# Investigation of Training Order Effects on Artificial Neural Networks for Image Recognition

By

Xiaotian Wang

Thesis
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
in
Computer Science

May 11, 2018

Nashville, Tennessee

TABLE OF CONTENTS

# LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Abstract

The stochastic gradient descent algorithm for training neural networks is widely used in many machine learning, especially deep learning tasks. The stochastic gradient descent algorithm operates by choosing a small fraction of the training data, called a mini-batch, at each iteration to compute an approximation of the gradient of the objective function to be optimized. In practice, researchers tend to use small batch sizes, and the training data fed into the neural network is usually of various categories and is in random order. Researchers have shown the advantages of smaller sizes of mini-batches quantitatively, yet in the past, there were very few formal investigations into the question of how the order of training data would affect the training efficiency and generalizability of the neural network. To gain more insight into this problem, we have investigated effects of training order and the composition of a mini-batch by conducting a series of controlled experiments. In our experiments, we retrained an existing neural network model for object recognition with images from the ImageNet dataset and from a newly-collected dataset called the Toy-Box dataset. We investigated using optimization techniques like genetic algorithms and simulated annealing to optimize the order of training data. Also, we compared training efficiency for different compositions of mini-batches.

## 1.2 Structure of the Thesis

The first chapter of this thesis will cover a brief introduction to the research problem that will be thoroughly discussed in the later chapters and the structure of the thesis. The second chapter will cover some background information and the goal of the research project. In the third chapter, we will do a literature review regarding some previous influential artificial neural network models and discussions regarding optimization algorithms. In the fourth chapter, we will cover the detailed problem statements, experiments completed, and report on the results as well as some data analysis. In the fifth chapter, we will have a conclusion and some discussion on the future research, followed by appendices and a list of references.

# Chapter 2

# Background

## 2.1 A Brief History of Modern Artificial Neural Network

In this section, we will briefly discuss the evolution of artificial neural networks that recently starts to draw a significant amount of attention not only from computer science researchers but also from industries and the general public. There are numerous influential papers regarding the artificial intelligence and neural networks. However, we will only mention some of them in the introduction based on some factors that can reflect their relative impacts, such as the citation number, recommendations, and references made in popular forums and blog posts [1][2][3][4].

The dream of bringing the intelligence to a machine can be retrieved from the early age of modern computing when scientists started to formulate the computing theory and build those gigantic programmable computers. The pioneer of modern computing, Alan Turing, in his famous paper *Computing Machinery and Intelligence*, even provided us a series of criteria to determine whether a machine was intelligent or not [5]. Since then, those criteria, also known as the "Turing Test," were frequently used in the pursuit of machine intelligence.

Before the modern artificial neural network came into being, scientists who were working in this field got most of the inspiration from cognitive systems. During this period, researchers studied brains as well as neurons and tried to mimic the mechanisms behind those complex yet well-organized infrastructures. In the early 1950s, Walter Pitts and Warren McCulloch in their influential paper *A Logical Calculus of the Idea Immanent in Nervous Activity* at first introduced the concept of a thresholded logic unit [6]. While it was not until the late 1960s that another researcher Frank Rosenblatt created the real precursor to the artificial neural networks we are using today, the perceptron. The concept of the preceptron was introduced in Rosenblatt's paper *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain.*, and it was also a biologically inspired algorithm [7]. However, compared to the thresholded logic unit, the perceptron went a step further, since it could learn more complex skills or features quantitatively through trial and error.

In 1986, decades after the introduction of the perceptron, a man called Geoffrey Hinton, together with two other researchers, published a paper *Learning representations by back-propagating errors*. In this paper, Hinton pointed out that people could use back-propagation algorithm to efficiently train networks of neuron-like units, or artificial neural networks as of today. Backpropagation algorithm operates by calculating the derivatives of the cost function of an artificial neural network, and back-propagate errors to adjust the weights of connections between layers in the network. The goal of this algorithm is to quickly optimize an artificial neural network so that the difference between the output of the network and the expected value could be minimized. By adding some "hidden" units that are not visible to both the input and output layers, people could implicitly encode many useful features, thus enabling the network to learn nonlinear functions [8]. Several years ago, with the proof of the universal approximation theorem, people found that the artificial neural network was able to learn not just nonlinear functions but virtually any functions [9] [10].

The backpropagation algorithm was then used to train the convolutional neural network for recognizing handwritten digits, and it started to show the potential in many other tasks since then [11]. However, the real strength of artificial neural networks was hidden due to the lack of computing power and collection of data until a breakthrough in 2012. This time, Geoffrey Hinton, again, showed people that some of the algorithms (backpropagation and gradient descent) used decades ago could also scale well using Graphics Processing Units (GPUs). Hinton used two GPUs to train a rather deep artificial neural network with two other researchers, and their resulting network model (AlexNet) was so influential that almost all the subsequent artificial neural networks were under the enlightenment of the AlexNet [12]. The success of the AlexNet demonstrated not only the importance of combining the power of hardware and software during the training process but also ways of improving the training efficiency.

With the fast iteration of computer hardware and simplification of machine learning related code libraries, more and more researchers in fields of computer science, mathematics, biology, finance and even social science start to build and train their artificial neural networks. Researchers would also spend the time to fine-tune the parameters used in their network models so that they could achieve higher accuracies. In practice, researchers found many rules to help accelerate the training process and prevent the training from several common problems like overfitting, vanishing and exploding gradient [13]. However, compared to the novel applications of the artificial neural network, very few formal investigation has been done in the past for those rules aimed at increasing the training efficiency. One reason is that the mechanisms behind the artificial neural network have not been fully understood so that it is hard to prove the effectiveness of those rules quantitatively. Another reason is that researchers not in the fields of mathematics and computer science tend to spend more time gathering information from various forums and blog posts for improving their network models, yet may not care so much about the validity of the information as long as some improvements are observed.

## 2.2 Goal of the Research

The goal of this research project is to find out how different properties of training data will affect the training efficiency when we are training on some datasets using a specific model (the Inception-v3 model in our case, a pre-trained network model built into the Google's TensorFlow library) for objects recognition. Specifically, we are going to investigate three aspects of the training data fed into the network model and examine their effects qualitatively and quantitatively. First and foremost, we will do experiments to find out, given various training orders, if the network model would prefer a specific order of training data from another regarding the training efficiency, and if it is possible to optimize the training order. Then, we will explore how the composition of a mini-batch (that we will use in the stochastic gradient descent algorithm) would affect the training efficiency. Last but not least, we will examine if the dataset we use would affect the training efficiency (e.g., if we could achieve different training efficiencies when we are using the ImageNet dataset and the Toy Box dataset).

# Chapter 3

# Literature Review: Advancements in Recent Neural Network Architecture and Discussions of Optimization Algorithms

## 3.1 AlexNet

The paper that started a whole new era of machine learning and the artificial neural network is *ImageNet Classification with Deep Convolutional Neural Networks* by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. These researchers used ImageNet dataset, which contains over 15 million labeled images in 22000 categories, and trained a convolutional neural network (AlexNet) on two Nvidia GTX 580 GPUs for five to six days [12].

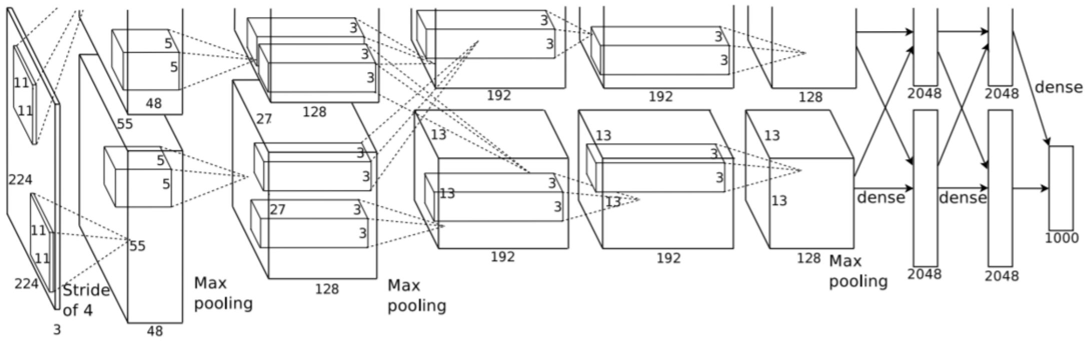

Figure 3.1: AlexNet Architecture

The CNN introduced in the paper contained five convolutional layers and three fully connected layers, shown in Figure 3.1. The final layer used a 1000-way softmax so that the model could calculate a probability distribution over 1000 class labels for prediction. In the paper, researchers used Rectified Linear Units (ReLUs), and the ReLUs could significantly accelerate the training process compared to a traditional network with tanh neurons. Besides, researchers applied data augmentation techniques and dropout layers to reduce overfitting problems that were common in training the convolutional neuron networks. Researchers used a variant of the CNN model in the paper to attend ILSVRC-2012 competition (ImageNet Large-Scale Visual Recognition Challenge) and achieved a winning top-5 test error rate of 15.3% (top-5 test error is the error that the correct label is not within the top five predictions from a model). Compared to 26.2% top-5 test error rate by the second-best model, the AlexNet was a big step ahead.

## 3.2 ZF Net

After the AlexNet, another influential artificial neural network was introduced in ILSVRC-2013, which won the first place with an error rate of 11.2%. This new network (ZF Net) was built by Matthew Zeiler and Rob Fergus by fine-tuning the AlexNet introduced in 2012. Matthew Zeiler and Rob Fergus wrote the paper *Visualizing and Understanding Convolutional Networks*, in which they reduced the filter size from 11*11 to 7*7, and the stride of convolution from 4 to 2, so that relevant information will not be easily skipped in the training process [14].

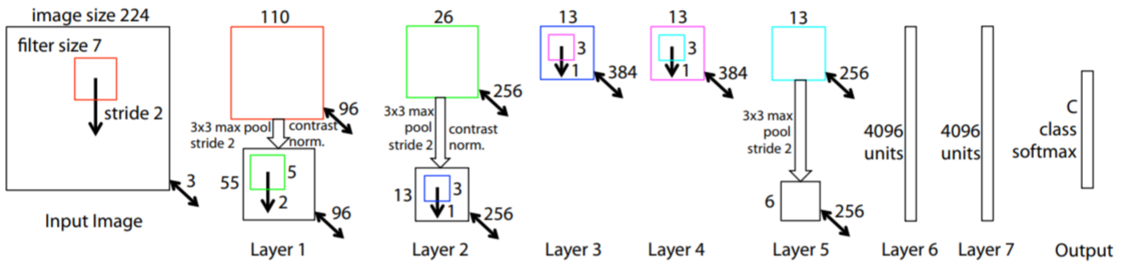

Figure 3.2: ZF Net Architecture

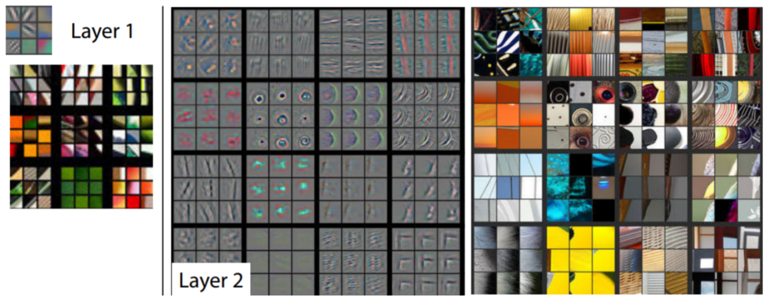

Figure 3.3: Visualizations of Layer 1 and Layer 2

The structure of the ZF Net was quite similar to that of the AlexNet, with some improvement, yet the more important contribution of both researchers was that they found a way (Deconvolutional Network, or deconvnet) to visualize and examine the feature activation. The deconvnet could help researchers not only to understand the infrastructure and mechanisms of a CNN better but also to gain insight for fine-tuning the network.

## 3.3 VGG Net

With the simplicity and depth in mind, Karen Simonyan and Andrew Zisserman built a 19-layer CNN (also known as VGG network, since both researchers were from the Oxford University's renowned Visual Geometry Group). Compared to the AlexNet's 11*11 filters and ZF Net's 7*7 filters, the very deep convolutional network created by two researchers in 2014 used small 3*3 filters and 2*2 maxpooling layers throughout the whole network. However, after each maxpooling layer, the number of filters doubled in the network. By shrinking the spatial dimensions, researchers managed to balance the network by increasing the depth of the network. To train such a deep artificial neural network, researchers used 4 Nvidia Titan Black GPUs, and it took over two weeks to train a single network [15]. It was apparent that with more complex network structures, it would take much longer time and higher-end hardware to train CNNs. The VGG network, although not the winner of ILSVRC-2014 with its error rate of 7.3%, showed us that the hierarchical representation of visual data could work better with a deeper network of layers.

## 3.4 GoogLeNet

In ILSVRC-2014, the first place was taken by the GoogLeNet, a CNN built by researchers from Google Inc., University of North Carolina (Chapel Hill), University of Michigan (Ann Arbor) and Magic Leap Inc. The GoogLeNet had a top-5 error rate of 6.7% in the ILSVRC-2014, and it contained 22 layers, which was even more than the number of layers in the VGG Net.

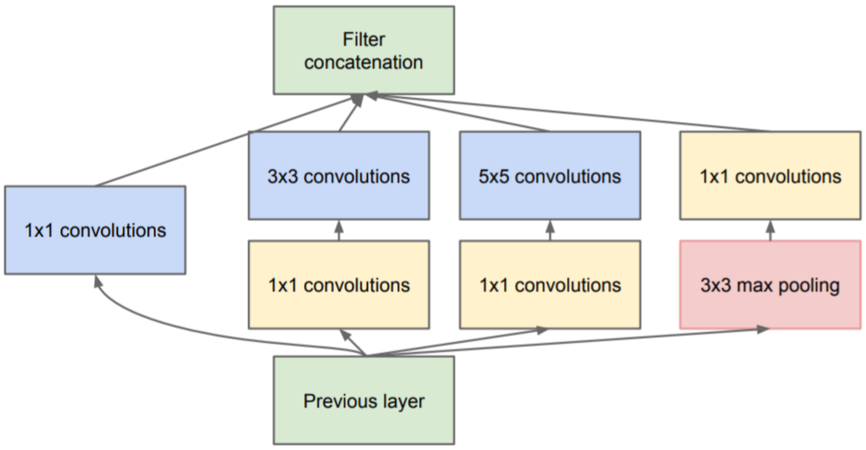

Figure 3.4: Full Inception Module of GoogLeNet

The GoogLeNet introduced the inception module, which, instead of stacking layers of neurons, used a spatially spread-out cluster. The inception module allowed pieces in CNN to operate in parallel. Researchers also solved the problem of having too many outputs from an inception module by applying dimensionality reduction techniques [16]. With the use of inception architectures, researchers were able to add more stages and increase the width of each stage in a CNN without largely increasing the computational resources and complexity. Thus, although the GoogLeNet had more layers than any other CNN models that won the ILSVRC in previous years, the GoogLeNet contains 12 times fewer parameters than the AlexNet from two years ago. Besides, it only took researchers a few high-end GPUs and a week to train the GoogLeNet, a much faster process than the training of the VGG Net.

## 3.5 Microsoft ResNet

In ILSVRC-2015, researchers from Microsoft Research Aisa introduced their incredible deep residual learning framework. An implementation of the residual learning framework was the Residual Network or ResNet, which won the first place in ILSVRC-2015 with a classification error of 3.6%. This new network contained 152 layers and was trained with 8 GPUs in no more than three weeks [17].

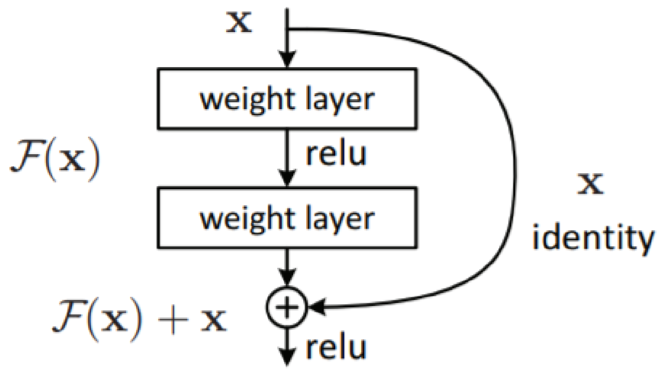

Figure 3.5: A Residual Block in ResNet

Researchers found that by letting stacked layers to fit a residual mapping, instead of the desired mapping, the CNN could get better optimization during the training process, and alleviate the network degradation problems. A building block of the residual network follows a Conv-ReLU-Conv structure, a rather straight-forward and elegant structure. This improvement on the neural networks, with more layers, helped the model to learn much more efficiently, not only on the ImageNet dataset but also on the CIFAR-10 dataset. Researchers also did some experiments and found that stacking layers on top of each other won't provide further improvements on a CNN; on the contrary, an overly-layered CNN would easily suffer from overfitting problems.

## 3.6 Insights into Stochastic Gradient Descent Algorithm

One of the pioneers of the artificial neural network, Yann LeCun, advocated the use of stochastic gradient descent in a chapter of the book *Neural Networks: tricks of the trade.* He systematically provided many heuristics for better training a neural network. One of the heuristics he gave was to shuffle the training data. The intuition behind this tip is that artificial neural networks learn the quickest through the most unexpected examples [13]. During the time LeCun wrote the book, it was still hard to tell whether a training example was information rich or not. Thus, LeCun pointed out that a straightforward, yet effective way was to make sure that the successive training examples are of different classes. Since from the intuition, data from different classes is likely to contain different features and vice versa.

Another researcher who has provided us some insight into the order of training data is Leon Bottou. Bottou did several experiments regarding the convergence speed of the stochastic gradient descent algorithm, in which he compared three different strategies for randomizing the order or training data. The first strategy is called random, and a specific number of training examples are selected uniformly from the entire training dataset. The second strategy is called cycle, and the training examples in each mini-batch are sequentially selected from the shuffled training dataset. The third strategy is called shuffle, and the training examples are selected sequentially from the training dataset, while the dataset is shuffled at each iteration in the training process. Bottou found that the second strategy, cycle, performed the best since the strategy could result in the fastest convergence of the stochastic gradient descent. The random strategy led to a slower convergence, and the shuffle strategy led to a more chaotic convergence, which was also not ideal [18]. The log convergence of the algorithm using various strategies are shown in Figure 3.6. The x-axis counts the number of epoch, and each epoch contains 781265 iterations; the y-axis counts the log convergence rate of the algorithm. Bottou did not, however, complete experiments to compare the speeds of convergence for any specific orders of training data; besides, he did not explore how the convergence speed would be affected without randomization of the training dataset.

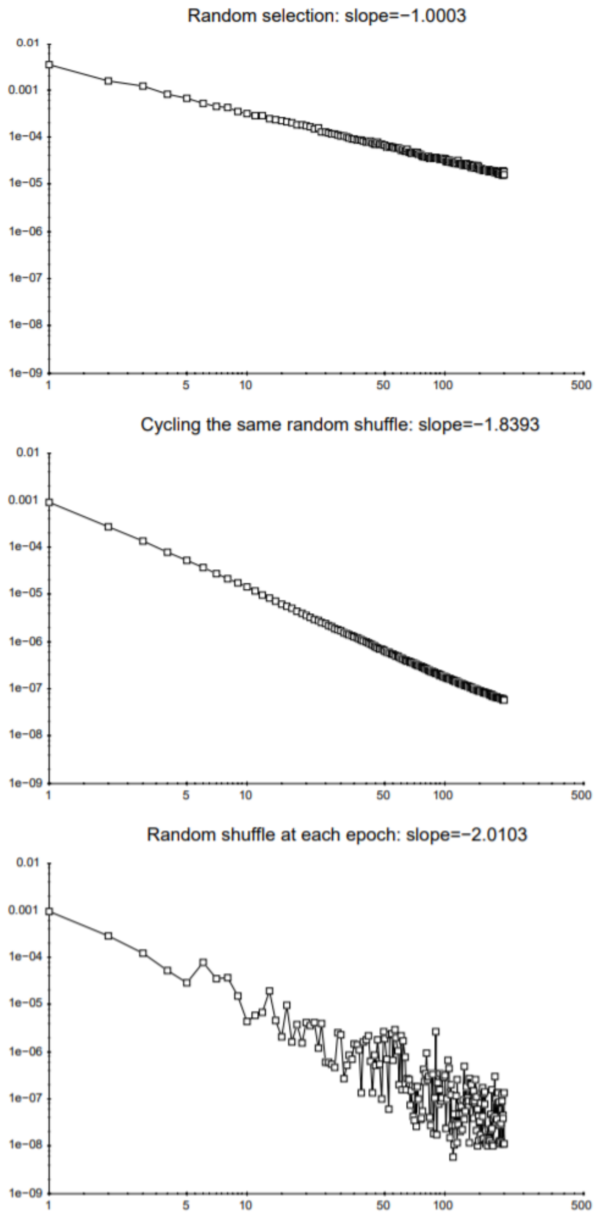

Figure 3.6: Convergence of the Stochastic Gradient Descent Algorithm for Three Example Selection Strategies

# Chapter 4

# Research Questions and Problem Solving

## 4.1 Phase I: Effects of the Order of Training Data on the Training Efficiency

### 4.1.1 Research Problems

Randomizing the order of training data has been considered to be one of the most useful approaches to prevent the training from overfitting. From the mathematical perspective, the cost function we use to measure an artificial neural network in a specific stage of the training process has non-convexity problems. If we do not randomize the order of training data, it is possible that we would finally get stuck in local minimum. Various orders of training data, by the end of a training process, would lead us to different places towards either local minimum or global minimum, resulting in various training and evaluation accuracies. If we define the training efficiency to be the decrease of evaluation accuracy in a unit of time, then, in theory, the order of training data should affect the training efficiency to some extent.

One of the pioneers of artificial neural networks, Yoshua Bengio, and his fellow researchers had the hypothesis that people could guide the training process by changing the order of training samples and dramatically increase the training speed. Researchers did experiments regarding the curriculum learning, in which they compared the speed for training a network for classifying geometric shapes with and without the curriculum learning technique. The result of the experiment showed that by feeding in the basic shapes first and then feeding in the more complex shapes, researchers could achieve a significantly higher training efficiency, compared to feeding in both kinds of shapes randomly (regardless of the complexity) [19]. Some following experiments completed by researchers from the DeepMind even showed the possibility of automating the curriculum learning process (with

LSTM) [20].

In the curriculum learning process, people would qualitatively classify training data into various categories based on complexity, and reorder the training data accordingly. However, MIT researchers found that to evaluate the quality of a particular training sample, they could calculate the performance of a subset of the training data, which includes that particular sample and all samples that have negative impacts on the training performance. This new criterion found by MIT researchers were proved to be very effective for optimizing the training order in the object detection and scene recognition tasks[21].

Although experiments have shown possibilities of reordering the training data based on specific criteria for improving the training efficiency, it is still difficult for us, in practice, to adapt those methods with our datasets and neural network models. One apparent reason is that either dataset we will train on shares different properties and image distributions comparing to datasets used in previous experiments by other researchers. For example, in curriculum training experiments, researchers worked with either moderately simple images (geometric shapes) or even non-images (texts, which should be trained with RNNs). Besides, although PASCAL 2007 dataset used by MIT researchers is similar to the ImageNet dataset, it is still different from the Toy-Box dataset in many ways. Another reason is that it is unclear if the training efficiency improvement would compensate for the time we use to find the optimal order of training data, considering that the reordering of a gigantic dataset itself is time-consuming, let alone the generation of such optimal order. To follow the procedure described by MIT researchers, we not only have to find all samples that could negatively affect the training efficiency at first but also complete an enormous amount of training processes to evaluate the quality of each sample, before we could even start the optimization of training order.

In the first stage of this research project, we will examine how the different order of training data will affect the training efficiency. We will use the stochastic gradient descent algorithm and compare multiple randomized order of training data, and check if any specific ordering is statistically better regarding the training efficiency than the others.

### 4.1.2 Experiments, Results and Analysis

To make the research project feasible, instead of training a CNN from scratch, we will use the Inception-v3 model provided by the Python TensorFlow library. The reason is that the Inception-v3 model has already been trained on the ImageNet dataset for its hidden layers, with the final layers (fully-connected and softmax layers) left out to be trained with our datasets, thus saving the time for experiments. Note that in the Inception-v3 model, all hidden layers together provide us various features of an object and training on the final fully connected layer is to map related features to specific objects. By simplifying the training process, we can save a considerable amount of time comparing to training from scratch, which can take up to two weeks for a single training process.

#### 4.1.2.1 General Hyperparameter Settings

Since the Inception-v3 model was trained with the ImageNet dataset, we decided to at first retrain the last layer of the network model with selected images from the ImageNet dataset. For the ImageNet images we selected, there were 12 categories of objects (airplane, ball, car, cat, cup, duck, giraffe, helicopter, horse, mug, spoon, and truck), and each category contained 1000 images for training and 100 images for evaluating. By completing some experiments, we found that the retraining process was relatively fast with a learning rate 0.1 and a batch size of 200. The resulting network model performed well in the evaluation, as shown in the Figure 4.1. Besides, since we use the mini-batch technique for stochastic gradient descent algorithm, to make sure that all the training data would be fed into the network model in a particular order, we could randomize the order of training data, and sequentially select images from the training dataset into the mini-batch.

To measure the training efficiency of a training process, we would use the value of average evaluation error in a certain number of training iterations as an approximation.

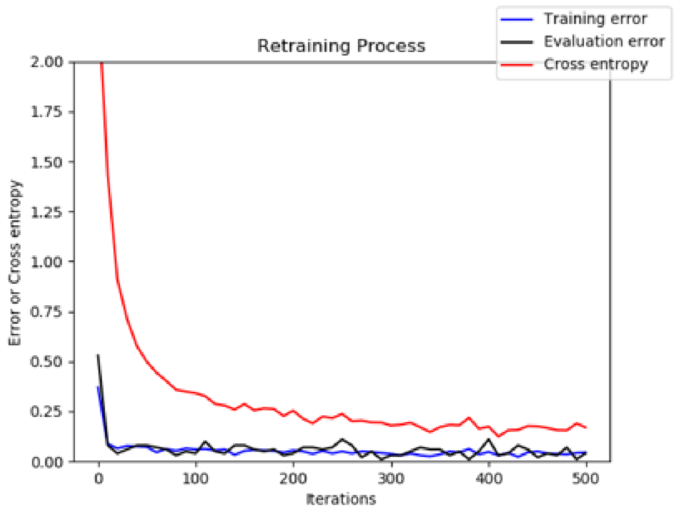

Figure 4.1: A Visualization of the Typical Training Process

With limited time and hardware resources, we would also restrict the number of iterations to retrain the Inception-v3 model to be 200, since most of time both the evaluation accuracy and cross entropy (output of our cost function) would become stable after 200 training iterations. As a result, we could complete as many experiments as possible without losing too much generality.

#### 4.1.2.2 Algorithms for Optimizing the Order of Training Data

Since we assume that order of training data would affect the training efficiency, it would also be reasonable to assume that with various algorithms for optimization used in artificial intelligence, we could gradually improve the quality of the training order, so that the training efficiency would increase over time. In other words, by optimizing the order of training data for a certain number of training processes, we could expect a lower average evaluation error within a training process.

For our control group, we would naively apply the randomization technique for the order of training data, and that is to say, we would randomize the order of training data before each training process begins.

One optimization algorithm we decided to use was the genetic algorithm, which was inspired by the natural selection, a key concept of the modern biology. The natural selection was popularized by Charles Darwin. This concept was elaborated in Darwin's influential book *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life,* and the concept then appeared in the published paper whose coauthor was Alfred Russel Wallace. The genetic algorithm mimics the behavior of the natural genes and follows the initialization, selection, crossover as well as mutation operations [22]. To apply the genetic algorithm in our experiments, as shown in Figure 4.2, we would run a certain number of training process (200 iterations each) each time, and sort their average evaluation error in the ascending order. Then we would pick a certain number of best ordering (e.g., first half), and do pairwise exchanges of some particular fragments of those ordering. For the mutation operations, we would choose a random fragment of each ordering and do extra randomization on the fragment. In practice, the selection operation requires us to complete multiple training processes, which means, for each training process with the random ordering in our control group, we would run multiple training processes using the genetic algorithm. Thus, the genetic algorithm tends to run much slower than our control group.

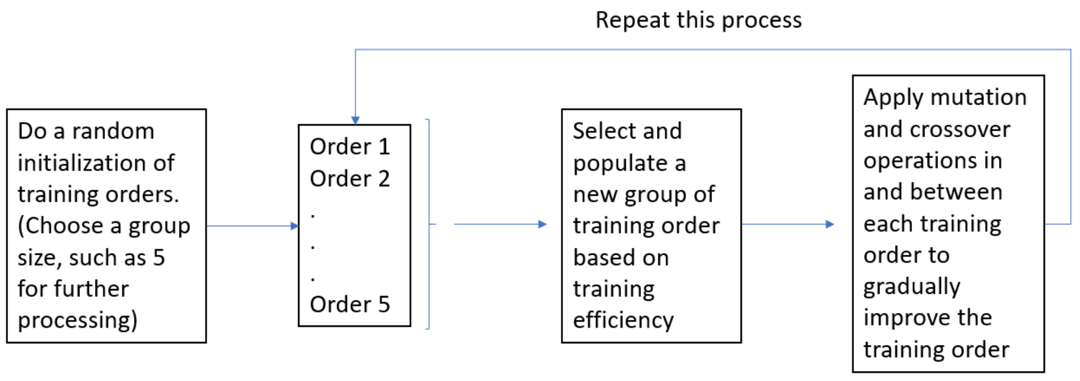
Repeat this process
Do a random initialization of training orders. (Choose a group size, such as 5 for further processing)
Order 1
Order 2
.
.
.
Order 5
Select and populate a new group of training order based on training efficiency
Apply mutation and crossover operations in and between each training order to gradually improve the training order

Figure 4.2: Genetic Algorithm Procedure

Another optimization algorithm we decided to use was a memory-less stochastic modification of the hill climbing algorithm, the simulated annealing algorithm. The simulated annealing is a technique used in metallurgy that alters the physical or chemical properties of a material to reduce its defects. The simulated annealing requires a heat treatment with controlled cooling, and the cooling process is usually interpreted as a gradual decrease in the probability of accepting worse solutions during the global search. This technique is often used as a heuristic to increase the chance of getting out of a local optimum [23]. To apply the algorithm, as shown in Figure 4.3, we should start with a random order of training data, and do randomization fragment by fragment on that ordering. At first, we will manually set a relatively high probability of accepting a worse solution (an order of training data that produces higher evaluation error). Each time we get a better solution, we will accept that better solution. However, if we get a worse solution, we will choose to keep or discard the worse solution based on the acceptance rate, after which we will decrease the acceptance rate until the acceptance rate becomes a small value that closes to zero. When the acceptance rate is so close to zero, or below a certain preset threshold, we will only accept the better solution at each step. With the assumption that the order of training data could affect the training efficiency, we could expect to get a better ordering using the simulated annealing algorithm compared to a random order of the training data.

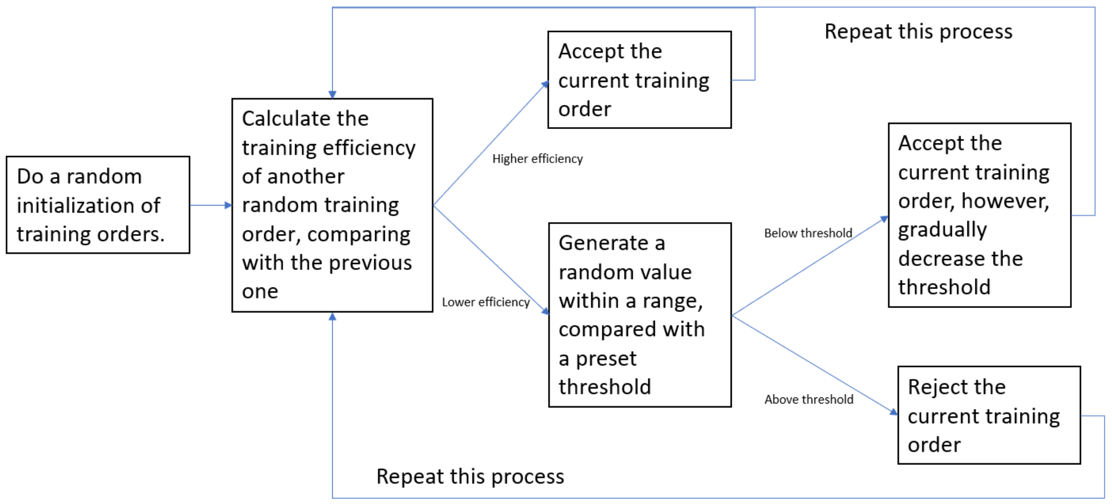
Do a random initialization of training orders.
Calculate the training efficiency of another random training order, comparing with the previous one
Higher efficiency
Accept the current training order
Repeat this process
Lower efficiency
Generate a random value within a range, compared with a preset threshold
Below threshold
Accept the current training order, however, gradually decrease the threshold
Above threshold
Reject the current training order
Repeat this process

Figure 4.3: Simulated Annealing Procedure

In the first series of tests, we assumed that the average evaluation error of a specific training order would be within a Gaussian distribution (which contains all the average evaluation errors a specific training order could generate). We used the ImageNet dataset and tested on the control group (randomizing the training order), the training order optimized with the genetic algorithm and the training order optimized with the simulated annealing algorithm. For these three groups of tests, we ran 500 trials of training processes each and calculated the average evaluation error for each process. During each training process, we ran 200 iterations using the Inception-v3 algorithm, whose learning rate was 0.1 and whose batch size was 200. Besides, to get 500 data points for the training process optimized by the genetic algorithm, since we only recorded the best one training order out of five each time, we will have to run 2500 trials of the training process. After running 2500 trials of training process for the genetic algorithm group, and 500 trials of training process for the other two sets of experiments, we used the linear regression algorithm to approximate the trend of the average evaluation error for each group. If the optimization algorithms are effective, average evaluation errors are expected to exhibit a decreasing trend, as shown in Figure 4.4. Three subplots in Figure 4.5 show the data of average evaluation errors (as black dots) and their trends (as blue lines). Sub-plots from left to right demonstrate average evaluation errors generated with strategies Random Selection, Genetic Algorithm and Simulated Annealing.

The visualization of these results shows that the average evaluation error is neither increasing nor decreasing with and without optimization. One possible reason is that a single average evaluation error of a specific training order may not be enough to reflect the quality of that training order. We may need at least an estimated distribution of average evaluation errors to denote the training efficiency of a specific training order. Another reason is that the optimization algorithm may not be effective for improving the order of training data, or it would require much more training processes for the improvement to be visible in plots, which impairs the purpose of the optimization.

In the subplot for training process optimized by the genetic algorithm, we can see an

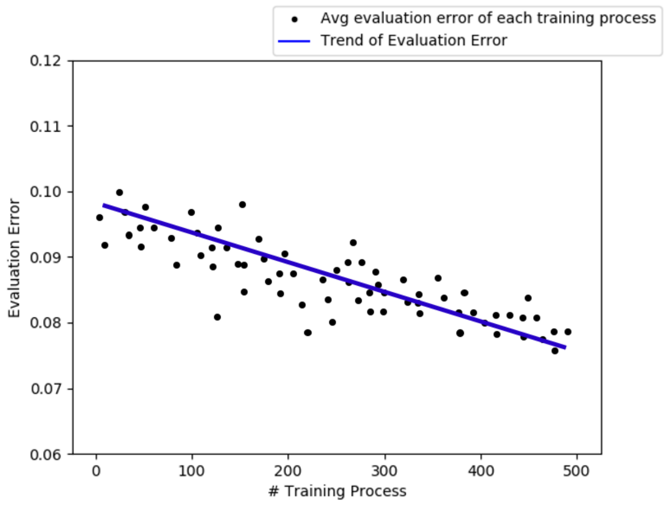

Figure 4.4: Expected Trend for Average Evaluation Errors with Optimization

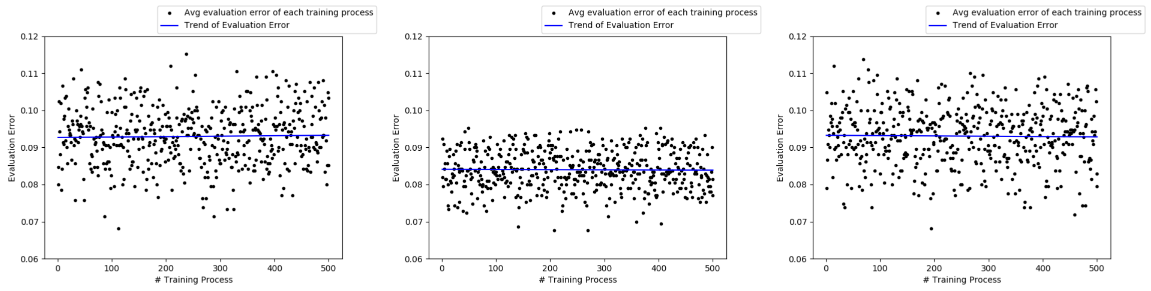

Figure 4.5: Average Evaluation Errors Generated Without and With Optimization on the Training Order

overall lower evaluation error, because each data point for evaluation error gets plot is the best one among the five being calculated. Besides, it took much longer (as much as five times) to optimize the order of training data by the genetic algorithm, which was much longer than the training process without optimization or optimized by the simulated annealing. Thus it is hard to justify if the results from training process optimized by the genetic algorithm is comparable to results from other two groups. To save the time for further experiments, we would only compare results from our control group and the group optimized by simulated annealing algorithm in later trials.

#### 4.1.2.3 Distribution of the Average Evaluation Error

From the previous experiments, we know that both optimization algorithms did not show clear signal that they could gradually improve the order of training data. Thus, our hypothesis might be flawed. Specifically, we need to test if average evaluation errors of a specific training order would fall into a Gaussian distribution and if lower evaluation errors would reflect overall better orders of training data.

We recorded the order of training data corresponding to the lowest and highest average evaluation errors (as shown in Figure 4.6) for both the randomized training order and optimized training order (by simulated annealing algorithm) in the previous experiments and ran 100 trials using each training order. We plot the average evaluation error of each trial and their means across 100 trials for the best and worst order without optimization in Figure 4.7, and with optimization in Figure 4.8. However, just from four plots, we could not tell if a given data point (an average evaluation error) is corresponding to the training order that originally generated the highest or lowest average evaluation error, and if that training order is optimized or not.

Note that in both the Figure 4.7 and Figure 4.8, it is apparent that even we only use a specific training order and fixed hyperparameter settings, we would get various average evaluation errors distributed in a particular range. The result we get is not a fixed number

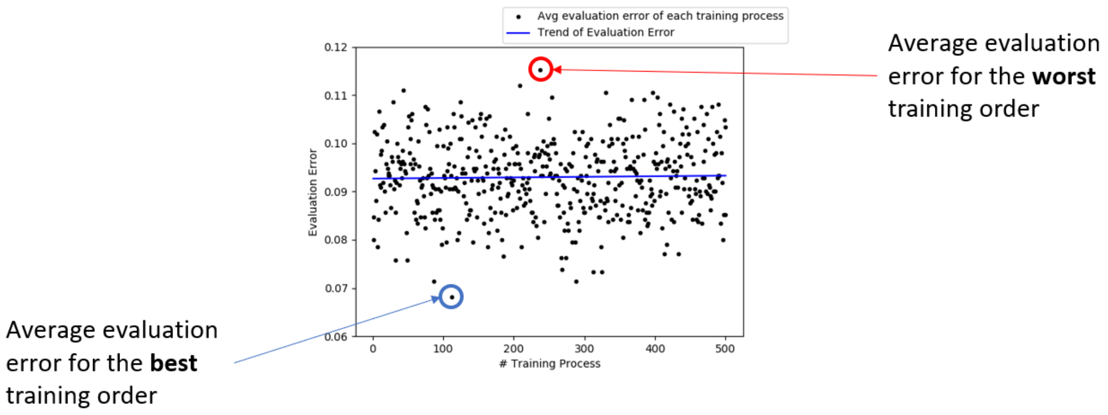

Figure 4.6: The Best and Worst Training order

since the network model would randomly initialize the weights for the final fully-connected layer before training. Thus, even following a fixed training order and updating weights for the final layer of neurons in the same way, with different starting weights at the beginning, we would get different average evaluation errors after each trial of the training process.

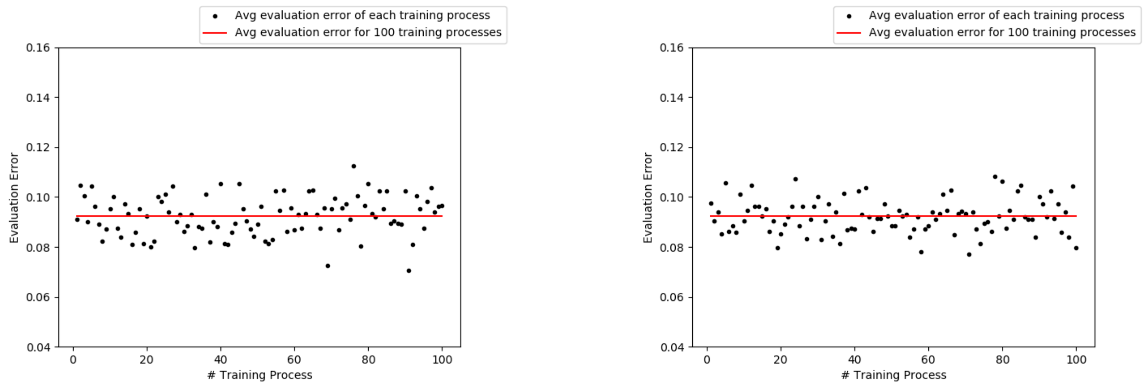

Figure 4.7: Average Evaluation Errors Generated By the Best and Worst Training Order (Random Selection)

We did further inspection of distributions by plotting the probability density estimation curves of average evaluation errors in four experiments done previously (100 trials each) in Figure 4.9. We can see that all four curves, although not perfect, appear to be Gaussian distributions with similar means and standard deviations, and we could not easily distinguish them from each other.

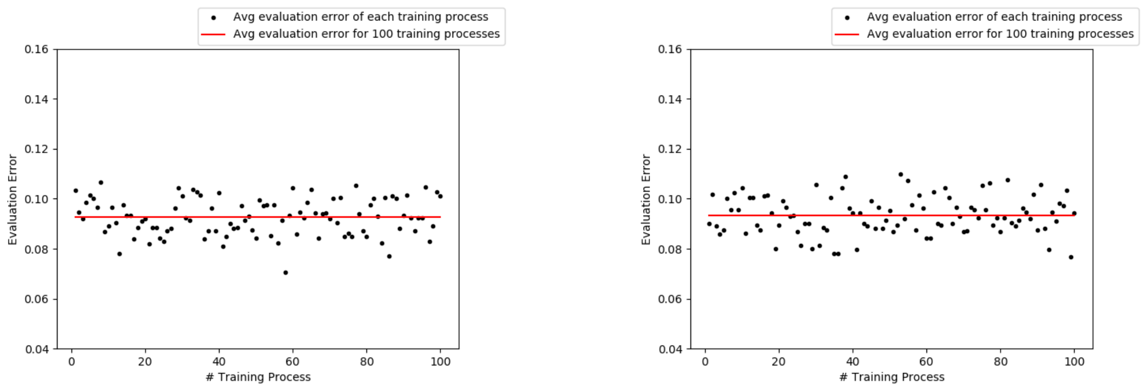

Figure 4.8: Average Evaluation Errors Generated By the Best and Worst Training Order (Simulated Annealing)

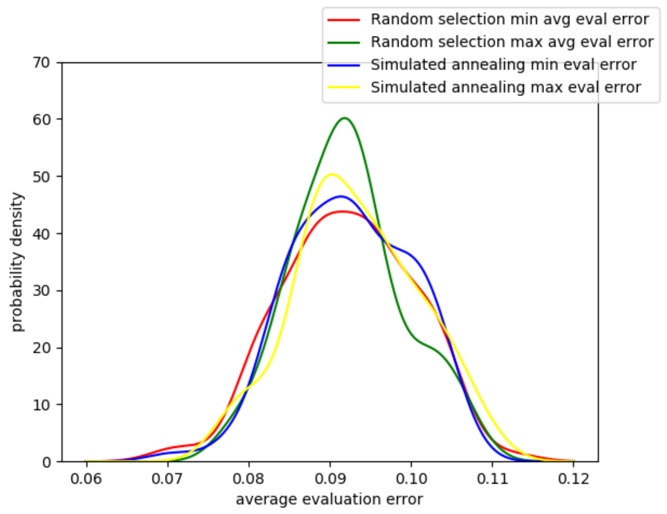

Figure 4.9: Probability Density Estimations of Average Evaluation Errors (Best and Worst Training Order, With and Without Optimization)

Since a specific order of training data might result in a range of average evaluation errors, the training order that happens to generate a single low average evaluation error (high training efficiency) is not necessarily a preferred training order. However, since the average evaluation errors that the order of training data can generate would roughly fall into a Gaussian distribution, we could still expect to see smaller means of such distributions that correspond to our preferred training orders (order that would result in lower average evaluation errors). Thus, in our next experiment, instead of using the training order that led to the lowest and highest evaluation errors, we sorted the orders of training data and picked top 10% and bottom 10% of them (top 50 and bottom 50 orders of training data). We used the training orders that resulted in top 10% and bottom 10% average evaluation errors and re-ran the training process for 50 trials each, and plotted probability density estimation curves of their average evaluation errors, shown in Figure 4.10.

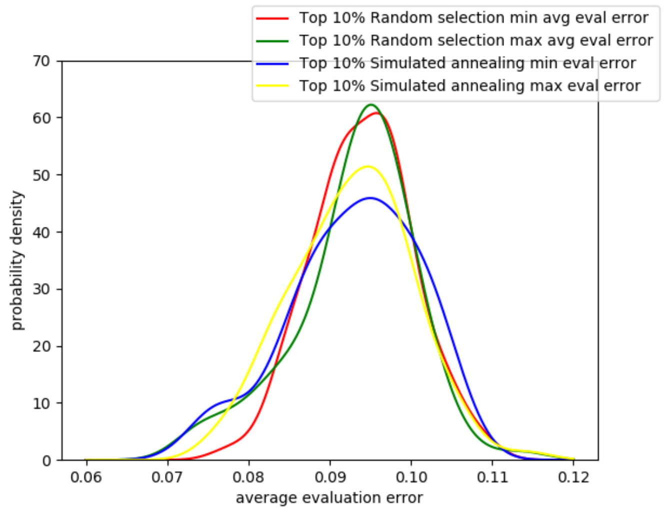

Figure 4.10: Probability Density Estimations of Average Evaluation Errors (Top and Bottom 10 Training Orders, With and Without Optimization)

From the result, we could still see that those distributions of average evaluation errors are roughly Gaussian distributions, and they share similar mean values and standard deviations. Thus, strictly from a statistical perspective, it is highly unlikely that average evaluation errors for the best or worst training orders with or without optimization are from multiple distributions. We are confident to say that the order of training data (the selected subset of the ImageNet dataset) has no noticeable effect on the training efficiency, and the simulated annealing algorithm could not provide an obvious improvement to the training order.

#### 4.1.2.4 Effects of Learning Rate and Batch Size

Our previous experiments showed that the order of training data has almost no effects on the training efficiency, and an algorithm like the simulated annealing cannot help to improve the training efficiency by optimizing the training order. However, in previous experiments, to better understand factors that would affect the training process, we used the average evaluation errors to measure the training efficiency and visualized the distribution of the average evaluation errors. It turned out that those methods were quite helpful.

We used the same technique and did the visualization to explore how the learning rate and batch size would affect the training efficiency. By randomizing the order of training data each time, we completed 1000 training processes for the learning rates 0.1, 0.4 and 1.6 respectively with a fixed batch size 200 and ran 200 iterations in each training process. As we can see from results in Figure 4.11 and Figure 4.12, with higher learning rate, we would get larger average evaluation error (or lower learning efficiency). The resulting distribution of average evaluation errors moves to the right as we increase the learning rate and the standard deviation of the distribution curve also becomes larger. Besides, we can see that the peak of the distribution estimation curve becomes lower as we increase the learning rate, which means that there are less average evaluation errors close to the mean of that distribution.

We completed 1000 trials of training processes using randomized training order for the batch size 200, 400 and 800 respectively, with a fixed learning rate 0.1 and ran 200 iterations in each training process. This time, from the results in Figure 4.13 and Figure 4.14, we found that with the same learning rate and the number of training iterations, we could get smaller average evaluation error (or higher training efficiency) with reasonably

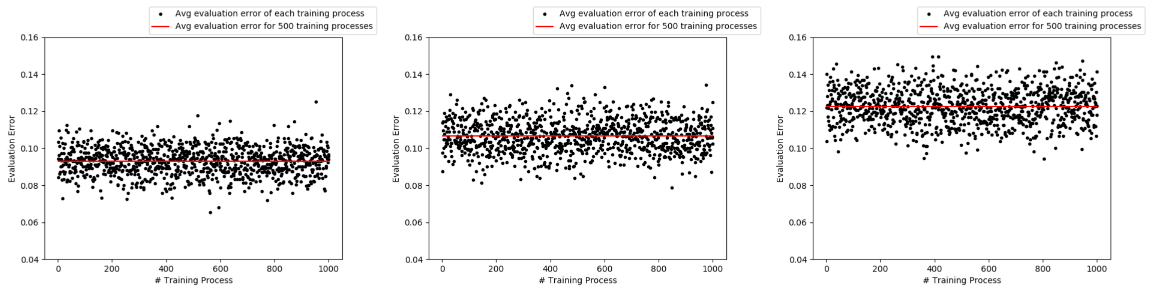

Figure 4.11: Average Evaluation Errors Generated by Random Training Orders with Various Learning Rates

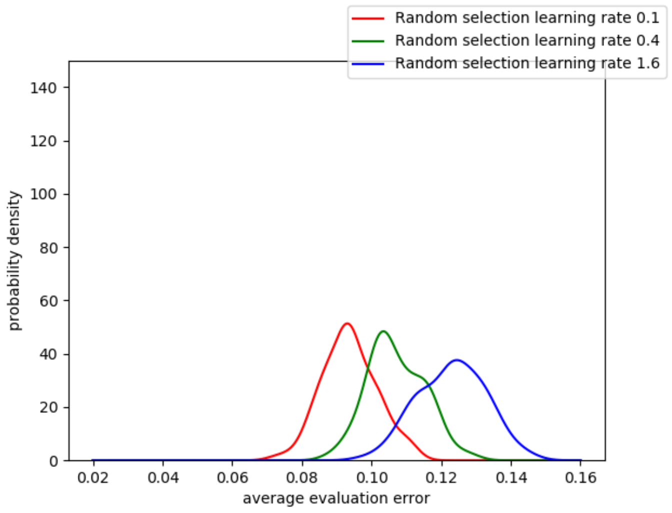

Figure 4.12: Probability Density Estimations of Average Evaluation Errors by Random Training Order with Various Learning Rates

larger batch sizes. In the plot for the distribution of average evaluation errors, we could see that the distribution curve would move to the left as we increase our batch size. This time, the standard deviation of the distribution curve would largely decrease as we increase the batch size. It is even more apparent that the peak of the distribution estimation curve becomes higher as we increase the batch size.

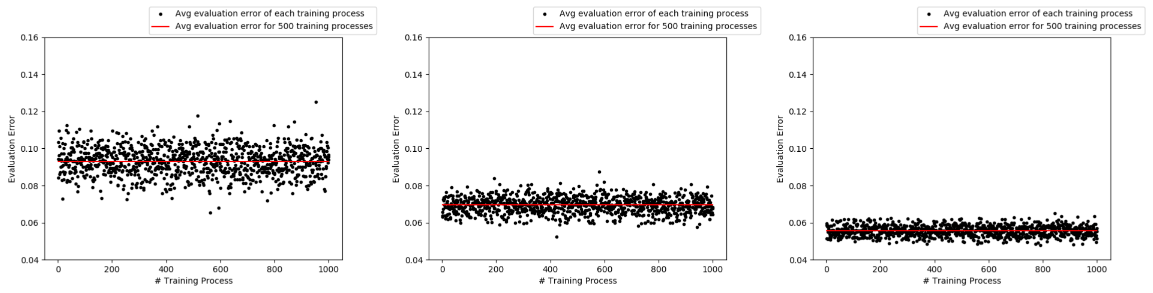

Figure 4.13: Average Evaluation Errors Generated by Random Training Orders with Various Batch Sizes

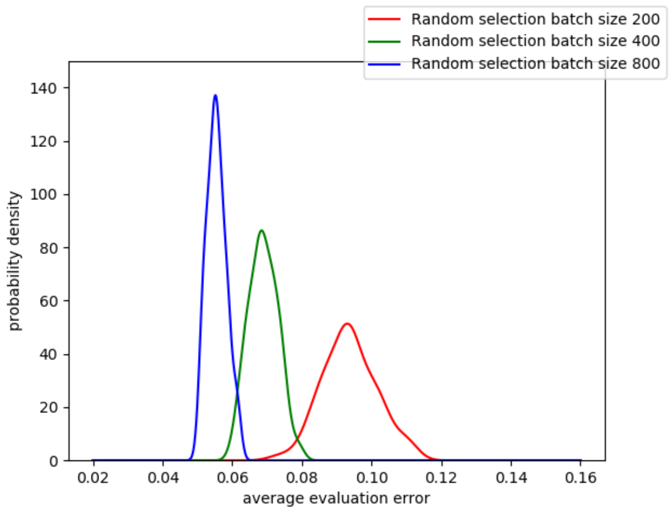

Figure 4.14: Probability Density Estimations of Average Evaluation Errors by Random Training Order with Various Batch Sizes

In all of our experiments above, we could observe that the standard deviation of a specific distribution (of average evaluation errors) is smaller when the mean of that distribution is also smaller, and this happens when we are using smaller learning rates or larger batch sizes while keeping other settings the same during the training process. This property of the result can be explained by the way we measure the training efficiency. For better explanation, we will divide a training process into two stages. In the first stage, the evaluation error drops quickly, and variances of evaluation error at each training step between different trials are large. In the second stage, the evaluation error would fluctuate within a small rage, and variances of evaluation error between different trails are relatively small compared to the first stage. It is apparent that within a certain number of training iterations (200 iterations in our case), the faster our evaluation error drops, the faster our network model can reach the stage where the evaluation errors become relatively stable. Thus, if it only take very few training iterations to reach the second stage of a training process, the evaluation errors within the second stage would contribute more to the average evaluation error of this training process. In this case (when we are using larger batch sizes or smaller learning rates, so that the evaluation error drops quickly during the first stage of training), by performing multiple trials of training process, we will have smaller variances between each trial, resulting in a smaller standard deviation in the distribution. Note that if we are using other criteria to measure the training efficiency, the plotting for the training efficiency distribution might show different properties.

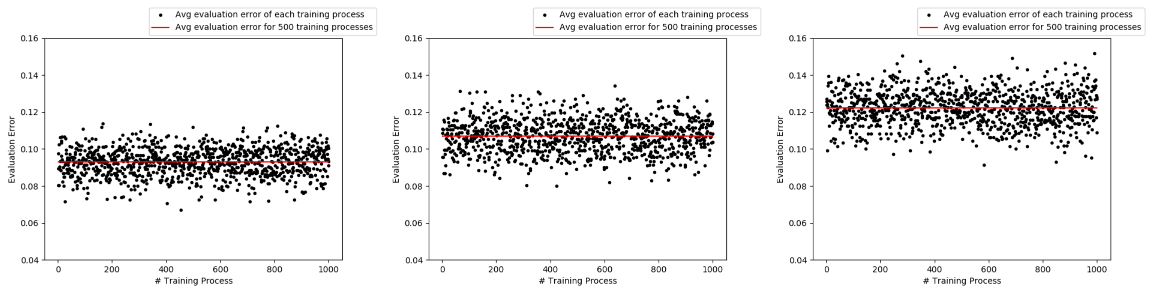

Figure 4.15: Average Evaluation Errors Generated by Optimized Training Orders with Various Learning Rates

With the same settings, we completed another two series of experiments, in which we used the training order optimized with simulated annealing algorithm and we examined how the learning rate and batch size would affect the training efficiency (average evaluation error). The visualization of the average evaluation errors and their distributions are

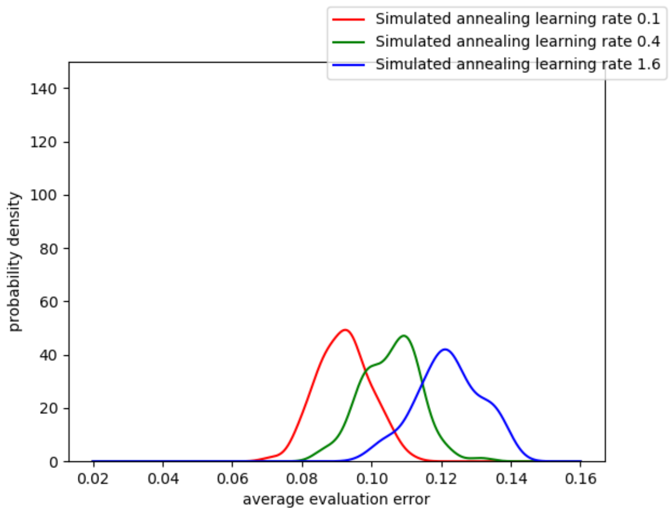

Figure 4.16: Probability Density Estimations of Average Evaluation Errors by Optimized Training Order with Various Learning Rates

shown in Figure 4.15 to Figure 4.18. From the result, we can see that the learning rate and batch size would affect the average evaluation errors the same way as they did without the optimization of the training order.

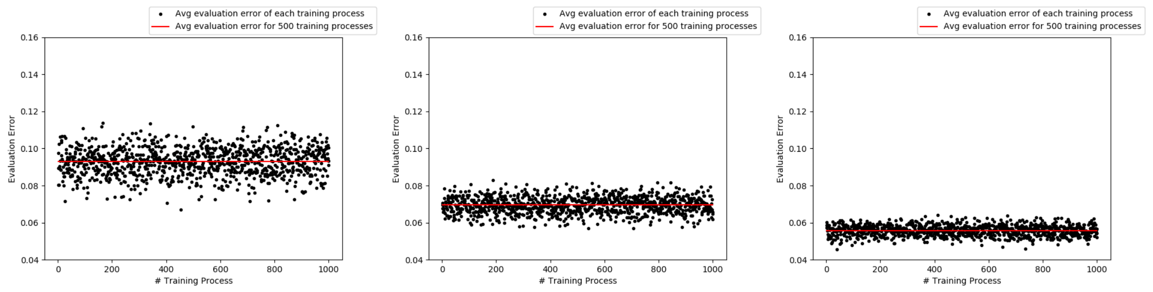

Figure 4.17: Average Evaluation Errors Generated by Optimized Training Orders with Various Batch Sizes

#### 4.1.2.5 Effects of Training Order on Toy-Box Dataset

Besides the data we selected from the original ImageNet dataset, we also used a newly collected dataset called Toy-Box to verify if the order of training data would affect the training efficiency. The Toy-Box dataset, which contains a collection of first-person s, is used to learn how the human beings would observe objects for image recognition tasks.

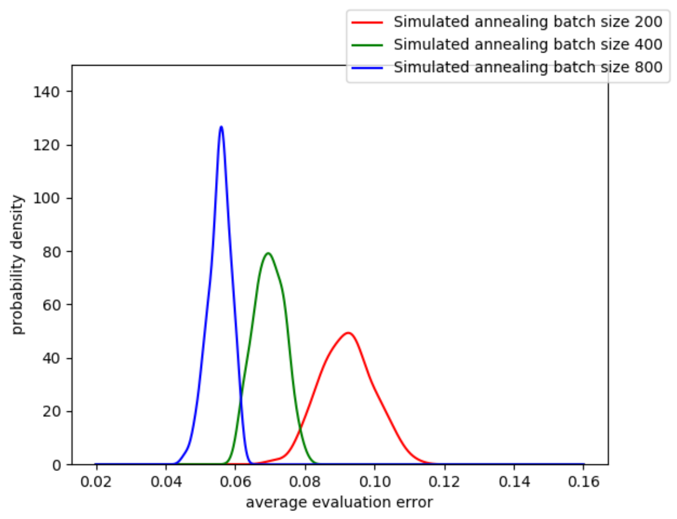

Figure 4.18: Probability Density Estimations of Average Evaluation Errors by Optimized Training Order with Various Batch Sizes

Researchers used wearable cameras to record videos in first-person views that contain different objects and gathered them into this new dataset [24]. With different frequencies of capturing frames from videos, researchers could get various numbers of images per object. Different from the images in the ImageNet dataset, which contains various objects with utterly different backgrounds, the Toy-Box dataset may contain many images of the same object in similar backgrounds. Both datasets contain 12 categories of objects. However, the Toy-Box dataset contains only 15 objects (each object appears in over 170 images in the dataset) for each category, while the ImageNet dataset we used contains 1000 objects per category. Given the difference in composition between two datasets, we decided to explore if the order of training data would also not affect the training efficiency using non-typical Toy-Box dataset.

To make the experiments comparable, we selected 1000 images per category for training and 100 images per category for validation from the Toy-Box dataset. We did this by holding 3 objects in each category to be used for validation, and the left 12 objects in each category for training. Besides, we did make sure that the number of each object used in the training and validation sets to be roughly the same. We ran 500 trials of the training

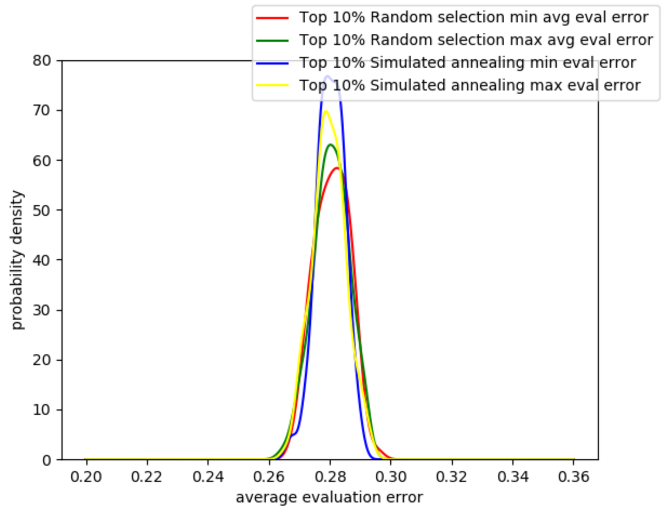

Figure 4.19: Probability Density Estimations of Average Evaluation Errors by Optimized Training Order with Various Batch Sizes (Toy-Box Dataset)

process with and without optimization by the simulated annealing and recorded training orders that resulted in the top 10% and bottom 10% of average evaluation errors (with and without optimization). Then we re-ran the training process 50 trials for each training order and plot probability density estimation curves of their average evaluation errors, shown in Figure 4.19. From the result, we can see that that all four curves (top 10% and bottom 10% training orders with and without optimization) denote Gaussian distributions with similar means and standard deviations, and we could not easily distinguish them from each other. Just like the result from the previous experiments in which we used images from the ImageNet dataset, we found that the order of training data would not affect the training efficiency. Besides, since it is hard to tell if a specific training order is better than another one, it is also hard to verify if we could optimize a training order or not.

## 4.2 Phase II: Effects of the Mini-Batch and Training Data Composition on the Training Efficiency

### 4.2.1 Research Problems

In Phase I of our research project, we explored how the order of training data would affect the training efficiency, in which we used the stochastic gradient descent and back-propagation algorithm to optimize weights of connections between layers in the artificial neural network. In Phase II, we will focus on the problem of how the composition of a mini-batch used in stochastic gradient descent could affect the training efficiency.

Stochastic gradient descent is a variant of the gradient descent, and another variant of the gradient descent is batch gradient descent. Batch gradient descent calculates the gradient using the entire dataset, while stochastic gradient descent calculates the gradient using a single or several samples from the dataset at a time. Stochastic gradient descent algorithm is empirically and theoretically better than batch gradient descent algorithm, and in practice, researchers use the mini-batch technique for stochastic gradient descent almost all the time. Batch gradient descent algorithm would drive the gradient more directly towards the global or local minimum, yet it takes too much time to apply. Stochastic gradient descent algorithm tends to drive the gradient in a zigzag manner towards the global or local minimum. However, an advantage of the stochastic gradient descent is that, with proper parameter settings, it can drive the gradient out of a local minimum.

For stochastic gradient descent algorithm, both the size and composition of a mini-batch can affect the training efficiency. In the past, it has been observed that the large batch size could cause significant degradation in the quality of a network model. With a series of experiments, researchers also showed numerically that large-batch methods would cause the network model to be lack of generalizability[25]. As a result, an inferior network model trained within a certain number of iterations indicates a lower training efficiency. For the composition of a mini-batch in the stochastic gradient descent algorithm, researchers showed convergence speed difference when using different methods for randomizing the training data in a mini-batch [18]. However, no experiments have been done systematically to investigate the effects of not using randomization in the stochastic gradient descent algorithm.

In the second stage of this research project, we will investigate qualitatively how a mini-batch can harm the training efficiency without data randomization. Specifically, we will design various compositions of a mini-batch and examine how different compositions of a mini-batch could affect the training efficiency using both the ImageNet dataset and Toy-Box dataset. Besides, we will explore if the different compositions of the training data, which would explicitly affect the composition of a mini-batch, would indirectly affect the training efficiency.

### 4.2.2 Experiments, Results and Analysis

#### 4.2.2.1 Three Different Compositions of Mini-Batch

To test how compositions of mini-batches would affect the training efficiency, we designed three different modes to form a mini-batch.

The first mode is called uniform mode. In each mini-batch, we will select the same number of images from each category, either sorting them based on particular criteria or keeping the order inside a mini-batch randomized, as shown in Figure 4.20 (a specific color denotes a specific category of objects). For example, if the size of a mini-batch is 100, and we have 10 categories of objects, then we will select 10 objects from each category for each mini-batch. To make sure that every single image in the training dataset will be used in the training process for the same number of times, we will at first randomize the order of images for each category, and sequentially select the same number of images from each category. Besides, we also have to carefully calculate and choose the proper batch size as well as the number of images used in training for each category.

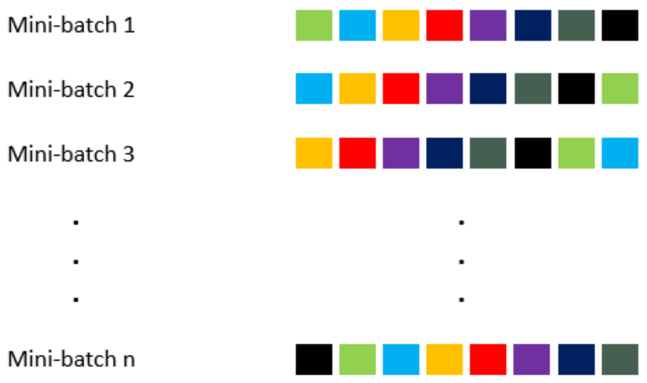

Figure 4.20: Uniform Mode

The second mode is called alternate mode. Images inside a mini-batch are of the same category. However, we will change to another category of objects in the subsequent mini-batch, as shown in Figure 4.21. For example, if the size of a mini-batch is 100, then at the first step of training, we will feed in 100 images of category A; at the second step, we will feed in 100 images of category B, and so on. For this kind of composition, we also have to apply similar method mentioned above to make sure that every single image in the training dataset will be used for the same number of times.

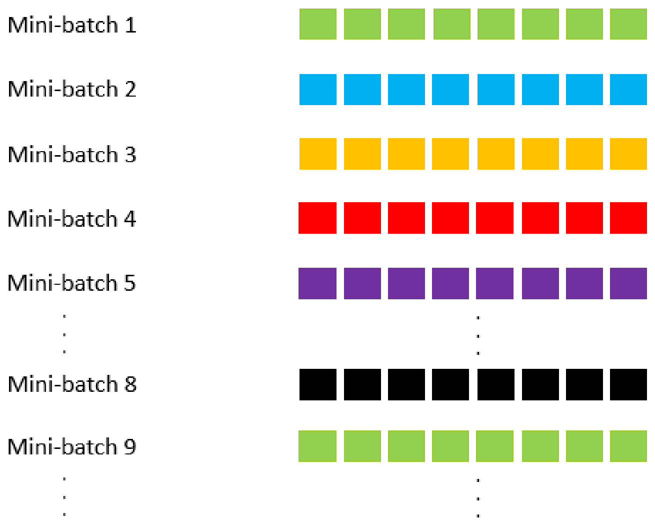

Figure 4.21: Alternate Mode

The third mode is called sequential mode. Images inside a mini-batch are of the same category, and we will sequentially pick images from a particular category before we start to use images from the subsequent categories, as shown in Figure 4.22. For example, if the size of a mini-batch is 100, and each category contains 1000 images, we will keep using 100 different images of the category A at each step, until we have used all images from the category A (which will take 10 iterations). Then we will start to use images from category B, and so on. For the third kind of composition, we also have to make sure that every single image in the training dataset will be used for the same number of times during the training process.

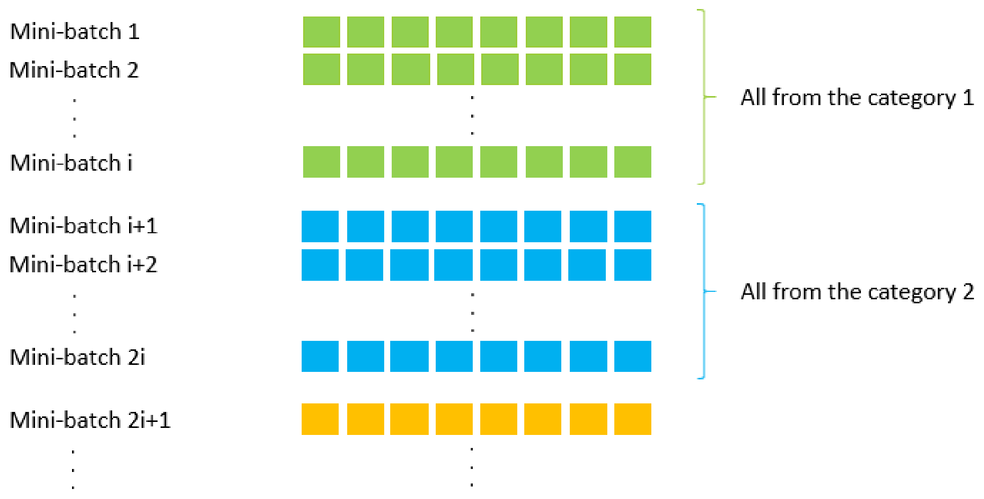

Figure 4.22: Sequential Mode

### 4.2.2.2 Effects of Mini-Batch Composition

In our experiments for exploring how the composition of a mini-batch would affect the training efficiency, we will train the Inception-v3 model on selected images from the Toy-Box dataset, and evaluate on both selected images from the Toy-Box dataset and ImageNet dataset. Specifically, for the training dataset, we will select 1080 images per category from the Toy-Box dataset. Each category will contain 12 different objects, and each object will appear in 90 different images in the training dataset. For each category, 3 objects, other than those 12 objects used for training, will be used for evaluating the network model. We will make sure that the number of images in the evaluation set to be 100 per category, and keep the number of images for each object in a category roughly the same (about 33 images per object).

For the training process, we will use a batch size of 45, a learning rate of 0.01, and 3600 training iterations in total, so that in every experiment to be completed, every single image in the training dataset will be used for the same number of times. Besides, during each training step, we will calculate the training error right before we feed a mini-batch of data into the network model to train so that we could know how the current model would perform on the current batch of data before the model is overfitting after the current step of training.

We plot curves of the training error, evaluation error (from both the Toy-Box and ImageNet dataset), and the cross-entropy for each mode of mini-batch composition in Figure 4.23, Figure 4.25 and Figure 4.27. For each mode, we also trained for another 240 iterations, so that we can get three extra plots to show more detailed trends of curves, as shown in Figure 4.24, Figure 4.26 and Figure 4.28.

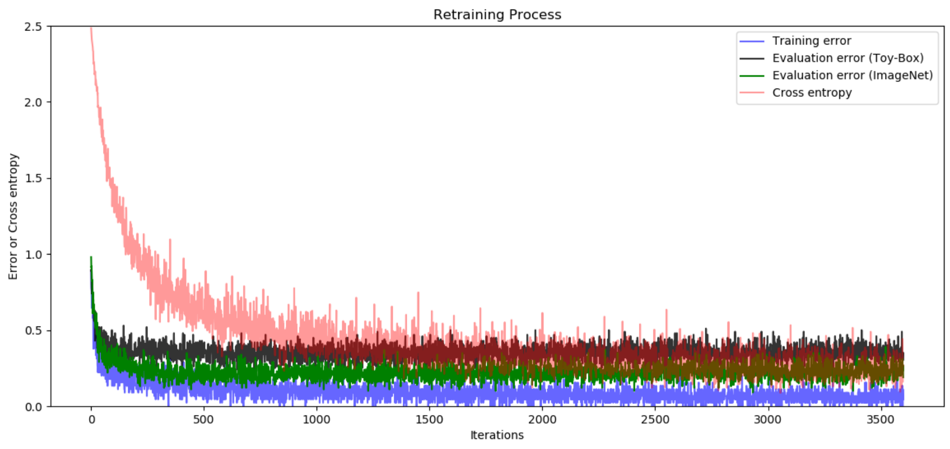

Figure 4.23: Training Process (Uniform Mode, 3600 Iterations)

Figure 4.23 and Figure 4.24 show training process when we were using the uniform mode for the batch composition. We can see that the training error, evaluation error, and the cross-entropy drop quickly during the first few iterations, and then slowly decrease,

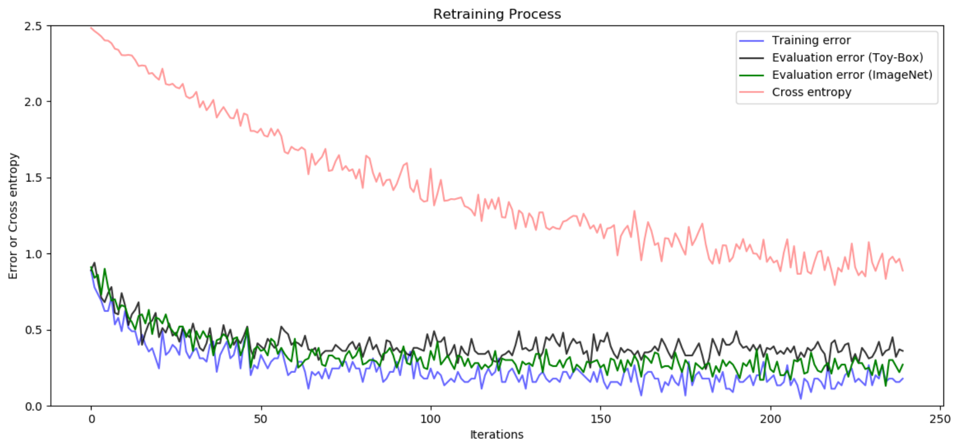

Figure 4.24: Training Process (Uniform Mode, 240 Iterations)

until the evaluation error of both the Toy-Box and ImageNet dataset reach 0.4 and 0.3 respectively. The convergence of the evaluation curves is quite chaotic, and it seems that at the later part our training process, the training accuracy of the network model is not further improved, or the improvement is too small to be visible from the plot.

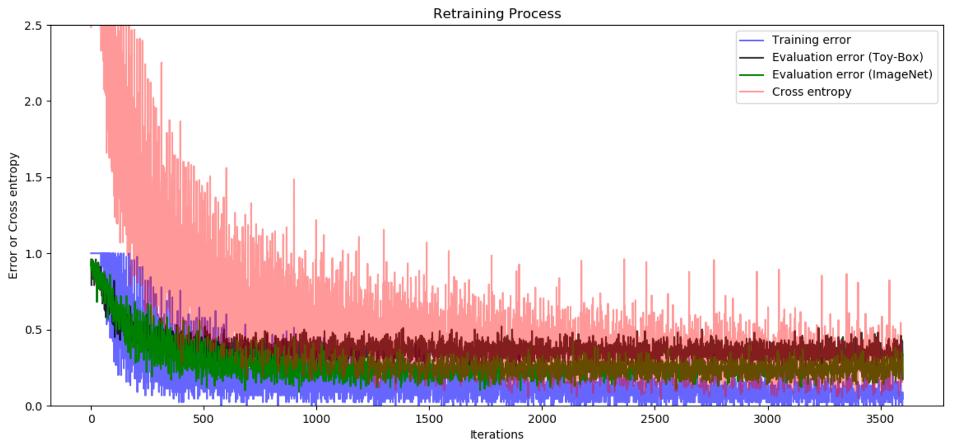

Figure 4.25: Training Process (Alternate Mode, 3600 Iterations)

Figure 4.25 and Figure 4.26 show training process when we were using the alternate mode for the batch composition. During the first few training iterations, the training error is pretty high. The reason is that we calculate the training error right before we train the network model, and the model is overfitting after each step since we would feed in the

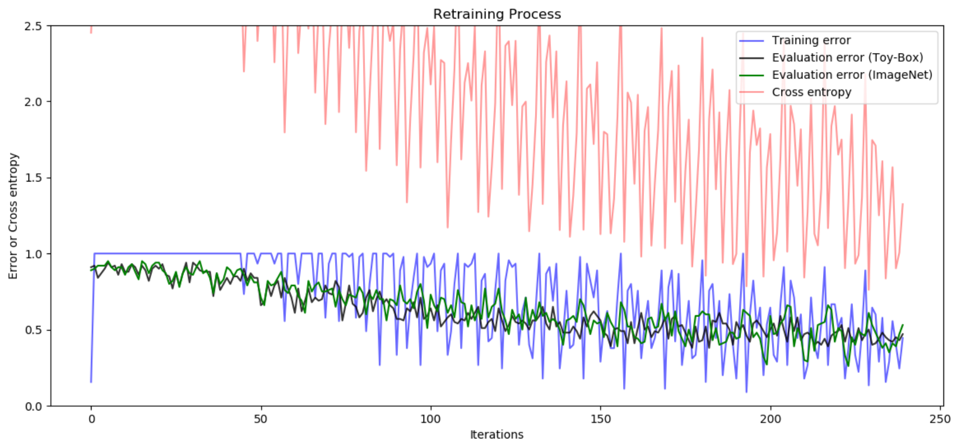

Figure 4.26: Training Process (Alternate Mode, 240 Iterations)

model with images of the same category every mini-batch.

Alternating the category of objects and the overfitting problems are also reflected in the sustained and substantial fluctuation of the cross-entropy, which is an indication that the way we feed in the training images is problematic. We can see that the cross-entropy, training error and evaluation error dropped the most in the first 1000 iterations, yet failed to maintain this trend later on.

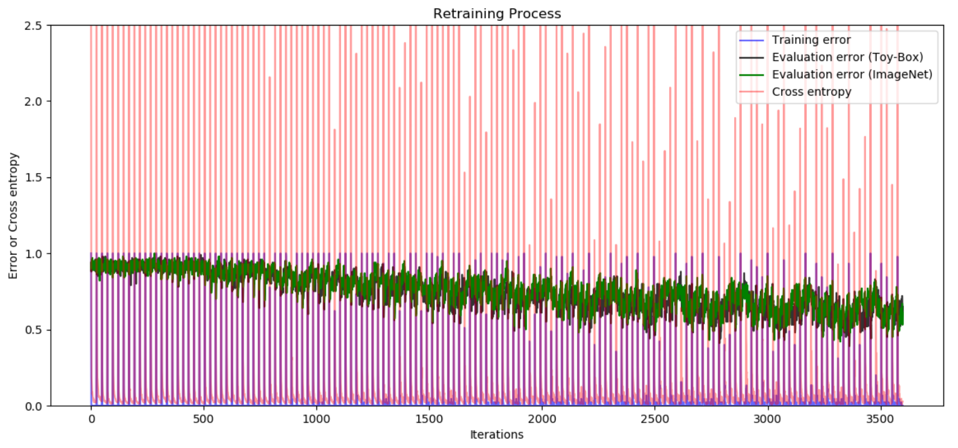

Figure 4.27: Training Process (Sequential Mode, 3600 Iterations)

Figure 4.27 and Figure 4.28 show training process when we were using the sequential mode for the batch composition. Both plots show apparent periodical patterns on both the

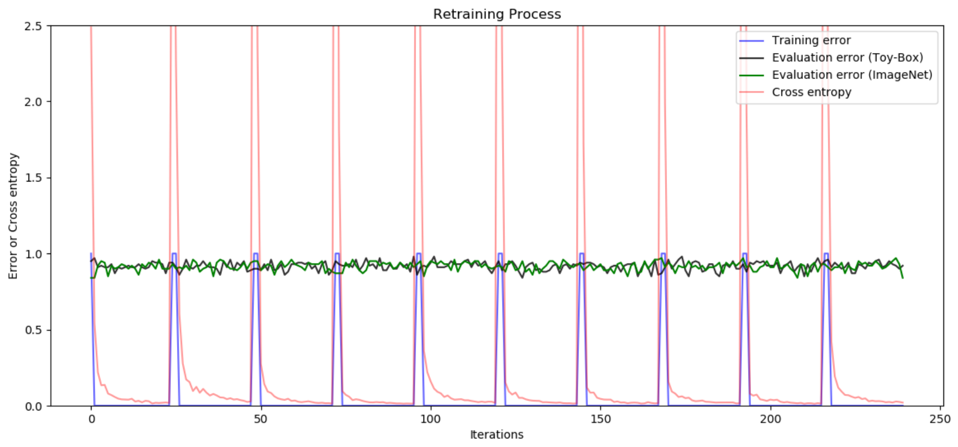

Figure 4.28: Training Process (Sequential Mode, 240 Iterations)

training error and cross-entropy curves. Peaks of both curves are corresponding to the first batches of each category used for training. In this mode, we also feed in the model images of the same category at each step. However, when we start to feed in a batch of images from a new category, the current network model is highly likely to categorize these images to be of the previous category, which might cause an extremely high training error and cross-entropy. The network model then quickly learn from the new category of objects and start to be overfitting again.

Comparing to the cross-entropy in the previous experiment, cross-entropy we got using the sequential mode showed even more substantial fluctuation, and it had a severe impact on the training efficiency. We can see that with the same number of training iterations, the network model could not even achieve the same evaluation error comparing to models we trained using two other modes.

To more directly compare the training efficiency between three different modes, for each mode, we completed the training process 50 trials using 1080 images per category from the Toy-Box dataset, with a batch size of 45 and ran 240 iterations of the training process. For reference, we also ran another 50 trials using the random mode in which we randomly selected training images for every mini-batch. Then we plotted probability density estimation curves of average evaluation errors (computed using the evaluation errors against Toy-Box validation dataset) for all four modes. Distributions of average evaluation errors are shown in the Figure 4.29.

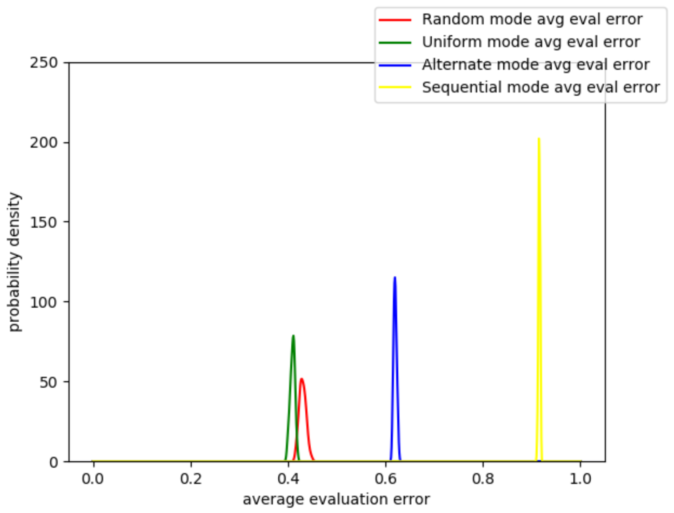

Figure 4.29: Probability Density Estimations of Average Evaluation Errors For Four Different Modes

Just as reflected in the previous plots, for a training process of only 240 iterations, we got the largest average evaluation errors with the sequential mode, followed by the alternate mode. We got the smallest average evaluation errors with the uniform mode and random mode. Surprisingly, comparing to the random mode, we got even smaller average evaluation errors on average using the uniform mode. The reason uniform mode could outperform random mode is that it is guaranteed in the uniform mode our network model could learn from each category equally at each step so that the network model could more easily overcome the initial overfitting problem, while it is not guaranteed when using the random mode. Thus, with our current hyperparameter settings, it is not difficult to observe that the more uniformly training data of each category we distribute within and across mini-batches, the higher training efficiency we can get.

However, in the long run, we would expect to have smaller average evaluation errors using the random mode, compared to the uniform mode. Considering that the network model (Inception-v3 model in our case) has a considerable amount of neurons, the model can learn the order we feed in training data, just as many other seemingly trivial features. It is possible that after epochs of training the network model would self-adjust to perform well on images fed in following the predefined order in the uniform mode, thus experiencing overfitting problems. As a result, although in the short run the uniform mode may benefit our network model by feeding in all different training images during each epoch, in the long run, the random mode is a better choice since it could more effectively prevent the network model from overfitting.

#### 4.2.2.3 Effects of Training Data Composition

Besides the way we form a mini-batch, to explore factors that may affect the training efficiency, we also have to take into account the overall composition of the training dataset. In previous experiments, we used 12 objects per category for training and got a reasonably good training accuracy and efficiency (using the random mode or uniform mode) considering that the diversity of our training dataset is much worse than that of the ImageNet dataset. In the following experiments, we will explore how the diversity of objects (number of different objects in the training dataset) and the number of different images per object in the training dataset would affect the training efficiency.

To explore the effects of objects' diversity, we will keep the number of images per category to be 1080, and change the number of objects each category contains in the training dataset. We will use the random mode (randomly select training images from the entire training dataset) to form the mini-batch, with a batch size of 45, and run the training for 2880 iterations. We will investigate how the training accuracy and efficiency would change when there are 2, 4, 6, ..., 12 objects per category (so that each object would appear in roughly 540, 270, 180, ..., 90 different images) in the training dataset. For each experiment, we ran 50 trials and plotted all the probability density curves of average evaluation errors (computed using the evaluation errors against ImageNet validation dataset) in Figure 4.30.

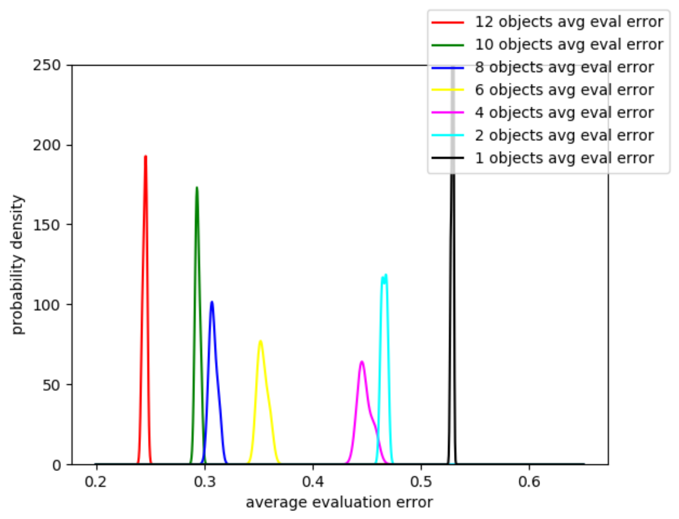

Figure 4.30: Probability Density Estimations of Average Evaluation Errors For Various Objects' Diversities

We can see from the plot that when the diversity of objects in our training dataset is relatively large, with the same number of iterations in our extended training process, the network model will reach lower average evaluation error, an indication of higher training efficiency. However, this result is corresponding to the training process with our current settings, which has a large number of training iterations, and we also want to know how the average evaluation error would change when we change the length of the training process (number of training iterations).

We plot the change of average evaluation error up to a specific training step in our training process. From the Figure 4.31, we can see that the relationship of average evaluation errors in the first 300 training iterations for various numbers of objects in the training dataset is not stable. Especially in the first 100 iterations, all curves intertwine with each other. However, after a certain number of training iterations (1500 iterations in our case), as shown in the Figure 4.32, we know that the relationship of average evaluation errors becomes stable. The more different objects we use in our training dataset, the smaller average evaluation errors we could achieve in the long run.

To explore how the number of images per object would affect the training efficiency, we

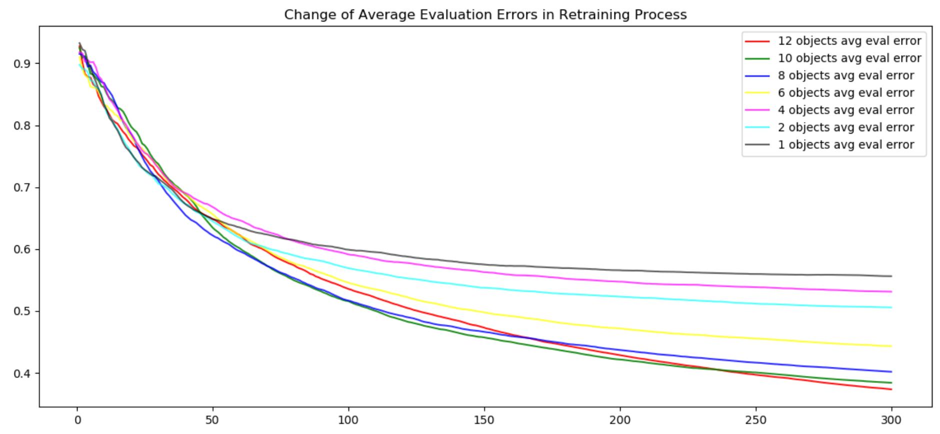

Figure 4.31: Change of Average Evaluation Errors in Retraining Process for Various Objects' Diversities (300 Iterations)

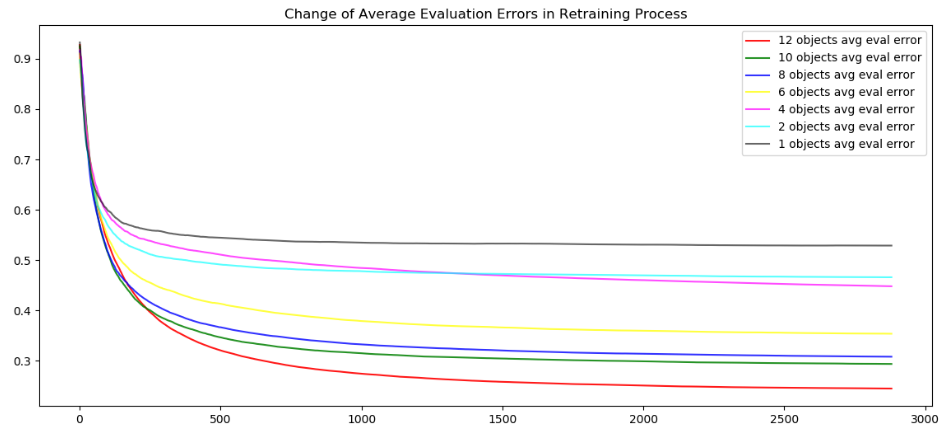

Figure 4.32: Change of Average Evaluation Errors in Retraining Process for Various Objects' Diversities (2880 Iterations)

only have to keep the total number of images used in the training process the same (batch size times number of training iterations) and change the number of images per object used in the training dataset. We will still use the random mode (randomly select training images from the entire training dataset) to form the mini-batch, with a batch size of 45, and run the training for 2880 iterations. In the training dataset, there will be 12 objects per category, and we will investigate how the training efficiency would change when there are 2, 4, 10, 20, 40, 60, 80, and 90 images for each object. For each experiment, we ran 50 trials and plotted all the probability density curves of average evaluation errors (computed using the evaluation errors against ImageNet validation dataset) in Figure 4.33.

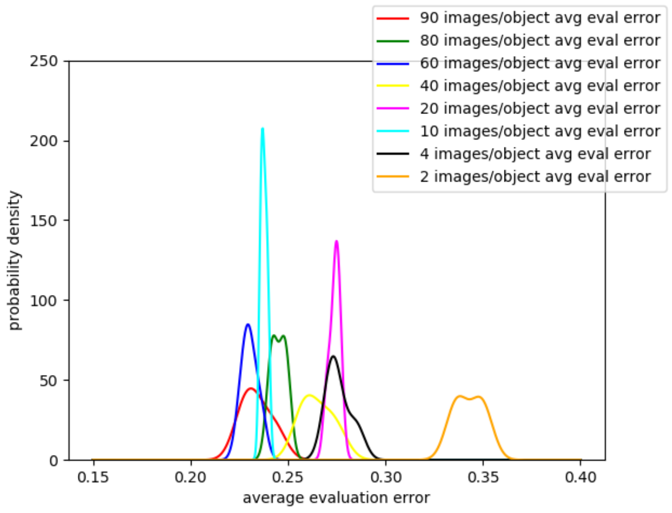

Figure 4.33: Probability Density Estimations of Average Evaluation Errors For Various Numbers of Images per Object

We can see from the plot that the distribution of average evaluation errors is much fuzzier than expected. We can still find out, though, when there are more different images per object in a category, we will get overall lower average evaluation errors, an indication of higher training efficiency. However, there are also exceptions. For example, when there are 10 images per object in our training dataset, we will get lower average evaluation errors than if there are 20, 40 or even 80 images per object in the training dataset.

To get more insight into the change of average evaluation errors during the training process, we plot trends of average evaluation error up to a specific training step for the

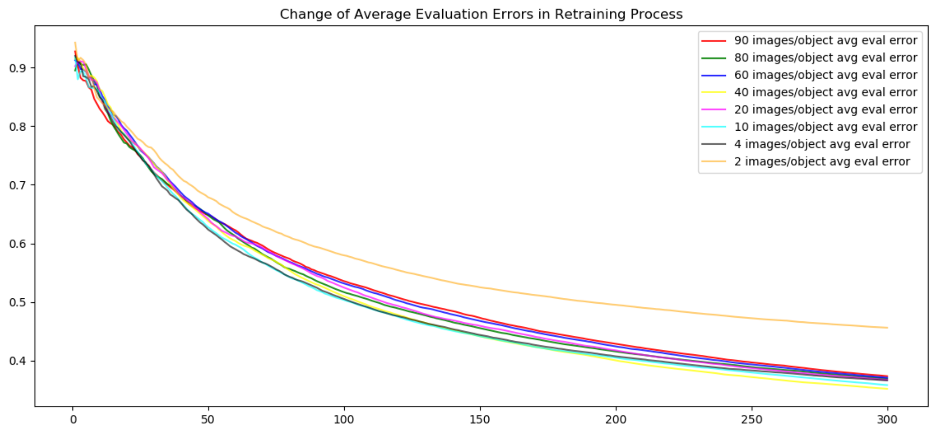

Figure 4.34: Change of Average Evaluation Errors in Retraining Process for Various Numbers of Images per Object (300 Iterations)

different number of images per object in Figure 4.34 and Figure 4.35 (for 300 iterations and 2880 iterations respectively).

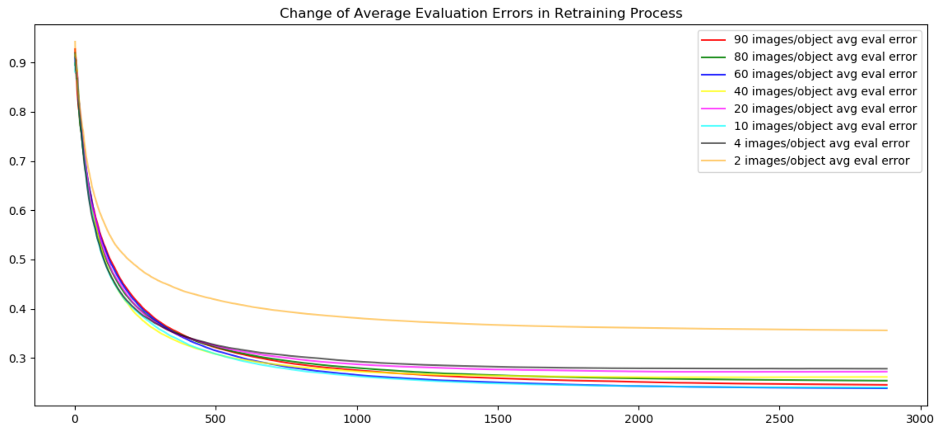

Figure 4.35: Change of Average Evaluation Errors in Retraining Process for Various Numbers of Images per Object (2880 Iterations)

From both plots, we can see that when we used at least four images per object in out training dataset, there would be no significant difference regarding the training efficiency. Thus, with our current hyperparameter settings, the number of different images per object in the training dataset does not have a significant impact on the training efficiency as long as that number is above some threshold, although it is still unclear what would contribute to that threshold.

One possible reason that increasing the number of images per object may not always help to improve the training efficiency could be that different images of an object are not distinguished enough for the network model to learn more features, a problem caused by the nature of the Toy-Box dataset. We might be able to provide cure to this problem, however, by manually selecting higher quality images for each object so that each of an object can be much more distinctive.

# Chapter 5

# Conclusions

## 5.1 Conclusions

From the phase I of this research project, we know that the order of training data, when we are using the stochastic gradient descent algorithm, will have minimal, if not none, effects on the training efficiency. As a result, it is hard for us to determine whether a specific training order will provide higher training efficiency or not. For the same reason, we know from our experiments that we would have a tough time to find proper criteria to evaluate the training order, which is necessary for further optimization used to improve the training efficiency.

We have not tried, however, either the method of curriculum learning proposed by Bengio [19] or the criteria of training value as shown in the paper by MIT researchers [20], using which we could reorder the training data by various optimization algorithms. Thus, by adapting those two methods, we might still have the chance to show that the order of training data from both ImageNet and Toy-Box datasets can be optimized.

It is also worth noting that in a series of experiments we have done, we only used a specific setting. Although we have done multiple experiments and verified that the evaluation error would drop significantly in a certain number of training iterations, it is still unclear if the relationship between a training order and its corresponding training efficiency would only be noticeable when using other settings. In other words, we might have to use other neural network settings and train either more or less number of iterations to show whether the training order would affect the training efficiency or not.

From the phase II of this research project, we know that not only the composition of a mini-batch but also the composition of entire training dataset will have a substantial impact on the training efficiency. Using mini-batches, in which images of all different categories are distributed evenly, we can in general, achieve higher training efficiency. In fact, all non-uniformly distributed mini-batches can impair the training efficiency to some extent, depending on how severe the overfitting problems they can cause in the training process. Specifically, we found that when we ensure that images of all categories in the dataset appear in every single mini-batch during the training process, the network model will achieve a training efficiency as high as, if not higher than, the training efficiency of randomly assigning images in a mini-batch. Although, in the long run, we expect to get lower average evaluation errors using the random mode, since it could more effectively help us prevent overfitting issues compared to the uniform mode.

Generally speaking, we would suffer from overfitting problem easily when we only allow a single or limited number of categories to be in a mini-batch. The situation will get worse if we further limit the diversity of categories between every mini-batch of data we feed into the network model. To ensure that the training efficiency is as high as possible, we have to distribute images of different categories evenly within and between mini-batches.

If we compare three different modes tested in our experiments to three modes mentioned in the Leon Bottou's paper, we could easily tell that the later three modes are superior regarding the training efficiency, since all three modes used by Leon Bottou made use of some randomization techniques, while three modes we tested did not [18]. Among these six different modes, we expect to achieve the lowest training efficiency using the mode in which we will apply randomization to the training order at the beginning of each epoch and sequentially select data for the mini-batches.

For the construction of a training dataset, especially the Toy-Box dataset that contains multiple images of the same object with a similar background, we should collect as many images of different objects as possible. Besides, more different images per object would also be helpful to improve the quality of our datasets, even though we found that sometimes with a smaller number of images per object, we might get even higher training efficiency. We could not tell, however, if the exceptions we got in our experiments were due to the quality issues of the dataset, or the fact that the number of images per object may not be a critical issue, as long as we could get enough of them (above a certain threshold).

## 5.2 Future Work

Given that researchers have shown the possibility of accelerating the training process by applying the curriculum training or similar technique from MIT researchers, we could do further experiments to see if we can adapt those methods in our training process (using ImageNet and Toy-Box datasets). We want to explore if we could find out a feasible and systematic procedure to quantize the difficulty or quality of a training image for both techniques. Besides, we will also have to check if the time we save with the optimized training order could compensate for efforts we put into the image quality evaluation and reordering process.

Besides, in the future, we will also do further experiments to explore with different neural network settings, how the relationship between the training order and training efficiency would change accordingly. This can be particularly useful for us to use a proper setting when examining the effect of a specific training order on the training efficiency.We could also get more insights by visualizing relationships between the training efficiency and each parameter we use to fine-tune our network models for a specific training order.

# Appendix A

## A Short Discussion: Why the Average Evaluation Error is a Good Enough Estimation of Training Efficiency

To explain why the average evaluation error can be a proper measurement or approximation of the training efficiency, we have to at first define what the training efficiency means in our context. When training a neural network model, the training efficiency means how fast can the evaluation error of that network model be reduced. Given this definition, we can calculate the training efficiency by taking the difference of the starting evaluation error and the evaluation error at which step the training stops and then dividing by the number of training iterations (suppose the model is not overfitting yet). However, this way of calculating the training efficiency is not flawless, since the evaluation error at the first and last step of a training process may vary in different trials. As a result, to reduce the uncertainty of training efficiency that we calculate using only two data points, we can take more evaluation errors of the training process into account.

We know that during a training process, the rate our evaluation error decreases is always changing. As a result, to estimate an overall evaluation error changing rate (training efficiency), we can calculate, on average, how fast the evaluation error should decrease at each training step. This calculation can be done by taking the definite integral of the change in evaluation error (which is to calculate the area under the evaluation error curve and above the x-axis from start to end of the training process), dividing by the number of training iterations. We can even simplify this estimation to the calculation of average evaluation error in a training process, an approximation to the definite integral. Let the function $f(x)$ to be the average evaluation error at a specific training iteration $x$, and there are $n$ iterations in total, then:

$$\int_1^n f(x)/n \, dx \approx \sum_{n=1}^{n} f(n)/n$$

Thus, we have shown that the average evaluation is a proper measurement of the training efficiency.

# Appendix B

## Code Changes to the Google's Inception-v3 Training Procedure

The first part of the code change in the official Inception-v3 retraining script is made to enable the model to read in evaluation data from a specific directory, instead of using a random selection of the training data for evaluation. This code change is useful since the Toy-Box dataset used for training contains multiple different images of the same objects with similar backgrounds, and we have a relatively high chance of evaluating the network model against images that are too similar to ones we used for training. Thus, errors can be too high for indicating the actual performance of the model due to the homogeneity of the data we used. To resolve this problem, we could assign a directory that contains objects that we have not used in the training process for evaluating the network model.

The second part of the code change in the official Inception-v3 retraining script is made to read and write log files (in JSON format) for serializing useful information. We wrote additional windows batch files (or Linux shell scripts) which could use the log files we stored to automate the optimization of the training order with the genetic algorithm or simulated annealing. In this part of the code change, we implemented the calculation of average evaluation error, a measurement of the training efficiency, and it is a necessary criterion for optimization algorithms.

The third part of the code change in the official Inception-v3 retraining script is made to allow the network model to read in the training data in a predefined order, and to control the composition of every mini-batch the model would use for training. In the official code, the network model would randomly select any image in the training dataset into a mini-batch, which means it is not guaranteed that every image in the training dataset will be used. At the end of the training process, the network model could use a particular image multiple times, yet never use another one. We designed a tracking mechanism so that the system could make sure that every single image is used at the end of an epoch, either in a random order or a predefined order.

Beside above functional changes, we also updated the retraining script for easier debugging and more elegant data visualization.

# BIBLIOGRAPHY

[1] Awesome - most cited deep learning papers. https://github.com/terryum/awesome-deep-learning-papers. Accessed: 2018-03-15.

[2] Deep learning 101 - part 1: History and background. https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html. Accessed: 2018-03-15.

[3] The 9 deep learning papers you need to know about (understanding cnns part 3). https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html. Accessed: 2018-03-15.

[4] Reading list). http://deeplearning.net/reading-list/. Accessed: 2018-03-15.

[5] A. M. Turing. Computers &amp; thought. chapter Computing Machinery and Intelligence, pages 11–35. MIT Press, Cambridge, MA, USA, 1995.

[6] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1990.

[7] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:65–386, 1958.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2:303–314, 1989.

[10] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[11] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann, 1990.

[12] 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.

[13] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

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

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[18] Léon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. 2009.

[19] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[20] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1311–1320, 2017.

[21] Àgata Lapedriza, Hamed Pirsiavash, Zoya Bylinskii, and Antonio Torralba. Are all training examples equally valuable? *CoRR*, abs/1311.6510, 2013.

[22] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[23] Hill climbing. https://en.wikipedia.org/w/index.php?title=Hill_climbing&oldid=799616668. Accessed: 2018-03-11.

[24] X. Wang, F. M. Eliott, J. Ainooson, J. H. Palmer, and M. Kunda. An object is worth six thousand pictures: The egocentric, manual, multi-image (emmi) dataset. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2364–2372, Oct 2017.

[25] Jorge Nocedal Mikhail Smelyanskiy Nitish Shirish Keskar, Dheevatsa Mudigere and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.