Going Deeper on the Tiny Imagenet Challenge

Andrew Zhai
andrewz@stanford.edu

Abstract

In this work, we investigate how fine-tuning from pre-trained models and network depth affect the classification performance on the Tiny ImageNet challenge. Starting with state of the art academic classification models such as VGG16 and ResNet-[50,101,152], we will explore how to adapt and fine-tune these networks trained on a different dataset with different input image sizes to our Tiny ImageNet classification dataset. We then train ResNet models of various depth and filter sizes from scratch to allow us to evaluate the effects of model depth on classification performance. From these experiments, we report insights into why fine-tuning works and the effects of network depth.

1. Introduction

Convolutional neural networks (convnets) have become the standard approach to classification tasks within the last few years. This transition from traditional hand labeled features is due to the overwhelming performance of convnets on classification challenges such as the ImageNet Large Scale Visual Recognition Challenge [9] with the best performing model achieving a top-5 error of 3.08% on the test set [10].

Over the years, there has been a trend where the deeper the model is, the better performance the model can get on the ImageNet challenge. In 2012, the AlexNet architecture with 8 layers resulted in a top-5 classification error of 16.4% on the ImageNet challenge [7]. In 2014, the VGG19 model with 19 layers resulted in a top-5 classification error of 7.3% [1]. In 2015, the GoogleNet model with 22 layers resulted in a top-5 classification error of 6.7% [11]. And finally in 2015, the ResNet model with 152 layers resulted in a top-5 classification error of 3.57% [3]. Using this as motivation we seek to investigate how deep models perform on the Tiny ImageNet Classification Challenge. In our case, we will be optimizing for top-1 performance on the Tiny ImageNet dataset, a dataset derived from the standard ImageNet dataset. This dataset is split into train, validate, and test components. The training set contains 200 classes with 500 images in each class for a total of 100,000 training images. Both the validation and testing datasets contain 50 images per class for a total of 10,000 images each. We will describe this dataset along with our preprocessing in more detail in Section 2.

We start off our investigation by describing our method of fine-tuning on classification models, such as VGG16 [1] and ResNet [3], of various depths known to perform well on the standard ImageNet challenge for the Tiny Imagenet Challenge in Section 3. Since our dataset is derived from the ImageNet challenge, a model tuned for the standard challenge should also perform well here. We will address fine-tuning challenges such as how to convert the standard ImageNet models trained on 224x224 images to our dataset which consists of 64x64 images.

In Section 4, we explore training ResNet models and variants from scratch for the Tiny ImageNet dataset. This allows us to be more flexible in the number of layers of our model to really investigate the affects of model depth on performance. Here we explore how the scale of the Tiny ImageNet dataset, with its much fewer images and classes than the standard ImageNet dataset, affects training for the ResNet architecture variants.

In Section 5, we describe our best performing models and dive deep into why finetuning works best for the Tiny ImageNet dataset along with a qualitative analysis into how we can improve our model in the future. Finally we summarize our learnings in Section 6.

2. Dataset

The Tiny ImageNet dataset is derived from the standard ImageNet challenge such that images are cropped into a size of 64x64. There are 200 classes in this dataset, a subset of which we visualized in Figure 1. We can see from the figure that the dataset consists of both coarse and fine-grained classes with coarse classes such as "comic book" and "lampshade" to fine-grained classes such as "German shepherd" and "standard poodle". Although smaller images contain less information, we have seen that high accuracy classification can still be done as shown through the CIFAR-10 dataset [6] where more than 90% accuracy classification can be achieved on 32x32 images. One concern with the Tiny ImageNet dataset however is that an object

is sometimes heavily occluded as some objects are cropped out such as the 4th lampshade example in the figure. This along other issues such as the fine-grained classes make this challenge difficult.

2.1. Processing

For the various models we train, we preprocess the Tiny ImageNet dataset by subtracting the channel wise mean as done in [1] for the VGG model and the pixel wise mean provided in the source code for [3]. We augment our dataset taking random 57x57 crops of the input images along with mirroring the images.

3. Fine-tuning

As we've seen from [2] [8], convnet features trained from large scale datasets such as the ImageNet dataset [9] can be used as generic image descriptors that outperforms traditional hand crafted features. We see that using such generic features as a baseline, we can bootstrap the learning process on new datasets by fine-tuning on existing classification models trained on the ImageNet such as VGG16 [1] and ResNet [3]. Because we have seen progressive improvements from VGG16 to ResNet in terms of classification performance on ImageNet, we predict that this trend in performance will continue to hold after fine-tuning on our new dataset, especially as our dataset was derived from ImageNet. We will investigate whether this is true or not.

3.1. VGG16

The standard VGG16 architecture as shown in Figure 2 is structured starting with a series of convolution + ReLU layers with max pooling layers occasionally to reduce the spatial dimension. Fully connected layers are then used before the final softmax layer [1]. The convolution layers use 3x3 kernels with a stride of 1 and padding of 1 to ensure that each activation map retains the same spatial dimensions as the previous layer. The max pooling layer uses 2x2 kernels with a stride of 2 and no padding to ensure that each spatial dimension of the activation map from the previous layer is halved.

We have described our fine-tune procedure in Figure 2. One problem with using the standard VGG16 architecture is that the input to the first fully connected layer depends on the spatial dimension of the activation maps from the previous max pool layer. This spatial dimension depends on the input image size. Because the standard VGG model requires 224x224 input images while we want the input to the network to be 57x57 crops of the 64x64 images in the Tiny ImageNet dataset, there is a mismatch with the number of weights required by the fully connected layers. As such we require retraining the fully connected layers, one layer with 4096 hidden units and one with 200 units to reflect the number of classes for the softmax layer.

Although we can't reuse the fully connected layers, we can reuse all the weights in the convolution layers because of their independence with the input image size as shown in the middle image of the figure. This is still desirable as these convolution layers usually learn low level descriptors for images such as edges and patterns [12]. We can imagine that similar descriptors would be learned for the Tiny ImageNet dataset and by reusing these weights, we can speed up training. We dive more into this in the Section 5.

When training, we add a cross entropy loss function on top of the softmax layer to get the following per example loss function:

$$L_i = -\log \frac{e^{s_{y_i}}}{\sum_{j=1}^{200} e^{s_j}}$$

The softmax function, quantity inside the log, measures the probability of the given correct class y_i from the scores vector s given by the previous fully connected layer. The $-\log$ then turns the function into one where we want to minimize, as per all loss functions. As such by minimizing the loss function, we are trying to increase the probability of the correct class. We average this per example loss for the batch example loss function.

We trained our model using Caffe [5] with the VGG16 model in the Model Zoo. For the new fully connected layers, we initialized the weights using MSRA fillers [4] and biases to zero. The first fully connected layer is trained with a 0.5 dropout ratio. Our fine-tuning was done in two phases. In the first phase, we set the learning rate of the convolution layers to be zero to only learn the fully connected weights. This is because the convolution weights have already been well learned and so we concentrate on learning representations for the fully connected layers to "catch up" to the convolution weights. We started training with a learning rate of 0.1 for 6,000 iterations with a step size of 2,000 iterations. Afterwards, for the second phase, we allow all layers to be trained so that the entire model is specialize for our current dataset. We start with a learning rate of 0.001 and train for 12,000 iterations with a step size of 4,000. For both phases, we train using momentum based SGD with momentum of 0.9 with a batch size of 256 and weight decay of 0.0001. We achieved a top-1 validation accuracy of 51.8% as shown in Figure 5. Note that the step sizes were chosen by dynamically visualizing the training loss and noticing when the training loss had plateaued.

We achieved a effective batch size of 256 through using the accumulated gradients technique where the effective batch size = actual batch size * itersize. This technique is implemented such that for our forward and backward pass, we run the computation for the actual batch size; the solver however does not update the gradients immediately and instead continues the forward and backward pass itersize times, accumulating the gradients for the parame-

Figure 1. Example images for six classes in the Tiny ImageNet dataset. The dataset contains 64x64 images of 200 classes. These categories are both coarse (poodle vs lampshade) and fine-grained (German shepherd vs poodle).

ters each time. Finally after the effective batch size number of images have been passed through the network, the solver updates the gradients. We then call this accumulated update of the effective batch size number of images an iteration.

3.2. ResNet

The main contribution of [3] when introducing the ResNet architecture was the idea of using residual connections as a manner of making it easier to train deeper networks that perform better. We see in Figure 3 that the basic building block of the ResNet architectures is designed to learn residual functions $F(x)$ where the residual function is related to the standard function learned $H(x)$ by $H(x) = F(x) + x$. One motivation for introducing these residual functions is that the authors believed that the ideal $H(x)$ learned by any model is closer to the identity function x than random and as such, instead of having a network learn $H(x)$ from randomly initialized weights, we save training time by instead learning $F(x)$, the difference of $H(x)$ and x. Additionally by introducing these identity functions x, we allow easier training of deeper networks by allowing the gradients to pass unaltered through these skip connections to alleviate traditional problems in training deep networks such as vanishing gradients.

The actual form of the $F(x)$ function learned with the input x is of the form 1x1 conv-bn-relu N $\rightarrow$ 3x3 conv-bn-relu N $\rightarrow$ 1x1 conv-bn-relu 4N where N is the number of filters learned and conv-bn-relu is convolution $\rightarrow$ batch normalization $\rightarrow$ ReLU. Padding and stride are chosen so the spatial dimension remains constant.

One variant of this architecture is shown in Figure 4 for the ResNet-152 architecture. Here we start off with the standard beginning of all ResNet variants, a 7x7 conv-bn-relu with stride 2 and padding 3 followed by a max pooling layer using a 3x3 kernel and stride 2. We then have 3 replicas of the building block with N = 64, then we have 8 replicas of the building block with N = 128, then we have 36 replicas of the building block with N = 256, then we have 3 replicas of the building block with N = 512, and finally we average pool spatially and have a fully connected layer followed by a softmax layer. Note that for every change in N in the building blocks, the first 1x1 convolution of the block uses a stride of 2 to lower the spatial dimension by half. To ensure dimension matching, the residual connection x is modified by a 1x1 convolution with stride 2 to match the reduced dimension of the learned $F(x)$. Here the variant is called ResNet-152 because $1+(3+8+36+3)*3+1 = 152$.

We fine-tune on the ResNet-50, ResNet-101, and ResNet-152 variants as they are the only models released for Caffe. (Unfortunately the shallower networks such as ResNet-18 and ResNet-34 were not released for Caffe. We attempted to convert the Facebook trained Torch models for ResNet-18 and ResNet-34 to Caffe with no success). Similar to VGG16, we cannot reuse the FC layer of the standard network and instead learn a new FC layer with hidden size 200. We adopt the same two phase fine-tuning method as before, training only new FC weights with a starting learning rate of 0.1 for 200 iterations with a step size of 100. We then train end to end with a learning rate of 0.01, step size of 5000 for a total of 17500 iterations. For both phases, we train using momentum based SGD with momentum of 0.9 with a weight decay of 0.001 and a batch size of 128 for the ResNet-50 variation and of 32 for the ResNet-[101,152] variations. We use accumulated gradients so that each parameter update is done after seeing a total of 256 images. Though this is incorrect because batch normalization breaks accumulated gradients since forward calculations are batch dependent, it was a compromise done due to not being able to fit the model in memory. We summarize the top-1 validation accuracy of all our fine-tuning in Figure 5 with our best performing model, ResNet-101, achieving a top-1 validation accuracy of 59.69% and top-1 test accuracy of 53.2%.

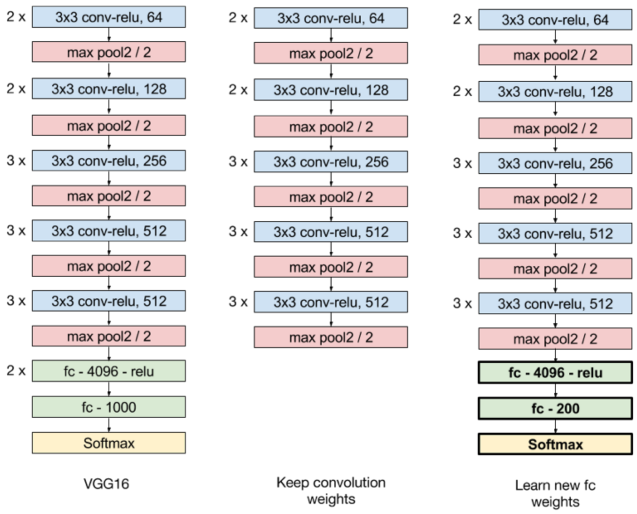

Figure 2. We start out with the standard VGG16 model on the left, re-use the convolution weights as shown in the middle, and learn new FC weights as shown on the right.

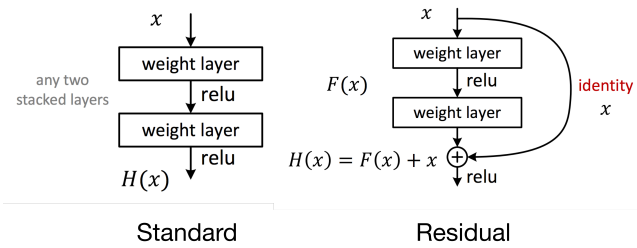

Figure 3. Here we see how residual connections differ from the standard connections. In particular for residual connections, we are learning a residual mapping $F(x)$ instead of $H(x)$.

4. Train From Scratch

One problem with fine-tuning is that our architecture is limited by what pre-trained models we find. Because pre-trained models were tuned for other datasets, they may not be the most optimal architecture for our specific task. As such here we investigate training models from scratch. We stick to the overall ResNet architecture due to the belief the general architecture improvements against predecessors such as VGG16 should hold even for new datasets. We however investigate how depth and less filters affect training and performance.

4.1. Original

Here we training the original ResNet-[50,101,152] architectures from scratch. In addition to the ResNet-[50,101,152] architectures, we introduce the ResNet-20 and ResNet-38 architectures. The ResNet-20 architecture is shown in Figure 4 and differs from the ResNet-[50,101,152] architectures by having less replicas and less spatial pool-

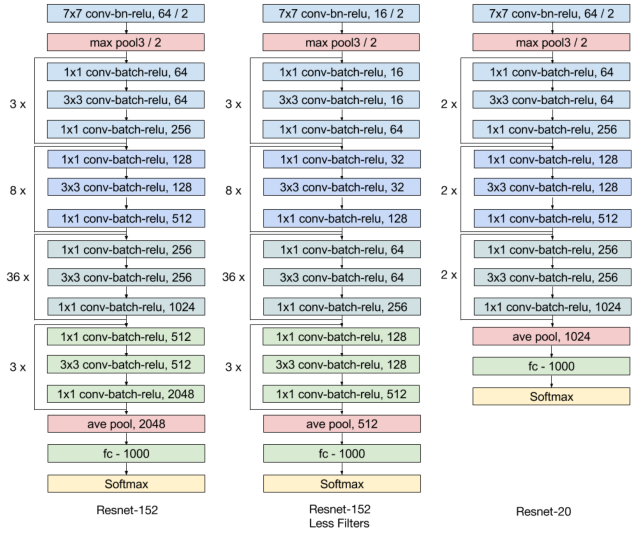

Figure 4. We show various variants of the ResNet architecture we trained. On the left is the standard ResNet-152, in the middle is the same architecture as ResNet-152 but with filter size divided by 4 throughout the network, and on the right is the ResNet-20 architecture involving less replicas of the buildings blocks and less spatial poolings. Note in the diagrams, we did not show the spatial pooling. This pooling occurs during the first conv-batch-relu for every basic block filter size change (from 64 to 128, from 128 to 256, and from 256 to 512).

Model	Validation Top-1 Accuracy / [Train loss]		
VGG16	51.8%	N/A	N/A
Resnet-20	N/A	45.59% / [0.167]	40.13% / [1.933]
Resnet-38	N/A	44.03% / [0.087]	38.28% / [1.675]
Resnet-50	59.11%	36.56% / [0.029]	32.75% / [0.919]
Resnet-101	59.69%	N/A	31.84% / [0.486]
Resnet-151	55.04%	N/A	29.77% / [0.392]
	Finetune	Original	Less Filters

Figure 5.

ings. The ResNet-38 architecture is similar to the ResNet-20 architecture but with 3x, 5x, 4x replicas instead of the 2x, 2x, 2x replicas in ResNet-20. Note that we chose not to base our architecture on the ResNet-18,34 variants introduced in [3]. This is because we did not want to confound our results with differences in how the residual function $F(x)$ is modeled.

Although we attempted to train ResNet-[101,152] from scratch, we did not finish the training due to computational constraints. The models did not seem like they were converging within a reasonable amount of time and so we did not finish training these models. For ResNet-[20,38], we trained the models using momentum based SGD with momentum of 0.9 starting with a learning rate of 0.1, batch size of 256, weight decay of 0.0001, and step size of 5000 for a total of 22500 iterations. For ResNet-50, we trained with

momentum of 0.9, batch size of 128, base learning rate of 0.1, weight decay of 0.001, and step size of 10000 for a total of 45000 iterations. The step sizes were chosen by dynamically visualizing the training loss and noticing when the training loss has plateaued. The weight decay parameter was chosen initially through hyper-parameter tuning on a fold of the training set through a grid search in log scale.

We show the results in Figure 5. Interestingly enough, deeper did not perform better in our case unlike the study done in [3]. We note however that we see trends of overfitting from Figure 5 in that our training loss is decreasing as we increase depth but our model's validation accuracy is also decreasing. Even the worse training loss achieved, a loss of 0.167 for the ResNet-20 variant, has a training top-1 accuracy of 98.86%, much better than the validation top-1 accuracy of 45.59%

4.2. Less Filters

To further investigate the effects of overfitting, we tried simplifying our model by reducing the number of filters in the ResNet variants by 4x, resulting in 16 being the lowest number of filters per any convolution as shown in the middle of Figure 4. Note that before changing the architecture, we tried other overfitting prevention mechanisms such as using stronger weight decay, smaller batch sizes for batch normalization to be more noisy, data augmentation techniques such as random distortions of the aspect ratio of the input image and random crops. None of these mechanisms resulted in better performance while the signs of overfitting still existed or the model simply did not converge for extreme weight decay and batch size values.

With less filters, we had the computation power to train the ResNet-[101,152] variants. As such for our study on how depth affects performance, we trained ResNet-[20,38,50,101,152] variants from scratch. Each model was trained with momentum based SGD using a base learning rate of 0.1, step size of 5000 for a total of 17500 iterations, weight decay of 0.0001, and momentum of 0.9. Through using accumulated gradients, we ensured that for each model, gradients are accumulated for 256 images. In other words, our effective batch size was 256 though the in-memory batch size varied due to memory constraints.

We show our top-1 validation accuracy results and final training loss in Figure 5 and plot the training loss and top-1 validation accuracy versus iterations in Figure 6. As we can see from the plot, by the end of training the training loss and the top-1 validation accuracy has plateaued for our various models. We can see that as we increase the depth of our model, both the training and validation loss decreases. This shows that as we increase the depth of our model, the model has more of a tendency to overfit with our training data as it achieves a better training loss but worse validation accuracy. This is interesting in that though somewhat expected that a more expressive model will overfit the data more, we would expect some signs that increased depth would increase performance as mentioned in [3]. This is especially true as the most shallow network, ResNet-20, had a final training top-1 accuracy of 57.52%, showing no extreme signs of overfitting to the training data. Overall reducing the number of filters resulted in worse performance compared to the original models trained from scratch, most likely due to the decrease in expressibility of the less filter variants.

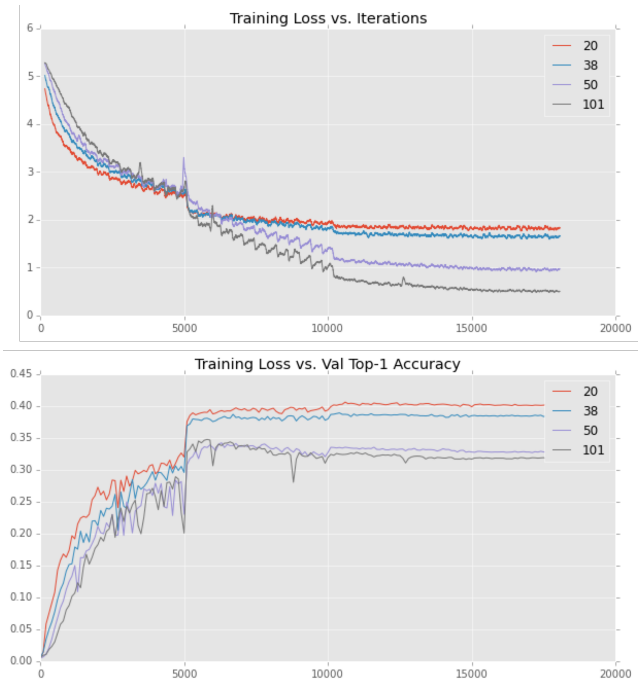

Figure 6. Visualization of the Training Loss and the Top-1 Validation Accuracy vs Iterations for the Less Filters variants. Here we see that increase in model depth results in lower training loss. Unfortunately, this trend of lower training loss also results in a trend of lower validation accuracy, showing that we are overfitting to the dataset more as we increase depth. Note: we did not plot the curves for the ResNet-152 model due to loss of the training logs.

5. Error Analysis

Our final best performing model was an majority vote ensemble of our finetuned ResNet-[50,101,152] models at the 3 latest snapshots each for a total of 9 models in the ensemble. (We snapshotted every 1000 iterations). For a given image, we would get the top-1 prediction of each of the 9 classification models and pick the majority agreed class. In case of no agreement, we took the prediction of the last snapshotted ResNet-101 model, our best performing individual model on the validation accuracy. This ensemble achieved a final validation top-1 accuracy of **61.53%** and a final test top-1 accuracy of **55.40%**.

5.1. Why Finetuning Performed Best

Here we investigate why we could not beat our finetuned models. It is well established that the first convolution in a convnet usually learns gabor like edge filters and color indicators [12]. When visualizing the learned conv1 filters in Figure 7, we can see when training from scratch, that the conv1 filters for ResNet-50 do not seem to converge well. Starting from random noise at Iteration 0, the conv1 filters for ResNet-50 become more defined at iteration 8000, showing some signs of gabor like filters and color indicators. However as we approach the end of training at iteration 200000, we see that still, these filters have not become very defined even though top-1 training accuracy has reached 97.43%. In contrast, for our fine-tuned ResNet-50 model, when viewing the conv1 filters at iteration 7200, we can clearly see well defined gabor like filters and color indicators.

One can make an argument however that perhaps our Tiny ImageNet dataset is different in that the optimal conv1 filters learned are not suppose to be gabor like or color indicators. We disprove this notion by also visualizing the conv1 filters for our shallower ResNet-20 model trained from scratch at iteration 15000. In this case, the filters for our ResNet-20 model look as expected, similar to the finetune filters, showing that the optimal conv1 filters for our dataset is still most likely gabor/color filters. This shows that our ResNet-50 model was able to do well on the training set without ever needing to learn generalizable representations, strengthening the belief that the ResNet-50 model trained from scratch was overfitting the training dataset. As for why our ResNet-20 trained from scratch performed worse than the fine-tuned models even though the conv1 filters look decent, we can see that the conv1 filters for the ResNet-20 model still look noisy, showing that perhaps we need to train the model longer. Note however that the ResNet-20 model is our best performing model trained from scratch.

More in general, we can see finetuning is powerful because we start training with representations that are already close to what we want as an output from training. The trained ResNet-20 conv1 filters look very similar to the ones in our finetuned ResNet-50's conv1 filters and in fact, our trained ResNet-20 filters look more noisy. As such it is not surprising that our finetuned models perform better as they were initially trained from a larger dataset, allowing general purpose representations to be learned.

When looking to improve our fine-tuned models, we believe that simply fine-tuning from shallower networks such as the ResNet-18 or ResNet-34 models released by Facebook for Torch could improve our accuracy. This is because when looking at the final training top-1 accuracy of our ResNet-50 finetuned model, we can see that the model is overfitting as we achieve an accuracy of 95.97%, much higher than our validation top-1 accuracy of 59.11%. Unfortunately by the time we came to this conclusion, we did not have enough time to either move our entire stack to torch, or convert the torch model weights into Caffe.

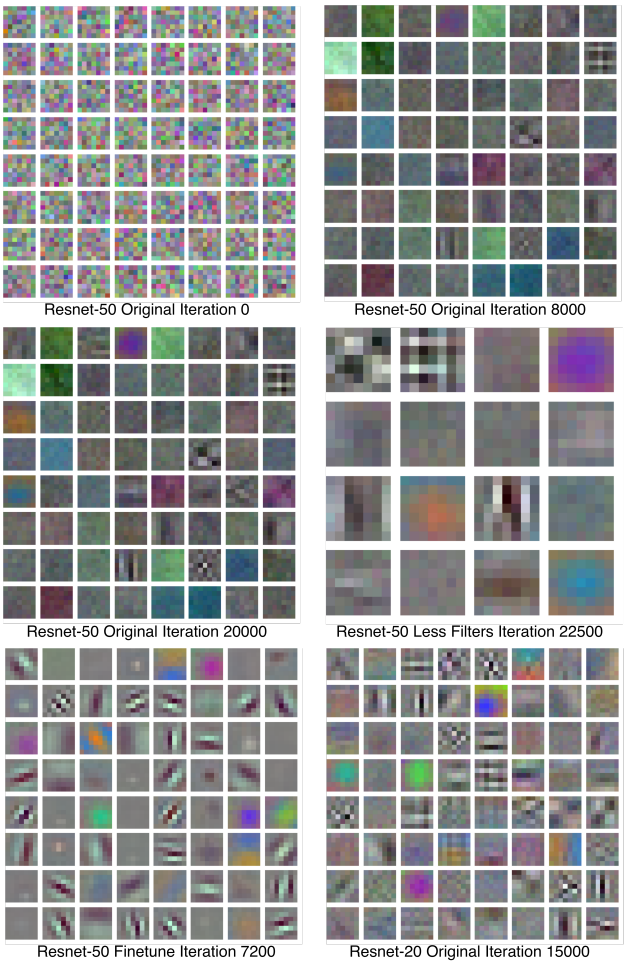

Figure 7. Visualization of the conv1 filters for the three Resnet-50 variants and the Resnet-20 model.

5.2. What Errors are Made

We visualize the top-1 errors made by our ensemble in Figure 8 with the true label on top and the predicated label on bottom. As we can see, in general the model's errors are due to fine-grained classes, misunderstanding the intent of the image, or simply getting confused with visually similar images.

We see examples of fine-grained classes errors with the scorpion versus centipede example. Because of the low resolution of the input image, the most distinct part of the image is the shelled body of the animal. In this case, both scorpions and centipedes have similar shelled bodies. Another example is the pig vs bison example. Both are similar in that they are four legged animals with brown fur. To understand

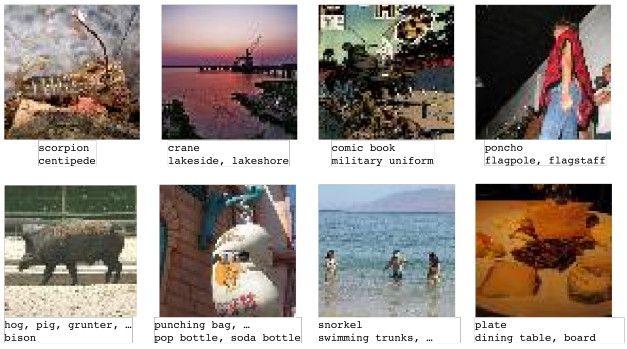

Figure 8. Images that our best performing model makes mistakes on. Each image is marked with the true class on top and the predicted class on the bottom.

the distinction between the two, our model needed to emphasize the shape of the animal as a whole more as bisons have a slightly different shape than a pig. One potential fix for this in our modeling may be to remove the initial 7x7 conv and maxpool with stride 2 each as this aggressively reduces the spatial dimension of our input. Starting from an input of size 57x57, after the maxpool layer, we already have reduced our spatial dimension by 4 as the spatial size becomes 14x14. This may remove the spatial information necessary to tell the difference in shape between a pig and a bison.

We can see misunderstandings of the intent of the image most clearly with the crane vs lakeside example. In this case, the crane is by a lakeside so it is not incorrect that our model produced lakeside as the top classification, especially because the lake is the majority of the image. The problem however is that in this image, our intent was to label the crane. Similarly for the snorkel vs swimming trunks example, it is perfectly plausible that the people in the image are going snorkeling. Instead however, our intent when labeling the image was the swimming trunks that they were wearing. This type of error is harder to fix as the classifier outputs are perfectly reasonable. Perhaps to improve upon this, we may want to learn an attention model for what the focus of an image is. This way, the model would better guess the intent of the image when multiple class predictions are reasonable.

Finally we see examples of mis-classifications due to visually similar images. For example in the poncho vs flagpole image, the way the person is standing and the loose shape of the poncho can make the person seem like a flagpole with the poncho as a red flag. Similarly with the punching bag vs pop bottle example, the punching bag happens to be in a bottle shape. For these examples, better modeling of coarse-grained classes can fix these errors. For example, see that the poncho image has a person in it which should lower the probability that the image is a flagpole.

6. Conclusion

We approached the Tiny ImageNet Challenge by looking towards the top performing academic classification models. By finetuning VGG16 and ResNet-[50,101,152], we were able to achieve our highest performing single model, the finetuned ResNet-101. In order to improve upon the finetuned models however, we attempted to learn representations from scratch with the hope that adjusting the scale of the ResNet architecture for the smaller scaled Tiny ImageNet dataset through changing the depth and filter sizes would improve performance. However, when implementing these solutions, we noticed strong signs of overfitting and a general lack of performance compared to the fine-tuned alternatives. By analyzing why finetuning was so much better, we saw that in general, finetuning starts with representations that are already near what we want to learn from scratch. Furthermore, by being pre-trained on a larger dataset, our finetuned models are able to learn more generic representations. Even for our fine-tuned models however, we do notice signs of overfitting in the shallowest model, ResNet-50, and as such, given more time we would immediately look into using Facebook's ResNet-[18,34] models to finetune.

References

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *ICML*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Arxiv*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, 2015.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[6] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[7] A. Krizhevsky, S. Ilya, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105. 2012.

[8] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CVPR*, 2014.

[9] 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.

[10] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, 2016.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[12] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *ECCV*, 2014.