ECE 5775 Lab2 Report

-Neil Adit (na469@cornell.edu)

Abstract: k-NN is a classification algorithm that uses distance metric of k-nearest-neighbors to find the closest matching group of a particular input. We have defined distance in our case to be the total number of mismatches in the two image bit sequences. For each input image, I update the distance of its k-nearest-neighbors among groups of already labelled data of 0-9 digits. I assign the input to the class (0-9) which appears the most in the k least distance list. The error rate of the model for the given test image sequence with k=3 is **6.11%**. Further, I have compared the error rates with varying values of *k* and **optimized the latency by an order** using hardware directives of loop unrolling and array partitioning.

Section 1: Describing functions used to implement k-NN algorithm

update_knn: The function calculates the distance between the testing and training instance by counting the number of corresponding different bits in the binary strings. This distance is then compared with all elements sorted list of *k* distances to maintain the list of k-nearest-neighbors only. If the distance is larger than all the *k* elements then it is not included in the list. If not, then it is inserted at the appropriate place and all other distances greater, are shifted one places each.

```
void update_knn(digit test_inst, digit train_inst, bit6 min_distances[K_CONST]){
    int dist = 0;
    for (int i = 0; i < test_inst.length(); i++)  {
        if (test_inst[i]!=  train_inst[i]) dist = dist + 1;}
    for (int i = 0; i < K_CONST; i++){
        if (dist < min_distances[i]){
            bit6 temp = min_distances[i];
            min_distances[i] = dist;
            dist = temp;}
        if (dist == 50) break;
    }
}
```

Code Snippet 1: Function update_knn

knn_vote: At this point, we have k-nearest neighbors among each of the 10 classes to the test image. We need to find the k-shortest distances among 10Xk distances. For this, I maintain pointers to all the 10 arrays of k-nearest distances. We find minimum among the 10 array pointers and increment that pointer. The group which had most elements closest to the test sequence is the class assigned to it. In case of a tie, I go ahead with the numerically highest number (since it gave better accuracy on test sequence with k=3).

```
bit4 knn_vote(bit6 knn_set[10][K_CONST]) {
    bit4 decider[10];
    for (int i = 0; i < 10; i++) decider[i] = 0;
    for (int i = 0; i < K_CONST; i++) {
        int min = 50; int ind = -1;
        for (int j = 0; j < 10; j++) {
            if (knn_set[j][decider[j]] < min) {
                min = knn_set[j][decider[j]]; ind = j; }}
        decider[ind] = decider[ind] + 1; }
    int max = 0; bit4 cls = -1;
    for (int i = 0; i < 10; i++) {
        if (decider[i] >= max) {
            max = decider[i];
            cls = i;
        }
    }
    return cls;
}
```

Code Snippet 2: Function knn_vote

Section 2: Comparing k-values with error rate, resource utilization and latency
The table below shows that accuracy does improve on increasing k from 1-3 but then it decreases on going from 4-5. Choosing *k* is a heuristic which might vary for different applications. The resource utilization and latency increase as we go down since we need to update more k-nearest distances per class.

Table 1: The table illustrates accuracy, resource utilization and latency on increasing k in the k-NN algorithm

Design	Accuracy	Clock Period (CP)	BRAM	DSP	FF	LUT	Latency
1-nn	9.44%	6.28	96	0	387	503	975667
2-nn	7.78%	6.5	96	0	444	587	1065742
3-nn	6.11%	6.28	96	0	448	599	1119785
4-nn	6.11%	6.5	96	0	450	600	1119828
5-nn	7.78%	6.28	96	0	456	617	1173871

Section 3: HLS directives to minimize latency of the design
Loop unrolling is the most effective when some things can be done completely parallelly. In the k-NN algorithm, finding and updating the distance of the test instance to a training instance from one of 10 classes available can be completely done in parallel. So ideally, we should get a 10x speed improvement since we have reduced a major chunk of updating work by a factor of 10. Hence, I **unrolled the loop L1** (refer code snippet 3 and 4) which updates distance values with the 10 classes completely. In order to make accesses to the array in the same clock cycle, I **completely partitioned the *knn_set*** array which holds k-nearest-neighbor distances for each class. The directives were added in the run.tcl file.

```
bit6 knn_set[10][K_CONST];
for (int i = 0; i < TRAINING_SIZE; ++i) {
    L1: for (int j = 0; j < 10; j++) { // Read a new instance from the training set
        digit training_instance = training_data[j * TRAINING_SIZE + i]; //Update KNN set
        update_knn(input, training_instance, knn_set[j]);
    }
}
```

Code Snippet 3: The loop L1 is unrolled and array knn_set is partitioned

```
set_directive_unroll digitrec/L1
set_directive_array_partition digitrec knn_set
```

Code Snippet 4: The run.tcl file with HLS directives of unroll and array partition

Table 2 shows an expected order of reduction in the latency of the design. The increase in the on-chip resources is due to the parallel hardware design squeezing more computations in one cycle.

Table 2: The table illustrates accuracy, resource utilization and latency for k=3 for baseline and HLS directives optimized case

Design	Accuracy	Clock Period (CP)	BRAM	DSP	FF	LUT	Latency
No directives	6.11%	6.28	96	0	448	599	1119785
Loop unroll + array partition	6.11%	8.51	480	0	2105	2315	111815

Summary: I have effectively implemented the algorithm of k-NN algorithm using functions update_knn and knn_vote. The accuracy of the algorithm was observed by varying k-values from 1-5. The accuracy peaked for values of k=3,4 at 6.11%, which shows that optimum value of k would vary with application and is not linear. We further optimized the design of the algorithm on hardware using HLS directives and utilizing the inherent parallelism in the algorithm. We got an expected 10x speedup after unrolling a loop of count =10, completely. The array was completely partitioned to enable unrolled loop to access local memory simultaneously in the same clock cycle.