Tiny ImageNet Challenge - Dissection of a convolutional neural network

Jean-Baptiste Boin
Stanford University
jbboin@stanford.edu

Abstract

In this project, we tried to achieve the best classification performance in the Tiny ImageNet Challenge using convolutional neural networks. Our constraint was a modest training infrastructure, as well as a limited time for training, which prompted us to try to get the most out of relatively shallow network architectures (compared to typical state-of-the-art networks). At the time of writing this report, our model achieved the fifth place in the class leaderboard, with a test accuracy of 44.0%. Another goal of this project was to see how different components of a network contributed to its performance. In order to achieve that, we evaluated different iterative versions of the same architecture, with increasing complexity. We then evaluated the validation accuracy of these different models so as to see how each added feature increased the accuracy. In particular, this analysis showed the importance of dropout units, especially in limiting overfitting.

1. Introduction

The Tiny ImageNet challenge is a smaller scope version of the ImageNet challenge (ILSVRC) [1] which was introduced for this class. We werre given 100,000 training images, with 500 images from 200 different classes, 10,000 validation images and the same number of test images. The images were color images of size 64×64. We were also given a bounding box around the object on the training images. Our goal was to guess the class that each test image belongs to.

In this report, we are presenting our results and justifying our approach. In part 2, we will review some useful material that shaped this project. Part 3 will present in detail the best performing architecture that we trained and how we trained it. Finally, part 4 will focus on our achieved results. In that part, we will also discuss our "convnet dissection" experiment, where we evaluated the accuracy of an iterative architecture in order to see what components were responsible for the largest gains in the accuracy.

2. Background

Our main goals in this project were to experiment on different techniques that are used for training a convnet (from scratch, since transfer learning was not allowed for this challenge). Starting from a relatively classic architecture, we added/removed different components and studied how they affected the performance of the network, by keeping track of the changes in validation accuracy. This iterative approach provided us with more insight on how to improve the accuracy for this dataset, and thus get a more discriminative network, without being crippled by overfitting.

In our limited time working on this project, we considered different techniques that could improve our network. Here, we will first give a some background of the techniques that we experimented on as well as the theoretical justification behind them.

- **Leaky ReLU nonlinearities**. The leaky ReLU nonlinearity was introduced by Maas et al. [6] Even though it is slightly more computationally demanding than the very straightforward ReLU nonlinearity, the fact that its gradient is nonzero on its own domain usually makes it more successful for training a network.
- **Tradeoff size of filter/depth**. Some models like Krizhevky's model [5] introduce a first layer with large filters. On the other hand, other successful architectures such as [7] proved that sticking with smaller filters while duplicating the layers could lead to an increase in performance. The reason behind this improvement is that successive small filters can describe more complex relations than a single large filter.
- **Data augmentation**. This is one of the most popular methods for dealing with overfitting, which was relatively useful here.
- **Dropout**. This regularization method was introduced by et al. [3] and was shown to give substantial gains as illustrated by this model by Krizhevsky et al. [5], which was the best classification algorithm in the 2012 ImageNet Challenge. As the previous method, it is expected to help mitigating overfitting.

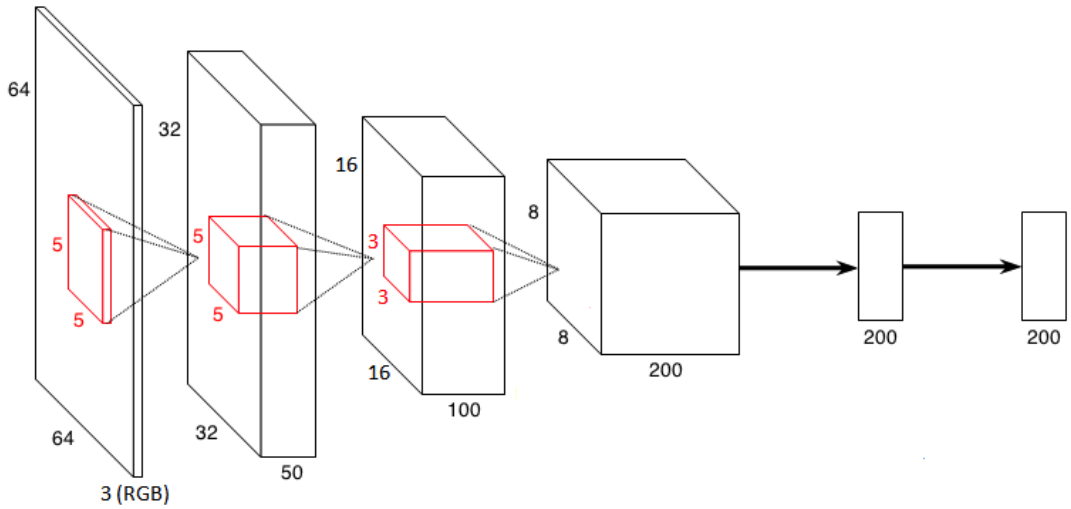

Figure 1. Architecture of our basic network on which we built the successive iterative architectures until our final model. The ReLU and pooling layers are not explicitly drawn.

- **Model ensembles**. This technique is relatively straightforward, but can still provide some improvements in our accuracy if implemented. Given the results of the 2014 ImageNet challenge, such as Google's successful submission GoogLeNet [9], it seems that this technique became very popular and consistently gives an additional boost of accuracy.

3. Approach

In order to spend more time on the higher level questions related to this project, we decided to work with the **CAFFE** framework [4].

For this project, we wanted to be able to train a network with our available infrastructure, without paying for additional resources such as Terminal or Amazon EC2 servers. We only relied at 100 % on the Stanford FarmShare *rye* servers, which provide us with a low number of GPUs that had to be shared with other students, mostly from this class. For that reason, we could not implement and train some of the most complex state-of-the-art architectures that would outperform our current results, but instead we strived at getting the highest accuracy from a model with a limited number of parameters and layers. This constraint was an interesting twist that forced us to focus on more refined techniques rather than a brute-force deep network.

Our first baseline network is described layer by layer as follows. An illustration is given in figure 1.

1. Data layer
2. Convolutional layer - filter size: 5, stride: 1, number of filters: 50
3. ReLU
4. Pooling layer - max pooling, stride: 2
5. Convolutional layer - filter size: 5, stride: 1, number of filters: 100
6. ReLU
7. Pooling layer - max pooling, stride: 2
8. Convolutional layer - filter size: 3, stride: 1, number of filters: 200
9. ReLU
10. Pooling layer - max pooling, stride: 2
11. Fully connected - 400 units
12. ReLU
13. Fully connected - 200 units
14. Softmax classifier

This is a relatively classic architecture, motivated by our infrastructure constraints. We introduced a pooling layer after each conv-ReLU block so that we would quickly decrease the size of the feature maps, and thus decrease the memory/processing time necessary for each image. The filters in the first two convolutional layers have a size of 5×5, which is motivated by the fact that we wanted to learn sufficiently discriminative models using our three convolutional layers only.

After training this model, we then implemented different additional in an iterative way, adding them one after each other until we reached our most complex, and also best performing, model. The way we present our roadmap is very

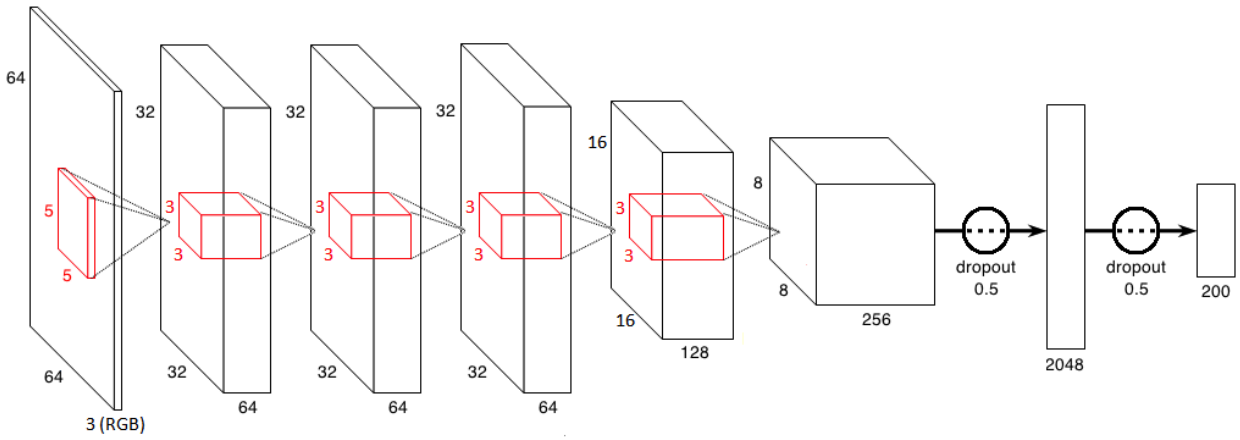

Figure 2. Architecture of an instance of our final network. Results from 11 similarly trained models with this architecture was used for our final predictions.

linear. However, in practice, we tried other different architectures that we did not list here, either because they were. More discussion about this will be included in the next section. Each model of the following list includes all components of the previous one, and implements an additional one that is given in the description.

1. **Simple model**: Architecture presented above

2. **+ dropout 1**: We added a dropout unit (probability: 0.5) between the two fully connected layers.

3. **+ data augmentation**: We generate random augmented data on the fly by applying random horizontal mirroring and cropping a 60×60 window out of the 64×64 images.

4. **+ leaky ReLU**: We replaced all the ReLU nonlinearities by leaky ReLU that have a fixed negative slope of 0.01.

5. **+ more parameters**: We increased the number of filters in the three convolutional layers, from 50, 100 and 200 respectively to 64, 128 and 256. We also considerably increased the size of the first fully connected layer, from 200 units to 2048 units.

6. **+ split second conv layer**: We replaced the second convolutional layer with three convolutional layers with filters of size 3×3, each separated by a ReLU (but no pooling layer in between them).

7. **+ dropout 2**: We added another dropout unit (probability: 0.5) between the last convolutional layer and the first fully connected layer.

8. **+ ensemble model**: We trained 11 models of the previous architecture with slightly different hyperparameters as well as a different initialization, and averaged the prediction probabilities within the models.

These models were fine-tuned by varying the learning rate and weight regularization parameters in a coarse to fine manner, using the validation data. As we go down the different models, a larger training time was usually needed, especially when there was augmented data or more parameters to train. The parameters given above, such as the dropout probabilities, were also chosen because they gave good results compared to other coarsely sampled values. It is to be noted however that we did not fine-tune these parameters. For all the models, we used a similar gaussian initialization for the weights, independently of the layer they were in. For practical reasons, we did not vary the standard deviation of the gaussian and assumed that our choice of parameter was good enough. Some papers, like [2], introduce a careful initialization method that seems to outperform a standard gaussian initialization. We did not follow their procedure so that we would have more time looking at the influence of other parameters. Finally, we used a momentum stochastic gradient descent with a momentum parameter of 0.9.

The final ensemble model was the one that we used for our submission. A simplified diagram of the architecture of a single model is given in figure 2.

4. Experiments and Analysis

4.1. Tiny ImageNet submission

We presented above the architecture of our submission. In this subsection, we give more information on our submitted predictions and analyze some data related to this network.

Among the models that we trained for the ensemble, the single model with the highest validation accuracy achieved

a validation accuracy of 44.90 %, while having a training accuracy of around 66.3 % and a validation loss of 2.368.

The weights from the first layer of this model are shown in figure 3. The reason why this figure has a low contrast is because the filter values are normalized so that the range of the values would fit in the $[0, 255]$ range, and despite our regularization, some of our filters had very peaked values. We can remark from this figure that this set of weights is qualitatively satisfying. Indeed, most of the filters are very different from one another, and they seem to capture varied information: edges of different orientation, color blobs, lines, high frequency texture, etc. Most filters look smooth, which proves that our regularization parameter was not too low, but some of them are also very peaked, which means that our regularization parameter was not too high either and our filters could capture noisy texture information.

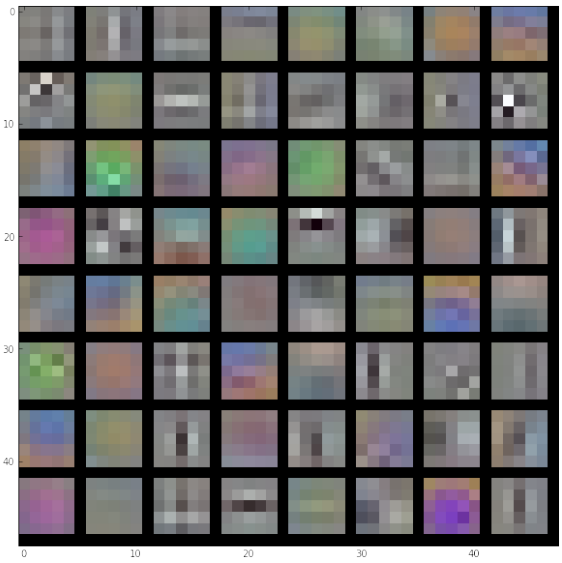

Figure 3. Filters of the first convolutional layer from our best single convnet (with respect to the validation accuracy).

We can verify this by looking at typical feature maps from this first layer applied to an example image, in figure 4. The responses are varied and capture information from varied parts of the image.

The validation accuracies given above correspond to the values we got from the predicted probabilities by feeding each image to the network. In order to improve the result of the prediction of one image, one way to go is to predict values for different oversampled versions of that same image, and then to average the probabilities. We used the default Caffe option for oversampling, which correspond to predicting 10 images: the cropped center as well as corners, and their mirrored versions. This typically improved our validation accuracy by around 1.5%. Indeed, for the best network described above, this additional step boosted our validation accuracy from 44.90 % to 46.4%.

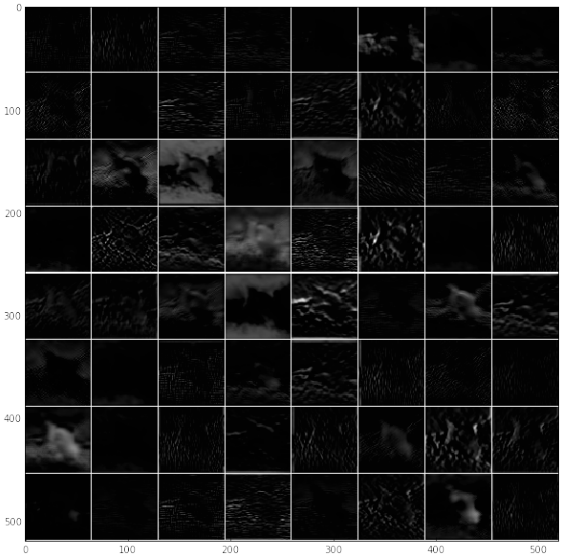

Figure 4. Feature maps from the first layer.

When preparing our ensemble model, we trained 11 similar networks with this architecture, and evaluated their validation accuracy using the oversampling above. We reused some of the code from assignment 3 to test all the $2^{11} - 1 = 2047$ possible combinations of models, and evaluated the validation accuracy of each of them. The plot of the performance of these models with respect to the number of models used is given in figure 5 (code from assignment 3 was reused here). As expected, we can boost our accuracy by a few more percents by averaging the values from several models. The best combination used 8 models out of 11 and achieved a validation accuracy of 49.06 %, which could imply that 3 of the models should be excluded from our ensemble. But the model ensemble made up of all 11 models achieved a validation accuracy of 49.02 %, which is almost equal, so we considered that it did not make sense to exclude the 3 aforementioned models. Thus, we used these 11 models for our final predictions.

In figure 6 we showed the test and validation accuracy achieved by our ensemble model, as well as a typical training accuracy of one single model (cf. caption for more information). We can see an interesting phenomenon: despite having a validation accuracy of 49.02 %, our ensemble model could only achieve an accuracy of 44.0 % on our testing images. The gap is considerable and cannot be just explained by statistical fluctuations. The usual explanations given for such a behavior are usually that we overfitted our hyperparameters when using the information from the validation accuracies, in this case these hyperparameters would be the ones that we fine-tuned: the learning rate and weight regularization. However, when training the 11 models that made up our ensemble (using different initialization and slightly different hyperparameters), all the validation

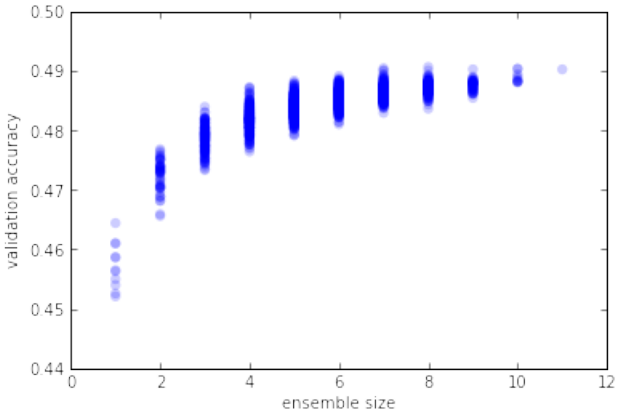

Figure 5. Validation accuracy for each possible combination of our models.

accuracies with oversampling had a low standard deviation, going from 45.21% to 46.44%, so we would expect these accuracies to be the same on the validation data, if the validation and test images have the same statistics. This was apparently not the case since the test accuracy of the ensemble model was of even lower than the validation accuracy of all the single models. To this time, we still do not have a convincing explanation for that, so we will have to accept the unconvincing explanation that we did overfit our hyperparameters.

One way to mitigate this problem would have been to use k-fold cross-validation, which would have confirmed if we really overfitted our data or if the test images have different statistics to the validation images. We unfortunately did not have time to implement that, so we had to stick with a final test accuracy of 44 %.

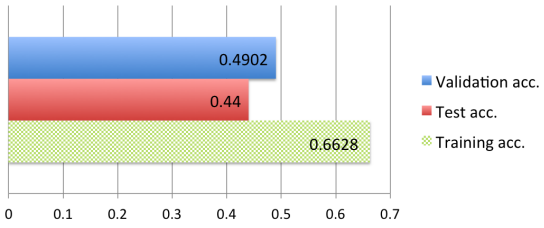

Figure 6. Accuracies obtained for our ensemble model. To be noted: the training accuracy does not correspond to the training accuracy of this model, but to the training accuracy of the best single model. It is just given as an order of magnitude of the typical training accuracy achieved by this architecture.

4.2. Dissection of our network

This subsection describes the result of our "convnet dissection" experiment. As described in section 3, we trained and fine-tuned many different architectures, going from a simple model to a higher and higher complexity.

Using the notations previously introduced, we show in figure 7 the different gains in validation accuracy. The "oversampling" step does not correspond to a different model to the previous one, which is why it was not listed previously, but it corresponds to a different way of evaluating the validation accuracy. Up to that point, all validation accuracies are evaluating using only one prediction per image, while the oversampling mode uses ten predictions per image, as explained in the previous subsection.

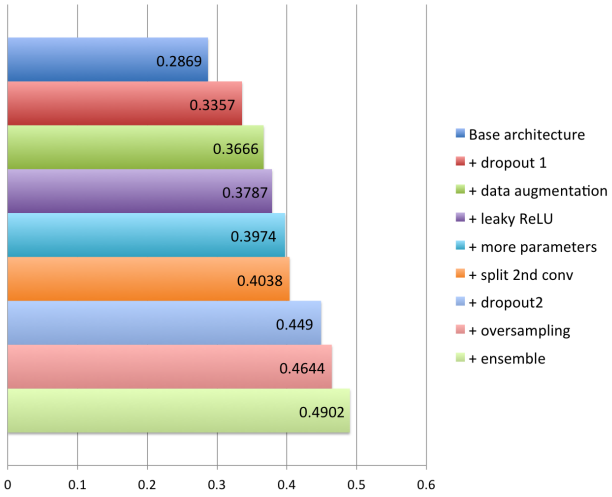

Figure 7. Validation accuracies obtained for each model as complexity is added (from top to bottom).

One caveat of our approach is that our results are noisy. Indeed, each of these bars correspond to the value that we get from fully training one model with a given set of hyperparameters. It is unknown if we would get similar values when training the same model with a different random initialization, andn if we had more time, a better experiment would have been to train a dozen instances of each type and to average the validation accuracies. Still, we can get an idea of the typical standard deviation by looking at all the models from our ensemble model. Indeed, for the ensemble model, we had to train 11 models with almost equal hyperparameters. These models had a standard deviation in the validation accuracy of 0.37%. This shows that even the values from one trained model still provide enough information to get some interesting insight for this specific architecture.

Another flaw in this approach is that we used the same validation set to tune our hyperparameters and evaluate the architecture, which means that we may have a bias that tends to overestimate these values. On the other hand, all the values are biased in a similar way, so the real gains we get from adding a component may not be that different from what we got here.

Because of these approximations, this experiment cannot be considered as completely accurate, but we can still

get very interesting insights by looking at the validation accuracy values.

As we can see from the plot, the two main gains occurred when we added the dropout units (4.88% and 4.52% respectively for the first and second dropout units). A hasty conclusion would be that these units by themselves contribute to some considerable gains. However, we can nuance it by looking at the theoretical effect of dropout on a network. As we saw, the main effect of dropout is that it prevents co-adaptation of features [3], and thus decreases overfitting. Knowing this, we can put the dropout units in context. We can see that in the case of the second dropout unit, we added it after splitting the second convolutional layer into three convolutional layers with smaller filters. That previous step increased the number of parameters (each 5×5 filter being replaced by three 3×3 filters) but only had a moderate effect on the validation accuracy. In fact, it turns out that the training accuracy went from 67.76% to 88.60% when we splitted this layer. That step considerably increased the capacity of the model, but also the overfitting. Thus, it made sense to add another dropout unit after that, so that we could decrease again the overfitting and collect the gains from our added capacity. This is exactly what happened. After adding this second dropout unit, the training accuracy went back down to a lower value: 66.28%, and meanwhile the validation accuracy went up. In the case of the first dropout, a similar phenomenon occured: the base model had enough parameters to have a relatively high capacity, but it was crippled by a large overfitting. Adding the first dropout unit helped to mitigate the problem.

Looking at the other large gains, we see that augmenting the data and averaging multiple models accounted for good improvements (resp. 3.09% and 2.58% increase in the validation accuracy). This is consistent with what we learned about how convnets work as well as what is actually done in practice in state-of-the-art papers. Going from a ReLU to a leaky ReLU gave an interesting 1.2% increase in our accuracy without much additional effort. Adjusting the slope could maybe even lead to more substantial gains.

Some other network architectures that we tried without much success include the following. We tried a version of the architecture where the second convolutional layer was split in two layers instead of three, but it seemed to be performing worse than the previous version. Looking back at it, it may be due to the fact that the overfitting increased, as it was the case for the 3-layer split (explained above). Another architecture tried to make use of multiple dropout units, as was done by Srivastava et al. [8]. This papers gives interesting guidelines when using many dropout units. In this case, it is indeed necessary to increase the learning rate, but care needs to be taken since a learning rate too large will lead to instabilities. It is thus important to normalize the weights in order to overcome this problem. Because of this complexity, we could not successfully train a network with many more dropout units.

5. Conclusion

It was interesting to learn how to train a large convolutional neural network from scratch and to get more insight on how its different components interact. By trying to get the best accuracy from a relatively simple architecture, we were forced to only consider the techniques that would not considerably increase our number of parameters. On the other hand, this was probably also one of the main reasons why we did not achieve higher accuracies in our final model, since our model capacity was limited.

The "dissection experiment" gave interesting results despite its known flaws, but studying its results also showed us something interesting about convnets: it is often impossible to evaluate the effect of a given technique by comparing the validation accuracy with or without it, because it may only work with other specific components. One example was shown for the dropout units: they gave good improvements because they were applied when we had an architecture with a large capacity, but also a large overfitting. The gains would have probably be smaller if we had not been in such a situation. Still, our belief in the usefulness of these units has been confirmed by our experiments, but it is important to know when their efficiency can be maximized.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[4] 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.

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

[6] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

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

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[9] 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.