Optimizing AlexNet on Tiny ImageNet10

Talal Alqadi Ella Lucas

June 19, 2020

Abstract

Image classification of Tiny ImageNet data by reproducing the state of the art convolutional neural network built by Alex Krizhevsky et al. called AlexNet [2]. Tiny ImageNet is a subset of ImageNet containing 200 classes, each with 500 images. After reproducing the model, we constricted the Tiny ImageNet dataset even further into the first 10 labels to speed up the model training and testing, to see how the model can perform with less labels but same amount of data. Our goal is for AlexNet to correctly classify even more test images with our new restricted dataset, as compared to its performances with Tiny ImageNet. Using Pytorch, a pipeline was built to train and validate different inputs into the AlexNet model to single out the best model parameters, and then finally create visualizations to compare accuracies between labels.

1 Introduction

As machines get stronger and as more data is available every year, deep learning has been able to accomplish huge feats. It has been able to overcome extremely complicated problems with many of these deep learning models' accuracies continuously increasing the more time passes. One such problem that deep learning had a huge hand in solving is in computer vision. The main topic of computer vision in this case, object recognition, has become pivotal by replicating human intelligence with significant applications in research, politics, defense, and industry. Convolutional Neural Networks (CNNs) are the architecture traditionally optimized for image recognition. At a high level, these networks are fed an input image and return either an output class or a probability that the input belongs to a certain class. There has been much research invested in the development of CNNs and many architectural variations including AlexNet, VGG, and ResNet, to name a few.

In this project we have created our own implementation of AlexNet (from scratch) as a solution for the Tiny ImageNet classification task. We are interested in developing a rich understanding on how we can refine existing deep convolutional networks to optimize for image recognition in different cases. The ImageNet challenge has quickly become the standard benchmark in object recognition and identification [1]. To precede any struggles with performance issues and runtimes, we constricted the Tiny ImageNet to contain only 10 labels of the 200 provided. This allows us to run many different trials, at many different value epochs and batch size, for final comparisons. All in all, the new dataset created called 'tinyimage10' contains 10 labels, 500 images for each of them. The data has been split into a training, validation, and test set with a ratio of 0.8:0.1:0.1. Sample pictures can be seen in Figures 2a and 2b. The results were what we were expecting as AlexNet performed significantly better with less labels, but the same amount of data. After a certain number of epochs, overfitting seemed to be a significant issue throughout all different parameter inputs.

2 Method Description

Since CNN's are the best present option for solving the image classification task, we explored successful variations in our research. The core components of a CNN include convolutional layers, activation layers, pooling layers, and a fully connected layer. Depending on the implementation, these components may be varied. AlexNet's original implementation consists of five convolutional layers and three fully connected layers (with Dropout), as well as overlapping max pooling layers, followed by ReLU nonlinearity (Krizhevsky et al). We build our instance of AlexNet with the same overall architecture. Changes to the size and dimensionality were made in order to accommodate differences between ImageNet and Tiny ImageNet data (ie: Tiny ImageNet contains images of size 64 x 64).

We considered implementing ResNet due to its tendency to obtain superior accuracy on test data but decided against it due to slower convergence during training. Overall, AlexNet met our goal of challenging the limits of basic CNNs while still training fast enough that we could run multiple experiments on our data.

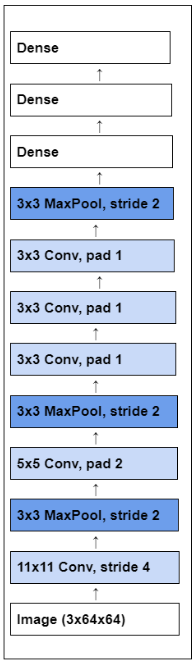


Figure 1: Model Architecture

2.1 Architecture and Activation

AlexNet is an eight layer CNN. Figure 1 exhibits the convolutional and pooling layers, as well as the three dense (fully-connected) layers that the input image passes through. One of the ways in which AlexNet distinguished itself from traditional CNN architectures was through its use of overlapping pooling. Pooling traditionally involved collecting outputs of neighboring neurons directly, with no overlap. Overlap was found to reduce the error rate and prevent overfitting [2].

AlexNet utilizes the ReLU activation function, which is a piecewise linear function that will return the input value if the summed weighted value is positive, otherwise it will output zero. This choice of activation function allowed CNNs to overcome the vanishing gradient problem characteristic in training. ReLU has also been found to reduce training time in comparison to other common choices such as the hyperbolic tangent function [2]. ReLU is applied after each convolutional and fully connected layer.

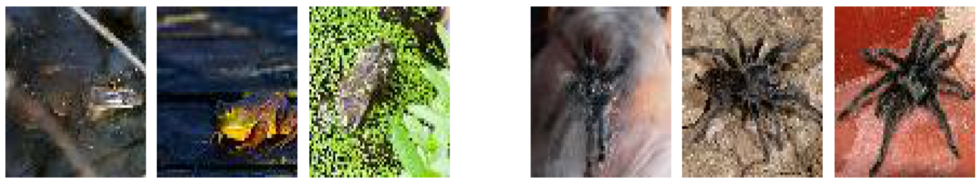
Dropout is applied before the first and the second fully connected layers to achieve regularization and reduce overfitting.

No normalization layers are applied because the original paper found that normalization did not result in higher accuracy on test data with AlexNet.

3 Experiment

3.1 Data and Preprocessing

Tiny ImageNet is a subset of ImageNet containing 200 classes with 500 training images, 50 of them validation images, and 50 of them test images. The dataset focuses on visual object recognition across thousands of labels, and more than 500 hundred images per label. Each class represents a different category, spanning across the animal kingdom, nature, as well as various physical objects. All images are colored and contain 64x64 pixels. Our dataset 'tinyimage10' is an even smaller subset of the Tiny ImageNet containing only the first 10 labels, with 500 images each. Similar to Tiny ImageNet, 'tinyimage10' contains 400 training images, 50 validation images, and 50 test images. Sample images from 2 of the 10 labels can be seen in Figure 2. In terms of preprocessing, the images were resized, cropped, and normalized. To normalize the images, the mean and start deviation of the pixels RGB data is used as provided by Krizhevsky et al (mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]).



(a) Bullfrog (b) Tarantula

Figure 2: Three Sample Images from 2 Labels

3.2 Hyper-Parameters and Metrics

3.2.1 Cross-Entropy (Log Loss)

Loss functions iteratively evaluate the performance of classification models that output a probability between 0 and 1. Cross-entropy loss increases when the model incorrectly classifies an input. As the probability output approaches 1, the log loss function decreases. When the probability decreases, the log loss increases rapidly, penalizing the model for incorrect classification.

Cross entropy is an effective loss function to use when the model's architecture incorporates activation functions in the output layer that model probability distributions. AlexNet's output from the last fully-connected layer is fed to a softmax which produces a distribution over the 10 class labels.

3.2.2 Optimizers

Stochastic gradient descent (SGD) is an iterative optimization method that has achieved success in decreasing the training time in high-dimensions by inducing randomness in the gradient calculation process. Our implementation of AlexNet was able to demonstrate consistent learning when tested against the Adam optimization algorithm during our initial experiments (see Appendix 5.3). We ran SGD with a learning rate of 0.001, weight decay, and momentum of 0.9.

The Adam optimization algorithm is a popular substitute for SGD in many settings. Adam has demonstrated success on non-convex optimization problems. It differentiates itself from SGD by combining two extensions of the latter during optimization: Adaptive Gradient Algorithm (AdaGrad) and

Root Mean Square Propagation (RMSProp).

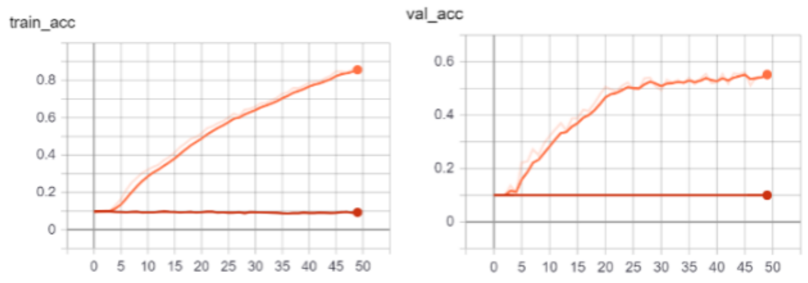


Figure 3: Comparing Accuracy using SGD (orange) vs. Adam (red)

As can be seen in Figure 3, the Adam optimizer performed extremely poorly when comparing to the stochastic gradient descent optimizer, which was initially surprising because it is essentially an extension of SGD. The model trained with a learning rate of 0.001 (omitting weight decay and momentum) using the Adam optimizer was spitting out the same label for every image, even 40 epochs in. A potential reason for AlexNet's inability to learn under Adam comes from the details of its original publication, where it is stated that weight decay acts not only as a regularizer in this setting, but is also essential for learning by reducing the model's training error [2].

3.2.3 Learning Rate, Momentum, and Weight Decay

The learning rate is a tuning parameter that exists within the optimization algorithm which determines the step size at each iteration during gradient descent (ie: while minimizing the loss function). During our initial experimentation, we tested with the standard learning rate of 0.001, and compared it with a learning rate of 0.005. During this experiment, the amount of decay and momentum were held constant. There was not a significant difference between varying the learning rate by this amount.

3.2.4 Batch Size and Epochs

Mini-batching is a popular technique that allows optimization to occur over a specified subset of the overall dataset. Once all of the data in a single mini-batch is processed once, the next mini-batch enters the neural network. Mini-batching allows for frequent updates to the network parameters as well as faster overall computations since the amount of data processes at a time is flexible.

The number of epochs is another flexible parameter that simply establishes the number of complete presentations of the data set to be learned to the network. In deep learning, it is common to need a large number of epochs for learning to occur.

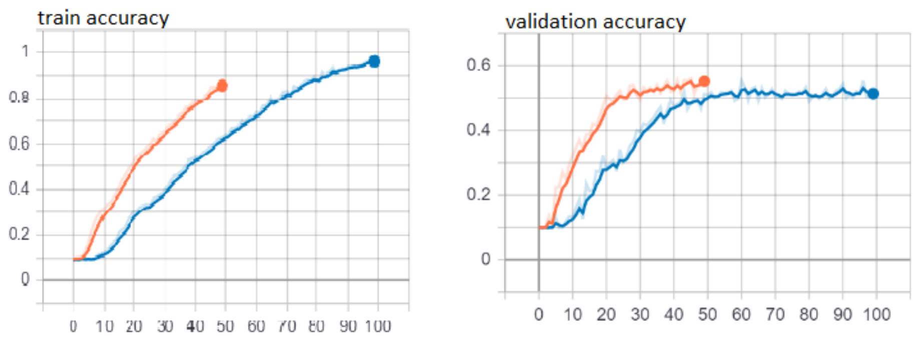


Figure 4: Comparison of Accuracy using batch size of 12 (Orange) vs 32 (Blue)

In our initial experiments, we were interested in establishing a baseline mini-batch size that would make sense for our training scenario. We experimented with batch sizes of 12 and 32, and were interested in the total number of epochs we could fit in before we saw overfitting on our validation data, a well-documented problem with CNN's including AlexNet on ImageNet/Tiny ImageNet data. We found that with a batch size of 12, AlexNet overfit validation data after 30 epochs and with a batch size of 32, overfitting occurred after 50 epochs. We found that approximately equivalent loss and accuracy metrics were reached in both training scenarios, the difference was the time that it took to reach them. Since we are interested in the training scenario that takes the least amount of time, we found that it suffices to train AlexNet with a batch size of 12 for 35 epochs. One of our comparisons can be seen in Figure 4

3.3 Gridsearching Parameters

In this section, we will briefly go over the pipeline that led us to the best accuracy score and the least loss value. After preprocessing the images and building the AlexNet model, we begin by Gridsearching across different inputs to see which would set of parameters would best fit our new tinyimage10 dataset. For each set of parameters, we train the model on the training data, retrieve its training, validation, and test metrics and then forwards that information for further analysis to Tensorboard.

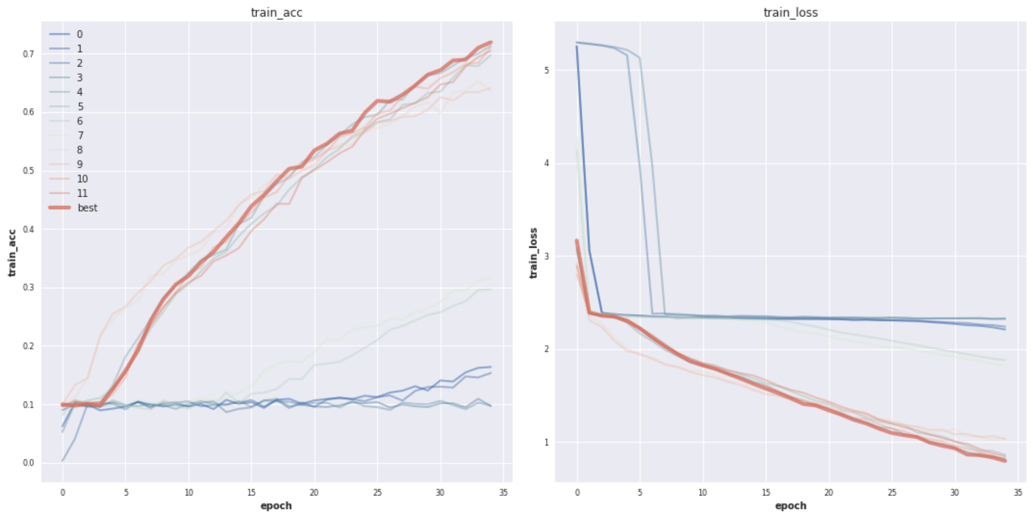


Figure 5: Training Accuracy and Loss across all trials

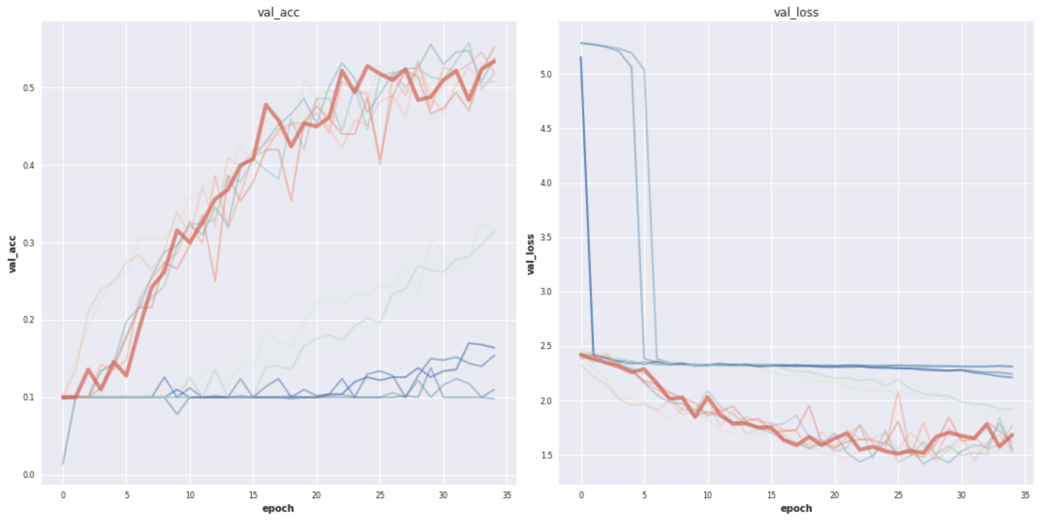


Figure 6: Validation Accuracy and Loss across all trials

Using the method above, we were able to test 12 different parameters inputs with 35 epochs in each trial run, leading to a total run time of approximately 126 minutes. From the parameters discussed above, we Grid-search AlexNet across the following values: 'batch size' : [12], learning rate : [0.0001, 0.001, 0.005], 'momentum' : [0.9, 0.5], and 'gamma' : [0.1, 0.01]. All of the values and results can be seen in the dataframe produced in Table 1. Figures 5 and 6 compare accuracy and loss across all the trials, using the training set and validation set, respectively. The corresponding tensorboard

batch_size	learning_rate	momentum	gamma	optimizer	time(m)	train_acc	val_acc	test_acc
12	0.001	0.9	0.1	sgd	10.82	**0.778**	0.538	**0.546**
12	0.001	0.9	0.01	sgd	10.82	0.728	0.518	0.534
12	0.005	0.9	0.1	sgd	11.17	0.685	0.526	0.534
12	0.005	0.5	0.01	sgd	10.59	0.737	**0.552**	0.534
12	0.005	0.9	0.01	sgd	10.67	0.702	0.508	0.518
12	0.005	0.5	0.1	sgd	10.63	0.728	0.52	0.502
12	0.001	0.5	0.01	sgd	11.01	0.326	0.316	0.304
12	0.001	0.5	0.1	sgd	11.05	0.317	0.314	0.276
12	0.0001	0.9	0.1	sgd	11.33	0.165	0.164	0.166
12	0.0001	0.9	0.01	sgd	10.76	0.16	0.154	0.154
12	0.0001	0.5	0.1	sgd	10.80	0.106	0.11	0.112
12	0.0001	0.5	0.01	sgd	10.88	0.1032	0.098	0.098

Table 1: 12 Trials of different inputs

files and code can be found in the repo [3]. With confirmed results that batch size of 12 fits our data better than others, we can see that best combination of parameters is: learning rate : [0.001], 'momentum' : [0.9], and 'gamma' : [0.1].

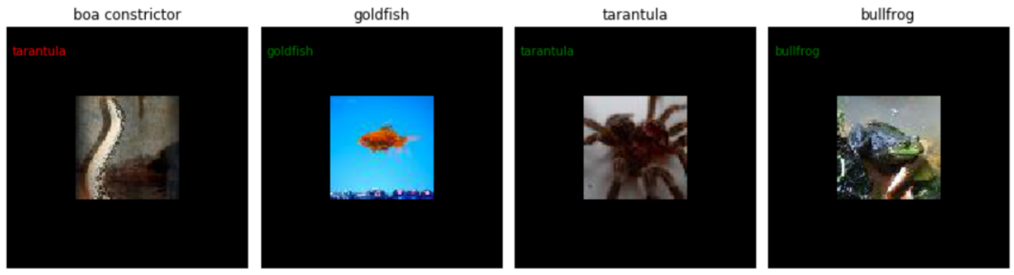


Figure 7: Sample Batch in Test Set

3.4 Final Model

With the gridsearch complete, we are able to identify the best combination of parameters to use. Figures 7 and 8 show a total of 8 images the model was tested on, along with their predictions and labels. The green caption on the images indicate the model predicted correctly, and red, otherwise. Figure 9 shows the models accuracy on each label across the entire test set. We can see that some labels performed way better than others, as can be seen in the 'goldfish' accuracy when compared to 'scorpion'.

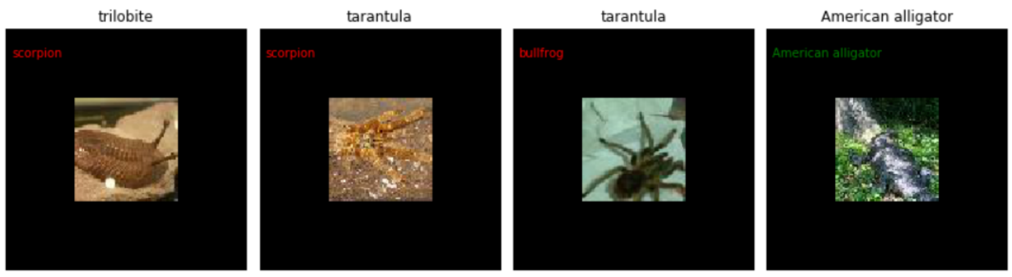


Figure 8: Sample Batch in Test Set

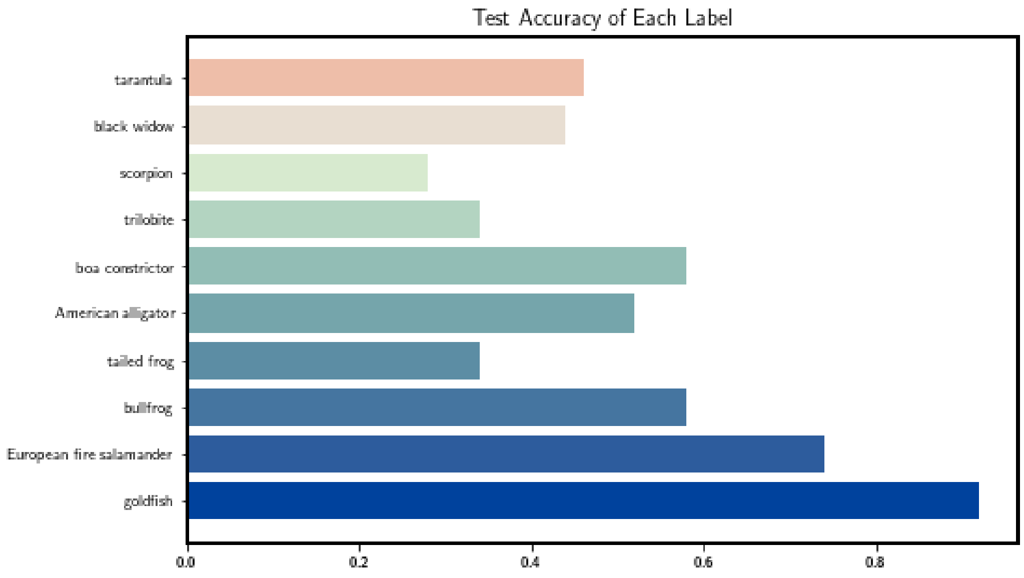


Figure 9: Test accuracy of each label

4 Conclusion

Our own implementation of AlexNet has given us great insight into how well developed deep convolutional networks function. Applying it to a subset of data created by us allowed us to experiment with parameters and understand which parameters have the most affect on the model training and testing. The results were similar to what we expected as AlexNet performed better on the subset of Tiny ImageNet when compared to the entire Tiny ImageNet dataset. Less labels, but the same amount of images for each, will always allow the model to train and test better as number of possible outputs is decreased exponentially. All in all, improvements can be made to the entire pipeline as all trials began overfitting by the 40th epoch of training. A solution to this problem would be to increase the number of images per label, or to search across even further parameters. Time and computer capacity issues held us back in these terms, and another combination of parameters or inputs will probably allow us see better performances. The code, results, and tensorboard files can be found in our shared repo [3].

References

[1] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009). DOI: `10.1109/cvpr.2009.5206848`.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. DOI: `10.1145/3065386`.

[3] talqadi7. *talqadi7/alexnet-implementation*. June 2020. URL: `https://github.com/talqadi7/alexnet-implementation`.