Deep Learning Project 2: CIFAR-10 Image Classification

Elliot Greenlee

October 13, 2017

Abstract

This report covers the implementation and optimization of various deep learning convolutional neural networks in order to classify RGB images. LeNet-5, AlexNet, ResNet, and VGGNet are applied to the CIFAR-10 [7] database, and performance is compared. In order to improve accuracy, hyperparameter improvements are made to LeNet-5 and data augmentation is used on AlexNet. The highest accuracy achieved overall is 93.6% by VGGNet.

1 Introduction

Convolutional neural networks have shown impressive performance on pattern recognition tasks like speech recognition and image classification. For these types of complex problems, hand designed features by human experts fall short of capturing the input, and so CNNs look at the raw data with minimal preprocessing. For image tasks, the inherent structure of the convolution operation has more prior knowledge about the domain than standard feedforward neural networks, and their more sparsely connected nature means training is less difficult or resource intensive. Over time, larger and more variable datasets, hardware improvements, and new algorithms have contributed to increased performance. New network structures have been well investigated, with most improvements pointing towards an increase in network depth. However, this trend has given rise to problems with resource requirements and overfitting. The most important advancements in deep learning have been the solutions to such issues, especially optimized for vision problems and the ImageNet challenge.

The objective of this project is to run multiple deep learning convolutional neural network structures in order to classify image data. Additionally, implementation and hyperparamter optimization are attempted for two of the networks, LeNet-5 and AlexNet in order to improve performance and enhance understanding. The highest accuracy achieved was 93.6% by VGGNet.

2 Cost and Activation Function Design

The classical choices for the activation and cost functions are the sigmoid and quadratic functions, respectively, but these suffer from a learning slowdown problem. Generally when humans learn, drastic incorrectness results in drastic changes in behavior, and more rapid learning; however, neural networks using these classical choices show the opposite behavior: when a prediction is far from the real value, learning begins slowly. This really means that the partial derivatives of the cost function with respect to the bias and weights $\frac{\delta C}{\delta b}$ and $\frac{\delta C}{\delta w}$ are small. When using the quadratic cost function

$$C = \frac{(y-a)^2}{2}$$

for the choice of activation function $a = \sigma(wx + b)$, those partial derivatives are

$$\frac{\delta C}{\delta w} = (a - y)\sigma'(wx + b)x = a\sigma'(wx + b)$$

and

$$\frac{\delta C}{\delta b} = (a - y)\sigma'(wx + b) = a\sigma(wx + b)$$

when $x = 1$ and $y = 0$. Considering the shape of the sigmoid function, it is obvious that when the neuron's output is close to one or zero, the partial derivatives all grow close to one or zero.

To address this problem, the cross-entropy cost function can be used instead of the quadratic cost function. The cross-entropy cost function is defined as

$$C = -\frac{1}{n}\sum_x [y \ln(a) + (1 - y)\ln(1 - a)]$$

where n is the number of training samples and the sum is over all training examples. The function is non-negative, and tends towards zero as the neuron's output is close to the real output for all training inputs. Computing the partial derivatives and simplifying gives

$$\frac{\delta C}{\delta w} = \frac{1}{n}\sum_x x(\sigma(wx + b) - y)$$

and

$$\frac{\delta C}{\delta b} = \frac{1}{n}\sum_x (\sigma(wx + b) - y)$$

. Both of these are controlled by the error in the output $(\sigma(wx + b) - y)$, so they do not suffer from the same issue as before. This makes cross-entropy nearly always the better choice. However, the quadratic cost is fine in the case of linear output neurons, where the sigmoid function is not applied.

Another option to address the learning slowdown problem is the use of softmax neurons in the final layer, and log likelihood as the cost function. The equation for softmax is

$$a = \frac{e^z}{\sum_k e^{z_k}}$$

where $z = wx + b$ and k is the number of other output neurons. Softmax has the property that all the outputs are positive and sum to one, which can be thought of as a probability distribution. Using the log likelihood cost function $C = -\ln(a)$, the partial derivatives become

$$\frac{\delta C}{\delta w} = a - y$$

and

$$\frac{\delta C}{\delta b} = a^{-1}(a - y)$$

where a^{-1} indicates the activation of the previous layer. Comparing this to the earlier equations for cross-entropy shows little difference, and so similarly this solution does not encounter a learning slowdown. This solution works well for classifications problems with disjoint probabilities.

Although the sigmoid neuron model is generally applicable, sometimes other neuron models can learn faster or generalize better to test data. Tanh is one of these variations, using the hyperbolic tangent function

$$\tanh(z) = \frac{e^z - e^{-1}}{e^z + e^{-1}}$$

where $z = wx + b$. It turns out that tanh is a rescaled version of the sigmoid function

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2}$$

Tanh has the same shape, but the output ranges from negative one to one rather than zero to one. This helps prevent a bias towards positive or negative for the weight updates, and also allows weights to the same neuron to increase or decrease depending on the sign of the input activation. For sigmoid neurons, all weights must either increase or decrease.

A second alternative is the rectified linear unit neuron, relu $= \max(0, wx + b)$. Currently there is not a lot of strong theoretical basis for the improvements provided by relu, but one suggestion is the solution to saturation. Sigmoid and tanh neurons learn more slowly when their output is close to one or zero, but relu does not. However, negative inputs to a relu neuron cause learning to stop. Across all these solutions, researchers are still unsure about the theory to back up performance [10].

3 Core Design Success

The original LeNet-5 paper focused on using machine learning techniques for feature extraction, rather than hand-designed methods by experts, used the convolutional neural network (CNN) structure that brought implicit prior knowledge to the image recognition space, and used simple backpropagation for training.

Before this paper, machine learning work focused on two separate steps, feature extraction and simple learning methods. Usually, this meant taking input and making low dimensional vectors using some expert prior knowledge, and using a general purpose classifier trained on the data. This led accuracy to be mostly reliant on good features, reconstructed for each problem, and as dataset sizes increased, the variability present became impossible to represent using hand-designed feature extraction methods. Instead, by applying CNNs directly on pixel images, they showed that with a good architecture minimal preprocessing is needed even for high dimensional input. Using backpropagation allowed them to synthesize and train their own feature extractor.

Classically a problem like image recognition faced issues because pictures are large, and fully-connected layers of neural networks have too many weights, increasing training capacity and requiring even more training examples. Computationally, the memory and calculation requirement for brute force methods was too high. Multilayer neural networks trained with backpropagation was really successful, and with the improvements in low cost machines and bigger databases, all that was required was a better method to bridge the gap. Using CNNs largely reduced the number of weights needing to be trained, and also brought prior knowledge into the structure. CNNs automatically obtain shift invariance without replication of weight configurations across the space, and without needing to perfectly center, size normalize, and rotate the data. If the image is shifted, the feature map will be shifted by the same amount, but will stay the same otherwise, proving robustness. By considering the topology of the input, they can find local features, then more global features, all relative rather than absolute [9].

The AlexNet paper focused on increasing the size of CNNs, and solving the myriad issues like efficiency, overfitting, and saturation that arise when using such large capacity networks. To deal with huge datasets with large variability like ImageNet, with 1.2 million samples and 100 classes, a model with a large learning capactiy was needed, and simple methods or small networks no longer worked.

In order to get their model to perform, it was implemented on a two-GPU system with their new optimization library for 2D convolutions. In order to prevent saturation, the rectified linear unit (relu) was used as the activation function, reducing training time by six times compared to tanh and sigmoid. More on this is available above in the section on activation functions. Overfitting was prevented by data augmentation and the new dropout method. In order to artifically increase the size of the testing data, images were flipped horizontally, translated, and had their RGP intensities altered. It was well established that combining the predictions of multiple models could reduce error, but this is too expensive for large networks. Instead, they used dropout where neurons have a 50% probability to not contribute in a training round, which reduced co-adaptations of neurons since they could not rely on the neighboring connections to be there. Overall, this caused more robust features to be learned [8].

The GoogLeNet paper focused on replicating a sparse architecture using dense, repeatable modules, and efficiently training much larger new networks. Since these algorithms are not helpful unless they can be used in real world applications, Google worked to keep the computational budget constant while increasing depth and width. Using 12 times less parameters than AlexNet allowed them to meet standard for mobile and embedded computing efficiency while also achieving the highest accuracy. This was achieved by diverting away from the bigger networks that lead to overfitting and better computer requirements.

Instead, dense components called inception modules make up the repeatable internal structure, approximating optimal local sparse structure. These modules fit the Hebbian principle of wiring and firing together and more closes mimic biology, creating a probability distribution represented by a sparse deep network of clustered neurons with highly correlated outputs. Each inception module does 1x1, 3x3, and 5x5 convolutions, passing on the results of each operation, which allows visual information to be looked at with different scales and aggregated. 1x1 convolutions are used for dimensionality reduction before the 3x3 and 5x5 convolutions to greatly reduce the number of operations needed. Separate from the normal network, auxiliary classifiers at the midpoints with discounted weight loss improved regularization and increased the gradient, providing faster learning by avoiding the vanishing gradient problem [13].

The VGGNet paper focused entirely on the effects of greater depth for image recognition. By using small, 3x3 convolutions with the same classical architectures, a depth of 16-19 layers was achieved, providing high accuracy on ImageNet and other datasets. Since so many other design components were well investigated, they kept everything else constant while increasing depth, even keeping about the same number of weights as a normal shallower network with larger widths and receptive fields. This paper shows that for the image recognition space, depth alone is a very important characteristic [12].

The ResNet paper focused on producing networks at a depth scale never before seen by solving the vanishing gradients problem. Deep networks led to amazing breakthroughs, using integrating low level to high level features through layers. However, vanishing and exploding gradients cause problems for deep networks, and normalized initialization and intermediate normalization layers could only do so much to solve the issue. Instead, the residual framework is used to provide a 152 layer network with a lower complexity than VGGNet. Layers learn residual functions with respect to layer inputs, not unreferenced functions. Where a normal network calculates $y = f(x)$, ResNet calculate $y = f(x) + x$; these identity connections propagate the gradient, fixing the gradient vanishing problem [6].

4 Hyperparameter Augmentation

The structures for convolutional neural networks come in an astounding and theoretically infinite variety; however, once the perfect structure is designed, it must still be initialized, fed data, and trained. Many parameters guide this process, but here is a quick description of those used for this project. The learning rate guides the speed at which weights and biases change based on the partial derivatives of the cost function, where a higher learning rate indicates a greater change that could be faster, but might lead to instability. The number of epochs controls the iterations of running batches of data through the model, where a higher number of epochs indicates a larger total amount of data fed through. The batch size controls the size of each batch in the stochastic process, where a higher size causes more data to influence the change in weights per epoch. The weight and bias initializations control the starting values, which have an effect on the training speed; generally random gaussian initializations between -1 and 1 provide the best initializations. The dropout keep probability controls the chance that a neuron will be kept in the network. Data augmentation can mean many things, but for image data of this type, distortions caused by flips, rotations, crops, and contrast/brightness manipulations can increase the size of the training data and prevent overfitting [10].

All of these individual hyperparameters can effect the learning speed and accuracy of the convolutional neural network, and this is after a specific structure has been chosen. Furthermore, the complex relationships between the parameters require specific combinations for maximum accuracy. One option for this sort of problem is gridsearch, an exhaustive search through a range of values for each parameter. This search is guided by cross validation accuracy on the training data [3]. Unfortunately, the run time for such a type of search can be enormous; in the case of these experiments, 245, 000 combinations parameters needed to be iterated through to get a comprehensive picture of the space. This was infeasible with the resource constraints. Even with improvements like stripping down the network structure and reducing the amount of training data which could alter results, training can still be slow. Fortunately, guided search methods can result in more promising results. The first goal for any search is to get any non-trivial results where the network achieves a higher accuracy than chance, but from there, progress depends on a combination of luck and investigation [10].

5 Results

Across all the networks, VGGNet performed the best with an accuracy of 93.6%; other accuracies can be observed in table 1.

Network	Accuracy
LeNet-5	53.0%
AlexNet	83.0%
VGGNet	93.6%
ResNet	92.3%

Table 1: Best accuracies for all networks.

5.1 Data

The data for this report is CIFAR-10, originally collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton[7]. It is a set of 32x32 pixel RGB images of ten different classifications: four modes of transportation and six animals. There are 60,000 instances divided into 50,000 training samples, 5000 from each class, and 10,000 testing samples, 1000 from each class. Each of the ten classes is completely mutually exclusive. The highest accuracy reported [2] on the database is 96.53% [5].

Distortion was used to augment the real data for AlexNet, VGGNet, and ResNet. Convolutional neural network performance on MNIST has been improved more by manipulation of the data-space than other types of manipulation, and while there is a theoretical limit, based on real data size, to the improvement that can be gained by data augmentation, most experiments do not reach this limit [16]. These types of manipulations include horizontal and vertical flips, rotations, cropping, brightness and contrast variations, as well as affine transformations and elastic distortions. For these experiments, TensorFlow [1] is used to perform random manipulations by flipping, rotating, cropping, and brightness/contrast alteration at a rate of 50%. Because these manipulations can be done in realtime, overfitting is reduced because the network rarely sees the same image twice.

5.2 LeNet-5

The LeNet-5 network was adapted from Liu's given code to work with the CIFAR-10 data.

In order to optimize the hyperparameters for LeNet-5, each parameter was investigated while keeping the others constant. First the initialization parameters were varied, then the dropout parameter, and then the training variables. Smaller variations were then made to find the optimal performance.

First, weight initialization was varied, with 0.01 resulting in the best accuracy of 29% with a learning rate of 0.001, 200 epochs, batch size 50, bias 0.1, and dropout keep probability 50%. Then, bias initialization was varied, with 0.1 resulting in the best accuracy of 31% with a learning rate of 0.001, 200 epochs, batch size 50, weight standard deviation 0.01, and dropout keep probability 50%. These results are shown in figure 1.

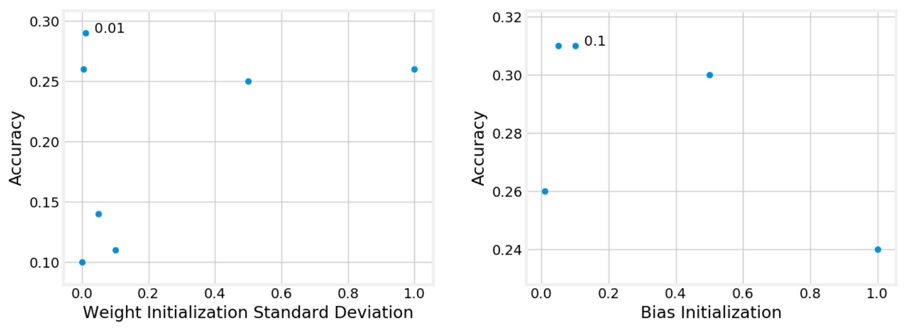

Figure 1: Initialization variable hyperparameter optimization by guided search as described above.

Next, dropout keep probability was varied, with 90% resulting in the best accuracy of 32% with a learning rate of 0.001, 200 epochs, batch size 50, weight standard deviation 0.01, and bias initialization 0.1. These results are shown in figure 2.

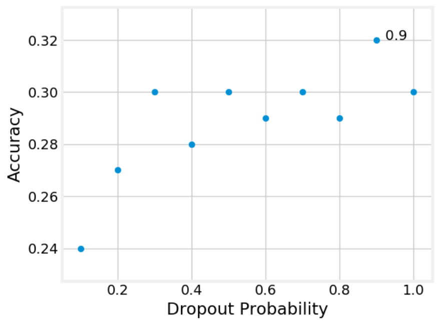

Figure 2: Dropout keep probability hyperparameter optimization by guided search as described above.

Next, the learning rate was varied, with 0.0003 resulting in the best accuracy of 39% with 200 epochs, batch size 50, dropout keep probability 90%, weight standard deviation 0.01, and bias initialization 0.1 . Then, batch size was varied, with 400 resulting in the best accuracy of 50% with 0.0003 learning rate, 200 epochs, dropout keep probability 90%, weight standard deviation 0.01, and bias initialization 0.1. Then, epochs were varied, with the value simply increasing as the training time increased, and all other values held constant at the best values above. Training stopped at 450 epochs with 53% accuracy. These results are shown in figure 3.

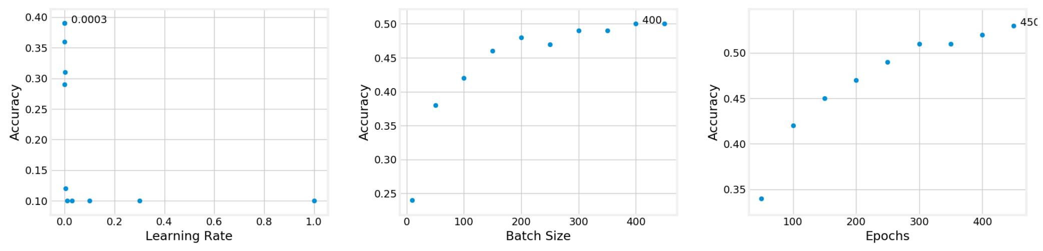

Figure 3: Training hyperparameter optimization by guided search as described above.

As the batch size increases, more training samples are used to compute the change in weights during the current epoch, more appropriately matching the distribution of the real data. This can lead to faster convergence, but increases the runtime. Figure 4 shows the balance to be struck between runtime per epoch and epochs to 25% accuracy, while table 2 gives exact information. From this it can be observed that the best efficiency is around a batch size of 40, but unfortunately this batch size does not allow the best final accuracy. Figure 5 shows multiple batch sizes and their validation accuracies over training.

	10	20	30	40	50	60	70	80	90	100	110	120	130
Runtime	0.07	0.13	0.19	0.26	0.31	0.38	0.45	0.52	0.58	0.65	0.72	0.83	0.90
Epochs	100	81	37	29	34	21	25	30	27	21	19	23	17
Total	7.0	10.5	7.0	7.5	10.5	8.0	11.3	15.6	15.7	13.65	13.7	19.1	15.3

Table 2: Runtimes per epoch over 20 epochs, and epochs needed to reach 25% accuracy at various batch sizes.

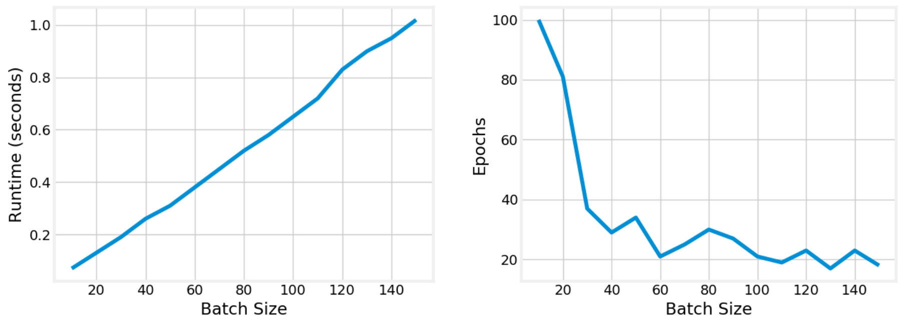

Figure 4: Runtimes per epoch over 20 epochs, and epochs needed to reach 25% accuracy at various batch sizes.

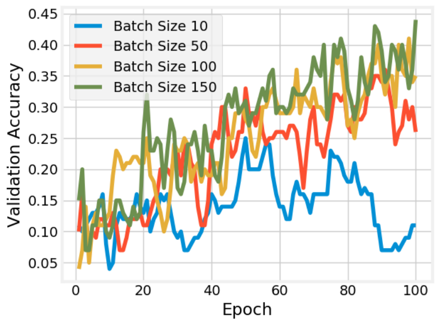

Figure 5: Training over 100 epochs for various batch sizes.

5.3 AlexNet

AlexNet was implemented according to the Tensorflow tutorial [14] and trained to achieve an accuracy of 83.0%. This model took 16,500 epochs, a batch size of 128, and a learning rate schedule of 0.1 with decay 0.1 every 350 epochs. Data augmentation through random horizontal flipping and brightness/contrast changes was used.

Different label-preserving distortions for data augmentation were used with accuracies compared. Accuracies after a certain number of epochs showed slight repeatable differences between setups, which can be observed in table 3. Across all the runs, center cropping with brightness distortions produced better accuracies and all other modifications produced worse accuracies; this is interesting because the actual AlexNet setup used recommended using horizontal flips and modifications to brightness and contrast. Perhaps at the 83.0% convergence these provided the best accuracy, but by 2000 epochs no differences in the trend was observed, with centering still higher than random cropping.

Crop Style (epochs)	None	HFlip	B	C	B, C	HFlip, B, C	VFlip
Center Crop (500)	53.0	52.9	54.1	52.7	52.3	53.6	45.6
Random Crop (500)	51.8	51.0	52.4	51.5	51.3	52.1	43.8
Center Crop (1000)	63.2	62.6	63.6	62.6	62.8	62.4	53.5
Random Crop (1000)	61.7	60.7	61.7	61.6	60.4	60.5	51.6

Table 3: Percent accuracy effects of various data augmentation techniques on AlexNet. HFlip and VFLip are horizontal and vertical flips, B and C are brightness and contrast modifications.

5.4 ResNet

For the ResNet testing a pretrained model was run across the test dataset and achieved an accuracy of 92.3% [15]. This model was ResNet-32 trained using 80,000 epochs, a batch size of 128, and a learning rate schedule of 0.1 at 0 steps, 0.01 at 40,000 steps, and 0.001 at 60,000 steps. The weight decay for l2 regularization was 0.0002. Data augmentation through random horizontal flipping at 50% and cropping of two pixel layers was used.

5.5 VGGNet

For the VGGNet testing a pretrained model was run across the test dataset and achieved an accuracy of 93.6% [4]. This model was VGG16 using 250 epochs, a batch size of 128, and a learning rate schedule of 0.1 with decay 0.000001. The weight decay for l2 regularization was 0.0005. Data augmentation through random 15 degree rotations, horizontal flips, and 10% image shifts was used.

6 Summary

LeNet-5, AlexNet, VGGNet, and ResNet were implemented and trained or run on the CIFAR10 dataset, and their performance compared. Hyperparameter investigation was performed to establish trends in performance. Overall, VGGNet achieved the highest accuracy at 93.6%.

The objectives of this project were to investigate multiple convolutional neural networks and their performance on the CIFAR10 dataset. Working through this project I learned more than I could write here, but I gained practice downloading and exploring advanced models from other researchers, discovered new hyperparameters to investigate along with general strategies for optimization, and figured out many new coding practices for working with convolutional neural networks and Tensorflow that I want to implement in the future. I was not able to complete everything I wanted to on this project. The training time for AlexNet was much larger than anticipated, so I was not able to spend time graphing trends in the hyperparameters. For LeNet-5, I wanted to implement data distortion techniques myself, but got stuck needing to re-implement the entire input structure; luckily I saw many more examples of how to do this properly in the other pretrained networks and will be able to in the future. I also wanted to look at the new hyperparameters I discovered along with way like regularization and learning rate schedule.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Rodrigo Benenson. who is the best in cifar-10? `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130`. Accessed: 2017-10-09.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[4] geifmany. cifar-vgg. `https://github.com/geifmany/cifar-vgg`. Accessed: 2017-10-11.

[5] Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey 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.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278-2324, November 1998. `http://yann.lecun.com/exdb/publis/#lecun-98`.

[10] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[11] Hairong Qi. Leaderboard. `http://web.eecs.utk.edu/~qi/deeplearning/project/leaderboard2.htm`. Accessed: 2017-10-13.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[14] Tensorflow. Convolutional neural networks. `https://www.tensorflow.org/tutorials/deep_cnn`. Accessed: 2017-10-5.

[15] wenxinxu. Resnet in tensorflow. `https://github.com/wenxinxu/resnet-in-tensorflow`. Accessed: 2017-10-11.

[16] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: when to warp? *CoRR*, abs/1609.08764, 2016.

7 Appendix

The code for this project can be found on GitHub at `https://github.com/LambentLight/692-cnn-comparison`. Please contact me with any questions or considerations.

Dr. Hairong Qi provided a class leaderboard for reported performance and hyperparameters which is shown in the figure below along with highest accuracies for this report [11].

Network	Mine	Leaderboard
LeNet-5	53.0%	69.2%
AlexNet	83.0%	N/A
ResNet	92.3%	95.3%
VGGNet	93.6%	N/A

Table 4: Class highest accuracy and my highest accuracy for each network.