracy. Due to the much more abundant computing resources now than 20 years ago, we are willing to pay for that cost.

Without whitening and global contrast normalization process, it seems hard to get state of the art results as the original paper.

Since our time was limited, we tuned the parameters for each model. Thus we could expect a better outcome if we have more time to do it.

5.3. Discussion of Insights Gained

Whitening and global contrast normalization might be a powerful tool to improve our results.

With modern methods of training CNN, a very simple architecture could perform very well.

Max-pooling operations in the networks do not necessarily improve the performance of CNN.

6. Conclusion

This project implemented 3 different architectures in the original paper. Based on the experiments, we found that a simple CNN would perform very well. Also the experiments explicit max-pooling operations in a network does not always improve performance of CNNs.

Moreover, we also want to experiment the effect of batch normalization and fully connected layer. Thus, we designed a new architecture derived from All-CNN-C. And we found that the result is not as good as All-CNN-C. It seems that adding fully connected layer after all convolution net does not guarantee to improve the performance.

7. Acknowledgement

Thank you very much for the help from professor Zoran Kostic and TAs. Also thanks the people who shared their questions in Piazza.

7. References

[1] https://bitbucket.org/e_4040_ta/e4040_project_xxxd

[2] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806 [cs]*, December 2014.

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning. http://www.deeplearningbook.org

8. Appendix

8.1 Individual student contributions - table

	hc2812	sl3903	hc2813
Last Name	Chen	Luan	Cui
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Build all neural networks. Designed our own model.	Help with building the models. Debug the code.	Help with building the models. Debug the code.
What I did 2	Train Strided-CNN	Train ConvPool-CNN	Train All-CNN and our own model
What I did 3	Write section 2 and section 4	Compare the models. Write section 3 and 5	Write abstract and introduction