Application of Convolutional Neural Networks to the Tiny ImageNet Challenge

Alex Martinez
Stanford University
450 Serra Mall
alexm711@stanford

Jacob Waggoner
Stanford University
450 Serra Mall
jacobw1@stanford.edu

Abstract

Combining results from state of the art convolutional neural networks, we train a network for image classification on the Tiny ImageNet dataset. We experiment with preprocessing the data by adding a gradient channel to the data. Our model uses use dropout, PReLUs, data augmentation, and model ensembles as ways to introduce and integrate over noise that better captures the space of input for each category. Using these methods, we were able to achieve an accuracy of 25.5% on the validation set. Although our results are unimpressive, we learned a great deal through the process of training the net and evaluating its (and our) errors, and expect future uses of CNNs to be substantially demystified by the experience.

1. Introduction

Although convolutional neural networks (CNNs) are not new, it was not until recently that they became computationally tractable and their efficacy gained recognition in the field. In recent years, CNNs have overtaken more traditional learning algorithms, including non-convolutional neural nets and regression models, in their capacity to achieve high accuracies at only the cost of increased training time.

Though not perfectly well understood, three broad themes have emerged as CNNs have gained popularity. First is the importance of retaining and curating relevant information in the training phase. Second is that deep networks with slim layers (few parameters) enjoy reasonable computation times while still yielding high classification accuracies. The third is that, in light of the vast space of possible inputs that correspond to just one category in image classification, CNNs generally benefit from the introduction of noise in training that is then integrated over in testing. The addition helps to increase the representativeness of each input image for its respective category. Our approach for designing a CNN was in keeping with these three trends, as is the format of this paper.

To that end, our approach was a three-tiered one. First, we attempted to extract the most relevant information from each training image by using preprocessing filter techniques and image flipping. To retain that information throughout the training process, we attempted to make use of state of the art advances from [1] and [2], PReLU and fractional max pooling, with varying degrees of success. Second, we designed a lightweight network architecture that fit with our computational (and therefore monetary) and memory constraints based on the results documented in [5] and [7]. Finally, we attempted to implement the advances made by [dropout reference], [ensemble reference], and [Szegedy et al] in an effort to increase the representativeness of the learned weights. These were, respectively, dropout, model ensembles, and adversarial examples.

2. Background / Relevant Work

Because the task of generic image classification is not a new one, our model uses tools designed specifically for the task and techniques that are informed primarily by the wealth of research that already exists.

The single most important tool that we used was Berkeley Vision and Learning Center's "caffe." It allowed us to quickly and easily prototype architectures and provided a suite of pre-built functions common to the CNN community. These implementations spared us from reinventing the wheel, and, in the case of PReLU, saved us from hopelessly implementing a python variant of PReLU within the Caffe architecture. Thankfully, a Caffe merge completed on March 11th included an implementation of PReLU.

Apart from those techniques that appear in Convolutional Neural Networks course at Stanford [3] (Dropout, ReLU, model ensembles), the inspiration behind several of our techniques came from outside work. Our use of a parametric parameter unit (PReLu) is inspired by the rectifier network in [2], which achieved a top-5 error of 4.96%. The use of fractional pooling to mitigate the destructiveness of downsampling (75% of your dataset is lost with the standard 2:1 max-pooling) is suggested by [1].

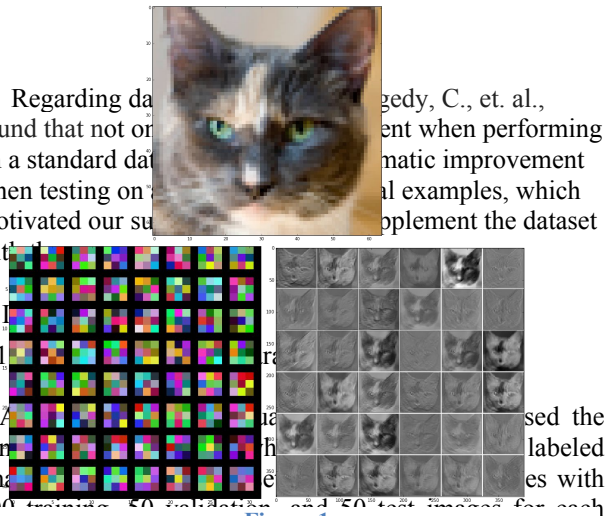

Figure 1

Top: Original image
Bottom Left: Visualization of filters from first ConvNet
Bottom Right: Images after application of kernels (filters)

We ultimately sought to minimize classification error on the testing set by training on the train set and performing hyperparameter validations on the validation set. We used Caffe running on a GPU to run and test the models. This required careful selection of several critical features and hyperparameters. The former list includes the network architecture, learning rate decay schedule, method of regularization, type of models to ensemble and their method of voting, and the weight update scheme. The latter includes the learning rate, regularization strength, dropout probability, filter size and number for each layer, batch size, number of epochs, and weight initialization.

3.2. Evaluation

Models	Best Accuracy
Learning Rate	1e-2
Dropout Probability	0.7
Filter Sizes	(5, 3, 3)
Filter Number	(32, 32, 64, 128)
Batch Size	150
Epoch Number	15
Weight Initialization	1e-2 / sqrt(fan_in)

successfully implement were limited in substantial part by our computing resources.

It is, moreover, less clear that our additional image preprocessing (above centering and scaling), such as adding the gradient image, would be effective, so we would need to compare test results on a small group with and without the feature.

In the process of obtaining final results, we made most use of the loss over and validation and training accuracies over time. The latter indicated to us early on that we needed stronger regularization. When we tested with regularization between 0.001, 0.001, and 0.0001 and

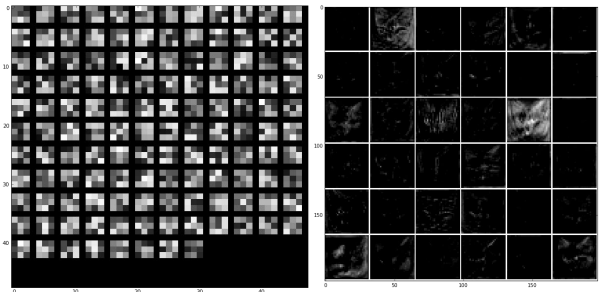

Figure 2

Left: Visualization of filters from third ConvNet
Right: Images after application of kernels (filters)

dropout probability 0.5, we saw trends as in Figure 3 part b. This indicated the need to increase regularization. We then tried a regularization of 0.05, 0.03, and 0.02. 0.02 performed the best, and its training/validation accuracies are shown in Figure 3 part d and f. The latter indicates to us the need to decrease our learning rate. That the loss so sharply decreases as in Figure 3 part e and then plateaus suggests that our results could be improved by an increase on the order of 10. Although we are confident in this hypothesis, we unfortunately did not have time to validate it.

One other measure we made some use of was the visualization of filters after the first convolutional layer. An example is shown in Figure 1, bottom left. Even after changing our regularization to produce the aforementioned closeness between the training and validation accuracies, the filters are scattered and produce no clear patterns. This suggests to us that our model could still benefit from further tuning of regularization; it is possible that the filters would benefit from greater run time, though this is unlikely given the plateaus in Figure 3, d and f.

4. Technical Details

Here, we provide a brief overview of the features we planned to implement, in the order we plan to address them.

4.1. Network Architecture

Our final CNN architecture consists of the following:

(Conv3-32 >> Conv3-32 >> ReLu-32 >> Pool-32) >>
(Conv3-64 >> Conv3-64 >> ReLu-64 >> Pool-64) >>
(Conv3-256 >> Conv3-256 >> Conv1-256 >> ReLu-256 >> Pool-256)
>> Affine-200 >> Dropout-200 >> Softmax-LogLoss

Our primary challenge when choosing CNN architecture was balancing the computational (and correspondingly, financial) cost with capacity and efficacy gains. Following [3] and [7], we therefore attempted to design a network that was as deep and robust as possible while working within our constraints. We initially tested a 9, 7, and 5

Figure 3
Hyperparameters

layer network, but quickly found that even after substantially scaling back the number of filters used at each layer, our Terminal GPU instance did not have the memory capacity to run networks of this size. We finally found that we were able to successfully run to completion with a three layer model. From there, we drew from [7] to use sequential convolutional layers with filters of size 3x3, and later in the network, an additional layer with filters of size 1x1. These chained small layers, as Simonyan suggests, serve to add nonlinearities to the model (e.g., increase its representational capacity) while maintaining a relatively low number of parameters. We then followed [4] in placing a dropout layer just before our fully connected layer, and [3] in using only one fully connected layer rather than the more traditional two, cognizant of the fact that we would be trading a small amount of accuracy for relatively large reductions in runtime).

4.

Classification Set	Classification Accuracy
Training Set (90,000 images)	30.5%
Validation Set (10,000 images)	25.5%
Test Set (10000 images)	N/A (See Discussion section below)

Ultimately, the only augmentation to our dataset was flipped images. The reason for this was two-fold. First, we found in preliminary testing that the addition of alternative augmentations did little (<1%) to improve our results. This result is confirmed by [3], which suggests that flipping contributes the most to generalization of learned features. Second, what little improvement we saw came at the cost of a substantial increase in runtime. We did not find the tradeoff to be worthwhile. These efforts are detailed below:

Gradient Image: We added a fourth dimension to our inputs, consisting of an approximation of the gradient of the image intensity function, as computed by the Sobel operator. Our hope was that it would allow the CNN to focus on salient features, under the assumption that regions of rapid gradient change generally contain particularly useful information. Despite the lack of literature on the subject, we were expecting the addition to make the model more robust - although a CNN can generate filters that approximate edge extraction, we hypothesized that having a fourth layer already containing that information might allow the net to devote fewer, but more accurate, filters to the task. Unfortunately, we found that this was not the case. Our validation accuracy was, in fact, slightly poorer (probably owing to variation in random initialization), which suggests to us that our CNN performed edge detection sufficiently well without the additional channel.

Mirror Image: We flip randomly selected images horizontally. This technique's impact on classification accuracy was strongly validated in [3], and so we did no comparison test to validate its efficacy ourselves.

Tinting and Contrast Modified Image: After implementing this technique, we saw no improvement over simply flipping the images that couldn't be attributed to variation in random initialization, and so chose to exclude it to spare the computation

Adversarial Examples: As training runs on the GPU, we were able to process adversarial images (imperceptible noise added to fool the network) and add them to our image set. This technique is validated in [6].

We tried other methods to improve robustness such as adding random croppings of the image (suggested in [3]), modulating its histogram, and warping the image, but we found improvement to be marginal.

4.3. Dropout

Fo
form
weig
p, an
the s
are d
prob
2^n
n is t

4.4. I

Th
impl
that t

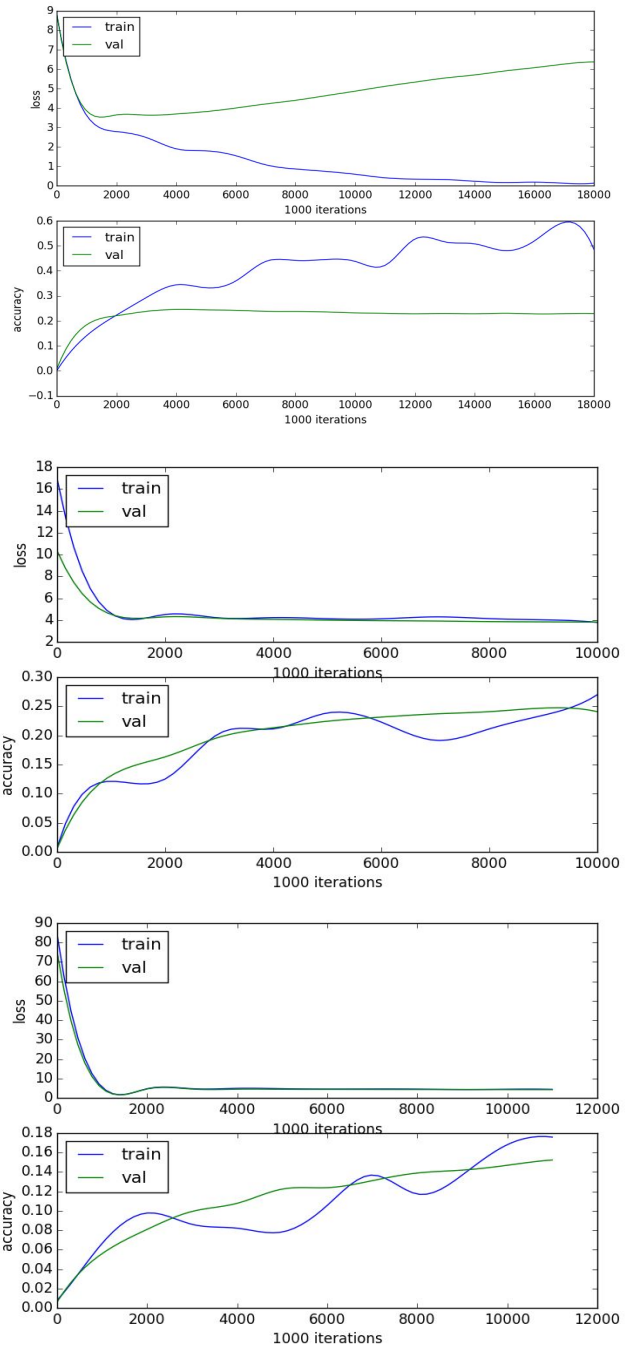

Figure 3
Top (a-b): ReLU, no Dropout
Center (c-d): ReLU, Dropout
Bottom (e-f): PReLU, Dropout

where a_i is determined on a per-channel basis at each layer by application of the chain rule during back-propagation followed by SGD + momentum, much like the weights and biases. In [2], it was demonstrated to have improved their results by nearly 1% over the previous state of the art, with very little additional computational cost or risk of overfitting because of the relatively small number of parameters it adds to the model (one/channel at each layer).

4.5. Fractional Pooling

It was recently demonstrated in [1] that max pooling too drastically reduces the amount of information available to be shared between layers that are separated by a pooling layer. In response, he formulated a fractional pooling method that reduces the amount of information in an image by some number between one and four. The method operates by applying max pooling to pseudorandomly chosen pooling regions in an image. These regions $P_{i,j}$ are determined by the equations:

where

for i = 0 … [output size]

These regions allow the network to learn some translation and elastic distortion without too drastically decreasing the amount of information shared between layers. Nonetheless, we expect to have to thoroughly tune our regularization parameters when we add in fractional pooling, as poor tuning has been reported to lead to easily lead to underfitting or overfitting.

4.6. Model Ensembles

[3] indicated that training multiple models to vote on the correct class for each image turning testing can substantially improve performance. We plan to ultimately train two models – one with a SVM linear classifier and one with a Softmax classifier – several times with different random weight initializations once our hyperparameters have been tuned.

We will then either average the output scores to determine the predicted class, or attempt to compare the models' confidence in their predictions and weight their vote appropriately. We are not yet certain that this latter voting procedure would be theoretically well-grounded, however, and so it may not appear in our final model.

5. Results

As shown in the table below, after a week of training, our best validation accuracy was 25.5% after 12,000 iterations (2.2 epochs). Our best training accuracy was a corresponding 30.5%. This model was not able to utilize adversarial examples or fractional pooling, as we suggest below.

6. Error Analysis

Our failure in ultimately implementing adversarial examples stems from the added computation making training within our given time frame infeasible. As demonstrated in assignment [3], the generation of adversarial examples with negligible visual deformations takes on the order of seconds in a naive implementation. Given the number of image we needed to operate on a naive implementation running on the gpu was unfeasible. It was suggested in [6] that the generation of adversarial examples occur on the cpu, and the output feed back into the pool of training examples that the CNN utilizes while training on the gpu. However, implementing this parallelization in caffe proved unfeasible for us as well.

Our failure in implementing fractional pooling stems from the added runtime as well.

Adding a fourth channel to our input data required a significant alteration of the caffe architecture that ultimately was unrealized.

The following two graphs compare models running with ReLU and PReLU. While the PReLU model demonstrates lower classification accuracy in both validation and training accuracies this is to be expected, as the variable negative slope in PReLU makes the model take longer to train, in the same way that dropout allows for a more robust model at the expense of a longer training period. In both graphs we can see that the learning rate parameter selected is too high, as there is little gap between training and validation. While a large gap between training and validation accuracies suggests too low a learning rate, a lack of gap suggests that the training accuracy is being compromised significantly, and the validation accuracy moderately.

7. Discussion

Compared to other Tiny Imagenet convolutional neural networks featured in this competition, we can see there's room for improvement.

Computational power was a major concern in training a convnet. There's little we can do to build a larger, more robust architecture if our machine cannot handle it or we cannot afford the cost of running Terminal for extended periods of time.

Another bottleneck was simply the amount of time we had to train. Both training time (number of epochs) and architecture size were constrained by the amount of time we had to build our network.

Additionally, fine-tuning was hindered by the time constraint. A suboptimal hyperparameter selection can compromise performance, however the time it takes in finding the optimal combination of just learning-rate, regularization, and number of filters at each layer, let alone other hyperparameters such as the weight initializations (Gaussian variance and bias) at each layer scales with the size of the architecture.

Finally, and perhaps most significantly, we were hindered substantially by Terminal.com. There were periods of time we had no access to our accounts and had to re-write already written code. We ultimately were not able to afford the expense, financially, that was required to run for long periods of time, and we unable to access the GPU instances nearer to the end of the project timeline for lack of available instances. In the future, we would prefer a more stable development platform.

For the reasons above, we were prevented from producing predictions for the testing set – Having left over 12 hours to compute the predictions, we were blocked from accessing the files need to compute a prediction until shortly before the deadline, which did not leave us time to run the prediction script to completion. We hope to be able to email it shortly.

References

[1] Graham, B. (n.d.). Fractional Max-Pooling. *ICLR 2015 - Under Review, ArXiv:1412.6071*, 1-8. Retrieved February 16, 2015, from ArXiv.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Microsoft, ArXiv:1502.01852*, 1-11. Retrieved February 9, 2015, from ArXiv.

[3] Karpathy, A. (Director) (2015, January 1). Introduction to Convolutional Neural Networks. *Course Lecture*. Lecture conducted from Stanford University, Stanford.

[4] Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems,* 1-9. Retrieved February 16, 2015, fromhttp://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] Lin, M., Chen, Q., & Yan, S. (2014). Network in Network. *ArXiv,* 1-10. Retrieved February 16, 2015, from ArXiv.

[6] Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut" *ACM Transactions on Graphics,*309-309. Retrieved February 16, 2015, from http://www.wisdom.weizmann.ac.il/~vision/courses/2006_2/papers/vid_seg/grabcut_siggraph04.pdf

[7] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR 2015,* 1-13. Retrieved February 16, 2015, from ArXiv.

[8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research, 15*, 1929-1958. Retrieved February 16, 2015, from http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[9] Jia, Y. and Shelhamer, E. and Donahue, J. and Karayev, S. and Long, J. and Girshick, R. and Guadarrama, S. and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *Jia2014caffe,* arXiv:1408.5093. Retrieved February 9, 2015, from ArXiv.