Tiny ImageNet Classification with Convolutional Neural Networks

Leon Yao, John Miller
Stanford University
{leonyao, millerjp}@stanford.edu

Abstract

We trained several deep convolutional neural networks to classify 10,000 images from the Tiny ImageNet dataset into 200 distinct classes. An ensemble of 3 convolutional networks achieves a test set error rate of 56.7%. The top single model achieves an error rate of 58.2%. It consists of 5 convolutional layers, 3 max-pooling layers, 3 fully-connected layers, and a softmax loss. The model is trained using dropout on the fully-connected layers and ℓ_2 weight decay to reduce overfitting. More refined analysis of the ensemble's classification errors yield insights into the strengths and weaknesses of the model architecture.

1. Introduction

A fundamental problem in machine learning is effective representation of complex inputs such as images or videos. Hand-engineered features have long been popular in computer vision, and machine learning systems based on these features have achieved excellent results on a variety of tasks. [2, 10, 13] Ultimately, however, these features can be brittle, difficult to design, and inadequately expressive for complicated, high-dimensional inputs.

Deep learning offers the promise of eliminating the dependence on hand-designed features and directly learning features from data. Deep architectures build up complex internal representations and fully learn a compositional mapping from inputs to outputs, from images to labels. While these methods have been known since the 1990's [9], recent improvements in computer hardware and optimization methods have led to a resurgence of interest in neural networks and many impressive results. Indeed, deep models have outperformed many hand-crafted feature representations and achieved state-of-the-art results on a significant number of computer vision benchmarks. [8, 14]

Motivated by the success of Krizhevsky *et al*. [8], Szegedy *et al*. [18], and several other groups in applying convolutional neural networks to image classification, and in particular, to the ILSVRC benchmark, [15] we apply deep, convolutional neural networks (CNNs) to the Tiny Imagenet Challenge. [19]

The 200 object classes that form the Tiny Imagenet Dataset are challenging and exhibit significant ambiguity and intra-class variation. To learn an image representation capable of accurately separating these classes, a deep, high capacity model is necessary. However, deeper, higher capacity models necessarily mean more parameters. This increase in parameters make over-fitting a significant concern, so we employ strong regularization techniques, *e.g*. dropout [6], to combat over-fitting and increase generalization.

With a model ensemble of the top 5 performing CNN's, we classify the 10,000 test images of the TinyImagenet dataset into 200 classes with an error rate of 56.7%. Examining the typical errors made by the model ensemble, we identify several broad classes of images and objects for which classification using CNN's is easy, as well as several classes where classification appears to be difficult. Focusing efforts on these problem classes suggests that significant reduction in the test error still is possible.

2. Data Set

Tiny Imagenet is a dataset of 120,000 labeled images belonging to 200 object categories. The categories are synsets of the WordNet hierarchy, and the images are similar in spirit to the ImageNet images used in the ILSVRC benchmark, but with lower resolution. [3, 15]

Each of the 200 categories consists of 500 training images, 50 validation images, and 50 test images, all downsampled to a fixed resolution of 64x64. The images are all pre-processed by subtracting the mean image from the entire Tiny Imagenet dataset from each image. No other pre-processing is performed, so the input to the CNN's are mean centered RGB pixel values for each images.

Several of the images are difficult to classify, even for humans. In Figure 1, example images for the class "popsicle" and "plunger", respectively, are given. In the Popsicle image, it is hard to even recognize the woman is even holding a Popsicle, and, in the plunger image, the object to classify is not clear, especially since the plunger is being used in an unorthodox fashion. These images highlight the challenges of building a robust model that achieves reasonable

Figure 1. 'Popsicle' and 'Plunger' Example Images

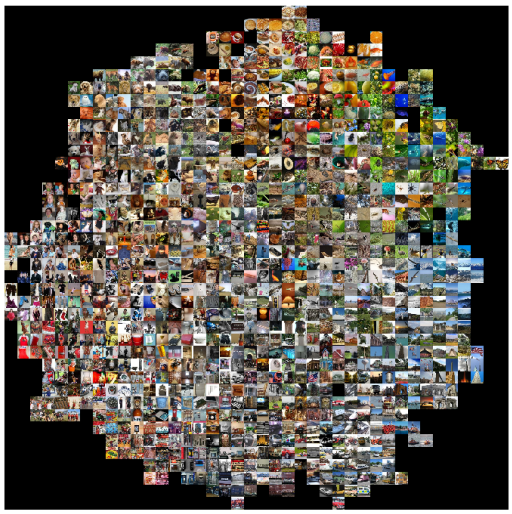

Figure 2. TSNE Embedding of Tiny Imagenet CNN Codes

accuracy on the Tiny Imagenet dataset.

Despite the obvious obstacles in classifying images such as those in Figure 1, the local geometry of the Tiny Imagenet image space appears to be favorable for image classification. To gain further intuition about this space, we visualized the images using t-distributed Stochastic Neighbor embedding (t-SNE). [11] Using the top-performing CNN as a feature extractor, we extracted 400-dimensional codes from the last layer, and computed a 2-d embedding of the codes. The result is visualized in Figure 2. The visualization shows that the space can roughly be separated into various semantic categories by a CNN. For example, there is a distinct section of the embedding consisting almost exclusively of automobiles and other consisting of people.

3. Selecting A High Performance Model

The space of potential model architectures for convolutional networks is quite large, and selecting a high performance model demands design decision along several different axes. In this section, we detail the most important considerations that produced our final model. Full details of the experiments and results mentioned below are included in the results section.

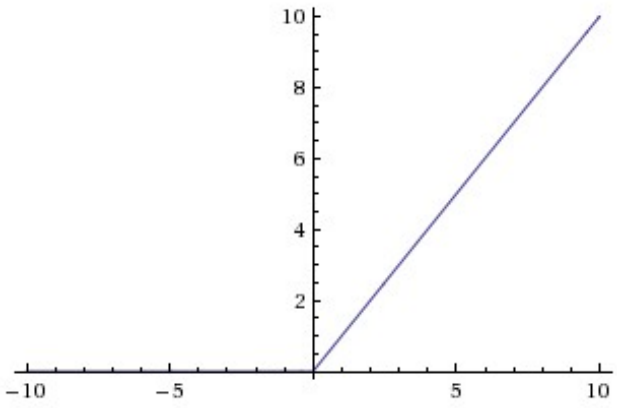

Figure 3. Non-Saturating Activation Function

3.1. Non-Linearity

Following the work of Hinton and Nair [12] and the empirical success of AlexNet and other models on ILSVRC, it has become standard to use rectified linear units, ReLU, as the non-linear activation function for CNN's. This is the approach that we take as well. For each neuron, the output f of its input is the non-saturating $f(x) = \max\{0, x\}$, graphically depicted in Figure 3.

As pointed out in [8], ReLU often improves the learning dynamics of the network, and this choice of activation function often significantly reduces the number of iterations required for convergence in deep networks. This is crucial for experimenting with larger models on a very limited computational budget.

3.2. Model Architecture

1. **Convolutional Layers**: Given the complexity of the image classification task and the small amount of available data, convolutional layers are critical to the success of the CNN's since they encode significant prior knowledge about image statistics and have far fewer parameters than other deep architectures. Experience of other researchers [8, 18] and empirical results suggest that deeper architectures are necessary to achieve high performance on this task.

   Our experimentation suggested that CNN's with 5 or more convolutional layers are needed to have sufficient model capacity. On the other hand, these layers require the largest amount of memory, so computational constraints limit the depth of the models.

2. **Pooling Layers**: Max-pooling (pool) layers are useful in reducing the number of parameters in the network by reducing the spatial size of the representation. Given the limited memory requirements of GPU's and the effectiveness of large numbers of filters, these layers are particularly important to ensure the connection

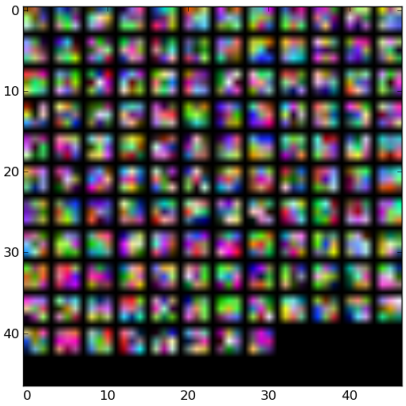

Figure 4. Trained First Convolutional Layer Weights

between the final convolutional layer and the first fully connected layer fits into memory.

We experimented with different numbers of pooling layers. For models with reasonable filter depth, having zero pooling layers exceeded hardware capacity. However, we found best performance was obtained using the minimum number of pooling layers that hardware supported, which was 3 for our model. This corresponds to the largest possible spatial representation, and significantly outperformed models with more pooling layers, *e.g*. those which pooled after each convolutional layer. To increase the amount of "nonlinearity" in the intermediate convolutional layers, the pool layers were placed after two intermediate convolutional layers.

3. **Filter Sizes**: The height and width of the convolutional filters play an important role in determining the capacity and power of the CNN representation. Stacking smaller convolutional filters on top of each other allows the CNN to develop more power representations of the input with fewer parameters. On the other hand, smaller filters requires more memory to compute the backpropagated gradients.

   We experimented with filters of size $k \times k$ for $k = 3, 5, 7$. Empirically, using several stacked, small 3×3 filters outperformed using filters of other sizes, even considering the constraints placed on model size due to memory consumption. For intuition about the learned filters of size 3×3, Figure 4 gives a visualization of the trained first layer filters from the best model.

4. **Number of Filters**: We performed experiments using filter banks of size $n = 32, 64, 128, 256$ and 512. Empirically, models with smaller filter banks lacked sufficient capacity and were unable to fully overfit the data. On the other hand, models with 256 or 512 filters were difficult to train and exhibited poor generalization. As a happy medium between the two extreme, filter banks with $n = 128$ exhibited good performance.

5. **Fully Connected Layers**: Fully-connected (fc) layers contain the largest number of parameters in the network. Therefore, more fully-connected layers directly corresponds to increased model capacity and representational power. However, as expected, increased capacity also leads to overfitting.

   We experimented with varying numbers of fully-connected layers, and empirically found that $3-4$ fully connected layers, combined with strong regularization to combat overfitting, produced the best results on the test set.

3.3. Final Model

We detail the architecture of the single top performing model, which achieved a test set error of 58% before being ensembled with other models to produce our final submission.

The network consists of 8 layers. The first five layers are convolutional (conv). These conv layers all have 128 filters of size 3×3 each. To ensure input height and width are preserved, padding of size 1 and a stride of 1 are used for the convolutional layers.

To reduce dimensionality of the representation and control the capacity of the model, the second, fourth, and fifth layers are followed by a max-pooling layer over 2×2 kernels with a stride of 1.

Finally, these convolutional and pooling layers are followed by 3 fully-connected layers. The first two layers both have 400 neurons, and the final layer has 200 neurons which are then fed into a softmax loss function.

Compactly, the final network architecture is

$$[[\text{conv}-\text{relu}]\times 2-\text{pool}]\times 2-[\text{conv-relu-pool}]-[\text{fc}]\times 3-\text{softmax}$$

4. Training

4.1. Initialization

Both the 3-layer and 5-layer model with 32 filters were randomly initialized using weights $W_{ij} \sim \mathcal{N}(0, 0.01)$. All of the biases were initialized to 0. This choice of initialization did not appear to have a significant impact on the learning dynamics for these smaller models.

However, Gaussian initialization was not successful for the 5 layer network with 128 filters and, more importantly, not successful for the deeper, 8-layer network. In particular, when the standard deviation was small, the loss function did not decrease. This is likely because the backpropagated error signal decayed too rapidly to be useful. On the other

hand, using a large standard deviation caused the loss to explode to infinity.

Problems initializing deep networks frequently arise in the literature, for example [1, 17]. To combat these issues, the method of normalized initialization, or "Xavier" initialization introduced in [1] is used. For each weight W_{ij} in the network,

$$W_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right],$$

where n_j is the fan-in of the neuron and n_{j+1} is the fan-out. As before, all of the biases are zero-initialized. This initialization allows training of the deeper models, which is important since our results will show that deeper models obtain higher test set accuracy.

4.2. Optimization

In addition to careful initialization choices, the results of [17] suggest that carefully tuned first-order momentum methods are important to training deep networks. Consequently, we experimented with both classical SGD with momentum and AdaGrad to train the deep 8-layer network. [4]

Classical SGD with momentum accelerates directions of low-curvature by building velocity in directions with a consistent gradient signal. This allows the learning process to avoid the problem of vanishing gradients and reach a better local optima. Given the objective $f(\theta)$, the update formula is

$$v_{t+1} = \mu v_t - \eta \nabla f(\theta_t) \quad (1)$$

$$\theta_{t+1} = \theta_t + v_{t+1} \quad (2)$$

The momentum parameter μ was set to the 0.9 during training.

For comparison, we also trained several deep networks with Adagrad. Adagrad is an adaptive, per-parameter gradient descent method that attempts to "find the needle in the haystack" to make effective parameter updates to sparse features. [4] Consequently, hyperparameter optimization is not as important since the algorithm is more influenced by historical gradients than the learning rate.The update formula is given by

$$G_{t_{i,i}} = \sum_{t'=1}^{t} \nabla f(\theta'_t)_i^2 \quad (3)$$

$$\theta_{t+1} = \theta_t - \eta G_t^{-1/2} \nabla f(\theta_t) \quad (4)$$

In both cases, the learning rate was initialized to $\eta = 1e-3$ and reduced by a factor of 10 every 10 epochs until convergence. For the final, top-performing model, after convergence, the learning rate was reduced to $5e-6$ and allowed to train for another 5 epochs with lower regularization. This fine-tuning procedure improved accuracy 1-2%.

4.3. Environment

Time and computational resources are by far the large biggest bottlenecks to successfully training CNN's. To speed up this process, all of our CNN models were implemented and tuned using Caffe, an open-source library for convolutional neural networks. [7] All of the training took place on a shared computer cluster using NVIDIA GTX 480 GPU's. This set-up achieved a speed-up of over 20x from the CS231n CNN implementation, allowing for faster development times.

Even with this set-up, computational resources proved to be a barrier to high performance. In particular, the limited memory of the GPU restricted model size and sharing the GPU with other users often restricted throughput and limited the number of possible training epochs.

It is likely that access to more powerful computing resources and a larger computational budget would immediately yield an decrease in test set error with larger models, more epochs of training and better cross validation.

5. Preventing Overfitting

Most image classification tasks contain significantly more images per class than the Tiny Imagenet challenge. For example, the CIFAR-10 data set contains more than 5,000 images for each of the ten classes compared to the 600 for each of the 200 Tiny Imagenet classes. The complexity of the various classes demands a large model capable of rich feature representation. However, the small number of training examples means that large models are prone to badly overfitting the data. In this section, we detail several techniques used to overcome these obstacles.

5.1. Data Augmentation

One of the easiest ways to prevent overfitting on an image dataset is data augmentation, whereby the dataset is artificially enlarged by using class-preserving transformations of the original images.

The first augmentation we used was taking horizontal translations and reflections of each image. We accomplished this by taking 10 random crops of size 56x56 from each 64x64 image and including it's horizontal reflection. We also changed the tint (adding a constant to each RGB channel) and the contrast (multiplying a constant to each pixel) of each image. Additionally, we added a random Gaussian noise with mean 0 and standard deviation 1 to each image.

We notice that all of these image augmentations should not change the class of the image, but changes the pixel values (and their relative relationships) dramatically and are essentially different images to the neural network. This allows us to artificially generate more data, although clearly these images are highly correlated with the originals. The

effect of these transformations is to increase generalization by making the model more robust.

For each image we add about 20 additional images to the dataset and expand the number of images we use to a total of 2 million.

5.2. Model Regularization

L2-regularization is commonly used in machine learning because it encourages smooth, diffuse weights and increases generalization of the model. In effect, regularization can be viewed as enforcing a Gaussian prior over the weights the model. The amount we regularize is determined by the regularization constant, λ. Larger values of λ correspond to stronger regularization, which makes overfitting more difficult, but limits model capacity. We initially used a value of $\lambda = 1 \cdot 10^{-3}$, but upon realizing we were regularizing too strongly, we also experimented with values of $\lambda = 1 \cdot 10^{-4}, 3 \cdot 10^{-4}$ and $\lambda = 5 \cdot 10^{-4}$.

5.3. Fine-Tuning

Given our large 8 layer convolutional neural network, with a data set of roughly 2 million images after augmentation, we determined it was not possible to cross validate the hyperparameters of our model while training from scratch. In order work around our limited computational capabilities, we trained 3 models from scratch using different learning rates and regularization constants for 40,000 iterations. We then looked at our loss and training/validation errors to determine new hyperparameter values. Each time we tried new values, we would continue training on a snapshot of the model that had already trained for 40,000 iterations. We believe that the early iterations only trained the basic features such as edges and colors and would successfully train under a wide range of hyperparameter values.

5.4. Model Ensembles

Model ensembling is a method of generating multiple fully trained CNNs and combining them all to obtain a aggregated prediction. Specifically, we run each test image through each of our predictive models and obtain its class probabilities and average them over each model. Model ensembles have been shown to reduce the testing errors of convolutional neural networks [8].

We did an ensemble over three models trained from scratch with different weight initializations. We can imagine that because these models are trained with different initializations and hyperparameters, the predictions it has at the end will be completely different from another model. One model might do well on certain classes while another model might do well on others. By combining these models, the ensembled model will do well on the classes each of the individuals models also do well on, and will generalize better than any of the individual models.

In order to improve our accuracy even further, we wanted to ensemble more models. However given time and computation constraints, it was not possible to train 5 separate models. Instead, we wondered if it was possible to ensemble various snapshots (10,000 iterations apart) from the same model as it trained. Our reasoning is that as a model trains, it will search through many different local optimum. Although each of these optimum may obtain the same validation error, the classes it succeeds in classifying correctly may be completely different. For each of our 3 models, we additionally ensembled three snapshots carefully chosen by looking through the loss history and training/validation errors to determine when the network was most likely in a different local optimum.

5.5. Dropout

Another widely used technique increasing regularization and preventing overfitting is Dropout [16]. Dropout sets the output of each neuron to zero with a certain probability, p. At test time we use all of the neurons but multiply the results by p. The majority of the parameters in the convolutional network are in the fully-connected layers, so in all of our models, dropout is ony applied to the fully-connected layers, and the early convolutional layers are not changed.

Dropout complements other regularization techniques such as L1, L2 and maxnorm. However the downside of dropout is that the network trains much more slowly, because it will roughly take $1/p$ more iterations to train the entire network. We experimented with different values of p for values $p = 0, 0.3, 0.5, 0.7$, but found that the default value of $p = 0.5$, as mentioned in [16] was the most effective.

6. Experiments

Name	Val	Test
Shallow Model	82.5	83.1
5 Layer CNN	77.6	77.8
8 Layer CNN	67.4	67.3
8 Layer CNN (Dropout, $\lambda = 1e-4$)	63.6	63.3
8 Layer CNN (Dropout, $\lambda = 3e-4$)	59.5	60.0
8 Layer CNN (Dropout, $\lambda = 5e-4$)	58.0	58.2
3 8-Layer CNNs Ensemble	56.9	56.7

6.1. Results

We see that the shallow model and the 5 layer CNN has significantly higher validation error, despite light parameter tuning and training over many epochs. This shows that the shallow models simply do not have enough parameters to fit the data. Furthermore, we did experimentation with the shallow networks on different filter sizes ranging from 3,5 and 7. We found that stacking 3x3 filters indeed does better than using a single larger filter. As a result, we have many

layers of 3x3 filters in our 8 layer model. We also experimented with the number of filters per layer. We generally found that the larger the number of filters we used, the better the model performed, because we increased the number of parameters being trained on. However, because of computation limitations, we chose to use 128 filters for our final model.

We also see that the 8 layer CNN without dropout does about 4 percent worse than the ones with dropout. For each of the other 8 layer models in the table above, we trained with a dropout probability of 0.5, a learning rate ranging from $1 \cdot 10^{-3}$ to $1 \cdot 10^{-4}$ and various regularization parameters, λ as seen in the table. Each model listed is without ensemble except for the last entry. We found that ensembling three snapshots of each model, dropped the test error by about 1 percent. The overall ensemble of all 9 models (3 models with 3 snapshots each) dropped our test error by an additional 1-2 percent.

We also notice that all of our models listed have near identical validation error and test error. This means that each of our models generalize extremely well.

7. Discussion

As we were training the 8 layer CNN models, we noticed that with a $\lambda = 1 \cdot 10^{-3}$ the training and validation errors were only differing by about 5 percent over 40,000 iterations. This means that our model was too heavily regularized so on top of the 40,000 snapshot, we trained an additional 20,000-40,000 iterations with $\lambda = 1 \cdot 10^{-4}, 3 \cdot 10^{-4}, 5 \cdot 10^{-4}$. However even with a lower regularization constant, the training accuracy was still only around 45 percent. We then decided to drop the regularization constant to $1e-8$ to see if it will overfit further. Using this model, we achieved a validation error of 45 percent without ensemble, but did not generalize well and had a test error of about 60 percent. Even with very little regularization our model did not overfit the data. This lead us to believe that our model was still not large enough and has significant room for improvement, if we did the same techniques on a 12 layer network. This optimistic assumption comes from the fact that even AlexNet is a much larger capacity model that achieved higher performance on a more challenging dataset.

However, the main problem with pursuing a larger model is our limited computational power. Our current model has over 400,000 parameters and took over 48 hours to fully train. A much larger model might take days to fully train and even longer to tune.

7.1. Error Analysis

A good way to tell whether our CNNs actually trained well, is by visualizing the weights of the first layer. From figure 4, we see that each 3x3 filter displays a clear image of various blobs of color. These blobs determine the type of features in our images that could help classify the classes. If our networks did not train well, ie had too much noise and was merely guessing, then the first layer visualizations would be random noise as well.

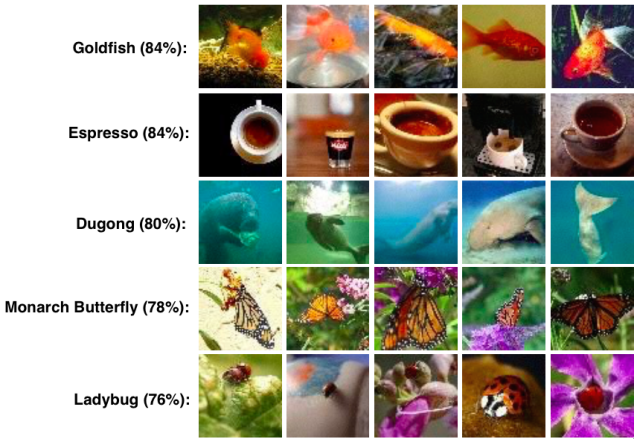

Figure 5. Top 5 Classification Accuracies

We could also see how our CNN is distinguishing between classes by looking at the t-SNE embedding in Figure 2. A shallow model would most likely only differentiate classes using colors, and have similarly colored classes next to each other. However we do not see that much of a color gradient in our t-SNE image. Instead, we see a lot of similar shapes and textures grouped together, which further proves our CNN actually trained well.

Another good way of analyzing our predictive models, is by identifying what types of image classes it does well on and what it does poorly on. From Figure 5 we see the top 5 classes our best model classifies and Figure 6 shows the top 5 classes it incorrectly classifies, and a few example images for each class. We notice that the top correct classes (goldfish, espresso, dugong, butterfly and ladybug) all have a lot of texture and similar telling colors between images of the same class. For instance all goldfish and monarch butterfly are orange and have distinctive scales or wing patterns. We also see that the orientation of these objects do not effect the CNNs ability to predict the class, meaning it learns more about the classes than just simple shapes.

However, for the classes we get wrong (Plunger, Pop bottle, Bucket, Reel, and Syringe), we notice that some of the images even a human cannot classify well. As a result, our CNN gets almost none of these correct. Some of the difficulties of these classes also stem from the fact that the color of the object (pop bottle and bucket) has almost no effect on the class. Also CNNs are known to perform poorly on clear transparent objects with no textures like the syringe and pop bottle. Our model also does extremely poorly on objects that are long and thin like the reel and the plunger.

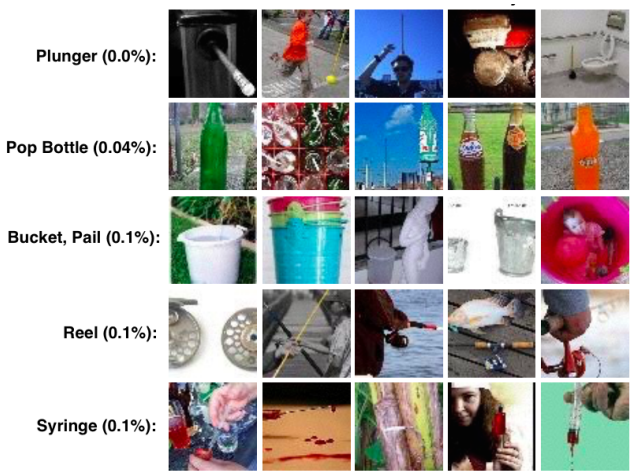

Figure 6. Bottom 5 Classification Accuracies

8. Conclusion and Future Work

The top performing ensemble of models achieved a test set error of 56.7%, which was sufficient for 4th place on the Tiny Imagenet leaderboard at the time of submission. The experimentation and results confirmed much of the common intuition about the performance of CNNs on image classification tasks. In particular, deep, high capacity models are required to handle to complexity inherent in natural images. However, given limited amounts of training data, strong regularization is necessary to prevents these models from severely over-fitting.

An analysis of the Tiny Imagenet dataset was also fruitful. As might be expected, many of the images were ambiguous and many of the classes exhibited high variation and noise. Further compounding these difficulties, the images are all initially size 64×64, which is low resolution for a difficult classification task. In this sense, it is somewhat remarkable that a CNN is able to achieve nearly 45% accuracy.

Given our limited time and resources for this project, there is significant room for improvement in terms of our test accuracy. Given more time, it is likely implementing additional techniques that are not currently native to Caffe would improve performance. For example, instead of just using a ReLu in our model, implementing a parametrized leaky ReLu that learns the best parameter α from our data. This was shown in [5] to achieve excellent results.

Additionally, upon viewing more of the images from the training set, we notice that a lot of the images even a human cannot identify correctly. This may be from heavy pixelation of the image or that the object being identified is simply not even in the picture. In either case, if a human cannot identify it, is it reasonable for a CNN to? Perhaps it would be better to remove these ambiguous images from the dataset. If our CNN only trains on standard easily identifiable images from a certain class, it may help our model at least identify the low hanging fruit for the classes that we obtain a near 0 percent accuracy. However, it is unclear how this affects generalization.

References

[1] Y. Bengio and X. Glorot. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, Proc. of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10), pages 249-256, 2010.

[2] Dalal, N.; Triggs, B., "Histograms of oriented gradients for human detection," Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on , vol.1, no., pp.886,893 vol. 1, 25-25 June 2005

[3] Jia Deng; Wei Dong; Socher, R.; Li-Jia Li; Kai Li; Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on , vol., no., pp.248,255, 20-25 June 2009

[4] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. JMLR, 12.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. arXiv:1502.01852, 2015

[6] 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.

[7] Y. Jia, Caffe: An open source convolutional architecture for fast feature embedding, http://caffe.berkeleyvision. org/, 2013.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in NIPS, 2012, pp. 11061114.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation, vol. 1, no. 4, pp. 541551, 1989.

[10] D. G. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91110, 2004.

[11] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

[12] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proc. 27th International Conference on Machine Learning, 2010.

[13] F. Perronnin, Z. Akata, Z. Harchaoui, and C. Schmid, Towards good practice in large-scale learning for image classification, in Proc. CVPR, 2012, pp. 34823489.

[14] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, CNN Features off-the-shelf: an Astounding Baseline for Recognition, CoRR, vol. abs/1403.6382, 2014.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575, 2014.

[16] Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from, Journal of Machine Learning Research, 2014 Overfitting,

[17] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, volume 28 of JMLR Proceedings, pages 11391147. JMLR.org, 2013.

[18] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. arXiv preprint arXiv:1409.4842, 2014.

[19] Tiny Imagenet Challenge [Online]. Available: https://tinyimagenet.herokuapp.com