A Dense Take on Inception for Tiny ImageNet

William Kovacs
Stanford University
kovacswc@stanford.edu

Abstract

Image classificiation is one of the fundamental aspects of computer vision that has seen great advancements due to the rise of deep learning. While this problem has been essentially solved within the past few years via the use of large networks, it still provides a good baseline for testing out new architectures. Here, we try to combine ideas from prior successes in order to leverage their strengths: the wideness that Inception modules provide with the feature reuse of DenseNets. To this end, we developed two dense-inception model that performs comparably with the inception one that it was based on. The models were tested on Tiny ImageNet, with test errors of 0.612, 0.664, and 0.623 for the Inception ResNet and two DenseNets, respectively. While the change in architectures does not provide much difference between the results, the overall filter distribution was changed significantly, suggesting further work would need to be done to boost the underlying model, but shows hope for their synergy.

1. Introduction

Over the past several years, deep learning has become the go-to method for most computer vision tasks due to its great efficacy. The most basic of these tasks is image classification, wherein a model is generated that is able to correctly identify the subject of various images. One of the most promiment competitions in this task is the ImageNet Large Scale Visual Recognition Challenge [16]. Here, small images from a very wide selection of objects, from boats to lamps to cows, are divided into labeled training and validation sets, and the goal is to develop a model capabale of learning from the data that can accurately label new images.

For this task, Krizhevsky et al. [11] popularized deep convolutional neural networks (CNNs) as the standard tool in this task. His initial model outperformed the other methods during the 2012 ImageNet competition and since then, deep learning has been the focus of computer vision. During the following years, new models have introduced novel improvements to the original CNN modek, such as residual connections or novel modules, that significantly improve results.

Indeed, image classification on ImageNet has become roughly as good as humans when using current state-of-the-art models [10]. Such a rapid increase in performance has been due to the focus on this model paradigm, with efforts to increase their depth, as well as the improvements in computation power that has enabled these methods to be trained in a practical amount of time compared to previous decades.

1.1. Related Work

In the following years since the initial breakthrough, there was great focus on creating deeper neural networks capable of leraning more complex functions. This paradigm can be easily seen in the corresponding 2014 challenge, where small, 3x3 convolutional filters were stacked to create a deep, 19 layer network to generate a competive network [19].

In the following year, an even deeper network, known as ResNet and consisting of 152 layers, was developed [6]. Increasing the depth of a network also increases the difficulty of training. To counteract this effect, ResNet learns residual functions based on that layer's input as opposed to learning an entirely new new functions.

There are a multitude of other network architectures that utilize a similar idea to ResNet of trying to include short paths from earlier layers to deeper ones. This can readily be seen in Highway networks [21], which uses a gating mechanism, similar to the Long Short Term Memory idea used in recurrent neural networks [7]. FractalNets also utilize a similar pattern, albeit the connections here consist of parallel paths where a single layer has two corresponding layers in an adjacent path as opposed to direct connections [12]. The inclusion of shortcuts can also be seen in Directed Acyclic Graph CNNs, which provide connections between every layer to the output layer [25]. The representation of hypercolumns as a CNN also follows this similar pattern, as the output of a few layers at varying depths all contribute to the final layer [5].

Similar to the above methods is the DenseNet; however, the new connections in this network are designed to promote feature reuse thereby enabling the creation of shorter, more efficient networks, as opposed to very deep ones [8].

There have also been focus on trying to increase the width of a network, instead of just increasing the depth. For instance, it has been shown that a wide ResNet variant can prove to be just as effective as a deep ResNet [26]. Indeed, in the 2014 ImageNet challenge, GoogLeNet introduced inception modules which increased the width of the network by performing mulitple, parallel convolutions on the same input before concatenating their results and using that as the input to the following layer [23]. By stacking such models, they were able to generate both a wide and deep network. Interestingly, an adaptation of these modules was introduced in [14] wherein the collaborative concatenation stage was replaced by a competitive maxout activation. Another interesting adaptation to this method is to include the aforementioned residual connections within the inception modules [22]. While an inspiring model, the addition of these residual paths led to similar results when compared against a similar, pure-Inception based network, though the performance was already so good on the image classification problem that their difference may not be visibile in this problem.

Another interesting architectural design that doesn't quite fit with the above categories is the Network In Network design that essentially replaces the standard, linear convolutional layers with convolution layers that utilize multilayer perceptrons [15].

Aside from architectural designs, recent work has also been focused on finding alternative ways to optimize the training of these networks. For instance, in ShiftCNN, the network is designed to utilize the less expensive shift and add operations over multiplications to increase the speed of training with little loss of accuracy [4]. Another method used to increase the speed of training is through the use of very large batch sizes, made possible by the use of a linear scaling rule to adjust the learning rate, correspondingly [3].

Other variants to the training method include the utilization of higher order features, instead of solely first-order ones [13]. A quite successful approach to altering the training method is to use stochastic depth, that is during training, a subset of layers are dropped to effectively only train small networks, and during testing, the full deep network is used [9].

2. Dataset

Instead of training on the full-sized ImageNet, our models will be trained on a subset of this collection known as the Tiny ImageNet. Images included in this dataset are 64x64 pixels. These patches represent a wide variety of object classes, and are labelled in a hierarchical fashion via the WordNet hierarchy. These classes can range from "tailed frog" to "oranges" to "poles," as can be seen in Figure 1.

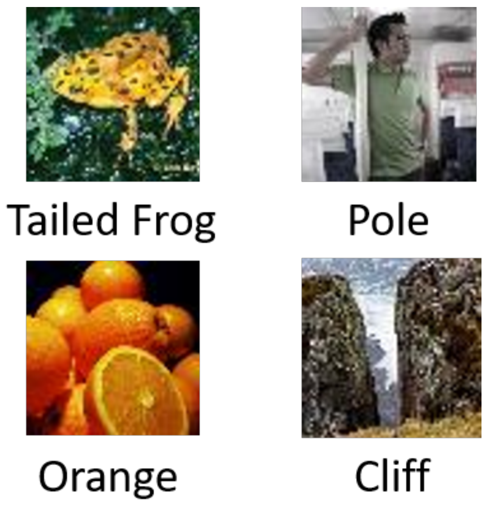

Figure 1. Example images from the Tiny ImageNet demosntrating the wide variety of classes that are contained. Furthermore, the pole example demonstrates an issue common with some classes: the presence of foreign entities that can intefere with classification.

This figure also highlights the complexity that some seemingly simple classes can exhibit: the image for a "pole" has a person as the center of the image that may confuse the network.

In the Tiny ImageNet data set, there are 100,000 training images, 10,000 validation images, and 10,000 test images. These images are split evenly into 200 classes. Prior year's results ranged from an error rate of 0.616 to 0.268.

To prepare our data for processing, they underwent some standard procedures. The first is to generate a 'mean' image whose pixel values correspond to the average of all the corresponding pixel values in the images. This 'mean' image is then subtracted from each image, similar to the preparation of the VGG-Net dataset. Then, to further augment the data, horizontally-flipped copies of the image were included as part of training.

3. Methods

Our model focuses on combining the core ideas of the inception models with those of the DenseNet. Specifically, the inception models utilize parallel paths with varying receptive field size in order to increase the width of the network. Meanwhile, the DenseNet uses concatenation of layer outputs to an overarching input for a block in order to promote feature reuse.

The combined model will be compared against two other models: a simple CNN model and a small Inception ResNet model. The simpleCNN model is used as a contrast to what a basic model composed only of convoutional layers can achieve, even if it was able to perform well on a previoous challenge, as well as demonstrating that the wide architecture of the other networks are important to their success. Meanwhile, the Inception ResNet model provides a good comparison to determine the effects of the dense connections.

All of the following models were constructed using Tensorflow (TF) Layers [1].

3.1. Simple CNN

Our initial baseline model was based on a model that was initially constructed for high accuracy on the CIFAR-10 dataset in assignment 2. It consists of 5 pairs of convolutional layers followed by maxpooling, and ending with 2 fully connected layers and a softmax classifier. The first two pairs had a filter size of 5x5 before shrinking to pairs of 3x3 filters, while the number of filters doubled after two of each pair, starting from 128, to 256 at the third pair, and ending with 512 at the final pair. The purpose of this baseline model is to serve as a skeleton constructed of the components of the more complex models to demonstrate the utility of the new connections, and to show that the non-inception convolutional layers are not the ones that dominate the power of the model, and so would be able to account for the similarity between the two different types of connections.

3.1.1 Regularization

There were two main regularization techniques employed, that will also be used in the following two models: adding an L2 regularization penalty and utilizing dropout.

The L2 regularization is simply the addition of the L2 norm of the weights to the loss function, weighted by the hyperparameter λ. That is, $R(W) = \lambda \sum_k \sum_l W_{k,l}^2$. This penalty encourages smaller, diffuse weights, especially compared against using the L1 norm. In the simpleCNN model, λ was set to 0.0001, which was determined empirically.

Srivastava et al. first demonstrated the power of dropout as a form of regularization [20]. Essentially, during each iteration of training, a random subset of neurons are selected to be 'dropped', and those neurons are not updated, while the remaining ones are. It's efficacy is thought to be due to the prevention of the neurons from co-adapting, that is subsets of neurons could update their features together. By preventing these neurons from co-adapting, dropout encourages more generalized classification. During testing, no dropout occurs. There is a single hyperparameter, p, that represents the probability that a given neuron will be dropped. In this model, p was set to 0.3, which was determined empirically between 0.3 and 0.5.

3.2. Small Inception Resnet

In the past year, the authors behind the Inception ResNet have released the code as part of the model zoo in the Tensorflow Slim library [17]. While the code itself was not used and all further models were constructed separately using TF Layers, it was used as a guide in order to get a good number of filters for the various convolutional layers in the model, which would have otherwise taken too long to determine if done from scratch. Furthermore, while the authors do provide a pre-trained version of the full Inception ResNet, it was deemed as an unfit model for comparison due to its distinct advantages over any model trained from scratch. That is, there is an inherent advantage that it provides with the size of the network (40 repetitions of inception modules that was trained over 20 GPUs), which is simply an infeasible size for us to train given the current hardware we could use. Furthermore, after a single epoch of fine tuning, the validation accuracy of the pre-trained model was already 0.531. Because the model was already trained on ImageNet beforehand, the intermediate weights are already good representations for the current subset of that collection, and all it has to do is essentially relearn the mappings that it would have had when used in the original ImageNet. While our model consists of similar building blocks as the full Inception ResNet, the differences are too great to get a useful comparison.

Instead of using such a large model, we constructed a customized small Inception ResNet with a vastly decreased number of layers to both make training from scratch feasible, as well as to provide a more apt comparison to the dense version of this model. Before describing the full model, an overview of the inception modules that are prominiently used is described in the following section.

3.2.1 Inception Modules

The core of the Inception ResNet are embodied in the special blocks introduced in [23], known as inception modules. As can be seen in figure 2, each block consists of parallel routes of varying receptive field size: 1x1, a 3x3, and a 5x5, as well as an average pooling layer. This parallelism increases the overall width of the network and has been proven to be an efficient alternative to the straight pathways of prior models [23]. As can further be seen in the figure, the 5x5 field size path is actually split into two 3x3 convolutions to be more efficient, as this style effectively covers the same region as a single 5x5 convolution, while requiring fewer parameters. Furthermore, to reduce the overall number of parameters, three of the paths utilize 1x1 convolutions that reduce the feature depth.

The ResNet version of these modules also feature a residual connection from the input of each module to the output concatenation, which can be seen as the orange line

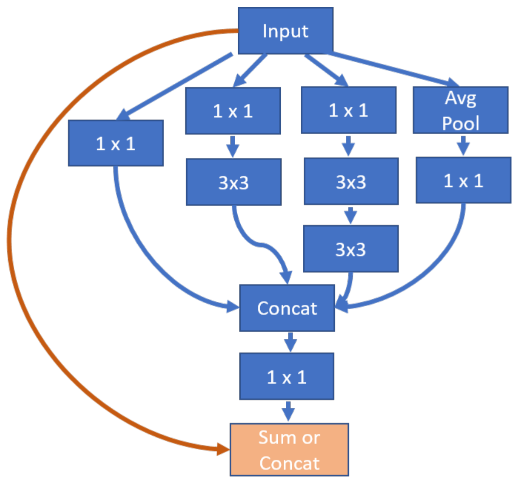

Figure 2. The inception module paradigm used in the networks. The orange line corresponds to the major change between the ResNet and the DenseNet version of the architectures: either addition or concatenation, respectively.

in figure 2 and using the sum operation. The utility of such connections has been discussed in Section 1.1. In [22], it's shown that the the addition of these residual connections offers similar results when compared to a network composed of modules without such connections. We chose to compare the dense version with the residual version because while both types (the vanilla and residual) show comparable performance, the residual method provides a cleaner transition between the two types of network with a simple change, which will be described in the DenseNet section.

3.2.2 Inception ResNet Architecture

The full architecture can be seen in figure 3, and is inspired by [22], but is scaled down in order to better match the smaller data, as well as to make training multiple models feasible with the given time and hardware. As can be seen, the model begins with several 3x3 convolutional layers with an increasing number of filters (32, 80, 192, 320), followed by a 2x2 max pooling with a stride of 2. This is followed by a triplet of inception modules, followed by a max pooling, another triplet, more max pooling, and ending with a fully connected layer and a softmax classifier. The number of filters for the convolutional layers within the modules in the first triplet were either 32, 48, 64, or 96, except for the last 1x1 convolutional layer, whose number of filters equaled those of the input tensor. In the second triplet, all filters, barring the last one, were either 80 or 96. The maxpooling after each triplet of inception modules actually consists of a similar inception-style block, where each of the three convolutional paths have a corresponding reduction: the 1x1 has its max pooling, the 3x3 and 5x5 paths use a convolutional layer with stride 2 in their final convolutional layer. For simplicity, this block is referred to as maxpooling.

This model uses the same regularization techniques as those described in section 3.1.1. For L2 regularization, λ was set to 0.0001, and for dropout, p was set to 0.3. Here, dropout was only performed on the final, fully connected layer. These methods of regularization were emprically determined. Indeed, without these techniques, there was a gap of 48% between training and validation accuracy (0.893 vs 0.411, respectively), but with them, it was reduced to only 1.4% (0.418 vs 0.404, respectively).

3.3. Inception DenseNet

This model attempts to implement dense blocks as described in [8], with the use of inception modules instead of single convolution layers. For each dense block, there is an overarching input representation that will be extended after each inception module. For example, the input starts at size n. This is fed through the first inception module, and its output is also size n and is concatenated to the end of the original input, which now has size $2n$. This process is then continued for however many modules are inside the dense block. In this model, each dense block consists of a triplet of inception modules, so the initial input is re-used twice, the second once, and the third simply undergoes maxpooling.

The inception modules used in the ResNet variant undergo a simple change to become part of a dense block: the residual connection is changed from a summation to a concatentation. This change is highlighted in figure 2 at the connection highlighted by the orange line.

The overall architecture remains the same as described in Section 3.2.2 and seen in figure 3. However, the number of filters for the convolutional layer just prior to the inception modules is reduced by 1/4, from 320 to 80. Furthermore, each of the layers within the inception modules and their respective max pooling is reduced by 1/4 as well, to demonstrate how the dense network can be used to achieve similar results with less filters. This means that the number of filters within these modules were now either 12, 16, or 24 in the first triplet (the convolutional layer used after the average pooling was only decreased by 1/2 from 32 to 16), and 20 or 24 in the second triplet. This network is known as DenseNet-A.

While the original intention was that each input would only grow by some constant addition, due to the implementation with the final 1x1 convolution being based on the input size to the layer and not the block, the input size actually doubled for each module in a dense block in one of the models. Fortunately, due to the reduction of filters by a quarter, the input into the third and final module for each block was actually the same size as in the residual network, meaning that there was a similar number of parameters between the

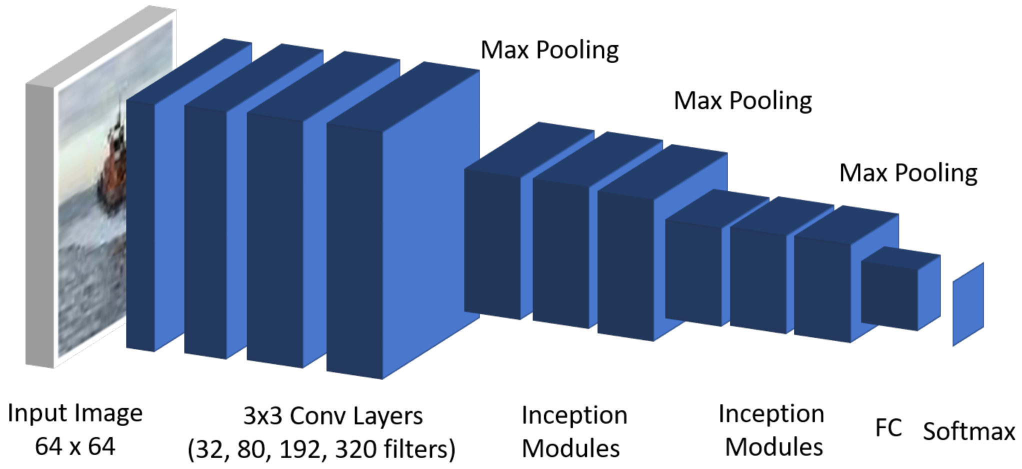

Figure 3. A schematic of the Inception networks that are used. Each triplet of inception modules are connected in a dense fashion in the dense network.

two models. The input propagation did not continue between dense blocks because the input size was reset during the max pooling. The increase in block size does provide an interesting comparison to the intended denseNet, and so we include the results as DenseNet-B.

4. Experiments

4.1. Training

Our models' weights were first initialized with Xavier initialization [2], which is aimed towards maintaining the scale of the gradients similar between different levels of the network by scaling the initializations according to the number of input neurons. The networks were trained using stochastic gradient descent (SGD). An adaptive learning rate method, RMS prop [24], with a decay value of 0.9, and momentum with a value of 0.9, was used for this method. Furthermore, the learning rate decayed by a factor of 0.1 if the loss would stagnate between multiple epochs. The learning rate was initially set to 0.0001, determined empirically between each magnitude of difference from 0.01 to 0.000001. A variety of batch sizes were tested (64, 128, 256), with 128 being selected as the final one, as it allowed to look at the largest amount of data per iteration, while retaining no performance issues, and prevented the model from running out of memory that the larger batch size could cause. The models were run for 20 epochs (about 4-5 hours), by the end of which they appeared to have converged.

In addition to the previously described models, a version of the Inception ResNet model was trained that used the same number of filters as the DenseNet models in order to further demonstrate the importance of the dense connections.

4.2. Testing

To evaluate our models, we compared the validation error and top-5 error, as well as testing error. Here, error is defined as the percentage of images that were misclassified using the model. Top-5 error refers to the percentage of images that did not have its label included as the model's top-5 possibilities for the given image. Only validation top-5 error could be recorded because the error of the testing data was performed on a server that we did not have access to and would only provide the error metric.

4.3. Visualizations

In order to better understand the differences and similarities between the models, saliency maps were generated according to [18]. Essentailly, a saliency map reflects the importance of a given pixel on the final classification score, and is calculated as the gradient of the correct class's score with respect to the pixel value. The code for these visualizations was adapted from assignment 3. In order to gain understanding into these models, the saliency maps corresponding to the best and worst classes are shown.

5. Results and Discussion

5.1. Classification Errors

The classification error of the different models can be seen in Table 1. As to be expected, the generation of a small, simple CNN performs rather poorly. However, it serves its purpose in that it provides a contrast to the inception model of similar depth, and to show that it is not just the straight, convolutional layers that provide the power behind these models.

It's interesting to compare the results of the various types

Model	Val Err	Val Top-5 Err	Test Err
Basic CNN	0.738	0.511	0.764
Inception Resnet	0.596	0.345	0.612
Inception Densenet-A	0.628	0.373	0.664
Inception Densenet-B	0.586	0.336	0.623

Table 1. Error metrics for the various models tested. Both of the inception models performed better than a simpleCNN model, while the dense network appears to yield similar results despite having fewer parameters.

of the constructed Inception networks. DenseNet-A, with its filter reduction, had only a difference of 0.05 accuracy compared to the ResNet version, and only a difference of 0.04 from the DenseNet version with its greater block sizes. The fact that both DenseNet versions are capable of achieving results similar to the residual network suggest that the dense connections are able to significantly contribute to the overall model. However, the difference between DenseNet-A and DenseNet-B suggest that there is a yet to be discovered optimal block size that can be used to more efficiently balance the number of parameters with the validation accuracy.

To further test the importance of the dense connections, a model using the Inception ResNet modules was trained, with the exception that the number of filters within the modules coincide with those used in the DenseNet model. After training such a model, the validation error was only slightly above that of the simpleCNN model: 0.742. This similarity suggests that the inception modules in the ResNet version, when not supplied with enough filters, contribute very little to the overall classification. This further suggests that the dense connections in the DenseNet models contribute significantly to their results.

Rather than just focusing on the accuracy of the models across all classes, it's helpful to examine the types of classes that were easier or harder to classify. For instance, across all of the Inception models, two of the highest accuracte classes were the lifeboat and goldifsh classes (with above 80% accuracy across all models). Meanwhile, two of the least accurate classes were the rocking chair and the lampshade classes, with less than 10% accuracy across all classes. Example images from these classes can be seen in Figure 4. Looking at the images themselves, there is a fairly intuitive explanation for why the models behave they do. Lifeboats and goldfish have finer details that the model can use to identify members of their class. Conversely, rocking chairs and lampshades consist of very basic shapes that may be difficult to distingsuish from other classes.

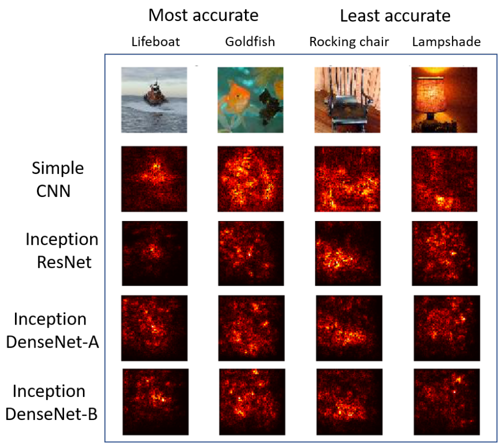

Figure 4. The image classes with the highest and lowest accuracies are presented, and their saliency maps are compared among the different models that were tested.

5.2. Saliency Maps

More intuition into why the models behave as they do can be garnered by looking at the generated saliency maps, which appear in Figure 4 as well. The first thing that becomes apparent is that the saliency maps for the simple CNN model tend to be considerably fuzzier, which correponds well with the results that we have seen. However, the saliency map for the lifeboat has a lot of focus on the lifeboat itself, suggesting that objects in this class are very distinct, and easy to identify, given that this simple model was able to easily identify it. The other classes are too fuzzy for this model to make any other reasonable conclusions.

The sharp focus on the lifeboat remains with the Inception models, further confirming its lack of difficulty. The DenseNet variants, while still generally focused, is fuzzier for this particular class, suggesting that it may consider the environment as well. The goldfish presents an interesting saliency map. For the ResNet variant, while there is a general fuzziness, there are distinct bright spots in the middle that correspond to the fish itself, particularly the eye, as well as the contours of the body. This correspondence can be more readily seen in Figure 5. For the DenseNets, there are bright spots corresponding to the goldfish as well, but there also appears to be higher intensity surrounding the fish as well, once again suggesting that the environment plays a larger role in the dense net variants.

The less accurate classes provide two different types of interesting saliency maps. For the rocking chair, there is a heavy emphasis on the seat of the chair, which appears to have very little distinguishing features. This could explain the poor performance for this class: rather than being able to develop features for the more unique aspects of a chair, such as the rails on the back, the model learned to

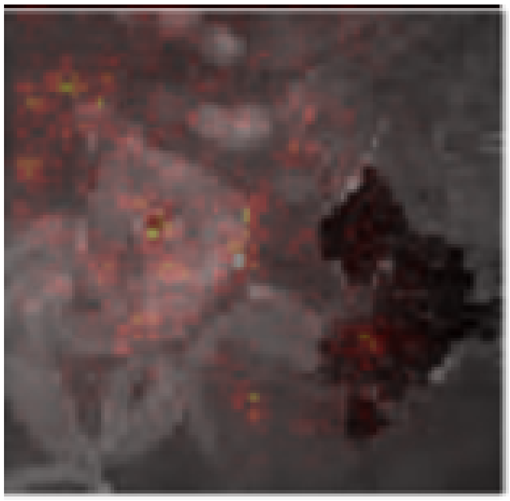

Figure 5. A close-up, grayscale image of a goldfish image overlayed over its corresponding saliency map for the Inception ResNet to highlight the high values along the head of the fish.

identify this sort of rectangle, which can belong to a variety of classes. Meanwhile, the lampshade example provides another paradigm for a saliency map on a misclassified image: there is simply a lack of learned features, and so a fuzzy blob appears. This is similar to the results of the simple CNN model for the other classes as well.

6. Conclusions

Using the Inception ResNet as a guideline, we were able to construct a model that performs okay on the Tiny ImageNet challenge. More interestingly, we were able to adapt this method to use dense connections that allowed for the reduction of filters within the inception modules while achieving similar results. This method relies on the conacatenation of filters to an overarching input representation per dense block, and we demonstrated that adjusting the size of these additions via the final 1x1 convolution can improve performance, even if the other filter numbers were reduced. It would be interesting to perform a wider variety of DenseNet architecture tuning to determine an optimal ratio of these filter numbers, which would require much more time than currently given.

Furthermore, the focus of this project drifted towards generating a dense version of an inception network that can achieve similar performance with fewer filters, so it would be interesting to focus on enhancing the original, ResNet model to include an extra block of inception modules, as well as increasing the overall number of modules within each block. Efficiently training such a network would require greater computing power, so configuring the network to run on distributed GPUs would be very helpful to reduce training time. If such a larger and more accurate network could be developed, applying the same conversions as performed here would be able to demonstrate its scalability. Overall, this project demonstrates the feasibility of extending the dense architecture to use inception modules, while still accruing average results for the Tiny ImageNet challenge.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[3] P. Goyal, P. Dollr, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[4] D. A. Gudovskiy and L. Rigazio. Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks. *CoRR*, abs/1706.02393v1, 2017.

[5] B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. *CoRR*, abs/1411.5752, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory, 1995.

[8] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[9] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[10] A. Karpathy. What i learned from competing against a convnet on imagenet, 2014.

[11] 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.

[12] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.

[13] P. Li, J. Xie, Q. Wang, and W. Zuo. Is second-order information helpful for large-scale visual recognition? *CoRR*, abs/1703.08050, 2017.

[14] Z. Liao and G. Carneiro. Competitive multi-scale convolution. *CoRR*, abs/1511.05635, 2015.

[15] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[16] 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.

[17] N. Silberman and S. Guadarrama. Tensorflow-slim image classification library. `https://github.com/tensorflow/models/tree/master/slim`, 2013.

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

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

[20] 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.

[21] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[22] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

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

[24] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning, 2012.

[25] S. Yang and D. Ramanan. Multi-scale recognition with dag-cnns. *CoRR*, abs/1505.05232, 2015.

[26] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.