Tiny ImageNet Challenge

Vani Khosla
Stanford University
vkhosla@stanford.edu

March 13, 2016

Abstract

This project aims to perform image classification using a Convolutional Neural Network in Keras on the Tiny ImageNet Dataset. The goal is to find a network architecture that provides the best accuracy on the validation and testing set. The results from this project include training accuracies up to 60% and validation accuracies up to 25%, however testing accuracies showed performances no better than random guessing.

1 Introduction

Image classification is a fundamental problem in machine learning. There have been many successful efforts to solve the problem of classification. This project aims to construct a solution to the image classification problem through the use of convolutional neural networks in order to provide a useful network and a learning opportunity for understanding the impact of convolutional networks. This project will use a specific image dataset to train and test the network with expected results in order to help evaluate the technical approach to building the network.

The overall plan to approach this problem is to create different network architectures consisting of convolutional layers, pooling layers, and fully connected layers to build the best performing network possible. Past performance on the Tiny ImageNet dataset ranges between 20% and 46% accuracies on the validation set. The model architectures selected are chosen to aim to reach the higher percentage, but is expected to be within this range. Overall, the goal of this project is to use this time as a learning opportunity for creating and understanding the performances of convolutional neural networks and working with visual datasets.

2 Relevant Work

Because image classification is fundamental and well approached problem in machine learning, there are many resources available for learning and analyzing results from networks that accomplish image classification. The classification problem can be solved in many ways, including a k-nearest neighbor algorithm. While an effective model can be created, it is not necessarily the most efficient model as the algorithm requires a comparison between the test image and every image in the training set. Convolutional neural networks are more commonly being used for the classification problem now because even though training test is significantly longer, the prediction of the test data is much faster. Within the work done with convolutional neural networks, most

works have increased performance of the network through the use of deeper and wider networks. Additionally problems arise from overfitting, which many models have accounted for by including techniques like dropout and other regularization methods.

The ImageNet Large Scale Visual Recognition Challenge provides a helpful layout of progress made for the image classification problem. Many new ideas are first applied in this "challenge" format, allowing standardized evaluation and comparison methods for improvements and radical ideas. While the use of a convolutional neural network has proven to have dramatic improvement in the results from the challenge, there are many techniques that have emerged from the challenge results. Methods such as max pooling, dropout, and batch normalization have proven to have an impact on performance. Additionally, changing filter sizes (sizes of 5x5 and 3x3 have been found as effective sizes) and the number of filters have also proven to have an impact on performance, which allows for changing the depth of width of the network.

3 Methods

3.1 Framework

The framework used for this project is Theano. Theano is written in Python and is a popular choice for machine learning applications. There are several libraries built for Theano that create an easier layer on top of Theano's interface, making the API easier to navigate. Overall, this framework allows for more comprehensive control over the network formation than both Caffe and Torch, and has many deep learning libraries including Keras and Lasagne. Keras was selected as the deep learning library to use for this project. Keras is a highly modular library written in Python and is capable of running either Theano or TensorFlow. It was selected for this project because it is easy to use and and allows for fast experimentation of different models. While Keras allows for training on both a CPU and GPU, all models for this project were trained on a NVIDIA GRID K520 GPU (Amazon g2 instance).

3.2 Network Layers

There are six main layers used in each network architecture:

1. Convolutional 2D Layers:

   Convolution operator for filtering windows of two-dimensional inputs. The number of filters and filter size are defined within each layer.

2. Pooling:

   Max pooling is a form of non-linear down-sampling. For all model architectures, a pool size of 2x2 was used.

3. Dropout:

   Dropout consists in randomly setting a fraction p of input units to 0 at each update during training time, which helps prevent overfitting. Dropout probabilities used in the model architectures were either 0.25 or 0.50 (generally at the fully connected layer).

4. Batch Normalization:

   A normalization method used which is done by a batch of activations with each dimension unit gaussian. It is defined as $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$

5. reLu:

   ReLu is the activation function that was selected for all model architectures. ReLu activation function is defined as $max(0, x)$.

6. Fully Connected Layers:

   A Fully Connected Layer works the same as in an ordinary Neural Network. Neurons in this layer have connections to all other activation layers, allowing to activations to be computed using matrix multiplication.

In addition to the layers used above, the loss function used in all models was a softmax loss function. Two different optimizers were experimented with:

1. Stochastic Gradient Descent (SGD) with Momentum:

   SGD with momentum is an update function to the weights by the following equation: $w := W - \eta \bigtriangledown Q_i(w) + \alpha \bigtriangledown w$

2. Adam

   Adam is an update step that combines momentum and an RMSprop like update with a bias correction. It is a popular choice to use as an optimizer.

Using the six main layers described and the two optimizers, three different model architectures were designed and are described in the next section.

3.3 Model Architectures

There were three different models built for experimentation. All models were built from scratch (including no pre-trained weights), using an L2 regularization rate of 0.01 and Gaussian initialization scaled by fan in for weight initialization. In addition, all models were trained using batches of size 128. The three models are described below.

Model 1:
conv3-32 - relu - conv3-32 - relu - max pool - dropout - conv3-64 - relu - conv3-64 - relu - max pool - dropout - fc(512) - relu - dropout - softmax

Model 2:
conv3-32 - relu - conv3-32 - relu - max pool - dropout - conv3-64 - relu - conv3-64 - relu - max pool - dropout - conv3-128 - relu - conv3-128 - relu - max pool - dropout - fc(512) - relu - dropout - softmax

Model 3:
conv5-32 - relu - conv3-32 - relu - max pool - conv3-64 - relu - max pool - conv3-128 - relu - max pool - fc(256) - relu - dropout - fc(200) - softmax

Note: conv3-32 indicates a convolutional layer with 32 filters of size 3x3, and fc(512) indicates a fully connected layer 512 units.

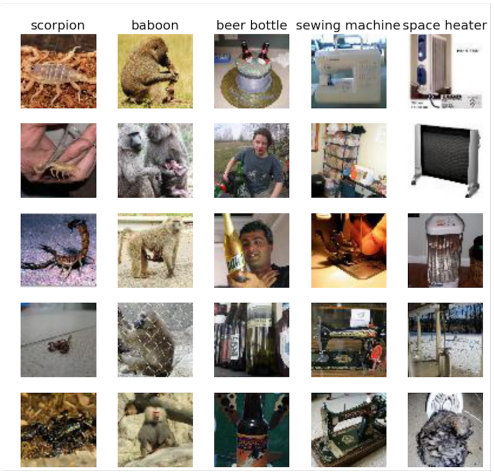

Figure 1: Five classes from the dataset, including five image examples of each class.

4 Dataset

The dataset used in this project is the Tiny ImageNet dataset, as provided by the Neural Networks class

at Stanford, is a subset of the ILSVRC2014 dataset. The dataset consists of 100,000 training images, with 200 classes (see Figure 1 for examples of classes and images that belong to each class). Each class has 500 training images, 50 validation images, and 50 testing images. Each image has an input size of 64x64. For this project, preprocessing of the images was done by subtracting the mean image and normalizing by the variance.

5 Results and Discussion

The results from the three models can be seen in Table 1. As the table shows, relatively good training accuracies were achieved (between 55% - 65%) on two of the models. However the best validation accuracy achieved was around 25%, which is not as accurate of a result as desired. In addition the test accuracies (not reported in the table) were no better than the result from random guessing, indicating that none of the models were sufficient after training.

Model	Training Accuracy	Validation Accuracy
1	0.5693	0.2435
2	0.1871	0.1799
3	0.6236	0.1560

Table 1: Training and Validation Accuracies for each Model.

In order to identify what went wrong in training of the models, an investigation into Model 1 was done. From the training phase of the model, the training and validation loss can be seen in Figure 2. As the figure shows, the training loss is going down over many epochs, but starts to plateau in it's decrease around epoch 100. This is explains why the training accuracy was able to produce reasonable results, as the loss is going down and there is improvement being made during the training phase. One thing to note is that this model used Adam as the optimizer, which explains why there is good decrease in the loss in the early epochs, and at later epochs there is less decrease in the loss as more fine tuning needs to happen. The training loss is not yet below 1, which is where it would be expected before seeing a good result. In contrast, the behavior of the validation loss is not what is expected from a reasonable model. The validation loss decreases in a reasonable fashion until around epoch 20, where it then takes a turn and steadily increases for the rest of training. This demonstrates why the validation accuracy is not improving even as training loss decreases and training accuracy increases. More thorough investigations need to be done in order to determine why the validation loss is not yet where it should be.

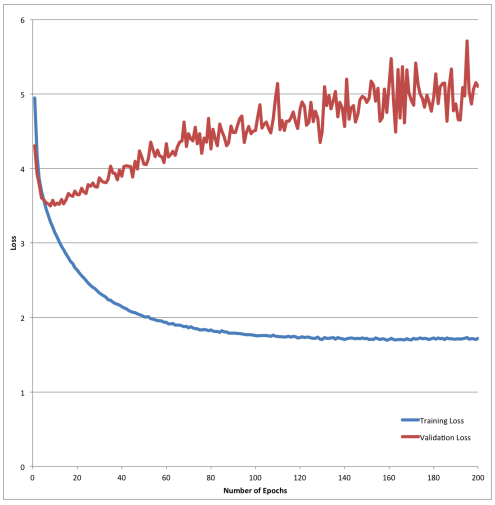

Figure 2: Training and Validation Loss for Model 1.

To understand why the validation loss and accuracies are not behaving in the expected manner, a visualization of the model was done to see what is happening to the filters at each layer. Two examples have been included in this paper (see Figure 3 and Fig-

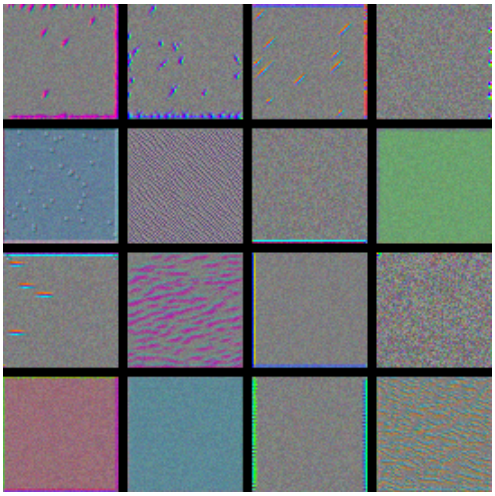

Figure 3: Visualization of a few filters from the first layer in Model 1.

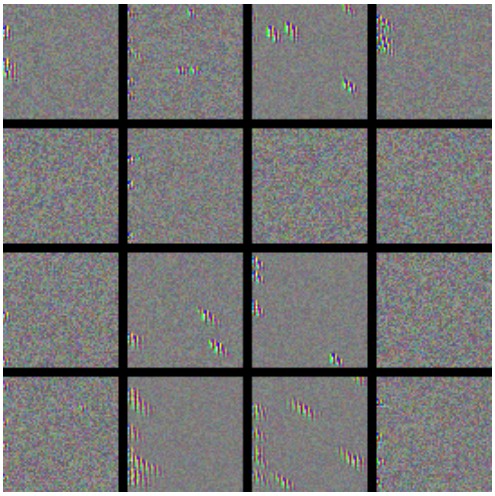

Figure 4: Visualization of a few filters from the seventh layer in Model 1.

ure 4). In Figure 3 we can see the filters from the first layer of the model. The filters in this image display moderate results, better visualizations of the filters are expected, but the results are not terrible. While the visualizations of the filters from the first layer have some interesting results, it is more the growth from layer 1 to layer 7 that is telling. Figure 4 shows visualizations of some of the filters from layer 7, and these visualizations demonstrate that the results are not good. Much stronger visualizations would be expected from a reasonably performing model, but from these two figures the progression shows that the weights are not yet being trained well enough as the filters do not show any strong improvement.

As mentioned earlier, the results provided from model 1 are from the training instances in which the optimizer Adam is used for training. While this provided good learning rates in the early epochs, it did not necessarily lead to great learning rates overall. Between the two optimizers selected (Adam and SGD with momentum) to investigate Adam performed significantly better than SGD on all models. This is most likely due to the fact that Adam was able to learn more quickly by taking larger steps (i.e. larger learning rates), which was seen as a better result in the first 200 epochs. Although Adam did perform better, overall with more training time and more epochs that expectation is that SGD would perform better over the long run since it would be able to fine tune a little more than Adam. However for the purposes of this project, Adam vastly outperformed SGD, as SGD failed to learn in some training instances of the models.

Given the results of this project, it is apparent that one of the key reasons the models underperformed with respect to the validation and training sets is due to the fact that all models were trained from scratch, there was no pre- training involved including the weight initializations. This served to show that training a model from scratch is a difficult thing to do, which helps explain why many models use pre-trained weights to help speed up the training process.

6 Conclusion

Overall the results found from the three models were disappointing as they all underperformed on the validation and test sets. While two of the models were able to achieve reasonable accuracies on the training sets, the visualization of the filters indicated that the models were not trained well. Moving forward there are a couple of ways to combat this: using pre-trained weights to help the learning rate achieve better accuracies by learning on the loss for the models or allowing for much longer training time since the models are being trained from scratch. In addition, this project demonstrated that performing well on the training set is not difficult to accomplish even with a bad model, and the validation and test sets are necessary to truly evaluate the performance of the model.

Even with good training accuracies there are many problems that persist in image classification; some of the things that make image classification a difficult problem include the scaling of images, size of the item being classified in contrast with the rest of the image, noisy images, distorted images, resolution of the photos, and many more. While this dataset has a good number of training items, these challenges can still persist across the validation and test sets. While the results from this project did not prove to be competitive in the Tiny ImageNet Challenge, this project highlighted the difficulty of training models from scratch, and indicated that training requires significantly more time before reasonable performance can be seen on the validation and test sets.

References

[1] Chollet, Franois. *Keras*, GitHub 2015. GitHub repository, https://github.com/fchollet/keras.

[2] Chollet, Franois. *How Convolutional Networks See the World*, 30 January, 2016. The Keras Blog, http://blog.keras.io/how-convolutional-neural-networks-see-the-world.html.

[3] K. He, X. Zhang, S. Ren, and J. Sun *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, arXiv preprint arXiv:1409.4729, 2014.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*, University of Toronto.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. *Imagenet large scale visual recognition challenge*, arXiv preprint arXiv:1409.0575, 2014.

[6] Simonyan, Karen and Andrew Zisserman. *Going Deeper with Convolutions*, arXiv preprint arXiv:1409.1556, 2014.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv preprint arXiv:1409.4842, 2014.