Techniques for Image Classification on Tiny-ImageNet

Zach Barnes
Stanford University
Stanford, CA
zbarnes@stanford.edu

Frank Cipollone
Stanford University
Stanford, CA
fcipollo@stanford.edu

Tyler Romero
Stanford University
Stanford, CA
tromero1@stanford.edu

Abstract

The goal of the image recognition task is to be able to correctly predict the subject of an image. The ImageNet Large Scale Visual Recognition challenge [7] is run every year to determine the state of the art in image recognition. For this project, due to the restrictions on time and resources, we worked with a smaller dataset, Tiny-ImageNet [1], and attempted to train an image classifier using this data.

We applied a wide variety of techniques to achieve a high classification accuracy on Tiny-ImageNet. These techniques include residual architectures, data augmentation, cyclic learning rates, and snapshot ensembles.

1. Background

1.1 Introduction

Image recognition is a prominent and important task for which convolutional neural networks have proved very effective. Furthermore, the task has numerous important practical applications. Many industries are affected by the task of image classification, such as security, entertainment, and augmented reality. A wide variety of computer systems are aided by this technology, and improvements in the technology are vital for improvement in these systems.

To frame the problem more precisely, we describe our inputs and outputs:

- Our inputs for this task are images from the Tiny ImageNet database. Each image is 64x64x3.
- Our output for this task is the class label corresponding to the class we predict the image belongs in. If this is the class associated with the image, we have made a correct prediction.

1.2 Related Work

Convolutional Neural Networks represent the state of the art in image classification. AlexNet [2] is a Deep Convolutional Neural Network that was used to win the 2012 ImageNet competition. It was the first of many Deep CNN architectures to win the challenge, and set the stage for the explosion of research in the area. The architecture is relatively simple when compared to the more complex architectures representing the state of the art. AlexNet contains multiple convolutional and max pooling layers, followed by three dense layers.

Another famous architecture for image classification is VGGNet [9]. VGGNet is similar to AlexNet, with some slight improvements, including using random scaling for data augmentation.

GoogLeNet [10] is another ImageNet winner that improved on AlexNet and VGGNet by using so called inception modules. Inception modules use multiple filter sizes at each layer and concatenate the results together. This allows the network to learn features of different sizes at each layer in the network.

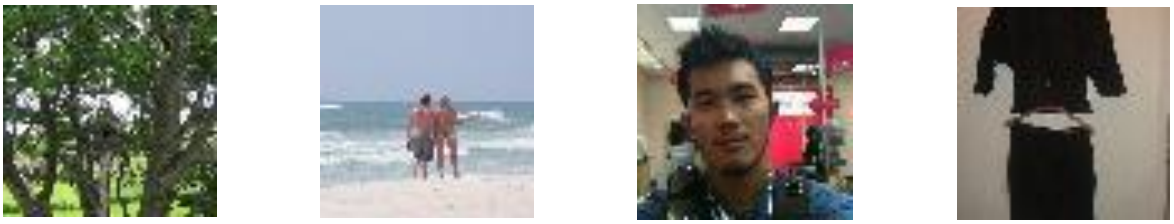
Recently, ResNets utilizing residual connections have become the most popular genre of model. Residual connections are special connections from the output of one layer of a network to the output of another layer further forward in the network. This allows gradients to propagate through much deeper networks more easily.

Cyclic learning rates (SGD with warm restarts) have been shown to help with model performance [11]. Collections of models, known as ensembles, are commonly used to make a joint prediction that is better than the predictions of any of the component models. This is because the different models in the ensemble converge to different local minima, and have different classification boundaries. Using many models together helps to lower the variance in predictions, resulting in a higher accuracy.

2. Approach

2.1 Dataset

We use the standard Tiny-ImageNet dataset. This dataset consists of 100,000 training images, 10,000 validation images, and 10,000 testing images that are all (64,64,3) and fall into 200 classes. These images are taken from the ImageNet dataset, cropped to be square, and resized to be 64x64. Due to this process, the images tend to be very pixelated, and occasionally missing key features. Examples of difficult to classify images from the training set are given in Figure 1.



birdhouse bikini sunglasses skirt

Figure 1

2.2 Data Augmentation

Heavy data augmentation was necessary in order to prevent our model from strongly overfitting to the dataset. We preprocessed the data by performing mean subtraction. When preparing each batch of images for training, we would horizontally flip each image with probability 0.5, and we would take a random crop of each image of size 56x56. These methods were suggested in [2].

When making a prediction on an image, we take ten crops (also of size 56x56) of that image (the four corners, the center, and their horizontal flips) and make a prediction on each one, and then average the SoftMax probabilities to make a final prediction. The colored section of figure 2 represents one of the original five crops. As we will show, these changes significantly reduced overfitting.

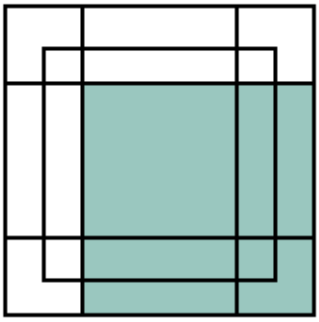


Figure 2

2.3 Models

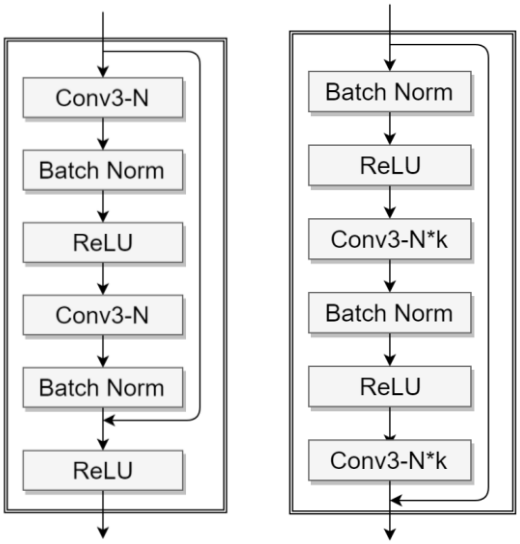
All our models take a 56x56x3 input image and produce a predicted class label by taking the argmax across a SoftMax probability distribution.

We first implement our baseline to get an understanding of the problem. Our baseline, our AlexNet model, uses an architecture identical to the original AlexNet, except that the filter sizes and strides are adjusted to be more appropriate for the smaller input images from Tiny-ImageNet. The exact architecture is available in the appendix.

We also have experimented with an inception network, based off the GoogleNet [10], to try and achieve a performance improvement over our baseline. We eventually abandoned InceptionNets for ResNets, which proved easier to train.

ResNets [3] were our primary model class. We experimented with various depths and residual block architectures to maximize our model accuracy. The residual block that we used in our experiments is depicted in Figure 2, and the entire ResNet architectures used are in the appendix.

A simple extension of ResNets are WideResNets [4], which have an improved residual block structure, and have more filters per layer. WideResNets are generally shallower than the ultra-deep ResNets that have been successful on the ImageNet challenge, but WideResNets have proven extremely successful on competitions related to Tiny-ImageNet, such as CIFAR-100. The wide residual block that we used is depicted in Figure 3.



Residual Block Wide Residual Block

Figure 3

2.4 Regularization

We found that our models continued to overfit, even when using data augmentation. Due to this, we experimented with L2 regularization as in [2] and dropout [15] as in [4]. In addition, we attempted to provide implicit regularization by shifting to thinner, shallower models with fewer parameters.

2.5 Snapshot Ensembles

One of the core features of our training framework was the ability to create snapshot ensembles [5]. Shapshot ensembling refers to a framework that allows for ensembles of models to be created in the time it would typically take to train a single model. Snapshot ensembles accomplish this using a cyclic learning rate that periodically dislodges the model from its local minima. A snapshot of the model parameters is taken immediately before they are dislodged. At the end of each cycle, the model converges to a different local minimum, giving a different decision boundary. The cycles are much shorter than a typical training session, but by ensembling the snapshots, a higher validation accuracy can be reached relative to a single model trained in the same amount of time.

We used the following formula from [5] for learning rate as a function of step number, where α is the initial learning rate, T is the total number of steps to train on, and M is the total number of cycles desired.

$$lr(t) = \frac{\alpha}{2}\left(\cos\left(\pi * \frac{(t-1) mod \left(\frac{T}{M}\right)}{\frac{T}{M}}\right) + 1\right)$$

Figure 3 is a plot of the learning rate formula. The shifted cosine formula allows for the model to dislodge itself with a high learning rate, and then quickly lower the learning rate so that the model can settle to a different local minimum. Snapshots are taken at the bottom of each cycle.

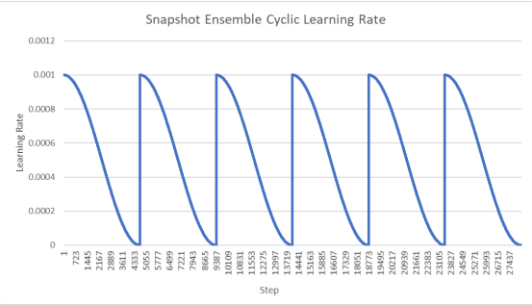


Figure 3

Snapshot ensembling was especially valuable due to our computational constraints.

3. Experiments and Results

3.1 Experimentation Details

We used the Adam Optimizer [8] with an initial learning rate of 0.001 to train our models. All nonlinearities are ReLUs [2]. Where L2 weight decay was used, we used a coefficient of 0.0001. In addition, all our models were given a training budget of 72 epochs. Our image preprocessing and augmentation consisted of mean subtraction, random crops, and random flips.

Our snapshot ensembles were trained such that there was a cycle every 12 epochs, for a total of 6 cycles (and therefore, 6 snapshots). We would disregard the snapshot generated by the first cycle because typically it had not yet reached an acceptable validation accuracy.

Our models trained using a standard procedure had the learning rate cut by 10 every 24 epochs. This would give the loss time to plateau before each decay. We used early stopping to help prevent overfitting.

3.2 Evaluation Metrics

Top-1 and top-5 validation and testing accuracies were used to evaluate our models. Top-1 accuracy refers to the accuracy a model achieves when it predicts only the class that it believes is most likely. Top-5 accuracy, however, refers to the accuracy a model achieves when a model predicts the five classes that it believes the image contains. If any of these five classes are correct, then that image is considered correctly classified. Top-5 validation accuracy leads to higher scores, but it also makes sense in the context of image classification due to the extreme similarity between certain classes, such as Siberian huskies and Eskimo dogs [10].

3.3 Results

Table 1 summarizes the results our models achieved. All values are accuracy rates. ‘a’ indicates that data augmentation was used, ‘d’ indicates that weight decay was used, and ‘snap’ indicates that snapshot ensembling was used.

Model	Top-5 Val	Top-1 Val	Top-1 Test
AlexNet	0.487	0.254	-
AlexNet + a	0.717	0.487	-
ResNet34 + a	0.734	0.523	-
WideResNet28-10 + a + d	0.773	0.564	-
WideResNet32-4 + a + d	0.778	0.571	-
WideResNet32-4 + a + d + snap	0.803	0.595	-
ResNet18 + a + d	0.795	0.589	-
ResNet18 + a + d + snap	0.814	0.602	0.536

Table 1: Results

3.4 Discussion

Adding data augmentation to AlexNet made an enormous difference in validation accuracy. Without data augmentation, AlexNet easily overfits the training data. We found this to be true for every model architecture we tried. Data augmentation is an extremely useful technique for the Tiny-ImageNet competition.

Switching to the ResNet34 architecture gave us a boost in performance over AlexNet. This could be because ResNet18 model was much deeper, or because it has less parameters and is easier to train.

Our models all tend to overfit in the second half of training, so we thought to add additional regularization in the form of L2 weight decay. Weight decay successfully helped reduce some overfitting, resulting in higher validation accuracies.

The use of dropout hurt performance in our ResNet models. We believe this could be due to the fact that our models have small sizes, which is not conducive to the use of dropout. We ended up not using dropout in any of our final models.

WideResNets offered some improvement over ResNet34, which could be attributed to the better design of the residual block.

Our models continued to overfit, even with all the regularization techniques we incorporated. To further regularize, we decided to try a smaller, shallower model: ResNet18. The smaller capacity of ResNet18 helps to combat overfitting. ResNet18 was our best individual model.

The addition of snapshot ensembling and cyclic learning rates gave us a significant improvement in validation accuracy. We found that using cyclic learning rates, even without snapshot ensembling, gave us validation accuracies that were higher than their counterparts trained using our standard method.

With snapshot ensembling, we saw jumps in top-1 validation accuracies of about 0.02. Due to the time constraints on class projects, we found snapshot ensembling to be a very efficient way to gain some of the benefits of ensembling without having to spend the time and resources required to train several individual models.

The biggest challenge on Tiny-ImageNet is overfitting. Even with our data augmentation and regularization schemes, our models could achieve a testing accuracy of 99% after 72 epochs. Additional regularization techniques and better hyperparameter tuning may help future models achieve a higher validation accuracy on Tiny-ImageNet.

4. Conclusion

4.1 Future Work

We have several ideas for improving validation accuracy on Tiny-ImageNet that we have not yet tested.

We believe that transferring a model that was trained on ImageNet to the Tiny-ImageNet task would be an effective way to achieve a higher validation accuracy. This is because a primary limitation of models trained on Tiny-ImageNet is the relative lack of data available, and the ImageNet model would be pre-trained on more data that Tiny-ImageNet contains, and therefore may have learned better features.

Full ensembling results in larger improvements to validation accuracy than snapshot ensembling according to [5], so using full ensembling may result in a higher top-1 validation accuracy for our current models.

Additional data regularization may result in some improvement. There are standard techniques such as scale and color augmentation that may prove effective.

4.2 Acknowledgements

We like the general framework for Tensorflow models supplied by [12], and we use a modified version of this framework as we implemented our models. We borrowed data loading functions for Tiny ImageNet and CIFAR-10 from [13]. Small sections of code, such as batch creation, were reused from a previous project of ours [14] that is unrelated to image classification.

Our models were implemented in Tensorflow [6].

4.3 Source Code

Our code can be found at:
https://github.com/fcipollone/TinyImageNet

5. References

[1] "Tiny ImageNet Visual Recognition Challenge." Tiny ImageNet Visual Recognition Challenge. N.p., n.d. Web. 05 June 2017.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[4] Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." arXiv preprint arXiv:1605.07146 (2016).

[5] Huang, Gao, et al. "Snapshot ensembles: Train 1, get m for free." arXiv preprint arXiv:1704.00109 (2017).

[6] Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* (2016).

[7] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." *International Journal of Computer Vision* 115.3 (2015): 211-252.

[8] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[9] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[10] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
[11] Loshchilov, Ilya, and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts." (2016).
[12] "Assignment #4: Reading Comprehension." CS224n: Natural Language Processing with Deep Learning. N.p., n.d. Web. 12 June 2017. <http://web.stanford.edu/class/cs224n/assignment4/index.html>.
[13] CS231n Convolutional Neural Networks for Visual Recognition. N.p., n.d. Web. 12 June 2017. <http://cs231n.github.io/assignments2017/assignment3/>.
[14] Cipollone, Frank, Tyler Romero, and Zach Barnes. "Squad-reading-comprehension." N.p., n.d. Web. <https://github.com/zabarnes/squad-reading-comprehension>.
[15] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

6. Appendices

6.1 SimpleAlexNet

Conv7, stride=2, filters=48
Conv5, stride=1, filters=48
MaxPool2, stride=2
Batch Normalization
Conv5, stride=1, filters=128
MaxPool2, stride=2
Batch Normalization
Conv3, stride=1, filters=192
Conv3, stride=1, filters=192
Conv3, stride=1, filters=128
MaxPool2, stride=2
Affine, 2048 hidden units, relu activation
Affine, 2048 hidden units, relu activation
Affine, 200 output units, no activation

Appd 1: SimpleAlexNet Architecture

6.2 ResNet34 Architecture

Conv3, stride=1, filters=64
Batch Normalization
ReLU
ResLayer, stride=1, filters=64
ResLayer, stride=1, filters=64
ResLayer, stride=1, filters=64
ResLayer, stride=1, filters=64
ResLayer, stride=2, filters=128
ResLayer, stride=1, filters=128
ResLayer, stride=1, filters=128
ResLayer, stride=1, filters=128
ResLayer, stride=2, filters=256
ResLayer, stride=1, filters=256
ResLayer, stride=1, filters=256
ResLayer, stride=1, filters=256
ResLayer, stride=2, filters=512
ResLayer, stride=1, filters=512
ResLayer, stride=1, filters=512
ResLayer, stride=1, filters=512
AvgPool of entire output
Affine, 200 output units, no activation

Appd 2: ResNet34 Architecture

6.3 ResNet18 Architecture

Conv3, stride=1, filters=64
Batch Normalization, ReLU
ResLayer, stride=1, filters=64
ResLayer, stride=1, filters=64
ResLayer, stride=2, filters=128
ResLayer, stride=1, filters=128
ResLayer, stride=2, filters=256
ResLayer, stride=1, filters=256
ResLayer, stride=2, filters=512
ResLayer, stride=1, filters=512
AvgPool of entire output
Affine, 200 output units, no activation

Appd 3: ResNet18 Architecture

6.4 WideResNet32-k Architecture

Conv3, stride=1, filters=64
WideResLayer, stride=1, filters=16*k
WideResLayer, stride=1, filters=16*k
WideResLayer, stride=1, filters=16*k
WideResLayer, stride=1, filters=16*k
WideResLayer, stride=2, filters=32*k
WideResLayer, stride=1, filters=32*k
WideResLayer, stride=1, filters=32*k
WideResLayer, stride=1, filters=32*k
WideResLayer, stride=2, filters=64*k
WideResLayer, stride=1, filters=64*k
WideResLayer, stride=1, filters=64*k
WideResLayer, stride=1, filters=64*k
AvgPool of entire output
Affine, 200 output units, no activation

Appd 4: WideResNet32-k Architecture