Parameter Optimizaion on Horizon Line Detecting NG-DSAC Algorithm

Xun Jia, Jonathan Nguyen, J. Michael Cox

Abstract

The horizon line is an important characteristic in many computer vision tasks. Many methods and algorithms have been implemented to estimate this line from images and videos. We are investigating some of these algorithms on a standard dataset and will be implementing optimization techniques on the top algorithm to see if we can improve accuracy of estimating the horizon line correctly.

Introduction

Horizon Line Estimation (HLE) is the process of taking input images or video and estimating the horizon, the line that divides the Earth from the sky. The horizon line can provide particularly useful information, as the line normally controls perspective and gives viewers a reference for other objects in the image.

Since the horizon line provides perspective, this can be used in many different types of applications. Such as, providing navigation in areas that do not have GPS, and creating 3D models from images that were taken from different perspectives. Additionally, knowing the location of the horizon could provide important, common sense data for image classification. Furthermore, HLE could be a useful tool in identifying potential pedestrians or other vehicles in scenarios containing an obstructed view or terrain based blind spot for autonomous vehicle navigation.

Background

HLE is one of the fundamental geometric problems in computer vision[6]. Progress has improved with recent methods, but has not reached standards that we believe would be valid to implement into certain applications.

In our original scope for this project, our plan was to investigate a leading tech website which proclaimed three algorithms as the best solutions for HLE [11]. Due to time restraints, we had hoped that the website would direct us to open source projects so that we could simulate those results on a standard platform and dataset. After the simulations, we would then choose the algorithm with the best results and go more into depth on how we could optimize it. However, upon analyzing and searching for open source projects, we were unable to find any for two of the algorithms, GoogleNet and CNN+Full. This led us to mainly focus our experiments on the only project we found, NG-DSAC.

In this project we hope to give an in depth understanding of all the algorithms from the website, what it is and how it is used to find the horizon line from an image or video. Afterwards, we will explain how NG-DSAC will be optimized and create different models from the variations of the algorithm. Visualizations of the results will be displayed to get a better understanding of the results and we will perform numerical analysis to find the algorithms optimal performance.

The Algorithms

NG-DSAC

The original RANSAC algorithm was created in 1981 by Fischler and Bolles. It's biggest strength lies in its simplistic ability to fit an estimation without interference from outliers [2]. RANSAC does this by creating multiple model hypothesis from random data points, scoring each hypothesis by taking a consensus, and selecting the hypothesis with the highest consensus score. In domains with many outliers, RANSAC requires an exponential number of iterations to find the optimal, outlier free set. For this reason, the classic RANSAC strategy limited the number of lines in its implementation and provides the best model up to that point.

A common deep learning strategy is to take the argmax of the weights to select the highest performing hypothesis. A newer strategy is to take the softmax, which takes the weighted average of arguments, but sacrifices the best hypothesis for a more generalized solution. Differentiable SAmple Consensus (DSAC), attempts to differentiate the expected loss of the pipeline with respect to all learnable parameters and ascertain a probabilistic maximum of weighted values. The probabilistic nature of DSAC can outperform softmax as an activation function for horizontal line detection and camera localization [9].

$$h_{DSAC}^{w,v} = h_J^w, with J\ P\left(J|\ v, w\right)$$

NG-RANSAC takes a different approach in (i) learning the weights to guide hypothesis search rather than using handcrafted heuristics, and (ii) integrating RANSAC itself in the training process which leads to self-calibration of

the predicted weights. In their proposal Brachman *et al.* let the neural network predict observations y(w) and, additionally, a probability associated with each observation p(y;w). Through this, the neural network can express a confidence level in its own predictions through this probability. This can be useful if a certain input for the neural network contains no information about the desired model h.

Traditionally,HLE is computed via vanishing point detection and geometric reasoning. Brachman *et al.* take an easier approach and use a general purpose CNN model that predicts a set of 64 2D points based on the image to which we fit a line with RANSAC. The network has two output branches predicting 1.) the 2D points y(w) within Y(w), and 2.) probabilities p(y;w) for guided sampling.

As briefly introduced above, the model consists of 10 2D convolutional layers with a batch normalization between them. From there the system forms two, fully connected branches consisting of 3 more 2D convolutional layers each. The point prediction branch uses ReLU and Sigmoid activations functions and the neural guidance prediction branch uses ReLu and LogSigmoid activation functions.

The training images are loaded into the system in mini-batches of four images and we consider the processing of each mini-batch as an iteration. Disregarding the time spent moving images from Google Drive to Colab and preprocessing functions, each iteration takes between 1.5 and 7 seconds depending on the GPU we are assigned at connection to the Colab server. Through 15,000 iterations, we complete 28 complete Epochs around the data. 15,000 iterations were not chosen because of any optimization strategy but as a practical limitation due to the Colab runtime cutoff of 12 hours.

Other than the hyperparameters we chose to optimize (learning rate, optimizer, and number of iterations), the algorithm contains other hyperparameters important in the point selection process, which we were unable to look at. beta determines the softness of the Sigmoid funtion, tau is the inlier threshold, and alpha determines the softness in the DSAC scoring function. Our implementation used the default Tau = 0.05, Alpha = 0.1, and Beta = 100.0.

GoogleNet

[6] uses convolutional neural networks (CNNs) to estimate the location of the horizon line from raw pixel intensities. It directly estimates the horizon line using convolutional neural networkswhich is quite competitive among the others. Using the GoogleNet architecture, it achieves similar accuracy to other architectures with fewer parameters. GoogleNet does not need to make explicit geometric assumptions on the contents of the underlying scene, and generates competitive, state-of-the-art results.

CNN+Full

[7] proposes candidate horizon lines, scores them, and keeps the best. It uses a deep CNN to extract global image context and guide the generation of a set of horizon line candidates, identifies vanishing points by solving a discrete-continuous optimization problem, and calculates a line based on the consistency of the lines in the image with the selected vanishing points. It first uses global image context to estimate priors over the horizon line. Using these priors, it uses a novel vanishing point (VP) detection method that samples horizon lines from the prior and search for high-quality vanishing points. The prior helps ensure a good initialization such that the detection method may obtain very precise estimate.

Dataset

The dataset that we will be utilizing is the ***Horizon Lines in the Wild***, a large dataset of real world images with labeled horizon lines, captured in a diverse set of environments[6]. The dataset utilizes 2018 testing and 16,906 training images. Although the version 1.2 of the dataset contains more diversity than previous horizon centric databases, it is lacking in some aspects. While it provides thousands of images from 15 cities, the cities are primarily in North America and Europe. Furthermore there are no images of mountainous regions or horizon scenes found in the actual wild, such as natural landscapes. We will be strictly controlling training parameters, and testing outcomes will be quantified using the AUC, Area Under the Region of Convergence Curve. Which we define in this instance as the measurement of the maximum distance between the estimated horizon and ground truth within the image, normalized by image height up to an error threshold of 0.25.

Horizon lines in the wild version 1.2 is a beginner level difficulty for horizon computation because the images in the training set and test set partially show the same scene (for example over 600 photos of the Alamo in Texas and over 400 images of Notre Dame in Paris).

Methods

There are many different ways that the algorithm will be optimized, and we plan to look into at least three different avenues to make the algorithms more efficient or improve performance. Some methods that we plan to implement will be to tune the gradient optimization algorithms, learning rate, and batch sizes. Optimization strategies will be based on numerical analysis.

Gradient Optimization Algorithms

Gradient descent algorithms are commonly used algorithms in order to optimize deep learning neural network solutions. The main idea of these gradient descent algorithms is to iteratively find the direction and step size of a point and checking to see if it has converged to its minimums. In python they offer a machine learning library called pytorch which offers many of these gradient descent algorithms such as Adam, SGD, L-BFGS, MultiplicationLR, and more. With this large library, numerous variations of the model can be tested with different gradient algorithms to see which one offers better results.

Our training included the vanilla versions of two optimization algorithms, Adam and Stochastic Gradient Descent.

Learning Rate

When deep learning neural networks are trained, they require a learning rate to determine how much a model changes. If a learning rate is too small then it will result in the training process taking too long and the model potentially getting stuck. When a learning rate is too large, then this could result in the model overshooting the optimal weights to minimize or maximize its function. In previous work done by Li et al. they state that although a small initial learning rate allows for faster training and better test performance initially, the large learning rate achieves better generalization soon after the learning rate is annealed[10]. As learning rate contributes to how well our model will perform, we plan on testing many different learning rates to see which one will provide the highest AUC.

Batch Sizes

In deep learning neural networks, batching or "batch means" is a well known method for estimating the variance of the sample mean of a stationary output process. In numerous studies this method has been shown to be robust and dependable[3]. The idea is to give the model a large enough batch size so that it will give average gradients that would represent gradients for the full dataset. When a batch size is too small, the model will return poorly developed average gradients, which will then result in a model returning a lower accuracy score. Batch sizes may vary significantly but a larger batch size does not equate to the best batch size, as the average gradient will only be improving considerably after a certain size. The model will be allocating more resources and time to achieve very similar results to a smaller batch. The goal of this method is to find a batch size that would give optimal results with the limited resources we have.

Plan and Experiment

Previous Experiment

In the previous work done by Brachmann et al., they explain that finding the horizon line of an image or video is typically solved by vanishing point detection and geometric reasoning. For their experiment they used a general purpose CNN which predicted a set of 64 2D points based on the image which a line would be fitted from RANSAC. They would use the HLW dataset but they did not specify if they used version one or version two. Other key metrics in their implementation is that they used Adam as their gradient optimization algorithm, 250,000 iterations, and a learning rate of 10^{-4}. In order to evaluate their model, they measured the maximum distance between the estimated horizon and the ground truth within the image. They would then calculate the AUC of the cumulative error curve up to a threshold of 0.25. With their metrics, the AUC accuracy they achieved was 75.2%[9].

Our Experiments

For our experiment we have broken it into four different steps. The first step would be to apply the same metrics that Brachmann et al. used for their experiment to see if the it would output the same results with our computational resources. Next would be to run different combination of models depending on gradient optimization algorithms, batch sizes, and learning rates. After gathering the data points from our models we would then perform numerical analysis to find a function that would fit our points. We would then test the function we have created by inputting a specific learning rate and batch size to see if the model would output a result close to what our function predicted. The goal of our project is not to find the optimal learning rates or batch sizes as our scope is relatively small and our data points are limited. What we hope to achieve is to show that our methods provided results which can then be replicated on a larger scale in order to find the functions optimal gradient algorithm, learning rate, and batch size.

Gradient Optimization Algorithms There are many different types of gradient descent algorithms but for the purpose of our project scope we go more into depth on Adam and SGD, stochastic gradient descent.

Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation [8]. One of the key advantages of Adam is the speed at which it converges to the minimum, but the use of Adam for optimization will likely make it difficult to predict the outcome of learning rate adjustments due to the stochastic nature of its adjustments and exponential decay rate based on estimates of both the mean and variance of the gradient.

Stochastic gradient methods aim to minimize the empirical risk of a model by repeatedly computing the gradient of a loss function on a single training example, or a batch of few examples, and updating the model parameters accordingly. SGD is scalable, robust, and performs well across many different domains ranging from smooth and strongly convex problems to complex non-convex objectives[5]. SGD can be traced back to the Robbins–Monro algorithm circa 1950, and the more modern implementation of SGD+Momentum was proposed by Rumelhart, Hinton and Williams in 1986 and has been used in many successful deep neural network solutions. While the use of SGD is more straightforward mathematically than Adam and easier to predict, we hypothesize SGD will not converge on a maximimum effective solution as quickly as Adam. Adding the momentum function as another hyperparameter, could make SGD a contender for maximizing our HLE program quickly, we don't have the computing resources needed to take on tuning the momentum hyperparameter.

Adam and SGD have been shown to be effective gradient optimization algorithms as they return fast results and the default parameters generally perform well on different problems. When analyzing the algorithms from the leading website, we noticed that they also chose to use Adam as their gradient optimization and we decided that we would create some models with this algorithm to see if we could produce an AUC greater than theirs. In order to see how our program

performed with other gradient algorithms, we created models using SGD to see if it performed better or worse than Adam.

Learning Rate After reading the literature by Li et al. and knowing the range of learning rates begin from 10^{-6} and 10^{-2}, we decided to test the opposite spectrum's of learning rates to see what results we would achieve. In order to get a well round understanding of how our model performs with different learning rates we tested three types of learning rates 10^{-2}, 10^{-4}, 10^{-6}.

Batch sizes The variability of batch sizes can range from a thousand to hundreds of thousands. In the experiment for NG-DSAC by Brachmann et al. they ran their model with an iteration size of 250,000. Due to the time restrictions of this project, we believed that an batch size of 250,000 was too large to train the models while having quality results. This resulted in choosing three different batch sizes of 5000, 10000, and 15000 to run our models. We hope to that these smaller batch sizes would result in producing training models in fraction of the time while also providing positive results.

Initial Results

Analyzing the table in Figure 1, we created fourteen different models with different variations of learning rates, iterations, and optimizers. The first model that we ran was a model with the same metrics as the experiment done by Brachmann et al., with a learning rate of 10^{-4}, 250000 iterations, and Adam optimizer and a learning rate scheduler (which we turned off for our tests in order to keep the project more in-line with the course objectives). Interestingly our duplication of their results scored lower with an AUC of 71.6% while their experiment scored a 75.2%. The cause of this difference could be due to multiple different factors such as usage of different datasets or computational resources. Some observations we also noticed was that SGD performed poorly compared to Adam as it created the model that scored the lowest AUC out of all models with a score of 1.1%. Comparing the models with Adam, we created a plot in Figure 2 to give a better visualization of how well the models performed. Analyzing the plot, we notice that as the iterations increase so does the AUC score. We also notice that when we tested the smallest and largest learning rates, the smallest rates would give us a significantly lower AUC score than the larger rates. The learning rate that produced the highest results from our model had a learning rate of 10^{-4}. From all the models that we created with Adam, the model with 10^{-4} learning rate, 15000 iterations generated the highest AUC accuracy with a score of 67.9%, while the model with 10^{-6} learning rate, 5000 iterations generated the lowest accuracy of 26.7%. From these initial results we decided to generate more points to get a better understanding how the AUC would score with higher iterations and learning rates between the extremums. From analyzing the table and plot we can assume that the optimal metrics for a model would be an Adam gradient optimizer, iterations greater than 15000, and learning rates between 10^{-4} and 10^{-2}.

Numerical Analysis

Although analyzing tables and plots to make assumptions on optimal metrics may seem feasible, numerical analysis methods can be applied to optimization problems to find its maximum and minimum values. There are many different methods such as gradient methods, Quasi-Newton methods, least squares algorithm, particle swarm optimization, and many more. For our analysis, we decided to go more into depth on the least squares method and a gradient method.

Least Squares Algorithm

With our inputs being learning rate and iterations, and output being AUC accuracy, the first analysis we thought to implement was the Least Squares algorithm, which is an algorithm that will create a regression line. This line will be the best fitting straight line through our data points. The idea of using least squares is that the regression line will predict the value of the AUC score when given a learning rate and iteration. The goal of this analysis is to find a vector x^* that minimizes $||Ax - b||^2$. Other key notes in our analysis is that we created a least squares regression line with all the data points excluding points from models with 10^{-6} learning rates, since it made a poorly fit line. To understand the proof of linear regression please refer to [4], Chapter 12.

$$x^* = (A^T * A)^{-1} * (A^T * b)$$

$$A = \begin{bmatrix} 0.01 & 15000 & 1 \\ 0.0001 & 15000 & 1 \\ 0.01 & 10000 & 1 \\ 0.0001 & 10000 & 1 \\ 0.01 & 5000 & 1 \\ 0.0001 & 5000 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 65.2 \\ 67.9 \\ 60.7 \\ 62.9 \\ 51.9 \\ 56.1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 0.01 & 0.0001 & 0.01 & 0.0001 & 0.01 & 0.0001 \\ 15000 & 15000 & 10000 & 10000 & 5000 & 5000 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$A^T * A = \begin{bmatrix} 0.0003 & 303 & 0.0303 \\ 303 & 700000000 & 60000 \\ 0.0303 & 60000 & 6 \end{bmatrix}$$

Learning rate	Iterations	Optimizer	Time	AUC @ 0.25
0.0001	250000	Adam	96 hrs	71.6
0.01	5000	Adam	6 hrs 14 min	51.9
0.0001	5000	Adam	4 hrs 40 min	56.1
0.000001	5000	Adam	5 hrs 5 min	26.7
0.01	10000	Adam	12 hrs 30 min	60.7
0.0001	10000	Adam	11 hrs 39 min	62.9
0.000001	10000	Adam	15 hrs 15 min	28.6
0.01	15000	Adam	18 hrs	65.2
0.0001	15000	Adam	18 hrs 5 min	67.9
0.000001	15000	Adam	19 hrs 15 min	29.9
0.01	5000	SGD	6 hrs 29 min	40.6
0.0001	5000	SGD	5 hrs 22 min	23.8
0.000001	5000	SGD	4 hrs 51 min	1.1
0.0001	15000	SGD		27.4

Figure 1: Table of Results

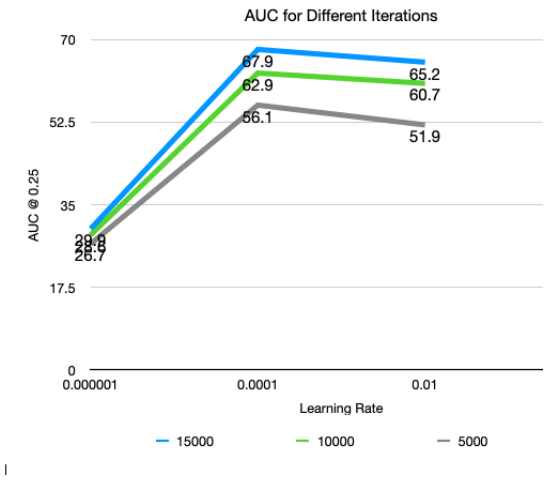

Figure 2: Initial Results with Adam Optimizer

$$A * b = \begin{bmatrix} 1.80 \\ 3772500 \\ 365 \end{bmatrix}$$

$$(A^T * A)^{-1} = \begin{bmatrix} 6.80 * 10^3 & 0 & -34.4 \\ 0 & 1 * 10^{-8} & -0.0001 \\ -34.4 & -0.0001 & 1.34 \end{bmatrix}$$

$$x^* = \begin{bmatrix} -306 \\ 0.001255 \\ 49.8 \end{bmatrix}$$

$$AUC = -306 * LearningRate + 0.001255 * Iterations + 49.8$$

Quadratic Function

To further our numerical analysis, we concluded that the real equation should be more complicated than the linear equation we found from the least squares analysis. The next method that we looked to implement was a quadratic formula where we first built a quadratic function for learning rate, number of iterations as input respectively and AUC accuracy as output as shown in Figure 3. With the data we collected, we got three quadratic functions for each of the variables.

For Learning Rate as x and AUC accuracy as y:

$$y = -3.841e + 07 * x^2 + 3.877e + 05 * x + 29.51$$

$$y = -3.467e + 07 * x^2 + 3.5e + 05 * x + 28.25$$

$$y = -2.947e + 07 * x^2 + 3e + 05 * x + 26.4$$

For number of Iterations as x and AUC as y:

$$y = -8.6e - 08 * x^2 + 0.00305 * x + 38.8$$

$$y = -3.6e - 08 * x^2 + 0.0019 * x + 47.5$$

$$y = -1.2e - 08 * x^2 + 0.00056 * x + 2$$

Upon finding the quadratic equations for learning rate and iterations, we tested different values from analyzing Figure 3. We would then calculate the mean of the maximums from our results and concluded that the optimal point for learning rate would be 0.005 and number of iterations to be 250000. Due to our computational resources we would not be able to test this optimal point, so we have adjusted our metrics to test a learning rate of 0.005 and iterations of 15000 to see if our model would achieve a maximal AUC compared to our points from our initial results. The results for this numerical analysis will be analyzed in the Testing Optimization Algorithms below.

Golden Section Method

Although least squares analysis and the quadratic formula allowed us to compare numerical analysis', we wanted to incorporate a third analysis which would be a polynomial regression formula. After finding the function, we plan to use golden section to find our optimal points. Golden Section Method[1] is a technique for finding an extremum (minimum or maximum) of a function inside a specified interval. For a strictly unimodal function with an extremum inside the interval, it will find that extremum. The method operates by successively narrowing the range of values on the specified interval, which makes it relatively slow, but very robust. The technique derives its name from the fact that the algorithm maintains the function values for four points whose three interval widths are in the ratio

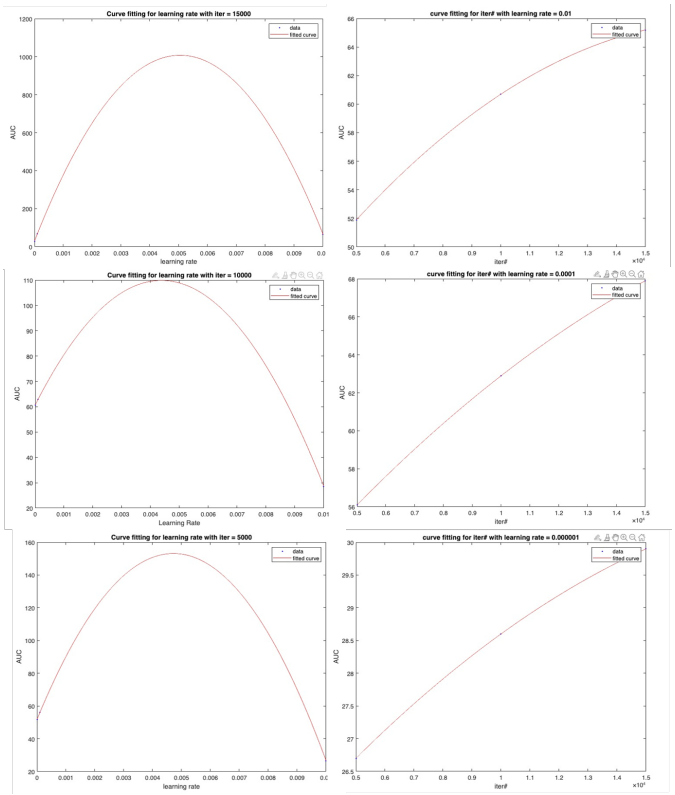

Figure 3: Quadratic Functions

$$2-\phi : 2\phi - 3 : 2-\phi$$

where ϕ is the golden ratio. These ratios are maintained for each iteration and are maximally efficient. Excepting boundary points, when searching for a minimum, the central point is always less than or equal to the outer points, assuring that a minimum is contained between the outer points. The converse is true when searching for a maximum. The algorithm is the limit of Fibonacci search (also described below) for many function evaluations.

Polynomial Regression

Fitting a 2-dimensional function would give us a better visualization about how the object function surface looks like. Polynomial functions shares the similar property that the higher the rank is, the more bizarre the surface looks like. So again, in order that the surface won't generate too many local maximum, we decide to fit a 2-dimensional cubic function which takes number of iterations and learning rate as input and AUC accuracy as output.

The goal of polynomial regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable x. It gives a better fit when the problem is none-linear than linear regression. Generating the polynomial regression function, with x_1 as learning rate, x_2 as number of iterations and AUC accuracy as Y, we got

$$Y = -(8.125e-14 * x_2^3 + -6.249e-6 * x_1 * x_2^2 + -859.4 * x_1^2 * x_2 + -2.607e-8 * x_2^2 + 8.82 * x_1 * x_2 + -2.537e+7 * x_1^2 + 0.000814 * x_2 + 2.553e+5 * x_1 + 23.14)$$

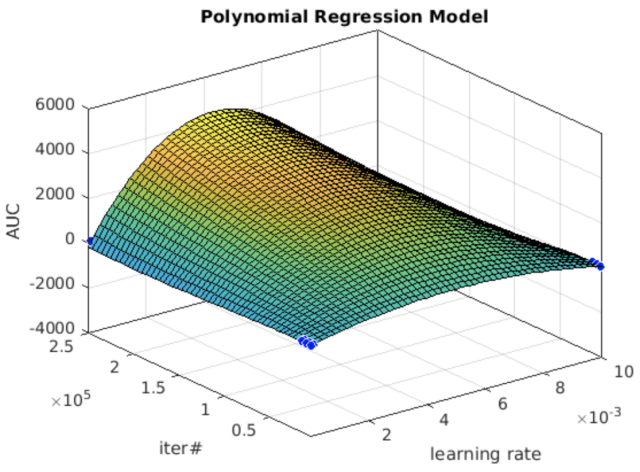

Figure 4: Polynomial Regression Model

Figure 4 gives us an estimate visualization of how the surface looks. Although this figure shows that the maximum value of AUC goes over than 100, it still gives us a thorough estimate of the maximizers. After finding the polynomial regression from our points, we performed Golden Section method and gradient decent method to get the maximum. However, due to the shape of this function, gradient method performs poorly. Golden section method was able to find the maximum of this surface and got and iteration of 10000 and learning rate of 0.005. We plan to run this model and compare our analysis from the results from our initial results and results from our other numerical analysis'.

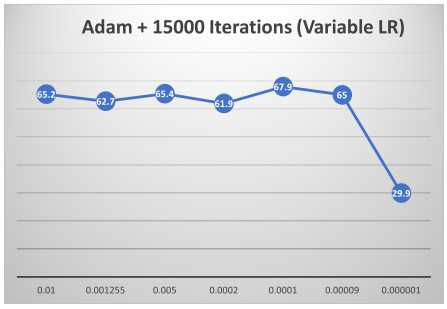

Figure 5: Final Result with Adam Optimizer and 15000 iterations

Testing Optimization Algorithms

We took several samples to get regression models in order to find out the potential maximizers. Least Square Method gives us learning rate = .001255 and iteration number = 15000, and we estimate a AUC accuracy of 68.24 and it turn out to 62.7. The quadratic forms gives us a rough estimation of learning rate = 0.005 and number of iterations = 15000. Golden section method finds a local maximizer of learning rate = 0.005 while number of lterations = 10000. Due to that we do not have the sample lying in between 0.0001 and 0.01, the model is not good shaped enough to give us an estimate of AUC accuracy. By testing those points, we got two local maximums. One is in the samples we took, while the other is generated by both quadratic functions. Golden section method on polynomial regression gives us the same maximun of learning rate.

Application

In order to check the outcome of our training we took 12 pictures at the NCSU Centennial Campus, and ran them through the NG-DSAC application. The outcomes of all 12 images on campus were within the .25 error threshold and provided a good horizon estimation even in challenging images with the majority of the skyline obscured by buildings (see Figure 6. The points in the central images are the hypothesis provided by the DSAC and NG-DSAC implementations. While they provide visually similar results the neurally guided portion was slightly closer to ground truth.

Conclusion

To conclude our experiment, our goal was to find methods that would be able to locate the optimal gradient optimizer, learning rate, and batch size that would produce the highest AUC. By applying visualizations and numerical analysis, we

believe that our findings provide promising results. As we narrowed our scope of only a couple gradient optimizers, learning rates, and batch sizes we believe we have shown different methods that would be able to find the optimizers. There were clearly limitations to our project, but we believe that our methods can be applied to a wider scope and that our experiment can be expanded in more directions.

Limitations and Future works

Due to the limitation of computation power and time, we were unable to test the amount of points we would have liked to perform a more accurate and comprehensive experiment. What's more, the initial bracketing scheme we chose was functional, but not proportional, this leads to a blank region in the sample space which made curve fitting a function to out data points difficult with the limited number of observations we were able to generate. There were a number of improvable areas found in the Horizon Lines in the Wild data set and the NG-DSAC horizon line estimation function which are displayed in Figure 7. In regards to the data set, there were no examples presented in the training set of undeveloped landscapes, such as the mountainous landscape displayed. The horizon line in this test was significantly out of alignment with ground truth. Furthermore, the program believed it had a good solution, which could lead to very bad results in low-level flight navigation. Another improvable area we found, which also highlighted the importance of optimizing the training was the observation that training at one image size limited us to testing with the same image size. If this anomaly persisted through further testing, it would indicate that the model would need to be retrained for every image size it would be expected to use.

References

[1] M. Avriel and D. J. Wilde, "Golden block search for the maximum of unimodal functions," 5, vol. 14, INFORMS, 1968, pp. 307–319.

[2] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi-org.prox.lib.ncsu.edu/10.1145/358669.358692.

[3] L. Goldsman and B. Nelson, "Batch size effects on simulation optimization using multiple comparisons with the best," 1990.

[4] E. Chong and S. Zak, "An interduction to optimization," 2008.

[5] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," 2016.

[6] S. Workman, M. Zhai, and N. Jacobs, "Horizon lines in the wild," in *British Machine Vision Conference (BMVC)*, 2016.

[7] M. Zhai, S. Workman, and N. Jacobs, "Detecting vanishing points using global image context in a non-manhattan world," *CoRR*, vol. abs/1608.05684, 2016. arXiv: 1608.05684. [Online]. Available: http://arxiv.org/abs/1608.05684.

[8] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2017.

[9] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother, *Dsac - differentiable ransac for camera localization*, 2018. arXiv: 1611.05705 [cs.CV].

[10] Y. Li, W. Colin, and T. Ma, "Towards explaining the regularization effect of initial large learning rate in training neural networks," 2019.

[11] "Horizon line estimation on horizon lines in the wild." [Online]. Available: https://paperswithcode.com/sota/horizon-line-estimation-on-horizon-lines-in.

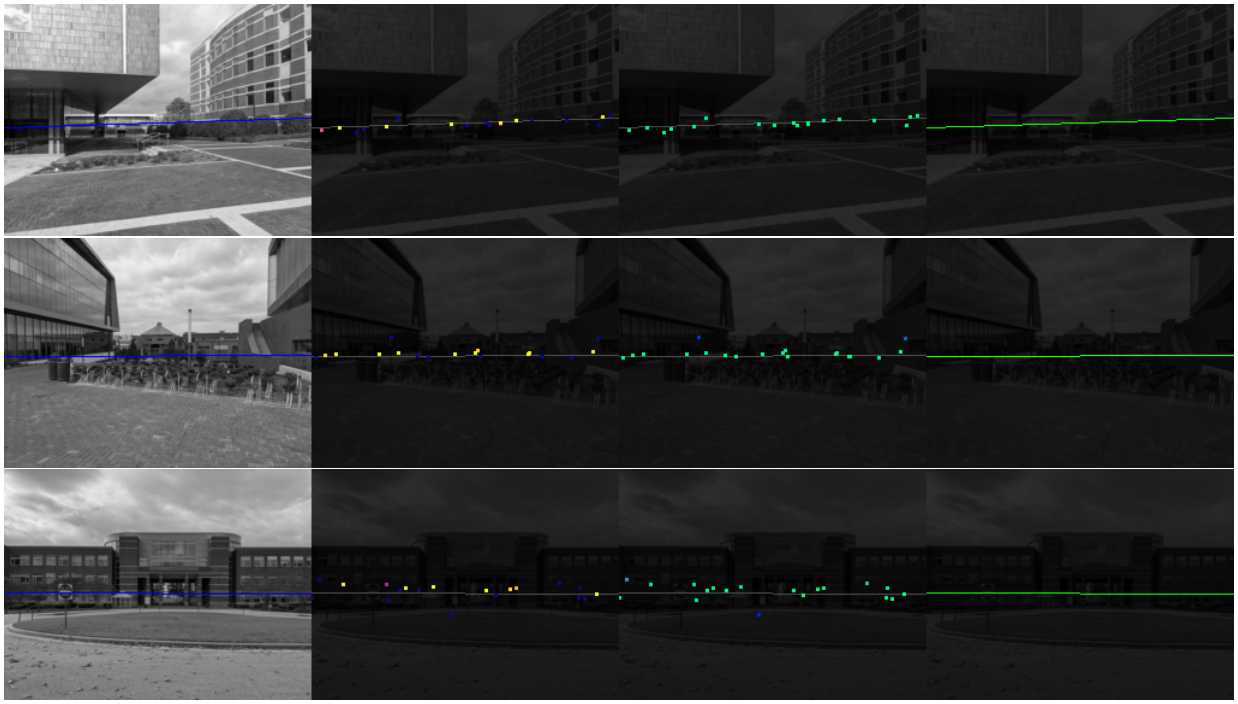

Figure 6: Examples of NG-DSAC horizon line estimation at NCSU campus performing optimally.

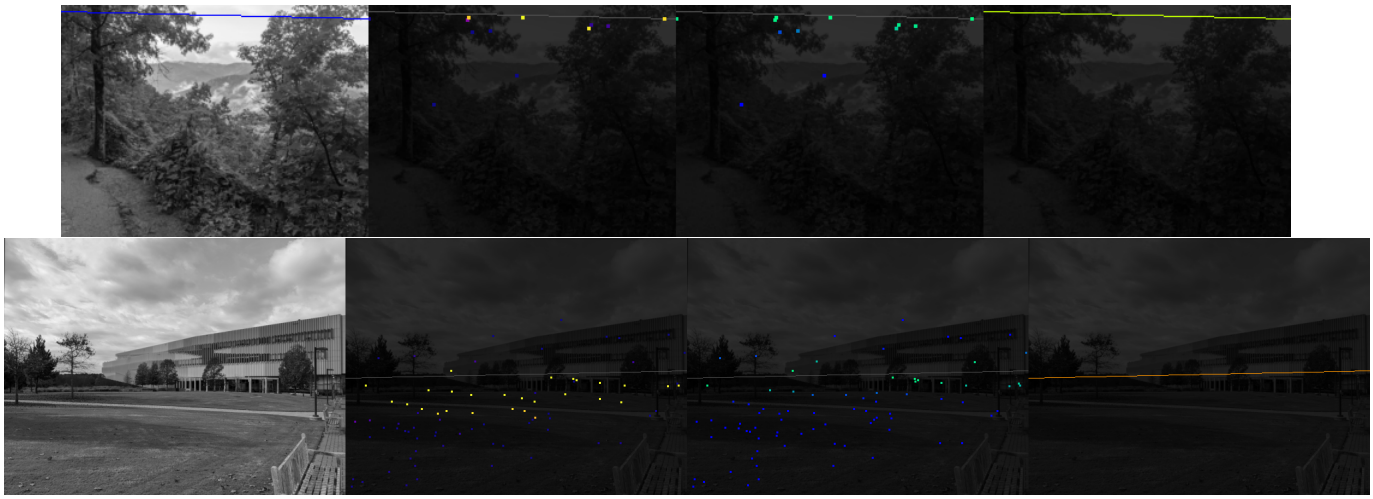

Figure 7: Examples of NG-DSAC performing sub optimally. (Top) Image of a rural mountain landscape (Bottom) An image requested to output at dimension 512x512 using a model trained at 256x256.

Appendix

Individual Contributions

Michael Cox Michael migrated the NG-DSAC code to a Google Colab server and oversaw the testing of initial and subsequent parameter tuning. He also created and analyzed application test images as well as collaborating on all written deliverables.

Jonathan Nguyen Jonathan oversaw team objectives and deliverable schedules to ensure the project moved at pace with our deadlines. He led the research on the GoogleNet algorithm and created the initial Least Squares hypothesis used to optimize from our initial guess. He performed a large amount of the research and writing.

Xun Jia Xun provided a great deal of the mathematical expertise needed to successfully navigate this project. He led the research on the CNN+Full algorithm testing, and found a local maximum in a region that was previously untouched by our test points by migrating away from a linear model into a quadratic model .