Tiny ImageNet Visual Recognition Challenge

Hadi Pouransari
hadip@stanford.edu

Saman Ghili
samang@stanford.edu

1. Abstract

In this work, we use Convolutional Neural Networks (CNNs) trained on GPUs for classifying images in the tiny ImageNet dataset. Specifically, we are pursuing two different goals. First, we try to train a relatively deep network with a large number of filters per convolutional layer to achieve a high accuracy on the test dataset. Second, we train another classifier that is slightly shallower and has fewer number of parameters several times, to build a dataset that will allow us to perform a thorough study of ensemble techniques. We introduce several notions, namely, "model confidence", "class-specific accuracy", and "prediction frequency". We use these quantities to study various ensemble methods, and provide some insight into the behavior of the CNN classifier for certain input images.

2. Introduction

In this work we use convolutional neural networks for image classification on the Tiny ImageNet dataset. Tiny ImageNet challenge is very similar to the original ImageNet challenge. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a well-known image classification and localization benchmark for large scale datasets [1]. Currently, the organizers provide a labeled dataset with 1000 categories for image classification. The training set contains over 1,200,000 images of the average image resolution of 256×256 pixels. There is also a dataset of 150,000 labeled images for validation and test. The state of the art implementations have a classification error bellow $\%5$. Figure 1 (taken from [1]) shows the performance of winning entries in the original ImageNet Challenge for the past 5 years.

The tiny ImageNet has a smaller number of classes: 200 instead of 1000. Each class has 500 training images (a total of 100,000), 50 validation images (a total of 10,000), and 50 test images (a total of 10,000). In the tiny ImageNet dataset, each image has a resolution of 64×64 pixels. The metric for evaluation of classifiers is the fraction of test images that are correctly classified. Notice that here, unlike the original ImageNet challenge, we confine ourselves to the classification task, *i.e.*, we do not perform the localization task.

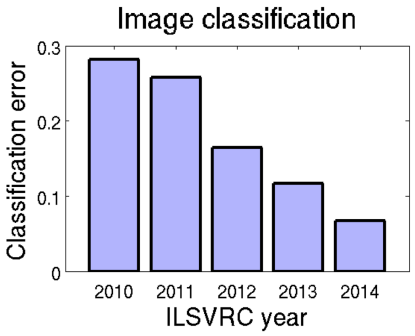

Figure 1: Performance of winning entries in the ILSVRC, years 2010-2014 competitions.

Although the tiny imageNet challenge is simpler than the original challenge, it is more challenging than some similar image classification benchmarks, such as CIFAR-10 [2].

Although Convolutional Neural Networks have been around for many years [3], until very recently, they were not commonly in use due to prohibitive cost of training. However, in the past few years using highly efficient parallel implementations, deep convolutional networks have been trained for relatively large datasets, including the ImageNet dataset [4]. For training the CNNs in this work, we use "Caffe", a publicly available parallel implementation of CNNs developed by the Berkeley Vision and Learning Center [5].

This manuscript is organized as follows. In section 3, we briefly describe the dataset, the algorithm, and the software that we use for training. In section 4, we present and discuss various network architectures that we have experimented with. In section 5, we present the results for some of these networks. We devote section 6 to the study of ensemble results from several networks, and section 7 to the performance of the CNNs for different specific classes. Finally, we conclude this work in section 8.

3. Dataset and implementation

In this section we describe the Caffe package and the tiny ImageNet data preprocessing, and the optimization al-

gorithm in more details.

3.1. Caffe

Caffe is a deep learning framework developed at UC Berkeley, and is in active development by the Berkeley Vision and Learning Center (BVLC). Caffe takes advantage of several software packages, including CUDA, BLAS, OpenCV, and boost in order to provide a high performance framework for neural network implementations. In particular, we use NVIDIA cuDNN in order to speed up our ConvNet model training [5].

3.2. Dataset and preprocessing

As we said earlier, we have 100,000 training images in the tiny ImageNet dataset. The number of classes is 200, and there is the same number of examples for each class in the dataset (50). The images are given in the ".jpg" format. We convert the JPEG files to Level-MDB files using the `convert_imagest` routine in caffe. In the LMDB format, the training data will occupy 1.6GB, while the validation and test data have sizes 160MB each.

3.3. Optimization algorithm

The training is done by minimizing an L_2-regularized softmax loss function (the regularization will be discussed in more details later). There are several algorithms that we can try for minimizing this loss function (denoted by L). All of these methods are based on stochastic gradient descent (SGD). The plain SGD updates the parameters by moving in the opposite direction of the gradient: $W \leftarrow W - \eta \nabla_W L$, where the learning rate η is a hyper-parameter. Momentum update rules simulate the motion of a particle in a force filed given by the gradient of L with a friction coefficient μ. In these methods, we first compute the velocity, and then update the variables based on that velocity. These methods include the momentum and Nesterov Momentum update rules. We use the momentum update rule in Caffe.

4. Network design

In this section, we describe the networks that we have used for image classification in more details.

4.1. Network architectures

We trained four convolutional networks (each trained several times with different initializations) that vary in both depth and the number of filters in each layer. These networks, called N1, N2, N3, and N4, are detailed in Table 1. Network N1 is the first and shallowest network that we use. It has only three convolutional layers and one fully connected one. Network N2 has one more convolutional and one more fully connected layers and achieves a better accuracy. Network N3 is the network that we used to generate data for the study of ensemble methods (see section 6). It is considerably deeper than N1 and N2, with 6 convolutional layers and two fully connected ones. We wanted this network to be deep enough to give us good accuracies, and at the same time we wanted it to be easy and fast to train since we need several trained versions of it for the ensemble study. Therefore we used a relatively small number of filters in the convolutional layers to make the training process faster. Network N4 is the network that we used to achieve the highest accuracy. This is the deepest network (with the highest number of filters) that we trained. Compared to N3, it has more and larger filters, an extra fully connected layer, and the pooling layers are in different places. A schematic of this network is depicted in Figure 2.

4.2. Nonlinearities

There are several nonlinear functions that are used at the output of convolutional layers in neural networks. One family of these nonlinearities consists of the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\tanh(x)$, which is very similar to the sigmoid function, but maps to $[-1, 1]$ instead of $[0, 1]$. We experimented with a few nonlinearities. The most commonly used function in this family is the rectified linear, or ReLU function which returns x if x is positive, and zero otherwise, *i.e.*, $\text{ReLU}(x) = x\mathbf{1}\{x\}$. It is similar to [4]. We also trird the leaky ReLU, which is like ReLU when $x > 0$, and instead of zero, returns αx with some $\alpha = 0.02$ when $x \leq 0$. This did not make a noticeable difference. We also experimented with the $\tanh$ function. In order to avoid the saturation problem, we first trained the network using ReLU nonlinearity for about 150 epochs, and then continued the training using $\tanh$ as the non-linearity for another more 150 epochs. This also did not result in significant improvement.

4.3. Regularization

When the network is deep and has many parameters, we will have to regularize the loss function in order to mitigate the overfitting phenomenon. There are several ways of regularizing the loss function. In the L_2-norm regularization, we add a term $\lambda \|W\|_2^2$ to the loss for each layer, where λ is a hyper-parameter determining the regularization strength and $\|W\|_2$ is the L_2-norm of the weights for that layer. The values that we used for λ were in the range 0.0005-0.01 for the various networks that we trained. Another method that can be used in conjunction with the previously mentioned regularization is the dropout regularization, where we keep a neuron active with a prescribed probability p, and use p to find the expected activation of the layer. As it can be seen from Table 1, we use a dropout layer (with $p = 0.5$) after every fully connected layer (except the very last one) in all of our networks.

Name	Architecture	Best test acc.
N1	IMG → ConvReLU(F5-16) → MaxPool → ConvReLU(F3-16) → MaxPool → ConvReLU(F3-32) → FC(200)	%25.5
N2	IMG → ConvReLU(F5-32) → ConvReLU(F3-32) → MaxPool → ConvReLU(F3-64) → MaxPool → ConvReLU(F3-128) → MaxPool → FCReLU(256) → DropOut(p=0.5) → FC(200)	%33.3
N3	IMG → ConvReLU(F5-16) → MaxPool → ConvReLU(F3-16) → ConvReLU(F3-32) → ConvReLU(F3-32) → MaxPool → ConvReLU(F3-64) → ConvReLU(F3-128) → MaxPool → FCReLU(256) → DropOut(p=0.5) → FC(200)	%37.0
N4	IMG → ConvReLU(F5-32) → ConvReLU(F5-64) → MaxPool → ConvReLU(F3-64) → ConvReLU(F3-64) → ConvReLU(F3-64) → MaxPool → ConvReLU(F3-128) → FCReLU(256) DropOut(p=0.5) → FCReLU(256) → DropOut(p=0.5) → FC(200)	%42.1

Table 1: Our four networks and their accuracies. The number of parameters are given for each layer. For example, FC(200) means a fully connected layer with 200 neurons, while ConvReLU(FX-Y) is a conv-ReLU layer Y filters of size X.

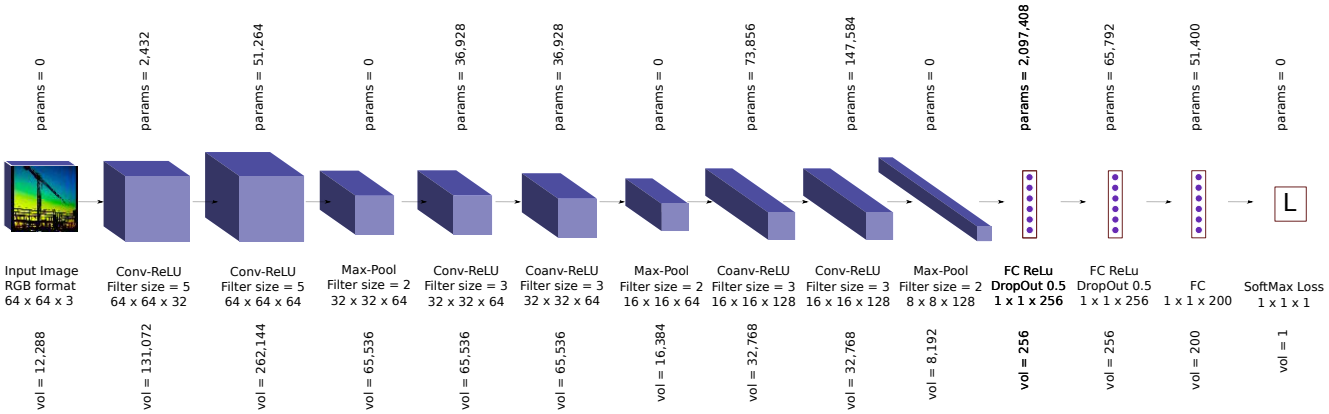

Figure 2: Schematic of the network N4.

4.4. Data augmentation

Data augmentation is in fact another way of regularizing the convolutional neural network. This can be achieved in several ways, including cropping the input images before training, mirroring them (flipping), tinting, etc. We used cropping and mirroring in training all of our networks, which seem to be extremely helpful when it comes to increasing the test accuracies. Specifically, we trained N1 and N2 without data augmentation, which lead to lower accuracies, namely, $\%17.3$ and $\%21.4$, respectively.

5. Analysis of the results for the network N4

In this section we discuss the performance of our deepest network, N4.

5.1. Training

We trained N4 using two different weight initializations: "Gaussian" and "Xavier". For both cases, we used a regularization strength of $\lambda = 0.01$, and ran the training optimizations for 60,000 iterations (over 150 epochs). The learning rate was initialized to 0.005, and multiplied by 0.85 every 5000 iterations. The μ parameter in the momentum method was set to 0.9. These parameters were chosen via cross-validation. The evolution of training and validation accuracies and losses are shown in Figures 3 and 4, respectively. The two initializations lead to very similar results, both in terms of the evolution of accuracies and losses, and the final accuracies, namely $\%42.1$ for Gaussian and $\%41.8$ for Xavier.

5.2. Visualization

Figure 5 shows the filters in the first convolutional layer of N4, trained from the Xavier initial parameters. We see a variety of filters that are useful in detecting coarse features of input images. Some of these filters are mainly color blobs and are used by the network to predict the class of images that can be easily classified by their colors, and some are more suited for detecting edges, coarse shapes, etc. Figure 6 shows the response of these filters for an image (shown

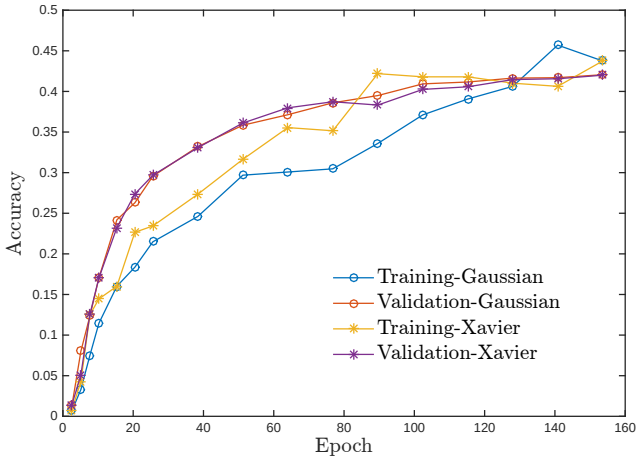

Figure 3: Time evolution of accuracies for the network N4, for two different initializations of the weights.

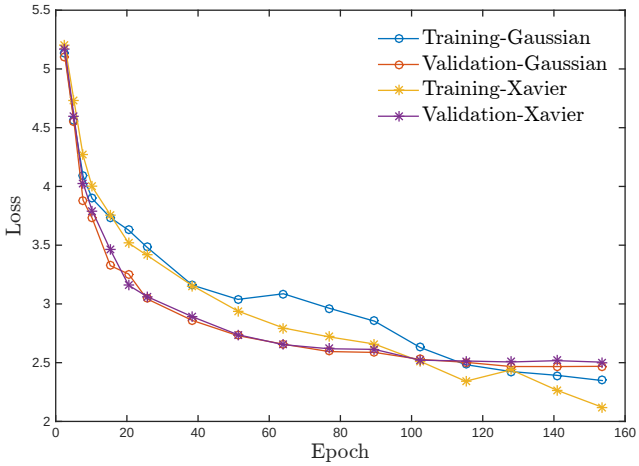

Figure 4: Time evolution of losses for the network N4, for two different initializations of the weights.

in Figure 7) of the class "monarch" (which is classified correctly by this network).

5.3. Easiest and hardest classes to predict

We can use the validation images to measure the accuracy of our trained CNN on different classes separately: we have 50 images in each class, and we can compute the fraction of the images in each class that are correctly classified by the CNN. The classifier N4-Xavier (N4 initialized by Xavier) has the highest accuracy for the class "goldfish" (%90), and the lowest accuracy for the class "chain" (%2). A few sample images from each of these two classes are shown in Figures 8 and 9, respectively. These figures suggest that there are two properties that make a specific class easier or harder to classify by the CNN. First, it seems that

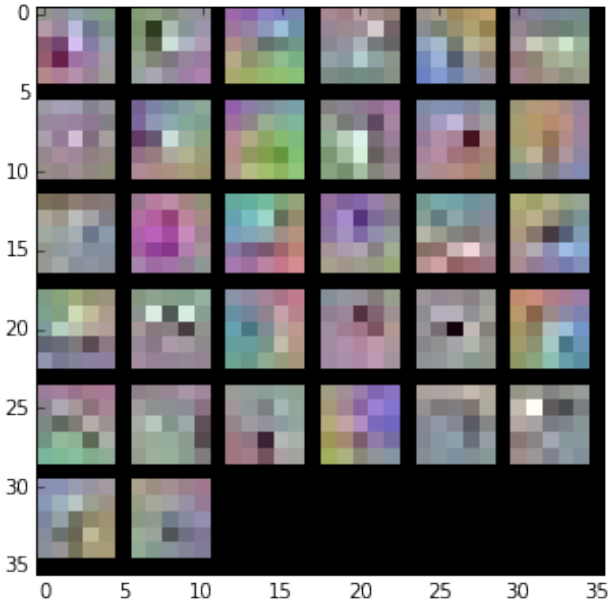

Figure 5: First layer filters for N4.

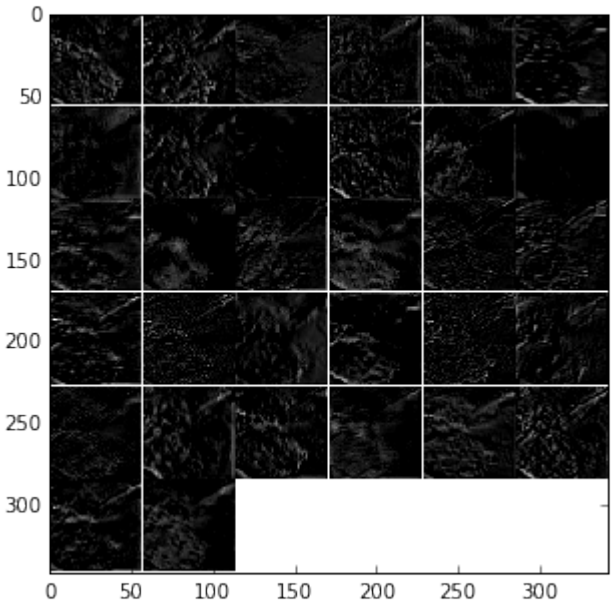

Figure 6: First layer outputs for the image shown in Figure 7.

the classifier relies heavily on the color composition of the images: almost all the goldfish are gold (or reddish gold), while the chains come in a variety of colors. Second, in almost all the goldfish images, the fish takes up a large part of the image, while in many of the chain images, the chain ap-

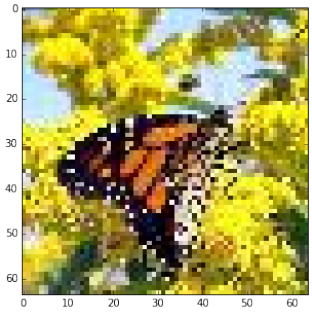

Figure 7: A sample image from the class “monarch”.

Figure 8: Nine sample images from the class “goldfish”.

pears only in a small region of the image. Since we are not performing localization here, this could drastically reduce the accuracy of the classifier when it comes to this particular class. We will discuss the class specific accuracy of a CNN classier in more details in section 7.

5.4. Ensemble averaging

For an image from the class “sandal” (Figure 10), the class probabilities predicted by the softmax layers of N4-Xavier and N4-Gaussian, and their ensemble average are shown in Figure 11. As it can be seen from the plots, N4-Xavier classifies this image as “barrel”, with a probability that is only slightly higher than that of “sandal”. Similarly, N4-Gaussian classifies this image as “sock”, with a probability that is only slightly higher than that of “sandal”. We also see that N4-Xavier and N4-Gaussian predict relatively small probabilities for “barrel” and “sock”, respectively. In

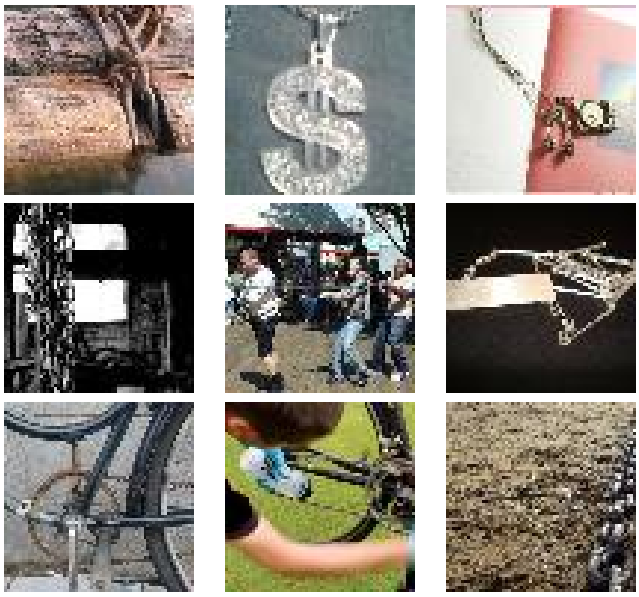

Figure 9: Nine sample images from the class “chain”.

the ensemble average of the probabilities predicted by these two classifiers however, the probability of “sandal” is higher than both “barrel” and “sock”. Therefore, even though both N4-Xavier and N4-Gaussian predict the wrong class for this image, the ensemble average of them predicts the right class. By ensemble averaging, we get the accuracy of $\%\mathbf{43.2}$, which is higher than what we get from N4-Xavier and N4-Gaussian separately. A more thorough study of ensemble methods (including averaging) is done in the next section.

Figure 10: Sample image from the class “sandal”.

6. Ensemble techniques

In this section, we study various methods to combine different models in order to obtain a model with higher accuracy. In order to obtain a set of classifiers and study different ensemble averaging methods, we trained one of

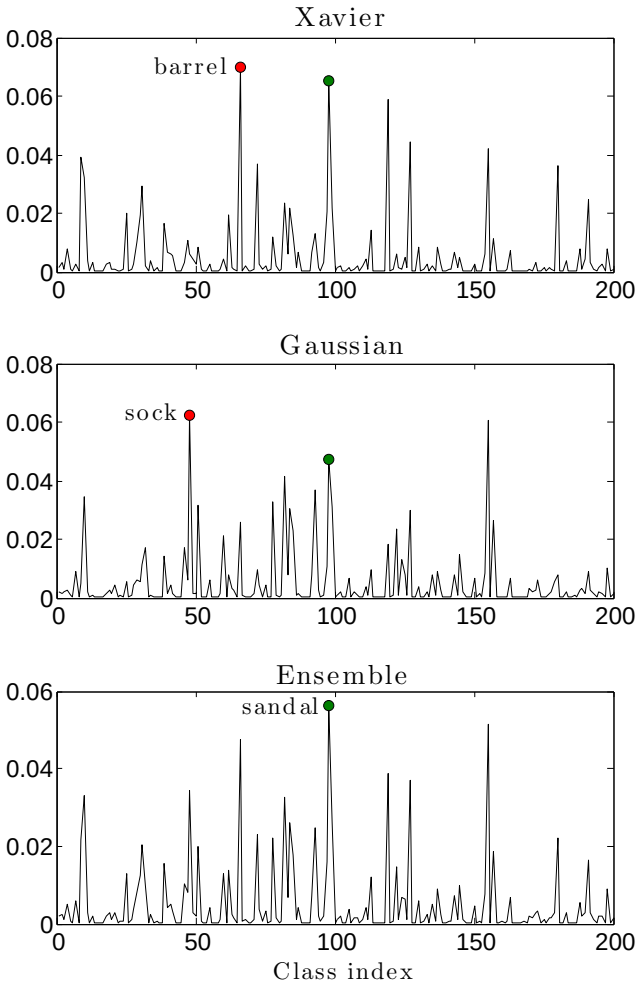

Figure 11: Predicted class probabilities by N4-Xavier, N4-Gaussian, and their ensemble average for the image shown in Figure 10.

our networks (network N3 in table 1) 9 times using random initial conditions. Note that even though the networks have the same architectures and are trained with the same hyper-parameters, the final models are different due to different initial conditions, and the randomness associated to the SGD optimization. Note that the purpose of this section is only to investigate different ensemble averaging methods, including some novel techniques, and not obtaining super high accuracies. Therefore, each model is trained for only a few number of iterations (about 50 epochs).

6.1. Simple averaging

Suppose we are given n models, $M_1, M_2, \ldots, M_n$, and we want to form a combined model (in our study $n = 9$). Also, assume there are a total of m classes (here, $m = 200$), and N validation images (here, $N = 10,000$).

For a given image, x and a model M_i, $\mathbb{P}(x = y_j; M_i)$ represents the probability that the input image x is classified as y_j ($j = 1, 2, \ldots, m$) under the model M_i. The simple ensemble average is then:

$$\mathbb{P}(x = y_j; \{M_1, \ldots, M_n\}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{P}(x = y_j; M_i)$$

In Figure 12 we demonstrate the ensemble model validation

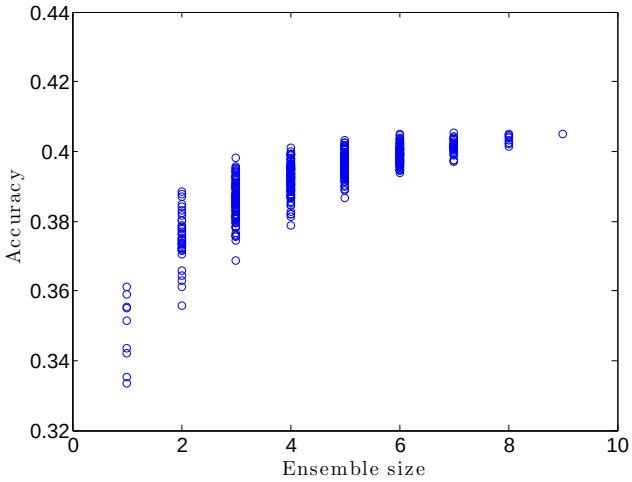

Figure 12: Validation accuracy of the ensemble model using different number (and combination) of models.

accuracy, when k different models are combined. For each k there are $\binom{9}{k}$ possible combinations of models. Each point on the plot shows the accuracy of one combination. The ensemble model with $k = 9$ has 40.48% accuracy, whereas the average accuracy of our 9 classifiers is 34.84%.

6.2. Weighted averaging

The simple ensemble averaging is often results in a combined model with better accuracy. However, we can incorporate some of the available information about each model in the averaging process. Let's assume r_i is the accuracy of the model M_i on the validation set, defined as follows:

$$r_i = \frac{\text{\# of correctly classified images by } M_i}{N}$$

Hence, for each image we have n models (experts), with different accuracies (reliabilities). A natural extension to the simple ensemble averaging is to use accuracies as weights as follows:

$$\mathbb{P}(x = y_j; \{M_1, \ldots, M_n\}) = \frac{1}{\sum_{i=1}^{n} r_i} \sum_{i=1}^{n} r_i \mathbb{P}(x = y_j; M_i)$$

The weighted ensemble averaging process is discussed in [6], and higher accuracies using the above method in comparison to the simple averaging is presented. If the model

accuracies are very close to each other, the weighted and simple ensemble averaging results are very close. The weighted ensemble model in our case has 40.53% accuracy, which is slightly higher than the simple ensemble.

6.3. Model confidence

We go one step further, and introduce the concept of *confidence* for each model. The notion of reliability only shows how accurate each model is in an average sense. However, looking at the softmax probability distributions (Figure 11) we observe that a model is sometimes very confident in its decision (*i.e.*, there is only one high probability class), and sometimes not (*i.e.*, many classes have got high probabilities). Let's define $\texttt{conf}(x; M_i)$ to be the confidence of model M_i in classification of image x. This information can be incorporated in the ensemble process using one of the following methods:

Method 1: The combined method for each image goes with the decision of the most confident model. Assume $f_i(x) = \text{argmax}_j \mathbb{P}(x = y_j; M_i)$ is the classification result of the model M_i on the image x. The ensemble prediction, $f(x)$, is then defined as:

$$f(x) = f_{i^*}(x) \quad \text{, where } i^* = \text{argmax}_i \texttt{conf}(x; M_i)$$

i.e., believe the most confident expert.

Method 2: The combined method uses model confidences as weights in the averaging of probability distribution. *I.e.*, the ensemble probability distribution over different classes is as follows:

$$\mathbb{P}(x = y_j; \{M_1, \ldots, M_n\}) = \frac{1}{\sum_{i=1}^n w_i} \sum_{i=1}^n w_i \mathbb{P}(x = y_j; M_i),$$

where $w_i = r_i \texttt{conf}(x; M_i)$. Note that we have also incorporated the reliability effect in the definition of weights.

The notion of confidence can be quantified in different ways. Here, we introduce some possible definitions:

Confidence based on variance: one can quantify the confidence of a model M_i in classification of an image x based on the variance of the output probability distribution, $\mathbb{P}(x = y_j; M_i)$, such that more variance means less confidence. *I.e.*,:

$$\texttt{conf}_1(x; M_i) = \frac{1}{\sum_{j=1}^m \mathbb{P}(x = y_j; M_i)^2}$$

Confidence based on the maximum probability: a very simple way to quantify confidence for a given probability distribution is to to look at the maximum class probability. *I.e.*, the probability of the predicted class:

$$\texttt{conf}_2(x; M_i) = \max_j \mathbb{P}(x = y_j; M_i)$$

Method	**Val. acc.**	**Method**	**Val. acc.**
simple	40.48%	weighted	40.53%
method1 with $\texttt{conf}_1$	37.07%	method2 with $\texttt{conf}_1$	39.94%
method1 with $\texttt{conf}_2$	37.72%	method2 with $\texttt{conf}_2$	40.1%
method1 with $\texttt{conf}_3$	37.7%	method2 with $\texttt{conf}_3$	40.01%

Table 2: Validation accuracies for different ensemble methods.

Confidence based on likelihood: The above definition of the confidence can be improved by incorporating the probabilities of all classes. For a given image and model (x, M_i), suppose $\mathcal{I}(j)$ is the index of the j'th most probable class, *i.e.*,:

$$\mathbb{P}(x = y_{\mathcal{I}(1)}; M_i) \geq \ldots \geq \mathbb{P}(x = y_{\mathcal{I}(m)}; M_i).$$

Then, another way of defining the confidence is as follows:

$$\texttt{conf}_3(x; M_i) = p_1(1 - p_2)(1 - p_3)\ldots(1 - p_m),$$

where $p_j = \mathbb{P}(x = y_{\mathcal{I}(j)}; M_i)$.

The validation accuracies of the ensemble models using the aforementioned methods are compared in Table 2.

7. Class-specific accuracy

In this section, using the models introduced in section 6 we analyze the accuracy of the models on different classes. We define the class-specific accuracy of a model M_i for class j as follows:

$$\mathcal{A}_j(M_i) = \frac{\text{\# of correctly classified images in class } j \text{ by } M_i}{\text{\# of images in class } j}.$$

In the tiny ImageNet dataset, there are 50 images per class in the validation set. In Figure 13 the statistics of class-specific accuracies are illustrated. In particular, we show the average (μ_j), maximum (H_j), and minimum (L_j) class-specific accuracies for different classes, where:

$$\mu_j = \frac{1}{n} \sum_{i=1}^n \mathcal{A}_j(M_i)$$

$$H_j = \max_i \mathcal{A}_j(M_i) \quad , \quad L_j = \min_i \mathcal{A}_j(M_i)$$

The classes in Figure 13 are sorted by the mean values, μ_j. The gap between H_j and L_j demonstrates the fact that different models have different accuracies for a given class. Basically, by ensemble averaging we take advantage of this fact. The more important message of Figure 13 is the fact

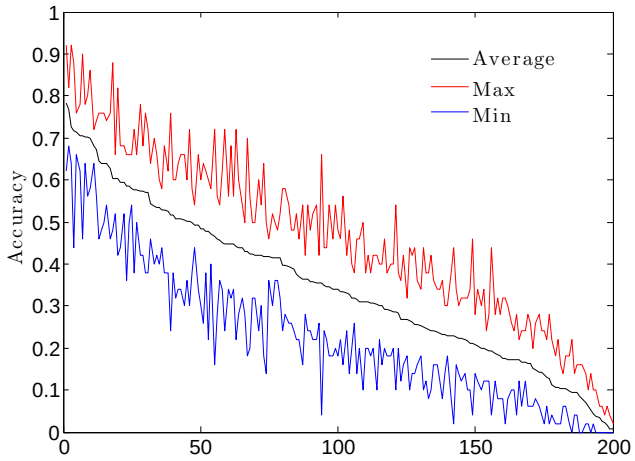

Figure 13: Class-specific accuracy sorted by mean accuracies.

that the average class-specific accuracy of all models is very poor for some classes, and very good for some other ones. *I.e.*, there are some classes that all models often misclassify. It means, features of those classes are not captured properly by any of the models. This issue can be solved by training a more complex network, for many number of epochs. In concert with our previous observation, Figure 14 demonstrates the average (over different models) number of times each class is predicted (prediction frequency of the class). For an ideal classifier, each class would be predicted exactly 50 times. Note that the classes in Figure 14 have the same order as in Figure 13. The descending trend in Figure 14 shows that those classes that are more accurately classified, are in fact predicted more frequently (i.e., more than the exact value of 50). Similarly, the classes with small class-specific accuracies are predicted less frequently. We also investigate the relationship between confidence (introduced in section 6.3) and accuracy. Essentially, we want to see if there is any correlation between being confident and being accurate for classification. This is shown in Figure 15. The horizontal axis is class-specific accuracy, $\mathcal{A}_j(M_i)$, and the vertical axis is the averaged confidence (based on likelihood), defined as follows:

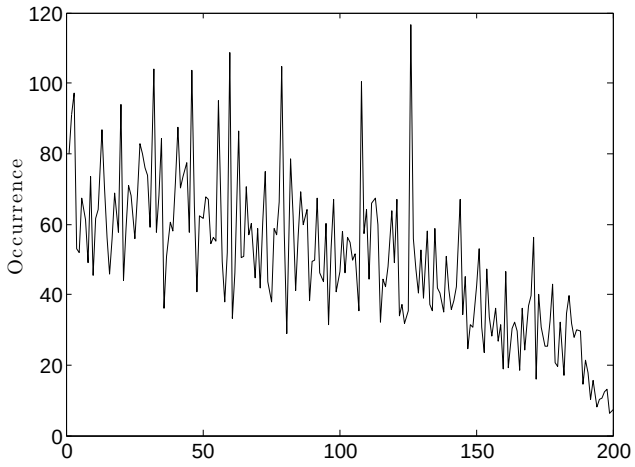

Figure 14: Averaged prediction frequency of each class. Classes are sorted by their average class-specific accuracy.

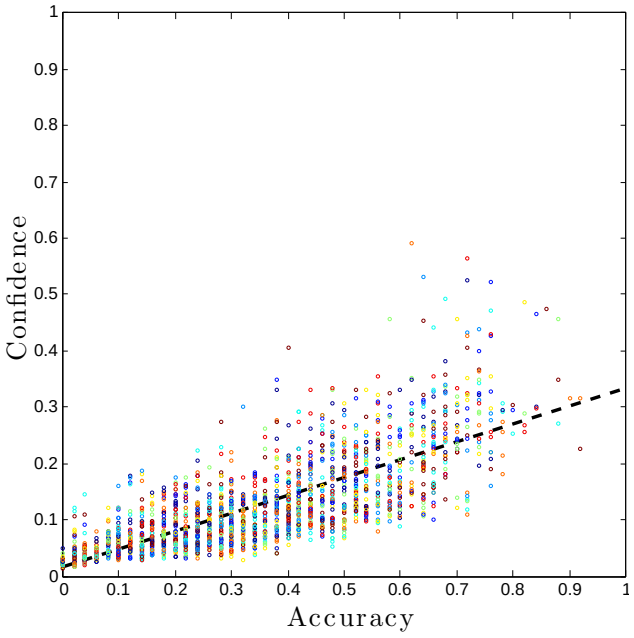

Figure 15: Averaged confidence vs. class-specific accuracy. The dashed line is the linear regression.

$$\texttt{conf}^j(M_i) = \frac{1}{\sum_{x:f_i(x)=j} 1} \sum_{x:f_i(x)=j} \texttt{conf}_3(x; M_i).$$

Therefore, there are $n \times m$ points in total. Points are colored by their associated models (*i.e.*, $i = 1, 2, \ldots, n$). As is evident from the plot, averaged confidence and class-specific accuracy are highly correlated (the correlation coefficient = 0.754). This means that in average, when a classifier is confident (i.e., the output softmax probabilities are close to a delta function) it is also accurate.

8. Conclusion

In this work, we used convolutional neural networks for performing image classification on the tiny ImageNet dataset. We trained several networks (called N1-N4), increasing in depth and number parameters for this task. By training the networks N1 and N2 both with and without data augmentation, we found out that the accuracy significantly

improves when we do cropping, mirroring, etc. We trained the network N4 with two different parameter initializations and ensemble averaging to achieve our highest test accuracy ($\%43.2$). We also trained the network N3 with 9 different initializations and used the results to perform a thorough study on ensemble methods.

We introduced and investigated various ensemble methods. The weighted ensemble averaging method, for particular test case, outperforms the other methods. In our study of ensemble techniques, we introduced and investigated several statistical concepts, namely, model confidence, class-specific accuracy, and prediction frequency. Using our case study results, we showed that some classes are accurately predicted by all models, whereas some other classes are hard to classify by every model. Moreover, we showed that classes that are easier to predict by the models, have high prediction frequency as well. Also, we demonstrated a strong correlation between class-specific accuracy and average model confidence.

References

[1] Leslie Lamport, Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." Computer Science Department, University of Toronto, Tech. Rep 1.4 (2009): 7.

[2] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge."arXiv preprint arXiv:1409.0575 (2014).

[3] Fukushima, Kunihiko. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position."Biological cybernetics 36.4 (1980): 193-202.

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

[5] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the ACM International Conference on Multimedia. ACM, 2014.

[6] Frazo, Xavier, and Lus A. Alexandre. "Weighted Convolutional Neural Network Ensemble." Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Springer International Publishing, 2014. 674-681.