CS231N Project Report - Tiny Imagenet Challenge

Arijit Banerjee
Stanford University
arijitb@stanford.edu

Vignesh Iyer
Stanford University
vigansub@stanford.edu

1. Introduction

Image Classification is a major problem in computer vision today, where rapid strides are being made towards progress. The idea of using many convolutional layers which involve heavy computation, and learning over these layers has drastically improved the performance of image classification and recognition algorithms around the world.

The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object classification and detection, with millions of images and hundreds of object classes. In the ILSVRC 2014, which is large-scale visual recognition challenge, almost every highly ranked team used CNN as their basic framework. The winner GoogLeNet reduced the mean AP of object detection to 0.439329, and reduced classification error to 0.06656, the best result to date. Its network applied more than 30 layers. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans.

Convolutional neural networks have revolutionized solutions to learning problems over the last few years. This has been particularly evident in the area of computer vision, and in using deep learning to solve problems. There is a wide database of information available today, including ImageNet for large image collections, or ShapeNet for large 3-D shape collections which can be used to understand various problems of different magnitudes of difficulty. For instance, deep learning has been widely used for shape reconstruction from an image network using orientation information [2]. There has also been wide-spread usage of deep learning in problems such as face recognition [3] and depth inference [4].

In this project, we have tried to look at understanding how we could perform recognition of various images using a deep learning model. We have tried to do this using various architectures, more specifically the Network In Network[5] and Alexnet architectures[6]. We have also tried to understand how we could combine both these models to perform better than the individual models.

2. Problem Statement

In this class project, the Tiny ImageNet visual recognition challenge, there are 200 different classes. The training data has 500 images per class, with 50 validation images and 50 test images, with the validation and training images provided with labels and annotations. The problem statement requires us to predict labels, without needing us to annotate the test images.

In our project, we used convolutional neural networks, experimenting with various convolution architectures, and toying around with learning rates, loss functions, regularization rates, so as to maximize our chances of accurate image recognition. We also used techniques such as dropout, and data augmentation to minimize the amount of overfitting.

In particular, we have adapted in our project ideas of NIN (Network in Network) and Alexnet architectures. We shall discuss broadly the salient features of this architecture in the next couple of sections , following which we shall discuss how we implemented our approach to solving this problem, and the results obtained thus.

3. Architectures used

3.1. Network In Network (NIN)

The NIN architecture is a method to replace the Generalized Linear Model in a Convolutional Neural Network with a non-linear model instead. The motivation behind using this architecture is that the convolution filter in a basic Convolutional Neural Network is a GLM, and that

the level of abstraction in this case is fairly low. The idea is to move to a more non-linear model where the abstraction capacity of the local model can be boosted.

The NIN architecture employs a learning setup with the Generalized Linear Model being replaced here by a micro-network. This structure approximates a general non-linear function. The multi-layer perceptron idea is used to generate this micro network, which is further used as a universal function approximator, and a neural network trainable by back-propogation. The resulting layer is called the mlpconv layer, as compared to the linear convolutional layer. A block diagram of the same is provided in Figure 3.1. Also, a general form of how the Network In Network model is structured in whole is shown in Figure 3.1.

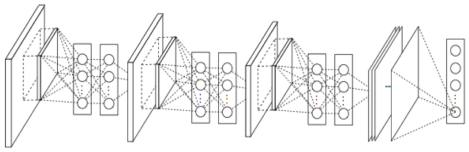

Figure 1. The Network In Network structure

The MLPConv maps input local patch to the output feature vector with a MultiLayer Perceptron (MLP) consisting of multiple fully connected layers with nonlinear activation functions. While ReLU (Rectified Linear Unit) is still used as the Activation function in also the multilayer perceptron, it follows the mlpconv layer here, as opposed to the linear convolutional layer. As the figure above indicates, there are fully connected multi-layer perceptron layers in between the individual set of layers, and this is how the non-linearity is achieved.

The NIN also uses a global average pooling strategy at the last mlpconv layer. Instead of adding fully connected layers on top of the feature maps, averages of each of the feature maps is computed, and results fed into the softmax layer. The below figure shows the overall structure of the NINs. This image shows the stacking of three mlpconv layers, followed by one global average polling layer. In traditional CNN, it is difficult to interpret how the category level information is passed back to the previous layers through the fully connected layers, and hence it works as a black box there. But here, global average pooling enforces correspondences between categories and features, because of the strong local modeling of the micro network, and therefore, seems more meaningful and interpretable. Also, while fully connected layers are typically cases of overfitting and dropout regularization is vital there, the global average pooling is a structural regularizer, which prevents overfitting for the overall structure.

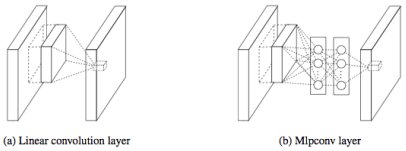

Figure 2. Linear Convolutional Layer as compared to MLPConv

3.2. AlexNet

The AlexNet is a CNN model that uses 8 layers, 5 convolutional layers and 3 fully connected layers. The last fully connected layer has it's output connected to an $n-$ way softmax, which produces a distribution over the n class labels. The network is designed to maximize the multinomial logistic regression objective, which is equivalent to maximizing average across training cases of the log-probability of correct label under probability distribution.

The Alexnet has response-normalization layers following the first and second convolutional layers. It also has max pooling layers following both the response-normalization layers and the fifth convolutional layer. These are followed by the three fully connected layers. It also minimizes overfitting by using dropout and data augmentation techniques.

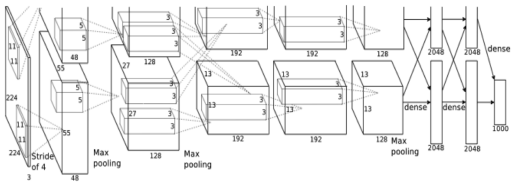

Figure 3. Linear Convolutional Layer as compared to MLPConv

The structure of the AlexNet is shown in Figure 3.2 above. This image from the AlexNet paper is representative of the case where there are 2 GPUs, while in our case we use only 1 GPU. In the Alexnet, we continue to use ReLU as the non-linear activation function as in traditional CNNs and NIN. ReLU does not require input normalization to

prevent saturation, but normalization aids generalization. Therefore, the output of the ReLU non-linearity is further normalized, and this is called the response-normalization, which has been discussed earlier.

We attempt to maintain the crop-ratio used in the Alexnet paper, by using a 58 x 58 cropped image from the 64 x 64 image, as compared to the 224 x 224 image from the original 256 x 256 in the paper.

Once we had both these models working, we used them on the Tiny Imagenet challenge, following which we also tried to do the same using ensembles, and observe if they perform any better.

4. Motivation and Technical Approach

We now explain the approach we used on the Tiny Imagenet dataset and explain our motivations for doing so. Our plan for maximizing accuracy within a limited time frame was to quickly implement a very deep convolutional neural network and train it for as long as possible. To minimize the problem of overfitting, we used dropout and data augmentation.

During the milestone, we mentioned that we used a **t2.medium** EC2 instance for implementing K-Means. This was no longer feasible since convolutions would be too slow on CPU. This time, we used GPU optimized implementations provided by Caffe on terminal.com for training. In total, we trained 5 models for a total of about 150 hours (for a personal cost of $15, after hitting the referral bonus limit).

As mentioned in the earlier section, the first thing we wanted to do was choose an architecture. Since we wanted to optimize for GPU cost (and thus time), it seemed prudent to train several architectures for a short amount of time, and choose the best one and train this for a longer period of time (although this method might cause slight overfitting on the validation set). We decided to compare the well known Network in Network (NIN) and AlexNet architectures against each other. We used 'prototxt' files provided in Caffe's Model Zoo for training, with minor modifications (we added cropping and mirroring for data augmentation, along with changing the size of the final layer to correspond to the 200 classes we have on Imagenet).

We chose a few different learning rates (in the 0.01 - 0.001 range) and trained our models on the above architectures. We observed that AlexNet tended to saturate after a few tens of epochs while NIN continued learning as evidenced by the training and val accuracy continuing to increase steadily. After a day of experimenting with this, we decided to stop training AlexNet and use the best parameters obtained using NIN to train an ensemble of classifiers.

An important aspect while training was to figure out when to drop the learning rate. We noticed that if the learning rate remained constant for too long, beyond a point, the training and val accuracy stopped increasing while the loss kept fluctating. In practice, we noticed that this happened every few tens of epochs. Since we wanted to minimize the amount of time spent on training, we thought that a general policy like 'reduce learning rate every K epochs' might end up wasting a lot of time before actually lowering the learning rate. Thus we manually looked at the learning rate every few hours and dropped it by a factor of 10 when we felt that the loss function was not steadily decreasing any more.

While training NIN, we used cropping (with a 'crop_size' of 48) in 2 of the models and no cropping in the third model. We enabled mirroring of the training data in all the models. After training these classifiers, we combined their results (along with the results of the two previously trained AlexNet) classifiers into an ensemble of classifiers. We implemented the ensemble by summing up scores for each class weighted by the accuracy of the corresponding classifier on the validation set. We found out that our ensemble performed best on the validation set when we used only the 3 models trained using NIN, and it is the score of this result on the test set that constitutes our final score.

5. Evaluation and Results

We now describe the main numbers and results along the way towards our final score. As mentioned for the milestone, K-Means achieved an error rate of $\mathbf{0.982}$.

For our first pass (trying out AlexNet and NIN for a few epochs), the best set of hyperparameters for AlexNet involved using a Learning Rate of $\mathbf{0.01}$. This achieved a validation set accuracy of $\mathbf{0.25}$ within 20 epochs after

which the loss started fluctuating. At this point, the Learning Rate was dropped to $\mathbf{0.001}$. Now, the loss function started steadily dropping again until about 30k iterations. At this point, the model had an accuracy of about $\mathbf{0.33}$ but the training and val accuracy showed clear signs of overfitting (the training loss was decreasing steadily while the validation loss was not). We decided to stop training AlexNet at this point.

While training AlexNet, we ran a NIN model on another machine for the same amount of time. At the time we decided to stop training AlexNet, NIN had achieved a greater accuracy on the validation set of $\mathbf{0.35}$, and was not showing any signs of overfitting. This could have been because it was a deeper model than AlexNet (epochs were talking longer to complete). Thus we decided to continue running this NIN model and spin up two other versions of this model with slightly different parameters to create our ensemble.

The 'prototxt' file for NIN has a few dropout layers, and we experimented with parameter values for dropout here. As mentioned earlier, dropout along with 'crop_size' and 'mirror' were the parameters we used to prevent overfitting.

All of our models used SGD with Momentum for gradient descent, with a momentum of 0.9. Snapshots were stored every 2 epochs so that we could recover from failures (we ran out of disk once, luckily, this did not create too many problems). A batch size of 100 occupied about $\mathbf{400\,MB}$ on the GPU and seemed to work fast in practice, though we could have probably optimized this a bit more (since the GPU had a much larger capacity) for faster training.

While training our ensemble, we dropped learning rates for each model manually as described previously. The first drop in learning rate occurred after about $\mathbf{40}$ epochs for each model when the val accuracy was about $\mathbf{0.37}$. After a few more hours (about $\mathbf{30}$ epochs), the learning rate was again dropped (at which time the accuracy was about $\mathbf{0.42}$) to squeeze out a few more percentage points in val accuracy. Our best single model achieved a validation set accuracy of about $\mathbf{0.47}$.

We submitted each of our models to the online judge, and received error rates on the test set of about $\mathbf{0.57}$, corresponding to an accuracy of $\mathbf{0.43}$. We are not sure where this 4% drop in accuracy came from, though we surmise it could be because of the randomness in the forward pass (because of cropping and mirroring) present during training but not during testing.

Finally, we modified our 'prototxt' file for testing to output losses rather than class labels. Using this, we built an ensemble of 5 classifiers (2 AlexNet + 3 NIN). We added up loss functions (with some hacks to neglect scores for a particular class if they came from really low ranks) and used combinations of different classifiers as ensembles. Our best ensemble used only the 3 NIN models (which achieved much greater accuracy than the AlexNet models) and got a final error rate of $\mathbf{0.53}$ on the online judge, placing us $\mathbf{4th}$ overall on the leaderboard.

We had a few more ideas for improving our error rate, but unfortunately at this point we were out of time and GPU credits, and decided to stop here.

6. Comments and Ideas for Future Work

Below are some comments and ideas we had in mind towards improving accuracy but could not really get around to implementing:

- Architecture: The AlexNet architecture is optimized for 256x256 images on the ImageNet dataset while the NIN architecture we used was optimized for the CIFAR-100 dataset, which contains 32x32 images. Since we were using 64x64 images, we got around this by changing the kernel size of the final pooling layer in NIN (and no change in AlexNet). However, our architecture was now not necessarily optimized for 64x64 images and this might have been something to look at.

- Test with Random Crops: We trained using random crops, but did not use a variety of random crops at test time. This would definitely lead to an increase in accuracy, but we ran out of time before we could try this out.

- PRELU units: In the milestone, we had mentioned wanting to try out Parametrized Rectified Linear Units [7], a new kind of RELU unit where the parameter is learned as training proceeds. We made some slight process on implementing this in Caffe, but could not figure out how to change parameters easily in the new layer we were adding. Had we completed implementing this, we might have got faster training rates. However this may not necessarily have helped much since our main problem was that our models saturated

(though it might have helped in creating more ensembles).

- Effective Ensembling: We did not exploit model wise bias towards each class while merging ensemble results. Some kind of EM algorithm to merge results based on class wise confusion matrices for each model was something we thought about trying but could not get around to (also, this isn't really related to CNNs).
- Distribution of Classes: We expect about 50 images per class on the test set labels, on average. We thought about breaking close ties while choosing labels to ensure that this holds, but this would not really have helped. In fact, we calculated the sum of absolute deviations of class count from 50 for each class for each of our submissions and plotted this against test set accuracy, but did not observe much correlation.

References

[1] http://arxiv.org/abs/1502.01852

[2] Scott Satkin, Maheen Rashid, Jason Lin, Martial Hebert , "3DNN: 3D Nearest Neighbor", International Journal of Computer Vision January 2015, Volume 111, Issue 1, pp 69-97

[3] http://arxiv.org/abs/1406.6947

[4] Hu Tian, Bojin Zhuang, Yan Hua, Anni Cai ; "Depth Inference with Convolutional Neural Network", VCIP 2014: 169-172

[5] arxiv.org/abs/1312.4400

[6] A. Krizhevsky, I. Sutskever, G.E.Hinton. "ImageNet Classification with Deep Convolutional Neural Networks", Eletronic Proceedings of Neural Information Processing Systems. 2012.

[7] http://arxiv.org/pdf/1502.01852.pdf Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification