Tiny ImageNet Challenge

Anna Shcherbina
Stanford University
annashch@stanford.edu

Abstract

An ensemble of three convolutional network architectures was used to classify 10,000 images from the Tiny ImageNet challenge into 200 distinct classes. A test error rate of 0.506 was achieved. The top single model in the ensemble achieves an error rate of 0.524. The top-performing architecture was a keras implementation of the VGG-16 architecture. To avoid overfitting to the training data, the data was augmented through image rotation, cropping, and color jitter. Additionally, dropout and regularization (L1 and L2) were utilized. Network performance was further analyzed by visualization of the filters, computation of saliency maps, and determining the per-class distribution of the model validation accuracy.

1. Introduction

Image classification is a fundamental problem in computer vision. The ImageNet Challenge[14] tasks participants with classifying 100,000 test images into 1000 classes, giving a training set of 1.2 million images. Top-1 and top-5 accuracy on the test dataset is used to rank performance. The Tiny ImageNet Challenge follows the same principle, though on a smaller scale – the images are smaller in dimension (64x64 pixels, as opposed to 256x256 pixels in standard ImageNet) and the dataset sizes are less overwhelming (100,000 training images across 200 classes; 10,000 test images).

Since the ImageNet Challenge was first held in 2010, a deep learning revolution has occurred in computer vision. Initial winners of the challenge relied on standard techniques in computer vision. In this approach feature such as SIFT [12], histogram of gradients [6], and local binary patterns [13] must be extracted from the training data and pooled for a global image representation using techniques such as spatial pyramid matching[11] and Fisher vector representation [7]. The pooled features are then input to a classifier such as a support vector machine to perform the image classification. These approaches work well on the scale of thousands or tens of thousands of images but do not scale efficiently for the 1.2 million images in ImageNet.

Convolutional neural networks help to overcome the limitations of traditional classification techniques. They do not require pre-defined input features, but rather learn them as part of the training process. Furthermore, the convolution and dot product operations of ConvNets can be highly vectorized, which allows for highly parallel and fast GPU implementation. The high suitability of convolutional neural networks for image classification is illustrated by the success of Krizhevsky et al [9], Szegedy et al [18] in applying these techniques to win the ImageNet Challenge.

In this project, I experiment with four convolutional neural network architectures to classify the images in the Tiny ImageNet Challenge. Pretrained weights are used when available; otherwise networks are trained from scratch. I focus on adding dropout, regularization, and data augmentation to prevent models from overfitting on the training data. Several approaches are implemented to visualize model performance and to identify classes where the model achieves high accuracy, as well as classes where the model performs poorly.

2. Methods

All models for this project were implemented in Python using the Keras library (v1.3)[5] running on top of Theano (v.0.7)[4]. All code and network weights for the project can be downloaded from Github: https://github.com/annashcherbina/cs231n_project.git.

The image data was pre-processed by calculating the per-channel mean of the training data and subtracting this value from each image in the training, validation, and test data. The dataset pixel values were subsequently normalized via division by the per-channel standard deviation of the training dataset. The skimage Python library was used for data loading and pre-processing[19].

2.1. Data augmentation

The training dataset provides 50 images for each of 200 classes, for a total of 100,000 images. Data augmentation was used to increase the amount of available training data. Five augmentation steps were performed (Figure 1). Images

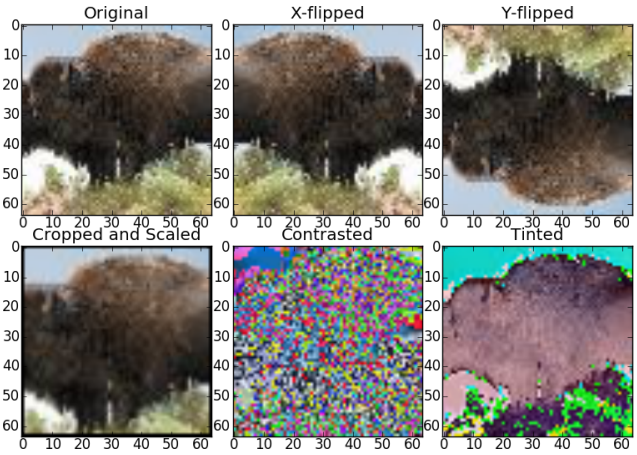

Figure 1. Data augmentation approaches that were applied to the training data.Images were flipped horizontally, flipped vertically, cropped, color-contrasted, and tinted.

were flipped about the vertical axis or about the horizontal axis. Random crops of 55x55 pixels were extracted from images, and the resulting smaller image was then scaled (via spline interpolation) to the original size of 64x64 pixels. The scaling was done with the resize function in the skimage library. Image contrast was adjusted randomly. For each input image in the training data, a number from the range [0.8,1.2] was selected at random, and each pixel of the image was multiplied by that number. Images were tinted at random. For each input image, a random color was generated whose red, green, and blue components were drawn uniformly at random from the range (-5,5). This color was added to each pixel of the image. Augmenting each image with all five techniques would have increased the training data size by 500%, which is too large to fit in memory for my available system. Consequently, 30% of the training data was selected at random for each of the augmentations, yielding a total training set size of 265,000 images.

2.2. Network architectures

Four ConvNet architectures were trained on the Tiny ImageNet dataset. These are illustrated in Figure 2.

2.2.1 Nine-layer network from CS231n assignment 3

The nine-layer network from assignment three was selected due to the availability of pretrained weights (Figure 2a). All convolution layer parameters were identical to values used in assignment 3. For the spatial batch normalization, the default Keras parameters were utilized (ϵ=1e-6,momentum=0.9). To reduce overfitting, the network was modified by adding 0.25 dropout after the third convolution layer, 0.25 dropout after the sixth convolution layer, and 0.5 dropout after each of the fully-connected layers[17]. L1 and L2 regularization was also added as a further step to reduce overfitting. The first convolutional layer was weakly regularized (L1 λ=1e-7, L2 λ= 1e-7). Stronger regularization was applied to each of the dense layers: L1 λ=1e-5, L2 λ=1e-5.

The network was trained via stochastic gradient descent for a total of 17 epochs. It was found that training for more than 17 epochs caused the validation accuracy to decrease (Figure 4), potentially a consequence of the network overshooting the global minimum. Learning rates of 1e-7, 1e-5, 1e-3, 1e-1, 5e-1 were utilized, and it was found that the network converged to the highest final accuracy with a learning rate of 1e-1. The default SGD parameters in Keras were tried (decay=1e-6, Nesterov momentum=0.9), but the network did not converge when these were applied. Removing decay and Nesterov momentum led to the highest validation accuracy and the fastest convergence time.

2.2.2 VGG-like network

The network illustrated in Figure 2b was provided as an example in the Keras documentation, and I used it for two purposes: to ensure that my Keras setup was working properly, and to learn the quality of results that could be obtained by training a network from scratch, as opposed to beginning with pre-trained weights. This was done because no pretrained weights were available for this network, which is a simplification of the VGG-16 network. In contrast to VGG-16, the VGG-like network utilizes only 4 convoluation layers. I modified the original network in [5] by adding 0.25 dropout after the second and fourth convolution layers, as well as 0.5 dropout after the first dense layer. To further reduce dropout, L1 and L2 regularization was added to the dense layers (L1 and L2 λ=1e-6 for the first dense layer; L1 and L2 λ=1e-5 for the second dense layer). The network was trained via stochastic gradient descent with learning rate 1e-1 for a total of 30 epochs (until validation accuracy plateaued).

2.2.3 VGG-16

The VGG-16 architecture described in [16] was implemented in Keras (Figure 2c.) Pre-trained weights for the architecture were downloaded from [1]. Because the original architecture was trained on the ImageNet dataset with 1000 output classes, the pre-trained weights for the Dense layers could not be used due to dimension mismatch. Consequently, the weights for the dense layers were initialized using the Glorot initialization [8]. The original VGG-16 architecture was modified by increasing dropout to 0.75 after the first and second fully-connected layers (as compared to 0.5 in the original architecture). This was done to mitigate the overfitting problem. L1 and L2 regularization was also added to the original VGG16 architecture to further reduce

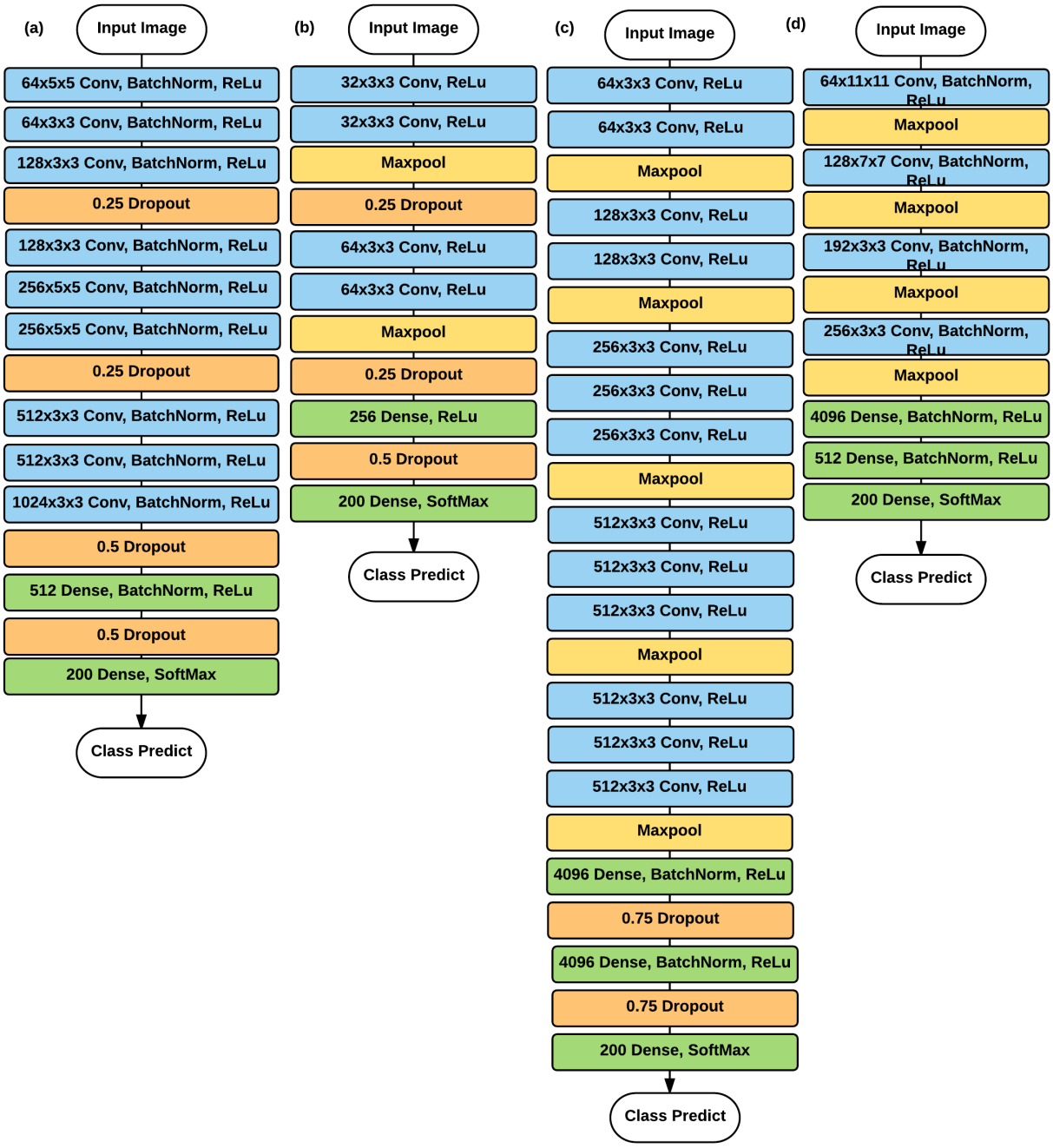

Figure 2. Four network architectures were trained as part of this project. a. Nine-layer ConvNet from Assignment 3. b. Simplified VGG-like network. c. VGG-16. d. AlexNet.

overfitting. Both weight regularization and activity regularization were added, and regularization parameters increased deeper into the network, however experiments showed that increased the regularization λ beyond 1e-4 prevented the network from learning, while using λ lower than 1e-7 was insufficient to have any meaningful effect on overfitting, leading to the following final set of regularization parameters:

- Conv Layer 11: L1 λ=1e-7, L2 λ=1e-7

- Conv Layer 12: L1 λ=1e-6, L2 λ=1e-6
- Conv Layer 13: L1 λ=1e-5, L2 λ=1e-5
- Dense Layer 1: L1 λ=1e-4, L2 λ=1e-5
- Dense Layer 2: L1 λ=1e-4, L2 λ=1e-4

The network was trained via stochastic gradient descent for 6 epochs, since the validation accuracy plateaued at this time, and training for additional epochs had no effect. Experimentation with learning rates of (1e-7, 1e-5, 1e-3, 1e-1) indicated optimal validation accuracy and faster convergence for a learning rate of 1e-3. For the VGG-16 model, the default Keras SGD parameters (decaye=1e-6, Nesterov momentum=0.9) did work better than SGD without these optimizations (in contrast to the 9-layer ConvNet described above).

2.2.4 AlexNet

I attempted to implement the AlexNet architecture[9] in Keras. The implementation is illustrated in Figure 2d. However, I was not able to evaluate this architecture because the memory required to run it was greater than that available on my computer system. This is due to the fact that AlexNet is a much wider architecture than VGG and the 9-layer ConvNet,though it is not as deep and utilizes only 4 convolutional layers. Experiments to replace the first convolutional layer of size (64x11x11) with smaller layers (64x9x9) or (64x7x7) continued to yield memory errors, preventing me from being able to evaluate the performance of this architecture.

2.2.5 Ensemble of architecture results

A majority vote of the three trained models was taken. If all three models disagreed, the VGG-16 class label was assigned, since that is the model with the highest validation accuracy. Pearson correlation was calculated to determine the degree of correlation in validation error rates across the models.

2.3. Optimization and Experimentation

2.3.1 Individual pre-training of layers

As an attempt to improve accuracy, layers in the 9-layer ConvNet were pretrained individually. Initially, the learning rate for all layers except the first convolutional layer was set to 0. Once that layer had been trained, the learning rate for the second convolutional layer was set to the default value (1e-1). The learning rate for the first layer was kept at 1e-1 to allow this layer to continue learning in the context of the full network as additional layers were added.

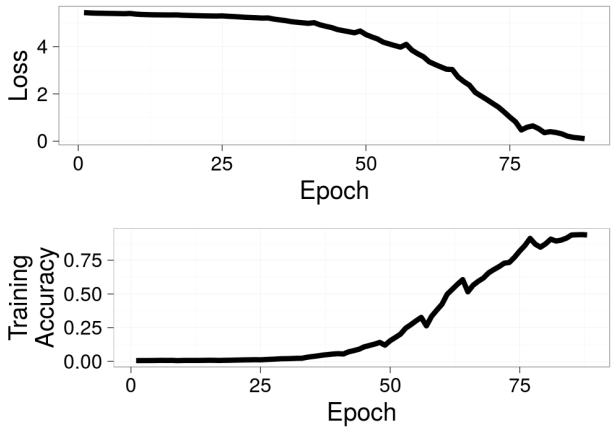

Figure 3. Accuracy and loss for the 9-layer ConvNet architecture with convolutional layers pre-trained individually.

In this manner, all remaining convolutional and dense layers were pre-trained and stacked on top of the previous layers. This approach follows one of the methods described by Larochelle et al [10]. The training accuracy and loss for this approach are illustrated in Figure 3. An accuracy of 98% is achieved on the training data, but the approach also suffers from sever overfitting, as illustrated by the 7.6% accuracy on the test dataset and 3.80% accuracy on the validation dataset (Table 1). Although these metrics were calculated prior to the addition of dropout and regularization, the performance was poor enough such that this approach was not utilized in the final set of models.

	Loss	Accuracy
Training	0.135	0.983
Validation	7.629	0.038
Test		0.0760

Table 1. Training, validation, and test accuracy of the 9-layer ConvNet architecture with convolutional layers pre-trained individually.

3. Results

3.1. Tiny ImageNet Classification Error

An error rate of 0.506 was attained on the Tiny ImageNet test dataset. The individual performance of the three network architectures illustrated in Figure2 is summarized in Table 2. The VGG-16 architecture achieved a test accuracy of 0.494, while the Assignment 3 ConvNet achieved an accuracy of 0.423, and the simpler VGG-like network achieved an accuracy of 0.237. Interestingly, a majority vote of the architectures achieved an accuracy of 0.492 percent, slightly lower than that of the VGG-16 architecture. This is due to the fact that the errors across the three archi-

tectures are highly correlated (Pearson correlation = 0.732), so we do not get the expected 2% accuracy boost from an ensemble classifier.

Architecture	Training Accuracy	Validation Accuracy	Test Accuracy
9-Layer ConvNet	0.988	0.455	0.423
VGG-like ConvNet	0.488	0.288	0.237
VGG-16	0.760	0.535	0.494
Ensemble			0.492

Table 2. Accuracy of the three network architectures on Tiny ImageNet training, validation, and test datasets.

The change in loss and accuracy over epochs of training is illustrated in Figure4.

3.2. Model Visualization

3.2.1 Saliency Maps

A function was implemented to compute saliency maps in accordance with Simonyan et al[15] and Zeiler [21]. Saliency maps for each of the 3 models on a test image from class n07768694 (”pomegranite”) are illustrated in Figure5. All three classifiers classified this image correctly. The absolute value of the saliency score is illustrated in gray scale, while areas of positive and negative saliency are shaded in color. The saliency maps for the three maps are different, but they are not readily interpretable.

3.2.2 Convolutional Filter Visualization

Convolutional filters for the first and last convolutional layer in each of the three architectures are illustrated in Figure 6. As expected, the filters in the first convoluational layer largely capture simple patterns, such as checkedred squares and circles or stripes of a particular color. The filters in the final convoluation layer (Figure 6 d,e,f) are more intricate and capture more complex patterns. All filters are somewhat noisy, which is indicative of overfitting and is in line with the disparity between the training and validation accuracies. In lecture, we learned that noisy filters might be an indicator that regularization is too low, which suggests that increasing regularization further might help to reduce filter noise and improve validation/testing accuracy.

4. Discussion

The difference in performance across the three models illlustrates the power of transfer learning. The VGG-16 architecture and the 9-layer ConvNet were both initialized with pre-trained weights for the convolutional layers. The simpler VGG-like model in Figure 2c was trained from scratch, as pretrained weights for this architecture were not available. The two models that used pre-trained weights achieved accuracies of 45%- 49.4% on the test data, after 5-10 epochs of training, while the model trained from scratch achieved an accuracy of only 24% on the test data, even after 20 epochs of training. This difference in performance illustrates the power of transfer learning and the great utility of ”model zoos”, such as the one being developed for the Caffe platform[2].

Conversely, however, the VGG-like model is the only one of the three that did not suffer from the overfitting problem – validation accuracy followed training accuracy very closely (Figure 4). This might be due to one of two factors. It is possible that the pre-trained weights used to initialize the other two models are contributing to the overfitting problem. A lot more likely, however, is the fact that the VGG-like model is a much less complex model,involving only four convolution layers with a relatively low number of filters for each layer (Figure 2b). Consequently, due to the lower number of parameters in the model, overfitting is less severe, but accuracy is also lower.

In [16], Simonyan and Zisserman implemented a VGG-16 net that achieved a top-1 accuracy of 24.4% and a top-5 accuracy of 7.5% on the ImageNet validation set. My Keras implementation of the VGG-16 network used the weights published in [16], but performed considerably worse, achieving a top-1 accuracy of 49.4%. The difference in performance can be accounted for by the fact that the Tiny ImageNet dataset differs from the full ImageNet dataset that the VGG-16 network was trained on. The Tiny ImageNet dataset is smaller – there are 100,000 available images for training and 200 object classes (50 images per class), as opposed to 1.2 million training images for 1000 object classes (over 1000 images per class) in the full ImageNet challenge [14].

Due to the lower amount of training data per class, overfitting on training data proved to be the most significant challenge in this project. My initial implementation of the 9-layer ConvNet in 2 achieved a training accuracy of 98%, but a validation accuracy of only 13.1% and and a test accuracy of 10.5%. The addition of dropout, regularization, and data augmentation helped to reduce the overfitting problem, and allowed the model to attain the higher accuracies illustrated in Table 2. However, as shown in Figure 4, all three models continue to suffer from overfitting on the training data. The validation accuracy plateaus after 5-10 epochs of training, depending on the model.

Though dropout and regularization helped, data augmentation was by far the most impactful step in helping to combat overfitting. Due to limitations on time and resources, random subsets of the training data were selected for each augmentation step (see ”Data augmentation” subsection in ”Methods”). However, in a future iteration of the project, a smarter approach to data augmentation would likely yield improved overall accuracy.

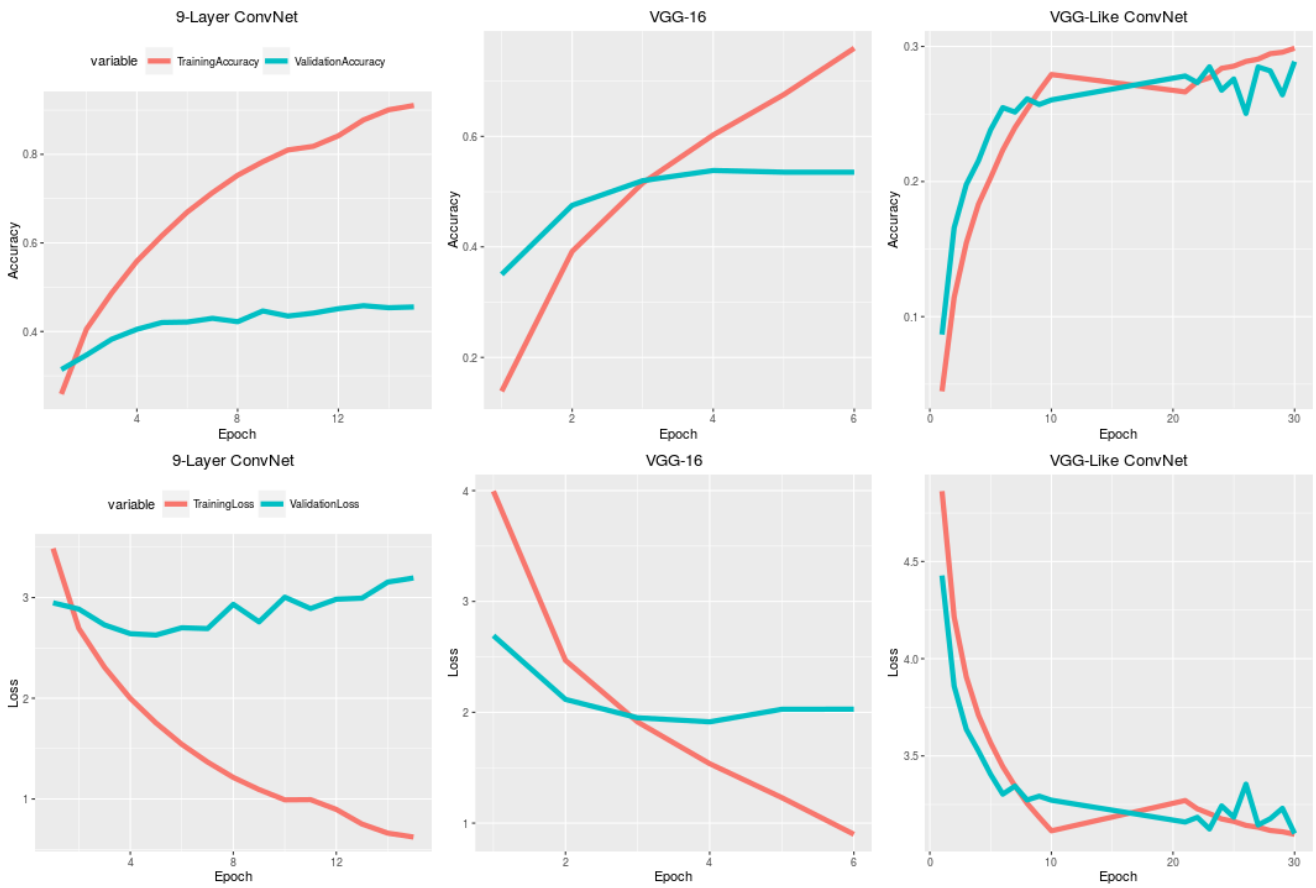

Figure 4. Accuracy and loss for the three network architectures that were trained in this project. The red line indicates training accuracy and loss; the blue line indicates validation accuracy and loss.

Specifically, examination of error by object class illustrates that the model is much better at identifying some object classes than others. For example, as shown in Table 3, the model does exceptionally well at identifying goldfish, school buses, trains, monarch butterflies and ladybugs. This finding is un-surprising – these image classes are all quite distinct from other image classes, and quite homogeneous within themselves. For example, Figure 7a illustrates that the goldfish images used in the validation dataset are fairly standard – the fish is center stage and the shape/coloration of the object remains consistent. The model had more trouble with objects of class umbrella, punch bag, wooden spoon, syringe, and plunger. As illustrated in Figure 7b, many of the problem images have multiple objects other than the object of interest. The object of interest is small, off -center, or portrayed from an unusual angle. These fail cases are in line with those described by Karpathy in [3] (i.e. small, thin objects such as the syringe or wooden spoon).

Class Label	Words	Number Misclassified
n01443537	goldfish	4
n04146614	school bus	6
n02917067	bullet train	7
n02279972	monarch butterfly	8
n02165456	ladybug	8
n04507155	umbrella	39
n04023962	punch bag	39
n04597913	wooden spoon	39
n04376876	syringe	41
n03970156	plunger	42

Table 3. Number of images that were misclassified in the validation dataset by VGG-16. The top portion of the table illustrates the 5 classes with the highest overall accuracy; the bottom portion illustrates the 5 classes with the lowest overall accuracy.

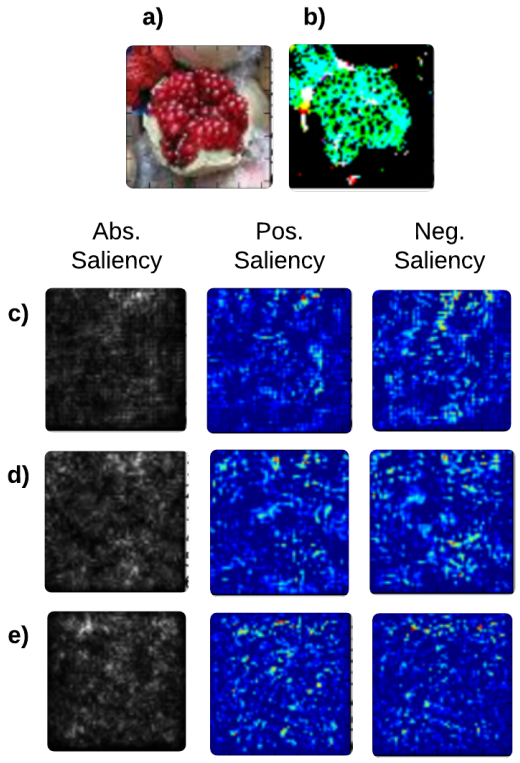

Figure 5. Saliency maps for test image test_8715.JPEG: a) Original image. b) Preprocessed image. Mean has been subtracted and image has been normalized by standard deviation of training data. c) Saliency maps from the 9-layer ConvNet model; absolute value in grayscale, followed by positive saliency and negative saliency. d) Saliency maps from the VGG-16 model. e) Saliency maps from the VGG-like model.

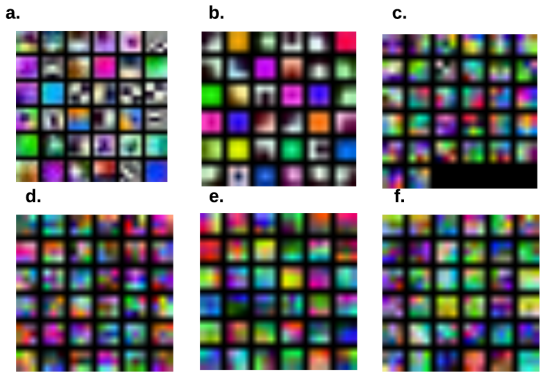

Figure 6. Convolutional filter visualization. a) 9-layer ConvNet, first convolutional layer. b) VGG-16, first convolutional layer. c) VGG-like ConvNet, first convolutional layer. d) 9-layer ConvNet, ninth convolutional layer. e) VGG-16, 13th convolutional layer. f) VGG-like ConvNet, fourth convolutional layer.

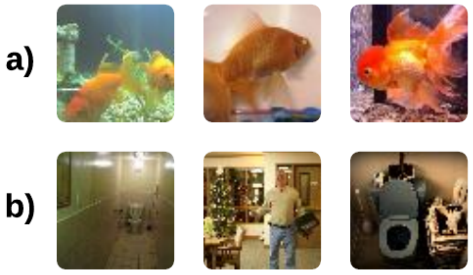

Figure 7. Representative images from the image classes in the validation dataset with best and worst accuracy. a) The "goldfish" class had the highest accuracy. b) The "plunger" class had the lowest accuracy.

5. Future Work

The biggest obstacle to high classification accuracy for the Tiny ImageNet challenge is overfitting – accuracy of 98% can be achieved on the training data with the pre-trained VGG16 and 9-layer ConvNet models, but validation accuracy plateaus at approximately 50%. Consequently, future work will focus on reducing the overfitting problem. Some techniques that may be of use include:

- I attempted to implement DropConnect[20] for the project, but unfortunately ran out of time. Whereas dropout sets a random subsets of activations in each layer to zero, DropConnect sets a random subset of weights to zero. This causes each unit to receive inputs from a random subset of units in the preceding layer, further increasing noise in the system, and helping to reduce overfitting.

- Smart data augmentation could also mitigate the overfitting problem. Currently, 30% of training images are selected at random to undergo each of the augmentations. However, as described in the Discussion section, some object classes are a lot easier to identify than others. Based on the initial results from the VGG-16 model, I would like to further augment the image classes where the model had the lowest accuracy (Table 3).

- I found that the time it takes to train the networks posed a bottleneck to development and iteration. To reduce the training time, I would like to prune redundant connections in the network, as described by Song Han et

al in [15]. In a first pass on the training data the important connections will be learned. Low-weight connections will then be pruned. Finally, the network will be retrained to learn the weights for the surviving connections.

References

[1] https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3. 2

[2] http://caffe.berkeleyvision.org/model_zoo.html. 5

[3] http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/. 6

[4] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation. 1

[5] F. Chollet. Keras. https://github.com/fchollet/keras, 2015. 1, 2

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society. 1

[7] V. Garg, S. Chandra, and C. V. Jawahar. Sparse discriminative fisher vectors in visual classification. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '12, pages 55:1–55:8, New York, NY, USA, 2012. ACM. 1

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May 2010. 2

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1, 4

[10] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009. 4

[11] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2169–2178, Washington, DC, USA, 2006. IEEE Computer Society. 1

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. 1

[13] M. Pietikäinen, A. Hadid, G. Zhao, and T. Ahonen. *Computer Vision Using Local Binary Patterns*, volume 40. Springer, 2011. 1

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 1, 5

[15] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013. 5, 8

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2, 5

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 2

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1

[19] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. 1

[20] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013. 7

[21] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. 5