Consistent Hashing in Redis

Manindra Kumar Moharan
University of California, San Diego
mmoharana@ucsd.edu

1. Introduction

Convolutional neural networks are a special kind of neural network designed for 2D image input. They've become very popular lately for image classification related tasks, outperforming other state of the art methods such as random forests and SVM. In this project I trained CNNs for classifying images in the CIFAR-10 dataset.

2. Dataset

The CIFAR-10 dataset [3] consists of 60000 32x32 images belonging to 10 classes with 6000 images per class. The training set consists of 50000 images and the test set consists of 10000 images. The classes are mutually exclusive with almost no overlap between images.

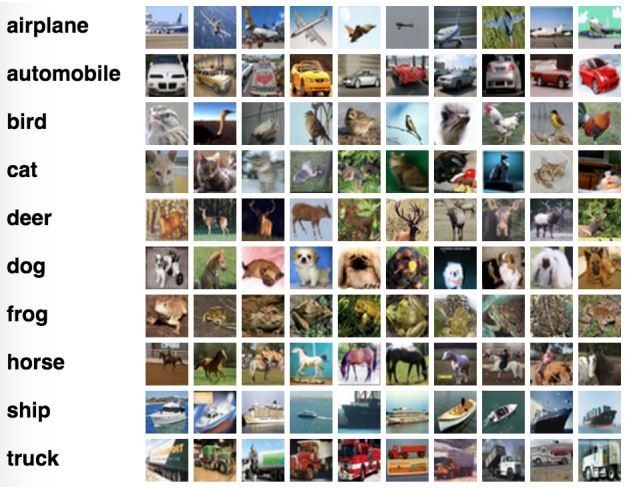

Figure 1. 10 random images from 10 classes of the CIFAR-10 dataset

3. Why CNN?

Traditional neural networks (NNs) receive some input and transform it through a series of hidden layers. Every layer consists of some neurons which are fully connected to all neurons of the previous layer. The neurons in a layer function independently without sharing connections.

In CIFAR-10, images are 32x32x3 (3 color channels), so a regular neural network would have 32*32*3 = 3072 weights. If we need multiple such layers, the number of weights could increase very quickly to an unmanageable degree. Therefore, this full connectivity is undesired and training too many parameters can lead to overfitting. CNNs have a special architecture to take advantage of the 2D shape of images. The layers in a CNN are arranged in 3 dimensions - width, height, depth. The neurons in a layer are connected to a small region of the layer before it, instead of a fully connected manner. Therefore they have lesser parameters than a fully connected network with same number of hidden units.

4. Recent Work with CNNs

We will now take a look at some of the existing work in image classification using CNNs. CNNs were first introduced by Kunihiko Fukushima in 1980 [1]. LeCun et. al. [5] improved the design and proposed LeNet-5, a 7 layer CNN for classifying handwritten digits in the MNIST dataset. Krizhevsky et. al. [4] broke new grounds in CNNs by achieving state of the art error rate on ImageNet LSVRC-2012 contest using a 8 layer deep CNN. Hinton et. al. [2] proposed an enhancement for CNNs using a random dropout layer that greatly reduced overfitting. The state of the art accuracy of 91.78% in CIFAR-10 was achieved by Lee et. al. [6] using their Deeply Supervised Net architecture which uses an additional companion objective function for the hidden layers, that is minimized along with the global loss. The network in network model by Lin et. al. [7] achieved an accuracy of 91.2%.

5. Layers in a CNN

A CNN consists of a number of convolution and subsampling(pooling) layers, followed by a fully connected layer as output.

5.1. Convolution Layer

The convolution layer can be interpreted as a 3D volume containing neurons along its depth [8]. Its parameters

consist of a set of learnable filters which extend along the depth. In the forward process, each filter is convolved with a particular patch of the image along the depth. Intuitively, the network will learn filters that activate when a particular feature is observed in that part of the image.

5.2. Pooling Layer

Pooling layers are usually added between convolution layers in a CNN. This layer is used to reduce the size of the representation so that the computation and number of parameters in the network are reduced. It also helps prevent overfitting. This layer operates on every depth slice of the input layer and reduces it spatially using a pooling function. Common pooling functions are max pooling, average pooling and L-2 norm pooling. For example, if the kernel size is 2x2, it reduces 4 inputs to 1 output by applying the pooling function on the inputs.

5.3. Fully connected Layer

The neurons of this layer are fully connected to all activations of the previous layer.

5.4. Loss Layer

This layer computes the loss with respect to a target and assigns the cost needed to mininmize the loss. The loss is computed in the forward pass and the the gradient w.r.t loss is computed in the backward pass. Common loss functions include softmax loss, euclidean distance loss and hinge loss.

5.5. ReLU Layer

The rectified linear unit is a recently introduced layer that has gained popularity. This layer computes the function $f(x) = max(x, 0)$. It's basically a zero thresholding layer. It has been shown to significantly decrease the training time (by a factor of 6 as reported by Krizhevsky et al.).

5.6. Dropout Layer

The dropout layer is a recent important invention in neural networks by Hinton et. al. [2]. It addresses a fundamental problem in machine learning - overfitting. It accomplishes this by setting certain activations to zero (dropping out) during the training phase. For each training example, a different set of random units is selected to drop. The number of activations to be dropped is controlled by a dropout ratio parameter.

6. Caffe

Caffe is an open source deep learning framework developed by the Berkeley Vision and Learning Center. It's implemented in C++/Cuda and provides command line, python and Matlab interfaces. Caffe is built with speed in mind, and provides seamless switching between CPU and GPU. Another advantage of Caffe is that models can be written entirely using schemas, without writing any code. For this project, I'll be making use of Caffe to train multiple CNNs for CIFAR-10 classification.

7. Experiments

I created and trained multiple Caffe models and evaluated their performance on the CIFAR-10 dataset. For maintaining consistency, I trained each network for 20K epochs using SGD and compared their performance on the test set. I used a learning rate of 0.001 for the first 10K epochs and 0.00001 for the next 10K epochs. I used the softmax loss function in the last loss layer.

The first network I tried is a 4 layer network consisting of 2 pairs of convolution and max pooling layers. It's architecture is described in table 7.

Layer Type	Parameters	Outputs
CONV	Kernel: 5x5, Pad: 2, Stride: 1	32
MAX POOL	Kernel: 3x3, Stride: 1	
CONV	Kernel: 5x5, Pad: 1, Stride: 1	32
MAX POOL	Kernel: 3x3, Stride: 1	
FC		10

Table 1. CNN 1

This network had an accuracy of 72.91% after 20K epochs. I had tried different kernel sizes of 3x3, 5x5 and 7x7, and zero padding values of 1,2 and 3. Kernel size of 5x5 for convolution layers and 3x3 for max pooling layers gave me the best accuracy.

I tried adding ReLU layers at the end of each max pooling layers in the above network. This improved accuracy by a decent margin - 75.81%. Thus, proving that ReLU units indeed decrease training time, as the network was able to achieve higher accuracy with same number of epochs.

After this, I tried the 6 layer network by Krizhevsky and also added ReLU layers, as described in table 2.

Layer Type	Parameters	Outputs
CONV	Kernel: 5x5, Pad: 2, Stride: 1	32 + ReLU
MAX POOL	Kernel: 3x3, Stride: 1	
CONV	Kernel: 5x5, Pad: 1, Stride: 1	32 + ReLU
AVG POOL	Kernel: 3x3, Stride: 1	
CONV	Kernel: 5x5, Pad: 1, Stride: 1	64 + ReLU
AVG POOL	Kernel: 3x3, Stride: 2	
FC		10

Table 2. CNN 2

This layer makes use of 1 max pooling layer and 2 average pooling layers. It has an accuracy of 77.39%. I tried adding dropout layers to the above network to try to improve accuracy. I modified the network to the following:

Layer Type	Parameters	Outputs
CONV	Kernel: 5x5, Pad: 2, Stride: 1	32 + ReLU
MAX POOL	Kernel: 3x3, Stride: 1	
CONV	Kernel: 5x5, Pad: 1, Stride: 1	32 + ReLU
AVG POOL	Kernel: 3x3, Stride: 1	
DROPOUT	Dropout ratio: 0.25	
CONV	Kernel: 5x5, Pad: 1, Stride: 1	64 + ReLU
AVG POOL	Kernel: 3x3, Stride: 2	
DROPOUT	Dropout ratio: 0.25	
FC		10

Table 3. CNN 3

Adding dropout further improved the accuracy to 79.57%. I tried other dropout ratios like 0.10, 0.20 and 0.50. 0.25 dropout gave the best performance. The filters of the three convolution layers are shown in figures 2, 3, 4.

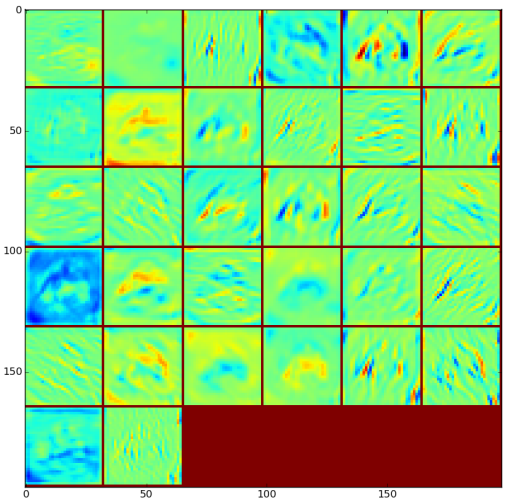

Figure 2. Filters in first convolution layer for CNN in table 3

As one can see from the filter images, the earlier stage filters activate on large image patches whereas deeper filters learn to activate on small/specific locations in the image. The intuition is that with successive layers, filters learn to focus on specific small features within an image.

Finally I compared my results to the reference CIFAR-10 classifier included in Caffe. This net makes use of 2 normalization layers in addition to the Krizhevsky network. This net achieves an accuracy of 77.48%.

8. Results

I trained 4 different networks in Caffe, successively iterating and improving the accuracy. The results are summarised in table 4. My final 6 layer model using relu and

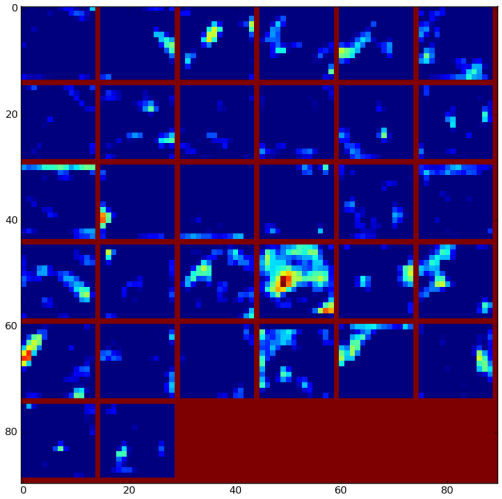

Figure 3. Filters in second convolution layer for CNN in table 3

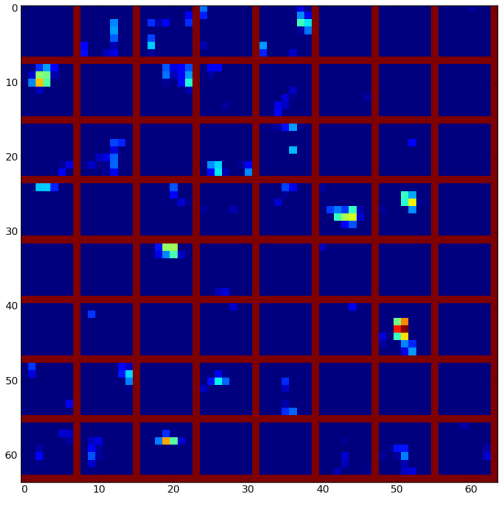

Figure 4. Filters in second convolution layer for CNN in table 3

dropout was able to beat the reference model included in Caffe by a small margin. I observed that adding higher number layers improves performance at the cost of training time. But adding too many layers can also have the potential downside of overfitting to the training data and performing badly on the unseen test data. Dropout can help mitigate this to a certain extent.

Model	Accuracy
4 layer	72.91%
4 layer + ReLU	72.91%
6 layer + ReLU, Avg Pooling	77.39%
6 layer + Dropout, ReLU, Avg Pooling	**79.57%**
Caffe Model + ReLU, Avg Pooling, Normalization	77.48%

Table 4. Results

9. Conclusion and Future Scope

In this project, I got to learn how to prototype models using the powerful Caffe framework and achieved decent results on the CIFAR-10 dataset. If I had more time, I would have tried to the train the models for much higher number of epochs. The state of the art models are trained for 150000+ epochs. Ensemble models are also known to improve performance. Data augmentation (mirroring, rotations, affine transformations) is another technique used to increase the size of training data and thus improve performance.

In the future, I would like to train deeper networks for higher number of epochs and also explore how to use ensemble models and data augmentation techniques for improving performance.

10. Acknowledgements

I would like to thank Professor Zhuowen Tu and 2Lab for providing access to their lab server with which I was able to use GPUs to speed up the training in Caffe.

References

[1] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

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

[3] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

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

[7] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[8] Stanford-CS231N. Convolutional neural networks. `http://cs231n.github.io/convolutional-networks/`. Accessed: 2015-03-20.