Exploration of Knn-Classifier Algorithm for Digit Recognition

Isaac Meza

Keywords:

Abstract. In this Lab, I implemented a K-Nearest-Neighbor Classifier function for identifying handwritten digit recognition while exploring the impact of differing k values and HLS Optimization techniques on Area, Accuracy, and Latency. K values negligibly impact area, while significantly impacting latency. K values 1-2 have a much higher error rate when compared to k values ≥ 3. However, there are no signficant improvements to accuracy past $k = 3$. Array Partitioning showed a large decrease in latency in comparison to an unoptimized program (k=3) with a modest increase to area. Loop unrolling had a neglible impact on latency while greatly increasing area. When these two optimizations are used together, there is an order of magnitude decrease to latency and a great increase to area.

1 K-Value Exploration

In this section I explored the impacts of different K values in terms of area, accuracy, and latency. I have plotted the results for each K value in Figure 1. I simulated for k values 1-3 as well as 4-5 to see a better picture on how these key metrics are impacted. For area, we can see that our area count fluctuates and increases by a small amount by the k-value. As a result, our k value has a small impact on Area needed. For latency, we see that our clock cycles increase dramatically as we move from $k = 1$ to $k = 2$. From there, our latency increases linearly as we increment our k value. Therefore, k values has a large impact on latency. Finally, we can look at our recognition error rate for each k value. Due to the constraints of the lab, we can not use error rates $> 10\%$ and therefore $k = 2$ is an invalid pick. We also get very close to our threshold with $k = 1$. For $k = 3, 4, 5$ we stabilize around an error rate of $\approx 7.2\%$. In conclusion, since we see worse latency and do not see any major improvements in accuracy past $k = 3$,this would be a good pick if accuracy was a priority. If minimizing latency is a priority, $k = 1$ would be a good pick since it offers the lowest latency without going past the 10% error threshold.

2 HLS Unrolling and Array Partitioning with k=3

For this section, I ran 3 simulations using loop unrolling, array partitioning, and a combination of both. The results were plotted in Figure 2. My process for this section began with running an unoptimized simulation and locating my bottlenecks. This first simulation gave me a very high latency of 91,968 clock cycles and a total area of 2617 components. I then analyzed the generated report and discovered that we have a very large loop trip count of 18,140. The vast majority (18,000) of the trip count resided in just the inner training loop, identifying our bottleneck.

2.1 Loop Unrolling

After identifying this bottleneck, I immediately unrolled this loop. Since we only care about minimizing latency for this lab, I also unrolled all the other small loops even though their contribution is negligible. We can see on our plots that this significantly increases area (24,433 components) as well as significantly decreasing trip count (1809). However, latency only dropped by a negligible amount at the cost of massive area increases. Since trip count is now reduced, our major bottleneck now is array access ports and thus array partitioning must be implemented to further reduce latency.

2.2 Array Partitioning

For this section, I avoided using loop unrolling and relied only on array partitioning to see the effects of this directive in isolation. I immediately partitioned knn_set, since this is our major bottleneck as it gets for every new training instance (18,000 instances of training data). Since area optimization is not the focus of this lab, I also partitioned all other arrays possible to further minimize latency. We can see that our area is much smaller than fully unrolling, however its still $\approx 3\times$ the area of our unoptimized program. We also see that our loop trip count is now our major bottleneck (18,922 trips). If we take a look at latency, we can see a major improvement by partitioning. However, this still isn't good enough as our lab constraints require us to have an order of magnitude lower latency.

Article version: 20th February 2026 h04:47 CET

2.3 Combined HLS Directives

For this section, I unrolled wherever possible and partitioned every array. Looking at our plots, we can see a very large area (26,172 components). We also see that our loop trip count has been minimized to 1800. This is the smallest we can get it as we are now completely bottlenecked by our outer training loop. We can not unroll this loop without vastly increasing area to an unreasonable amount and therefore can not optimize any further. We can see that our latency is very small (5,443 cycles) and is an order of magnitude smaller than our unoptimized program.

2.4 Conclusion

From my simulations, I discovered that on its own, loop unrolling on its own did not have any neglible improvements to latency while masssively increasing Area and would not recommend using this on its own. Array partitioning on its own was effective at reducing latency while still maintaining a modest area. However, when loop unrolling is used in combination with array partitioning, we can get incredibly fast latency at the cost of Area. If Area is a priority, array partitioning might be optimal, however, if latency is the priority then implementing partitioning in combination with unrolling is extremely effective.

3 Appendix

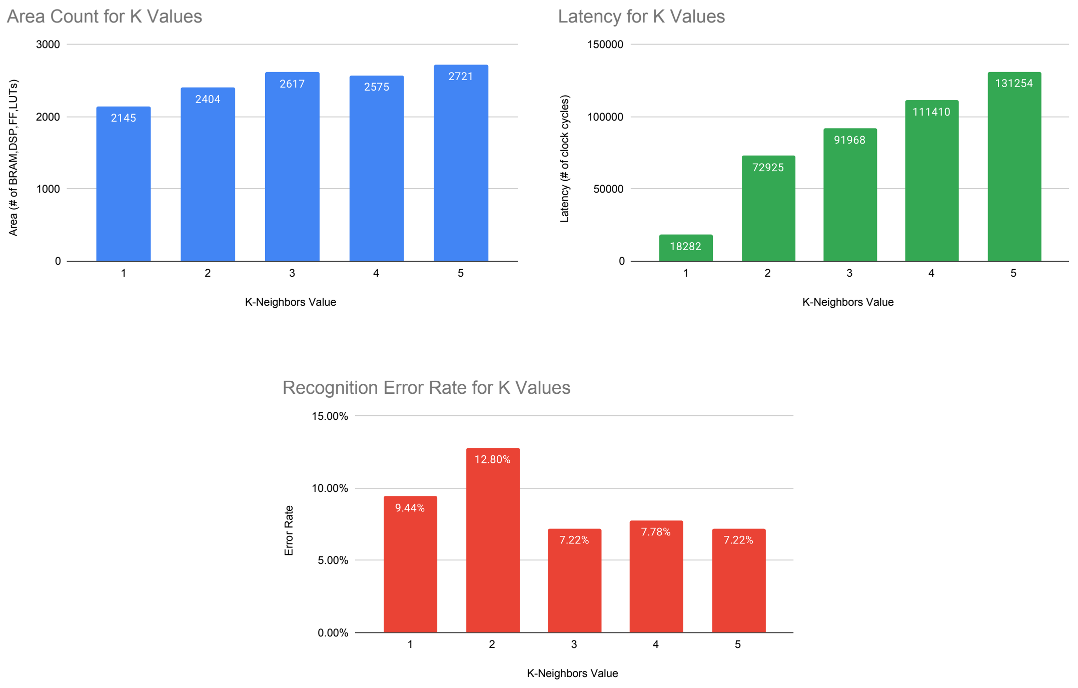

Figure 1: K-Value Exploration

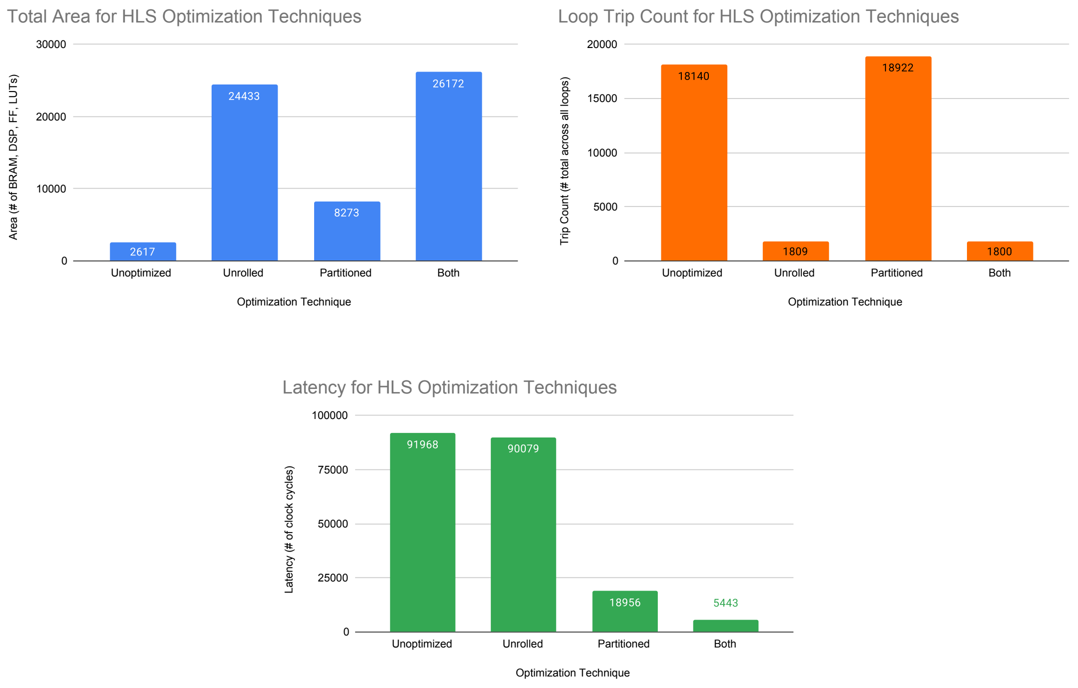

Figure 2: HLS Optimization Exploration