**KNN Accuracy vs. Latency vs. Nearest Neighbors:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nearest Neighbors** | Accuracy  (%) | FPGA Latency (s) | ARM Cortex  Latency (s) |
| 1 | 72.8 | 0.012 | **1.73** |
| 3 | 79.60 | 0.012 | **1.75** |
| 5 | 81.86 | 0.012 | **1.77** |
| 7 | 82.15 | 0.012 | **1.79** |
| 9 | 83.56 | 0.012 | **1.83** |
| 11 | 83.56 | 0.012 | **1.85** |

**KNN Accuracy vs. Latency vs. Nearest Neighbors:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Size of Training Dataset** | Accuracy  (%) | FPGA Latency (ns) | ARM Cortex  Latency (ns) |
| 50 |  |  |  |
| 100 |  |  |  |
| 150 |  |  |  |
| 200 |  |  |  |
| 250 |  |  |  |

**KNN**

**A hardware implementation of KNN algorithm is shown in Fig 2. In the built prototype architecture, two Rams are used to store the trained input feature and label trained data in order to make prediction on testing data. KNN algorithm consist of 3 main parts. In the first part absolute distance is calculated and concatenated with label trained data. After storing the concatenated values in ram, sorting algorithm is performed to sort values in ascending order. In sorting algorithm two counters are used to access the concatenated data from ram and sort the data using comparator. After sorting the values initial k nearest values are accessed again and evaluated with respect to their label.**

The data access pattern of KNN data path is shown in Fig 5, where Dist is the Manhattan distance between train and test data point, Index is the index of the each Dist stored in Ram, temp is the value used for sorting values, D is the final Decision signal.

|  |
| --- |
| **Algorithm1** Modified ANN Algorithm |
| **Input:** Cp-close prices; Hp-High prices; Lp -High prices; N-Total number of Test Data; TrainC-Trained Coefficients; Traini-Trained Intercepts; FX-Input Features; respectively X =1,2  **Output:** D signal  **for** i = 1 to N **do**    F1(i) = Hp (i) - Cp (i);  F2(i) = Cp (i) - Lp (i);  **for** j = 1 to N **do**  Dist(j) = |F1(j) - F1train(j)|;  index(j)= j;  **end for**  **for** k = 1 to N **do**  **for** l = 1 to N-1 **do**  **if** Dist(index(j))> Dist(index(j+1))  temp = index(j);  index(j) = index(j+1);  index(j+1) = temp;  **end if**  **end for**  **end for**  Buy = 0;  Sell = 0;  **for** l = 1:15 **do**  **if** ytrain(index(l)) == 1  Buy = Buy+1;  **else**  Sell = Sell+1;  **end**  **end for**  **if** Buy > Sell  D(i) = 1;  **else**  D(i) = 0;  **end if**  **end for** |
|  |
|  |