What's the Best Mini-batch? Empirical Study on Image Classification

Stephen Lasky, Vartan Kesiz Abnousi, Yufeng Ma, Yanshen Yang
{slasky, vkesizab, yufengma, yanshen}@vt.edu

1 Introduction

The initial goal of this paper is to analyze and see if there is any pattern that we as humans can pick out by looking at the images that would have been the best training mini-batch at different phases of training. Subsequently, we compute the full gradient, and then see which mini-batch of examples would have led to the best approximation of the true gradient. Stochastic Gradient Descent (henceforth SGD) has been a standard approach in machine learning. Mini-batch Gradient Descent (henceforth MBGD) is another popular approach, that is a compromise between SGD and the standard (Batch or Full) Gradient Descent (henceforth GD). While there has been extensive research on efficient Mini-batch Training for Stochastic Optimization [16], there is a paucity of research in discovering image patterns that could direct us to the best mini-batch at different phases of training. To this end, we explore the existence of patterns in MBGD images in relation to the GD, with the goal to find the best training mini-batch at different phases of training. We compare the true gradients (ones from GD) with the mini-batch gradients (ones from MBGD) on the CIFAR-10 dataset by developing an algorithm that finds the best results. Next, we implement several heuristic metrics in order to discover patterns on these images, as it's not only subjective but also hard to discern objective patterns in images directly. To the best of our knowledge, there has been no similar research in the past. Statisticians have used for years' heuristics, such as looking at plots, to quickly uncover patterns in their data without formal statistical tests. A successful recognition of some patterns in this framework could save practitioners a significant amount of time, since the optimization of the parameters is mostly done within a highly non-linear surface that is computationally expensive for learning tasks with big data.

2 Related Work

Traditionally, it is well known that the objective function, F, should satisfy convexity and Lipschitz continuity [1]. A problem with the traditional GD, in case the assumptions are violated, can converge to a saddle point, which are particularly common in non-convex optimizations [3]. However, recent advances prove that gradient descent converges to a local minimum, almost surely with random initialization, by applying the Stable Manifold Theorem from dynamical systems theory [15]. Visual representations learned by deep convolutional neural networks show excellent performance on previously challenging tasks like ImageNet classification. He et al. (2016) [8], proposed a residual learning framework to ease the training of networks that are substantially deeper than those used previously by explicitly reformulating the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Krizhevsky et al. (2012) [13] trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes and show that a large deep convolutional neural network is capable of achieving record breaking results on a highly challenging dataset using purely supervised learning.

Gotal et al. (2017) [6] showed that on the ImageNet dataset, large mini-batches can cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, they demonstrated no loss of accuracy when training with large mini-batch size up to 8192. To achieve this result, they adopted a linear scaling rule for adjusting learning rates as a function of mini-batch size and develop a new warm-up scheme that overcomes optimization challenges early in training. Simonyan et al. (2014) [17] investigated the effect of the convolutional

network depth on its accuracy in the large-scale image recognition setting. Their main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth up to 16–19 layers. Szegedy et al. (2014) [18] proposed a deep convolutional neural network architecture codenamed Inception, which was responsible for setting a new state of the art milestone for the image classification, and their results provided a solid evidence that approximating the expected optimal sparse structure by readily available dense building blocks is a viable method for improving neural networks for computer vision. The main advantage of this method is a significant quality gain at a modest increase of computational requirements as compared to shallower and less wide networks. Zeiler et al. (2013) [22] introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Used in a diagnostic role, these visualizations allow us to find model architectures that outperform Krizhevsky et al [14] on the ImageNet classification benchmark. Xie et al. (2016) [21] introduced a highly modularized network architecture for image classification. Our network for final experiments is the one called Dual Path Networks [2], which is a combination of ResNet [9] and Densenet [10].

Glorot et al. (2010) [5] tries to understand better why standard gradient descent with random initialization is doing notoriously poorly with deep neural networks, which can help design better algorithms in the future. Furthermore, their study how activations and gradients vary across layers and during training, with the idea that training may be more difficult if the singular values of the Jacobian matrix associated with each layer are far away from 1.

Wilson et al. (2017) [20] manually construct an illustrative binary classification problem where the data is linearly separable, both GD and SGD achieve zero test error, and AdaGrad [4], Adam [11], and RMSProp [19] attain test errors arbitrarily close to half. They additionally study the empirical generalization capability of adaptive methods on several state-of-the-art deep learning models. They demonstrated that the solutions found by adaptive methods generalize worse (often significantly worse) than SGD, even when these solutions have better training performance. These results suggest that practitioners should reconsider the use of adaptive methods to train neural networks.

Hardt et al. (2016) [7] show that parametric models trained by a stochastic gradient method (SGM) with few iterations have vanishing generalization error. Applying the results to the convex case, they provide new insights for why multiple epochs of stochastic gradient methods generalize well in practice. While in the non-convex case, they give a new interpretation of common practices in neural networks, and formally show that popular techniques for training large deep models are indeed stability-promoting.

3 Approach

As we have stated before, it would be hard to identify patterns by just subjectively looking at mini-batch images. Different individuals would perceive image content differently. In oder to find more objective patterns, we plan to approach it from the opposite perspective. Here we will first propose several heuristic metrics, based on which we can rank the training samples. Afterwards, the whole training dataset with fixed order can be divided into multiple mini-batches. This is different from the typical training of deep models, as its training examples are shuffled at the very beginning of each epoch while we will fix the order for later interpretation. Therefore, during iterations in one epoch the metric values of mini-batches will gradually change. Besides, we will also evaluate the distance of each mini-batch's gradient to the full-batch gradient. Then based on how far away different mini-batches are to the full-batch and their corresponding metric value in different training phases, certain policies for choosing best mini-batches might be identified. What's more, it's also essential to ensure that the gradient of each mini-batch and full-batch are estimated based on the same parameter value. As we know, typical deep model training will update the parameters after each mini-batch, which inevitably corrupts the estimation of gradients for later mini-batches. It would be important for us to also pay attention to this during training.

3.1 Training Framework

In typical training of deep neural networks, we will run for quite a few epochs, each of which will go over the whole training samples once. While during each epoch, as normally constrained by the

computational resources and parameter updating efficiency, training examples are basically divided into multiple mini-batches. Then for each mini-batch, a forward pass is executed to compute the loss value of a predefined objective function, which is followed by a backward process for parameter gradient estimation. Finally, parameters are updated with gradient-based optimizers upon current parameter values.

However, in our experiments we would want to compare gradients of mini-batches with the one of the full-batch. This constrains us to calculate the gradient of every mini-batch based on the exactly same parameter value as the full-batch, i.e., the starting parameter value for each epoch. Moreover, it's probably infeasible to place all training examples into memory to estimate the gradients of the full-batch. As such, we'll have to estimate it based on the gradients of each mini-batch.

Suppose we are doing image classification with training samples $\{x_i, y_i\}_{i=1}^{N}$. Our final objective function can be formulated as follows:

$$R(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i; \theta), \tag{1}$$

in which L stands for the cross entropy loss and θ contains all model parameters. While during the mini-batch training, if the batch size is set to m, then Eq 1 can be re-arranged as

$$R(\theta) = \frac{1}{N} \sum_{k=1}^{N/m} \left[\frac{1}{m} \sum_{i=1}^{m} L(x_{ki}, y_{ki}; \theta) \right], \tag{2}$$

where x_{ki}, y_{ki} corresponds to one specific training sample i in batch k. We are abusing notation usage here for better explanation. Based on this reformulation, if we have already cached the gradient for each mini-batch k, i.e., $\nabla_k = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(x_{ki}, y_{ki}; \theta)}{\partial \theta}$, then

$$\frac{\partial R(\theta)}{\partial \theta} = \frac{1}{N} \sum_{k=1}^{N/m} [m \times \nabla_k]. \tag{3}$$

This should give us the full batch gradient on all training samples. Then we may compare each mini-batch's gradient to this and estimate the 'best' approximations/similarities based on some quantitative metrics like euclidean distance or some kinds of norms.

On the other hand, to deal with the comparison based on the same parameter point, we propose to incorporate another model **net_compare** during the training, for which our normal training model is called **net_normal**. More specifically, at the beginning of each epoch, both **net_normal** and **net_compare** should share the same parameter values. While with the training of each mini-batch, **net_normal** would go through forward pass, backward pass, and parameter update steps. But for this new incorporated model **net_compare**, it will only experience forward pass and backward pass for the purpose of estimating the gradient of each mini-batch based exactly on the starting parameter value of this epoch. As there is no update step, parameter values are always the same. Moreover, we'll also cache each mini-batch's gradient to compute the gradient of full-batch with our previous strategy. It's worth noting that we'll need to save these gradients onto disk, since caching in memory can still cause out of memory issue. After we have iterated through all the mini-batches within one epoch, we shall read in all the cached gradients and average them to estimate the full-batch gradient. Now before we jump to the next epoch, we will then copy the normal training model's parameters (**net_normal**.θ) to this new incorporated model (**net_compare**). This makes sure **net_compare** is always tracking the parameter change during optimization. Then based on the full-batch gradient, we can calculate the distances of each mini-batch's gradient to the one of the full-batch. The distance of each mini-batch with different metric value at different epochs (training phases) can be visualized for interpretation. Detailed pseudo code is depicted in Algorithm 1.

Algorithm 1 Mini-batch and Full-batch Gradient and Distance Calculation

Require: training samples $\mathbf{X}$ and $\mathbf{Y}$
Require: loss function criterion$(\cdot,\cdot)$, two models **net_normal**, **net_compare**
for e=1 $\rightarrow$ Epochs **do**
  # copy the normal training parameters
  net_compare.θ = **net_normal**.θ
  for k, (inputs, targets) $\in (X, Y)$ **do**
    # typical training procedure
    outputs = net_normal(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    net_normal.θ.update()
    # gradient estimation at starting parameter value of each epoch
    outputs = net_compare(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    ∇_k =net_compare.θ.grad
    Cache ∇_k and inputs on disk
  end for
  $\nabla_e = \frac{1}{K}\sum_{k=1}^{K} \nabla_k$
  # calculate distance of each mini-batch to full-batch
  $D_{e,k}$ = norm($\nabla_k - \nabla_e$)
end for

Meanwhile, it's essential to define the training phases more rigorously. Modern training process of deep neural networks would usually follow some learning rate decay strategies. For example, we may decay the learning rate every 10 epochs or if there is no performance improvement in the last several epochs. During our exploration, we would define training phases based on epochs with the same learning rate. Finally, we may investigate on how the distance of mini-batch gradient to true gradient changes within such training phases. Details of model architecture and training phase definition will be elaborated in the experiment section.

3.2 Heuristic Metrics

With tens/hundreds of images in one mini-batch, patterns of details are really hard to intuitively recognize. Among different image categories, people should successfully extract common details buried in images. Not everyone is a vision expert. Instead of directly approaching this challenge, we would deal with it based on already know heuristic metrics. More specifically, if gradient of some mini-batch with certain metric value bear high similarities to the full batch, it implicitly indicates such metric value for a mini-batch could help build a good mini-batch during training.

3.2.1 Image Intensity

Our first metric scores an image by summing all of the color channels at all positions. For an image D, with width w and height h, this means that the metric M_1 would equal:

$$M_1 = \sum_{i=1}^{w}\sum_{j=1}^{h}\sum_{k=1}^{3} D_{i,j,k},$$

in which $D_{i,j,k}$ stands for the pixel value at position (i, j) within channel k. As such, given that lower pixel values produce darker colors, and higher pixel values produce lighter colors, images sorted by this metric could be considered to be ranked by intensity. Thus, darker images, particularly those with black backgrounds, would be expected to have the lowest scores, and lighter images, particularly those with white backgrounds, would be expected to have the highest scores.

3.2.2 Color

The second metric we propose scores an image by taking the average color of the entire image. However, this then raises an interesting question: how does one sort by color at all? Of course, the

naive attempt at sorting color is to simply sort by the RGB value. Unfortunately, this produces an unintuitive, noisy and overall poor-looking result. This is due in part to that the RGB color space is appearing unnaturally and unintuitively to humans. Here, we try to choose a more natural-looking color space to remedy this problem. HSV (hue, saturation, value) creates a more intuitive color space by introducing hue, which is the base color, saturation, which makes the base color more or less intense, and then value, which determines how dark the color is. Thus, for this metric, the average color of the image is computed, and then converted to the HSV color space, where the images are then sorted by this value.

3.2.3 Edges

Convolutional Neural Networks (CNNs) contain what are called convolution layers, which form the core building blocks of the network. In short, convolution layers run filters over the image, which produce activation maps based on the the filter's spatial interaction with the pixel intensities. As such, it is expected that pictures with more "edges" will have stronger interactions with the convolution layers. We attempt to test this interaction by sorting images based on the "quantity" of edges in the image. For this third metric, an image D is scored by running a Canny edge detector over the image, and then summing all of the pixels in the result. In particular:

$$M_3 = \sum_{i=1}^{w} \sum_{j=1}^{h} CANNY(D).$$

3.2.4 Perceptual Similarity with Average Image

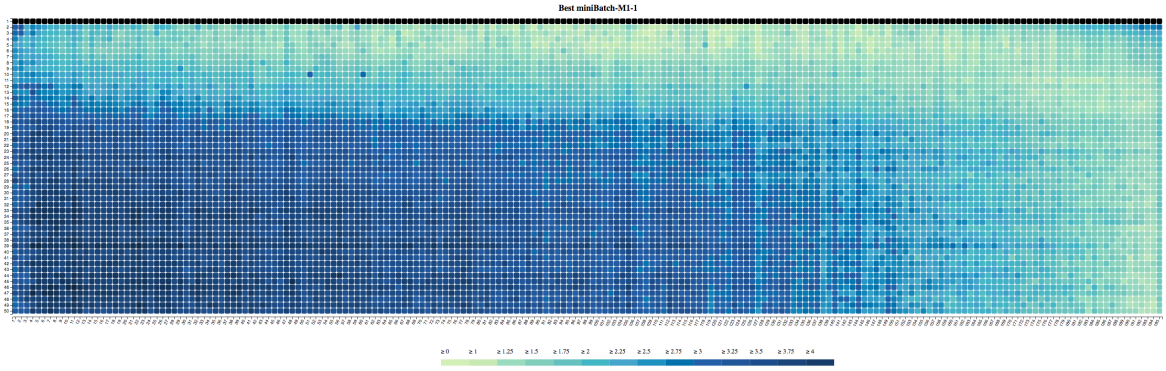
Finally, another metric that we can think of is to compare each image with the average image. Instead of using the Euclidean distance, we plan to apply deep feature based distance metric, which seems to provide more details and high-level insights about the image content. Specifically, the perceptual similarity metric proposed in [23] will be used in our experiments. This metric has a high correlation with how humans would perceive the images as stated in their paper. In our procedure, each image and the average image are fed into their model with the AlexNet [14] backend. Based on their distances, images are then sorted from ones with small distances to large distances. So during one epoch, our model will go through mini-batches of similar images to ones that are quite different from the average image.

4 Experiments and Results

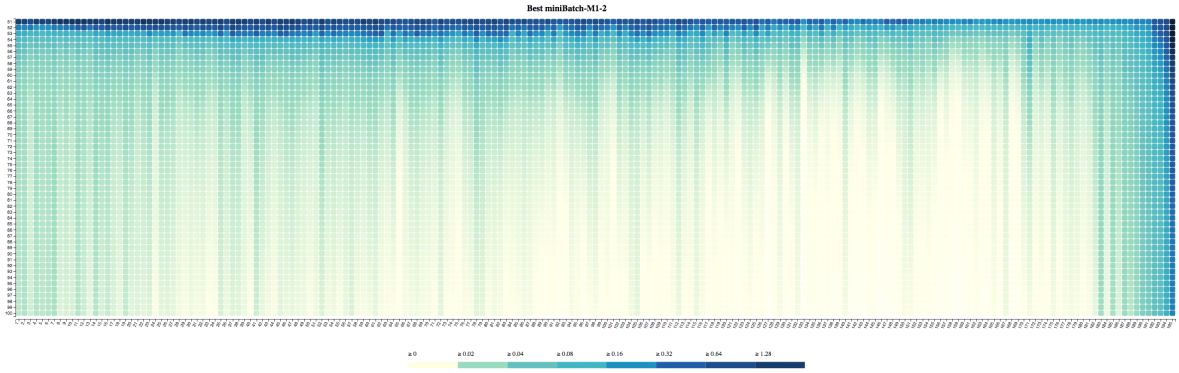
During the experiments, we take the CIFAR-10 [12] image classification as our test vision task, in which we deploy the Dual Path Network [2] as our CNN model. It has 92 layers in total and is currently the state of the art model for image classification on CIFAR and ImageNet. In terms of training phases, we define it based on the learning rate decay strategies as stated before. With a total of 150 epochs, the initial learning rate 0.1 is decayed by 10 after every 50 epochs, which provides us 3 training phases. Our source code is developed based on the CIFAR-10 image classification code on https://github.com/kuangliu/pytorch-cifar. We choose a batch size of 256 to accelerate the training.

Four distance files were generated following the algorithm described above, and each file contains a 150×195 matrix (150 epochs and 195 mini-batches). Each entry represents the distance of one mini-batch's gradient at some epoch to the full-batch's gradient. In order to obtain better insights of the results, we generated heat maps to visualize each matrix. One matrix was divided into three parts: epoch 1–50, epoch 51–100, and epoch 101–150, which constitute our 3 different training phases. Each sub figure shows the heat map of one training phase. Meanwhile, we employed heat maps with color schemes to illustrate the distances in matrix. Light squares represent shorter distance, and dark squares are ones with longer distance. The annotation legend is shown below each corresponding heat map.

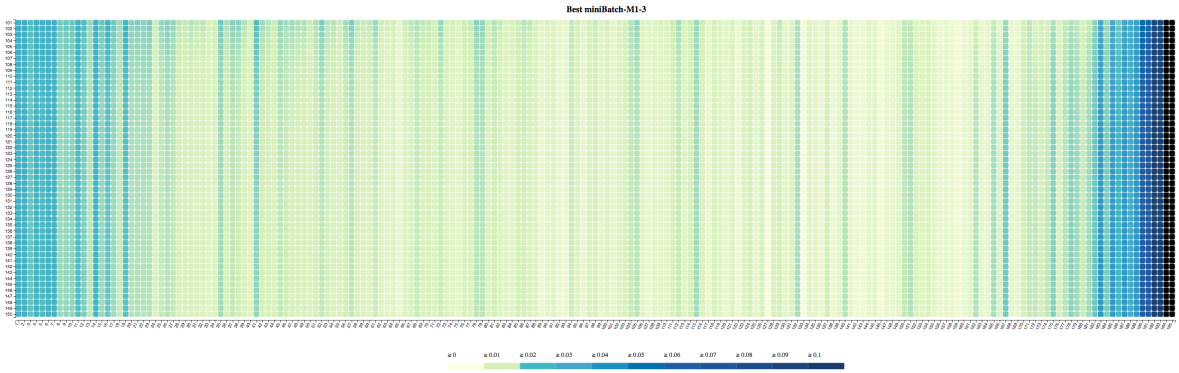
For the results in Figure 1, it displays three heat maps of the distances for metric 1, and the metric 1 was generated based on image intensity. It's obvious that in the early training phase, mini-batches of images with high intensities tend to be much closer to the true gradient. This implicitly indicates selecting high intensity images in the very beginning of the training. However, when the model gradually converges, such mini-batches becomes far away from the true gradient.



(a) Epoch 1 – 50 for Image Intensity Metric



(b) Epoch 51 – 100 for Image Intensity Metric

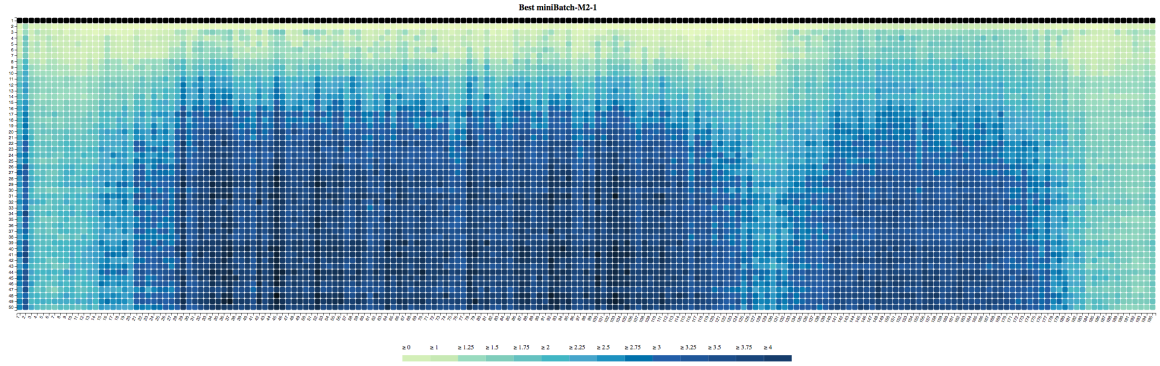


(c) Epoch 101 – 150 for Image Intensity Metric

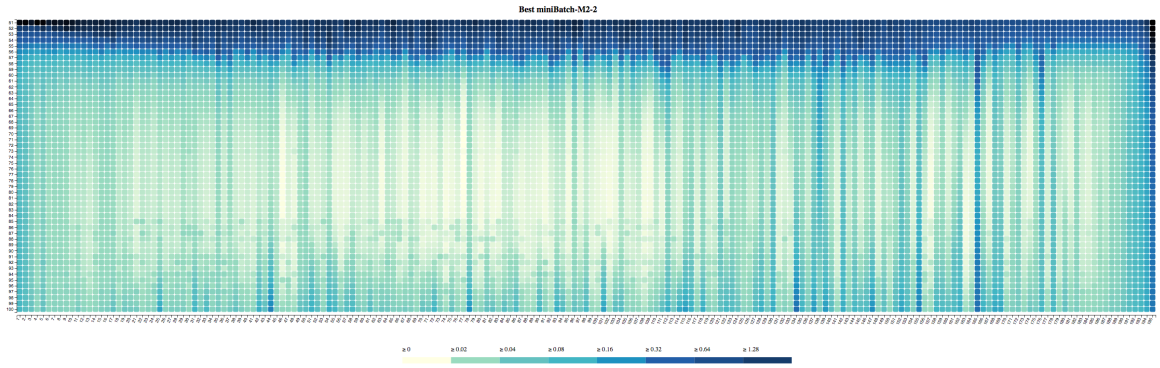
Figure 1: Heat maps for the distances of image intensity metric.

In terms of this, for later training phases, we may need to turn to image batches with intermediate intensities.

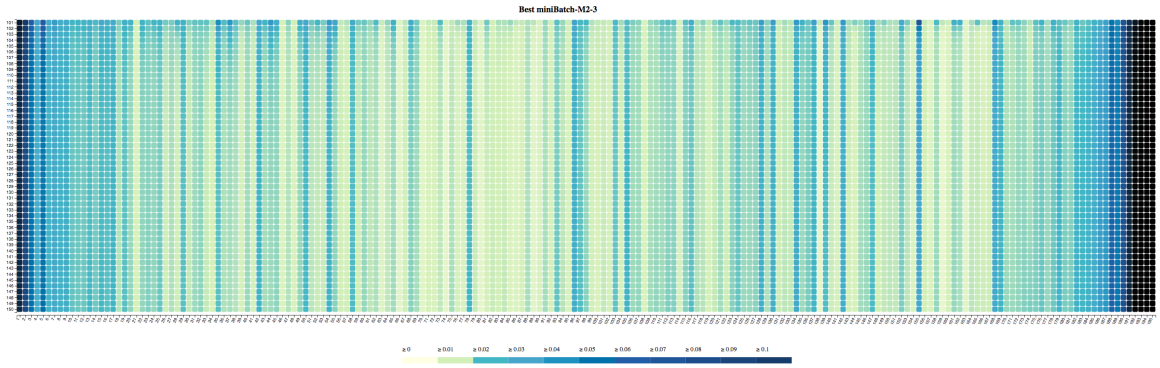
Figure 2 shows three heat maps of the distances for metric 2, i.e., the image color metric. As we can see, during the early training phase, mini-batches with both high and low color metrics are fairly close to the true gradient. But for later training phases, distances of these mini-batches are



(a) Epoch 1–50 for Image Color Metric



(b) Epoch 51–100 for Image Color Metric

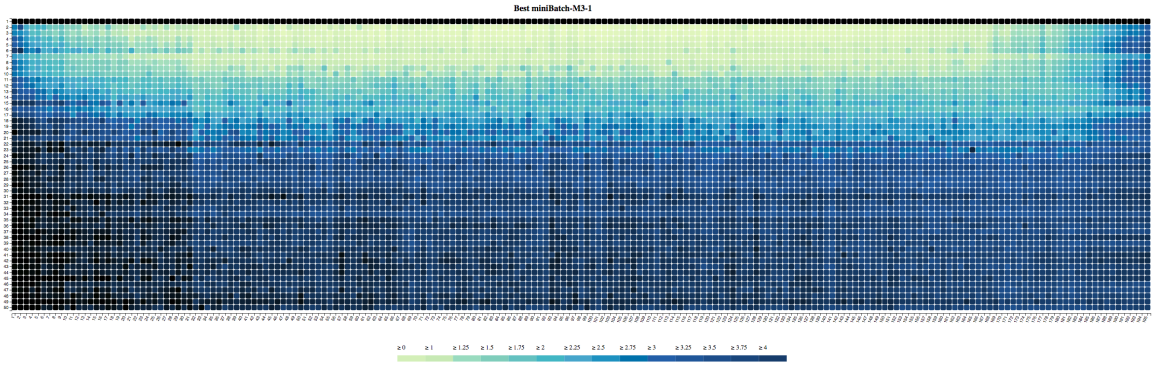


(c) Epoch 101–150 for Image Color Metric

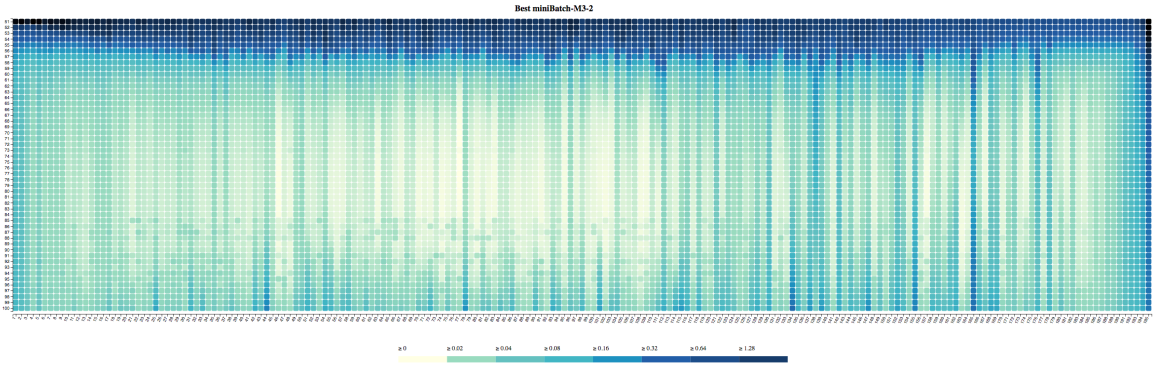
Figure 2: Heat maps for the distances of image color metric.

gradually increasing.

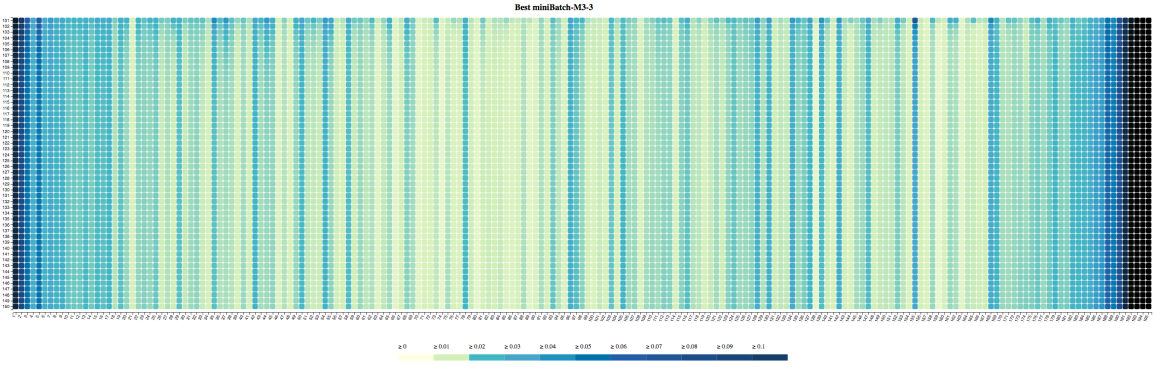
Figure 3 contains three heat maps of the distances of metric 3 reflecting the total edge magnitude of an image. It's pretty clear that in the early phase of training, images with too many or too few edges can't provide good estimation of the true gradients. This is intuitive as such images will either have blank backgrounds or too much details, which could be hard for deep neural networks



(a) Epoch 1–50 for Edge Metric



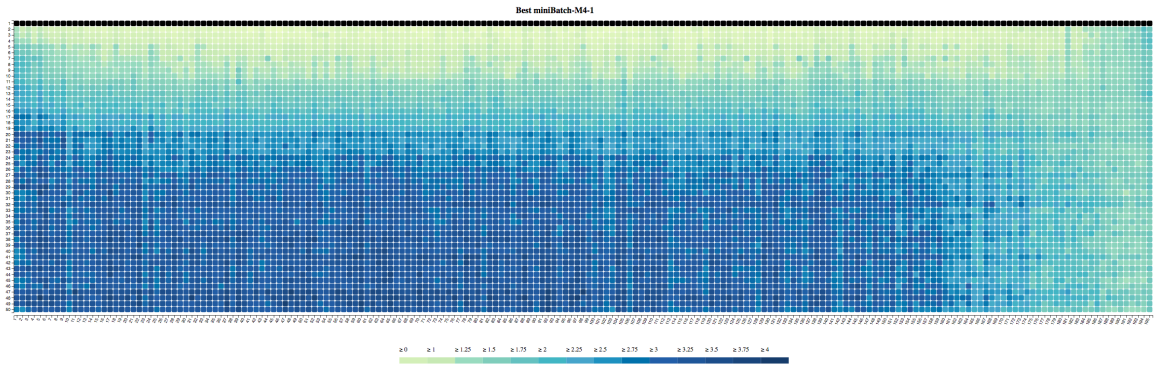
(b) Epoch 51–100 for Edge Metric



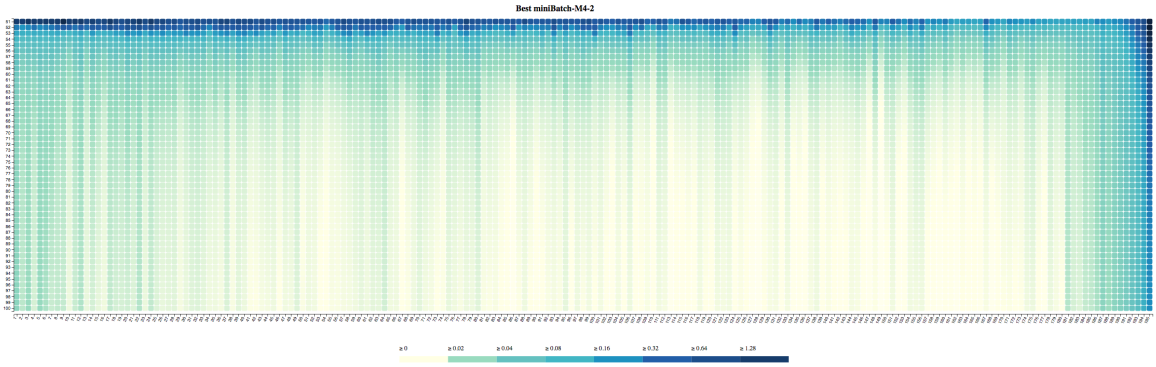
(c) Epoch 101–150 for Edge Metric

Figure 3: Heat maps for the distances of edge metric.

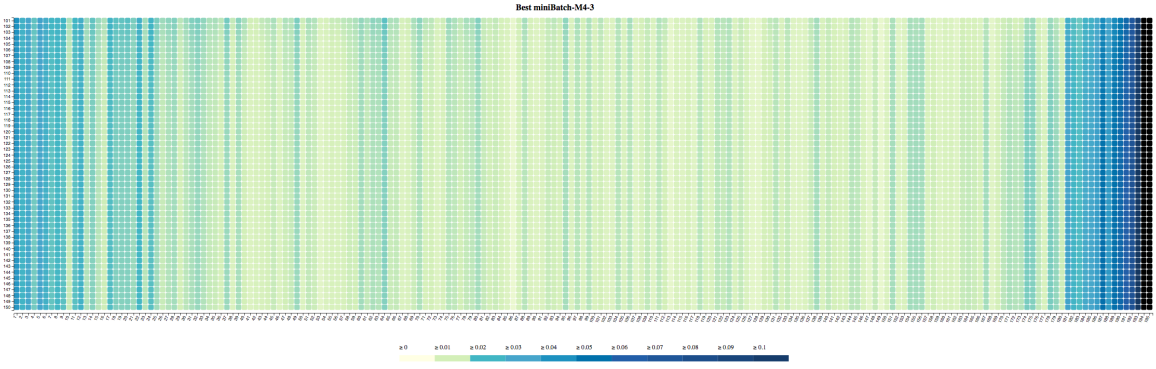
to identify. As the training goes on, in later epochs all mini-batches would share approximately same distances to the true gradient. But as the learning rate is decayed further, the marginal mini-batches are gradually far away in terms of true gradient estimation.



(a) Epoch 1–50 for Perceptual Similarity Metric



(b) Epoch 51–100 for Perceptual Similarity Metric



(c) Epoch 101–150 for Perceptual Similarity Metric

Figure 4: Heat maps for the distances of perceptual similarity metric.

Finally, Figure 4 shows three heat maps of the distances for metric 4, which is generated according to the perceptual similarity. The behavior here is kind of similar to metric 1, which is

show in Figure 1. Mini-batches of images that are very different to the average image should be quite important in the first 50 epochs. Later on, they are bad at estimating the true gradients.

5 Conclusions

All metrics produce different result based on the different training phases. Recall, the learning rate is updated twice, to produce a total of three training phases. For the first training phase, metrics 1 and 4 clearly show that the best results during the first training phase come from mini-batches in the upper quartile. Metrics 2 and 3 have less profound results, but respectively suggest that the best mini-batches are produced from the outside the middle 80 percent of mini-batches, and inside the middle 80 percent of data during the first training phase. For the final two training phases, all four metrics roughly agree that the metric extremes produce poor mini-batches after the first training phase.

Based on our observations from the results, there are two rough conclusions that can be drawn:

1. We suggest that future researchers can produce slightly faster convergence by removing mini-batches with extremely low or high intensities, (HSV) color, edges or deep feature image similarities.

2. We also recommend that future researchers can achieve slightly faster convergence by only considering mini-batches with high intensity images, and with images that are very distinct during the first training phase. They can further prune these mini-batches by selecting those with the highest and lowest color (HSV), and removing those with very few or very many edges.

During this project, we were hoping to be able to empirically show some conclusive metrics for selecting the best mini-batch when training convolutional neural networks. Unfortunately, due to time limits, as well as the limit on access to computational resources, we failed to accomplish the following:

1. We were only able to produce results that are specific to one CNN architecture and, as such, we cannot definitively generalize our results to other CNNs.

2. In short, we were hoping to use our observations to manually construct a subset of the "best" mini-batches and then attempt to leverage our observations to demonstrate that we can converge slightly faster.

3. We were only able to gather results for 4 different metrics. Unfortunately, the process of choosing, implementing and running a metric, as well as analyzing the results, is a work-intensive and time-consuming process.

On the positive side, the shortcomings of this project open nice doorways to potential future work. In particular, we would be particularly interested in seeing:

1. What are the results for other CNN architectures? (e.g., ResNet, DenseNet)

2. Can manually selecting a subset of best mini-batches allow for faster convergence?

3. Are there other heuristic metrics which can produce more profound results?

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, NY, USA, 2004.

[2] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, *Dual path networks*, in Advances in Neural Information Processing Systems, 2017, pp. 4470–4478.

[3] Y. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, CoRR, abs/1406.2572 (2014).

[4] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, 12 (2011), pp. 2121–2159.

[5] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Y. W. Teh and M. Titterington, eds., vol. 9 of Proceedings of Machine Learning Research, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010, PMLR, pp. 249–256.

[6] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, *Accurate, large minibatch SGD: training imagenet in 1 hour*, CoRR, abs/1706.02677 (2017).

[7] M. Hardt, B. Recht, and Y. Singer, *Train faster, generalize better: Stability of stochastic gradient descent*, CoRR, abs/1509.01240 (2015).

[8] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, CoRR, abs/1512.03385 (2015).

[9] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, *Densely connected convolutional networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, vol. 1, 2017, p. 3.

[11] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[12] A. Krizhevsky, V. Nair, and G. Hinton, *The cifar-10 dataset*, online: http://www. cs. toronto. edu/kriz/cifar. html, (2014).

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, USA, 2012, Curran Associates Inc., pp. 1097–1105.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[15] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, *Gradient descent only converges to minimizers*, in 29th Annual Conference on Learning Theory, V. Feldman, A. Rakhlin, and O. Shamir, eds., vol. 49 of Proceedings of Machine Learning Research, Columbia University, New York, New York, USA, 23–26 Jun 2016, PMLR, pp. 1246–1257.

[16] M. Li, T. Zhang, Y. Chen, and A. J. Smola, *Efficient mini-batch training for stochastic optimization*, (2014), pp. 661–670.

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

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, CoRR, abs/1409.4842 (2014).

[19] T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, 4 (2012), pp. 26–31.

[20] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, *The marginal value of adaptive gradient methods in machine learning*, in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, Inc., 2017, pp. 4148–4158.

[21] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, CoRR, abs/1611.05431 (2016).

[22] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, CoRR, abs/1311.2901 (2013).

[23] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The unreasonable effectiveness of deep features as a perceptual metric*, arXiv preprint arXiv:1801.03924, (2018).