COGS 181 Final Project

Sialoi Ta'a staa@ucsd.edu
Department of Electrical and Computer Engineering, University of California, San Diego
San Diego, CA 92093 USA

Jay Buensuceso jbuensuceso@ucsd.edu
Department of Cognitive Science, University of California, San Diego
San Diego, CA 92093 USA

Abstract

This journal includes our hypothesis, solution and findings when looking at trying to capture the complexities of CIFAR-10's dataset. We will be using the MobileNetV1 architecture founded in 2017 [1] in an attempt to capture these complexities and record our results. The decision of using MobileNets over any other architecture is that it claims to be accurate and lightweight. We wanted to test that claim on the CIFAR-10 dataset. We include our modified implementations of MobileNet that fits the dataset, the insights of the results from those implementations and how we would improve upon it going forward.

1. Introduction

Convolutional neural networks are a very popular neural network for classification problems. In 2012, AlexNet achieved a top 5 error of 15.3% in the ImageNet's 2012 Challenge leading to a renaissance in the development of convolutional neural networks.[3]

One such neural network is MobileNets developed in 2017 by Howard et. al.[1] This neural network is a mobile neural network developed by google meant for portable devices such as mobile phones or robotics applications. The primary differentiating feature of this neural network compared to other contemporary convolutional neural networks is its usage of depthwise separate convolutions in order to reduce the amount of parameters needed to maintain accuracy.

Depthwise convolutions separate each input layer into separate convolutions as opposed to traditional convolution layers where each filter convolves across all input layers. This defining feature leads to less computationally intensive operations and significantly reduces the amount of computations required to get to the same accuracy of a more traditional convolutional neural network.

Another feature of MobileNets is the usage of pointwise convolution layers. Where this comes into play is after a depthwise convolution is done, a 1x1 convolution is applied to increase the dimensionality back to a workable level and combine information from the different channels created from the depthwise convolution.

CIFAR-10 is a benchmark dataset of 60,000 images of size 32x32 pixels classified into 10 classes. Due to its manageable dataset, it is a common dataset used for the benchmarking of machine learning algorithms for computer vision tasks.

In our exploration of convolutional neural networks, we developed two different models closely based on MobileNetV1, the algorithm Google developed in 2017. [1]

2. Architecture Definition and Method

When we looked at *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* [1] by Howard et. al.[1] it quickly became clear that the neural network would need significant modifications before it could be trained on the CIFAR-10 dataset.

2.1 First Neural Network

Therefore, in order to properly read our images, we modified the amount of layers in the neural network from 25 layers to 12 layers.

Table 1. Our initial neural network:

Type	Filter Shape	Input Size
convLayer1	3 x 3 x 32	32 x 32 x 3
convDWLayer2	3 x 3 x 32	16 x 16 x 32
convLayer3	1 x 1 x 64	16 x 16 x 32
convDWLayer4	3 x 3 x 64	16 x 16 x 64
convLayer5	1 x 1 x 128	8 x 8 x 64
convDWLayer6	3 x 3 x 128	8 x 8 x 128
convLayer7	1 x 1 x 128	8 x 8 x 128
convDWLayer8	3 x 3 x 128	8 x 8 x 128
convLayer9	1 x 1 x 256	8 x 8 x 128
convDWLayer10	3 x 3 x 256	8 x 8 x 256
convLayer11	1 x 1 x 256	8 x 8 x 256
convDWLayer12	3 x 3 x 512	8 x 8 x 256
avgPoolLayer	4 x 4	4 x 4 x 512
FC1	512 x 1000	1 x 1 x 512
FC2	1000 x 10	1 x 1 x 1000
Softmax	Classifier	1 x 1 x 10

For this neural network we chose to compare the Adam or Adaptive Moment Estimation Optimizer and Stochastic Gradient Descent (SGD) with a learning rate of 0.0001 each. This is primarily due to our theory that due to its adaptive nature, Adam would help in a much faster convergence when compared to a SGD optimizer.

Additionally, Much like most contemporary classification neural networks, we used a cross entropy loss as our loss function of choice. This function is defined as:

$$L(y,t) = -\sum_i t_i ln(y_i)$$

Where t is the target and y is the predicted distribution.

The reason for choosing cross entropy loss is due to its popular usage in neural networks dealing with classification problems. This is due to its ability to quantify the probability between the predicted distribution and the given target labels.

2.2 Second Neural Network

This isn't the only convolutional neural network we developed during this exploration into neural networks. Much like our first neural network, the architecture of this neural network is closely based on MobileNets, making use of depthwise and pointwise convolutional layers.

Table 2. Our second neural network:

Type	Filter Shape	Input Size
convLayer1	3 x 3 x 32	32 x 32 x 3
convDWLayer2	3 x 3 x 32	16 x 16 x 32
convLayer3	1 x 1 x 64	16 x 16 x 32
convDWLayer4	3 x 3 x 64	16 x 16 x 64
convLayer5	1 x 1 x 128	16 x 16 x 64
convDWLayer6	3 x 3 x 128	16 x 16 x 128
convLayer7	1 x 1 x 128	16 x 16 x 128
convDWLayer8	3 x 3 x 128	16 x 16 x 128

convLayer9	1 x 1 x 256	16 x 16 x 128
convDWLayer10	3 x 3 x 256	16 x 16 x 256
convLayer11	1 x 1 x 256	16 x 16 x 256
convDWLayer12	3 x 3 x 512	16 x 16 x 256
convLayer13	1 x 1 x 512	8 x 8 x 256
5 x convBlockLayer	3 x 3 x 512dw, 1 x 1 x 512	8 x 8 x 512
convDWLayer19	3 x 3 x 512	8 x 8 x 512
convLayer20	1 x 1 x 1024	4 x 4 x 512
convDWLayer21	3 x 3 x 1024	4 x 4 x 1024
convLayer22	1 x 1 x 1024	4 x 4 x 1024
avgPoolLayer	4 x 4	4 x 4 x 1024
FC1	1024 x 1000	1 x 1 x 1024
FC2	1000 x 10	1 x 1 x 10
Softmax	Classifier	1 x 1 x 10

As opposed to our first neural network, the stochastic gradient descent optimizer was our chosen initial optimizer with a learning rate of 0.0001. The reason for using SGD in this neural network is due to having more opportunities in hyperparameter tuning with pytorch's implementation of the SGD algorithm, enabling us more flexibility in increasing our accuracy score.

Other hyperparameters such as loss function and epoch each remain the same from the first neural network and the second neural network.

2.3 Method

In order to test both neural networks, we ran multiple different hyperparameters in testing with the goal of as high of an accuracy as possible. Parameters such as activation functions, epochs, learning rate, and even momentum were modified with each iteration of our neural networks running.

Table 3. Neural networks and hyperparams used for each experiment

Neural Network	Hyperparams Tuned	Epochs
1.NN1	Adam Optimizer	10
2.NN1	SGD w/ 0.95 momentum	10
3.NN1	Adam w/ Leaky ReLU	10
4.NN1	Adam w/ Leaky ReLU	20
5.NN2	SGD w/ 0.95 momentum	10
6.NN2	SGD w/ 0.95 momentum, 0.00005 LR	60
7.NN2	SGD w/ 0.95 momentum, 0.00005 LR	125
8.NN2	Adam w/ Leaky ReLU 0.0001 LR	60

Each neural network was executed using these parameters, displaying the mini-batch loss at the end of each epoch. Subsequently, upon completion of network execution, an average loss graph and the accuracy corresponding to each label were generated. Furthermore, by dividing the neural networks into two sets, we could conduct tests concurrently, optimizing our group's time and significantly expediting testing compared to sequential execution of individual neural networks.

3. Experiment

In this section, we investigate each neural network's performance with the given hyperparameters. We utilize the two neural networks in the battery of tests outlined in Table 3. Upon the completion of each test, a comparison is made between the previous iterations of the neural networks.

3.1 Neural Network 1 Results

We conducted our battery of tests on our first neural network by starting out with modifying the optimizers of our first two executions. Table 4 describes the results of

the accuracy rating after 10 epochs. Comparing the Adam optimizer with Stochastic gradient descent, we see a 10% increase in accuracy when using Adam vs. SGD for neural network 1.

Table 4. NN1 Adam vs SGD at 10 epochs

Adam Optimizer	50.9%
SGD Optimizer	40.5%

This suggests that Adam is more effective in optimizing the neural network parameters, resulting in a better convergence and improved performance. Therefore, for this specific neural network and training task, Adam appears to be a more favorable optimizer choice compared to SGD.

Another hyperparameter we looked into in this neural network was comparing the performance of ReLU and Leaky ReLU. A ReLU or rectified linear unit is defined as:

$$f(x) = max(0, x)$$

Albeit simple, one of the main drawbacks of ReLU is the idea that a neuron can hit a negative value and set itself to 0. This leads to a problem where the neuron essentially vanishes or inactivates itself and stops updating its weight parameter entirely.

A simple way of solving this problem without refactoring the neural network is to utilize a different activation function such as Leaky ReLU. By using a small, positive gradient when the unit is not active (in this case 0.01), it can mitigate the issue of a neuron vanishing or dying. This function is defined as:

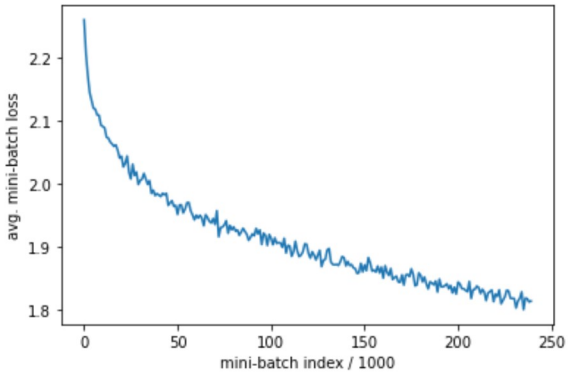
$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

By using LeakyReLU, the neural network's performance increases only a miniscule amount from 50.9% accuracy to 51.9% accuracy with a mini batch loss of 1.9. This led us to believe that the activation function for this dataset is not a high priority hyperparameter and thus, we focused our attention to other hyperparameters.

For this neural network, our last remaining tests involved increasing epoch amount and comparing the accuracy of a neural network trained on 10 epoch to 20 epoch.

Our findings showed an increase from 51.9% to 60% with a mini batch loss of 1.81. This makes sense as increasing the epoch means that we are increasing the amount of passes the neural network does on the CIFAR-10 dataset.

Figure 1. Average mini-batch loss of NN1 at 20 epochs

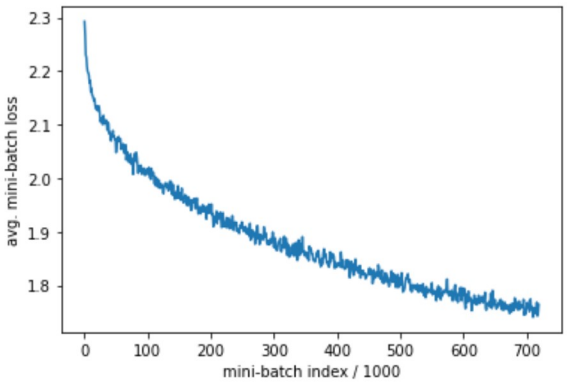


Concurrently, while this neural network was undergoing its testing, we revisited the MobileNets and took even further inspiration by adding additional layers, and more importantly, 5 convolutional block layers into the design of our 2nd neural network which more closely matched the architecture laid out by MobileNetV1.

3.2 Neural Network 2 Results

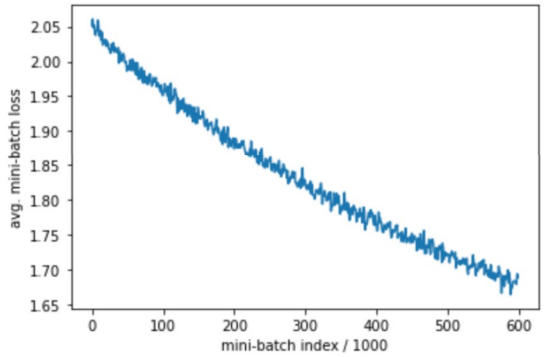
The second neural network used the MobileNetV1 but it used the convolution block of 5 depthwise convolutions and 5 regular convolutions. Because it has this extra block of hyperparameters, it was hypothesized that our network should be able to capture more complex patterns in the images and increase the accuracy. However, there was a concern of whether we would either increase the test accuracy or the testing accuracy with this decision as adding a deeper network generally causes overfitting to occur. This network is the closest implementation we have comparable to the original from the academic journal *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*.[1]

This the loss plot for the best model:



This model achieved an accuracy of 65% using Adam optimizer with a learning rate of 0.0001and 50 epochs.

Next is a loss plot with stochastic gradient descent for the optimizer and 60 epochs:



This model achieved an accuracy of 64% with the hyperparameters being learning rate is 0.00005 and momentum was 0.95.

Looking at both cases, Adam optimizer is better than SGD (stochastic gradient descent). The concern for overfitting was found unnecessary and our hypothesis was correct. Adding more layers such as that convolution block showed improved performance and has led us to new insights on the data. One insight is that if there's improved accuracy after adding more hyperparameters, then that could mean that the patterns in the dataset are more complex than previously assumed and a more complex model is needed to fully capture the most ideal solution. Another interesting insight is questioning if there were any patterns that couldn't be captured because of dead nodes? This is something that we would consider for the structure in the next model we create for this dataset.

4. Reflection

Reflecting on our results and our insights from the experiments, we have some ideas on how to make the network architecture better. The next idea would be to implement the block structure again but also add skip connections to make it similar to a ResNet as talked about in *Aggregated Residual Transformations for Deep Neural Networks*.[2] There's three benefits in doing this: creating a broader block to capture higher complexities, reviving any dead nodes back to life from activation functions such as ReLU, and not adding any depth to the network. These are the hypothesized likely benefits of adding skip connections to our network in order to maximize accuracy. The down side of this is that the added complexity of the network does lower the benefit of having a fast lightweight model that gives generally good accuracy. Another idea that could give us higher accuracy is if there were more data in the dataset.

5. Conclusion

In our experiments, we've found a good starting point in constructing a network that captures a majority of the patterns in the CIFAR-10 dataset. The second network was able to accurately identify the correct classes more than not. However, this should only be seen as a starting point and we've learned that while the network did better than what we originally thought, it could do better. MobileNets have a lot of potential to do better with the right techniques being used to structure the neural network such as adding skip connections, a special characteristic of residual networks.

5. References

[1] Howard, A. G., Zhu, M., Chen, B., Wang, W., Weyand, T., Andreetto, M., Adam, H., & Kalenichenko, D. (2017, April 17). Arxiv. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/pdf/1704.04861.pdf

[2] Xie, S., Girshick, R., Dollar, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1611.05431.pdf

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386

[4] Professor Tu, Zhuowen